Les grammaires de configuration : un cadre grammatical moderne

Mathieu Estratat, Laurent Henocque

Les grammaires de configuration : un cadre grammatical moderne

Mathieu Estratat **Laurent Henocque**
LSIS UMR CNRS 6168
Université Paul Cézanne
Avenue Escadrille Normandie Niemen
13397 Marseille cedex 20
{mathieu.estratat,laurent.henocque}@lsis.org

Résumé

Ce papier présente une nouvelle approche pour la description de grammaires du langage naturel et pour leur utilisation lors de l'analyse. Nous définissons un modèle objet contraint qui peut être étendu pour implémenter les formalismes grammaticaux et être ensuite exploité par un configurateur pour analyser syntaxiquement des phrases. Le cadre général se nomme « Grammaires de Configuration » et peut être utilisé pour la description de grammaires de dépendences comme pour celle de grammaires syntagmatiques. Nous proposons un exemple détaillé de cette application à un formalisme génératif à base de contraintes nommé les grammaires de propriétés.

Abstract

This paper presents a novel approach to the description of natural language grammars and their use for parsing. We define an abstract constrained object model that can be extended to implement grammar formalisms, then exploited by a configurator for parsing input sentences. The whole framework called "Configuration Grammars" is suitable for the description of generative as well as dependency based grammar formalisms. We illustrate its application to a constraint based generative formalism called property grammars.

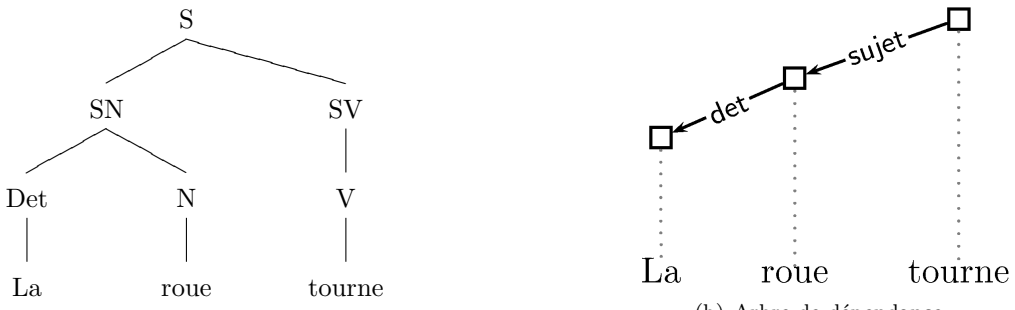
1 Introduction

Dans le domaine de l'analyse du langage naturel, dans la plupart des cas, définir une grammaire et l'implémenter sont deux activités différentes. D'un côté les grammairiens étudient le langage pour en extraire des règles, de l'autre, les informaticiens développent des algorithmes et des programmes qui utilisent ces règles pour analyser syntaxiquement (mais également sémantiquement, prosodiquement, phonétiquement, ...) des phrases. Peu de formalismes sont capables de décrire plusieurs grammaires avec un seul et même langage. Les grammaires de configuration lexicalisées [9] en sont un exemple. Ces dernières utilisent des forêts ordonnées pour représenter l'information, des principes généraux de bonne formation de ces forêts et un ensemble de contraintes pour chaque mot, définies dans un lexique, pour exploiter les propriétés de la forêt. Cependant, à notre connaissance, aucun cadre grammatical existant n'est capable de générer, pour toute grammaire, un analyseur à partir d'une description de grammaire. Notre approche le permet et nous pouvons considérer que dans celle-ci « la grammaire EST l'analyseur ». La configuration [12] est un outil pour la recherche de modèles finis qui consiste, dans le cas général, à simuler la réalisation d'un produit plus ou moins complexe à partir de composants, choisis dans un catalogue de types de composants. Différentes approches, basées sur les CSP[1] ou la programmation logique comme [14, 1, 16] par exemple, ont été proposées pour résoudre les problèmes de configuration.

1.1 Objectifs de cet article

Notre approche consiste à décrire les grammaires en tant que modèles objet contraints (en utilisant un modèle objet contraint abstrait) et permet d'obtenir directement un analyseur (également appelé parseur) à partir de cette représentation. Ce parseur est implicitement fourni par la recherche de modèles finis

[1] Constraint Satisfaction Problem



(a) Arbre syntagmatique

(b) Arbre de dépendance

FIG. 1 – Arbres syntaxiques pour la phrase « La roue tourne »

appliquée au modèle objet contraint. Nous présentons le formalisme des *grammaires de configuration*, un modèle objet contraint abstrait et les règles d'extension de ce dernier afin de capturer toute sorte de formalisme grammatical. Un modèle objet contraint est un modèle objet au sens classique du terme (représentable notamment par des diagrammes UML2 [10]) dont les règles « d'assemblage » des objets sont soumises à des contraintes. Aucun langage n'est adopté par toute la communauté pour représenter les modèles objet contraints. Pour cela, nous utilisons un langage formel « standard », le langage Z [15], pour assoir formellement les descriptions des modèles objet contraints. Ce langage possède l'avantage de reposer sur la théorie des ensembles (un modèle objet est un ensemble d'objets) et de disposer d'outils comme le « typechecker » *fuzz* qui assure une correction des types dans la définition des modèles objets contraints. Ainsi, décrire une grammaire avec le formalisme que nous proposons revient à définir sa sémantique formelle.

Les théories grammaticales se partagent en deux familles de formalismes, la première se nomme « generative-enumerative syntactic framework » (GES), la seconde, « model-theoretic syntactic framework » (MTS). Leur comparaison est proposée dans [13]. Ces deux classes héritent toutes deux de la logique classique. En effet, le formalisme GES comprend les théories utilisant des règles de réécritures, tandis que le formalisme MTS est associé à la recherche de modèles et donc à la sémantique des formules. L'approche que nous proposons appartient par définition au cadre MTS.

Les différences entre les théories grammaticales ne reposent pas seulement sur l'opposition règles d'inférence / recherche de modèles mais également, sur la manière dont elles considèrent les éléments de la phrase et les relations entre ces éléments. De nouveau, deux groupes sont à exhiber. D'une part, les grammaires syntagmatiques, héritées des travaux de Chomsky [5] et d'autre part les grammaires de dépendance, définies formellement pour la première fois par [17]. Les grammaires syntagmatiques s'interessent à la structure des phrases et vérifient que les groupes de mots sont bien formés et autorisés dans une construction donnée. Les grammaires de dépendance quant à elles, s'intéressent aux relations qu'entretiennent les mots entre eux dans la phrase. Considérons la phrase « La roue tourne », d'un côté l'analyse se fera en terme de groupe nominal (constitué d'un déterminant et d'un nom) + groupe verbal (constitué uniquement d'un verbe), de l'autre en terme de relation *sujet* entre tourne et roue et de relation *det* (pour « déterminé par ») entre roue et la. Ces deux analyses fourniront les arbres syntaxiques présentés sur la figure 1.

Cet article présente un nouveau cadre générique de formalisation dans le domaine de l'analyse du langage naturel. Ce cadre est suffisamment expressif pour les problèmes traités et il génère automatiquement, à partir de la description de toute grammaire, l'analyseur correspondant. Nous détaillons un exemple d'application provenant des grammaires syntagmatiques : l'application aux grammaires de propriétés [4]. Nous illustrons de plus la facilité d'extension de notre formalisme pour prendre en compte tout type de grammaire et sa généricité par son application aux grammaires de configuration lexicalisées. Cette dernière remarque est accentuée par la présentation dans [7] de l'application des grammaires de configuation à la grammaire de dépendence définie dans [6].

1.2 Plan de l'article

La section 2 présente la formalisation des grammaires de configuration, la section 3 décrit la formalisation de toute grammaire de propriété en grammaire de configuration. La section 4 propose l'application de la configuration comme outil de recherche de modèles finis sur le modèle objet contraint résultant. Enfin, la section 5 conclue sur l'utilisation, les performances et les limites de ces techniques dans le cadre de ces recherches.

2 Formalisation des grammaires de configuration

Nous proposons une spécification en langage Z du modèle abstrait des grammaires de configuration.

2.1 Définition des types

Tout élément utilisé dans notre approche est du type *Object*, définit ci-dessus :

$$[Object]$$

La notation ci-dessus est la déclaration d'un type non-interprété en Z. Les objets utilisés sont groupés en classes [2] :

$$Class == \mathbb{P}\ Object$$

Les classes définissent des ensembles d'objets qui partagent les mêmes propriétés. Un objet membre d'une classe est appelé une instance de cette classe.

2.2 Définition du modèle objet

Le noyau de notre formalisme est constitué des quatre classes suivantes :

$$\begin{array}{|l} Text : Class \\ Sentence : Class \\ Word : Class \\ Cat : Class \\ \hline Sentence \subseteq Cat \end{array}$$

Un texte (*Text*) est une séquence de phrases (*Sentence*), elles-mêmes construites à partir de séquences de mots (*Word*). L'expression $Sentence \subseteq Cat$ signifie que toute instance de la classe Sentence est aussi une instance de la classe Cat, on dit que la classe Sentence hérite de la classe Cat ou encore qu'elle en est une classe fille ou une sous-classe. Nous allons voir plus loin comment définir des attributs et des relations entre les classes, cependant nous pouvons dire d'ores et déjà qu'une classe fille hérite des attributs et relations définis pour sa classe mère (ou super-classe).

2.2.1 La classe Cat

La classe *cat* est la classe abstraite dont hériteront les futures classes adjointes au modèle lors de la définition de grammaires, elle est le lien entre la donnée brute que constitue le texte à analyser et la grammaire utilisée. Le nom *cat* donné à cette classe est issu des théories linguistiques qui appellent en général les éléments syntaxiques des « *catégories* ». Les classes filles de cat diffèreront en fonction du formalisme décrit :

- pour une grammaire syntagmatique, les sous-classes de cat seront des classes telles que N, Pro, etc... qui caractérisent les éléments terminaux (les mots) et SN, SV, etc... qui représentent les symboles non-terminaux ou groupes.
- pour une grammaire de dépendance, elles seront telles que N, Pro, etc... qui caractérisent chaque mot et det, sujet, etc... qui caractérisent les liens entre ces mots.

2.2.2 Attributs

Un attribut d'une classe est une variable locale à cette classe qui pour une instance donnée va prendre sa valeur dans son domaine de définition. Un attribut d'une classe est défini par une fonction de la classe vers le domaine de valeurs de l'attribut. Seules les catégories ont des attributs [3] :

$$\begin{array}{|l} phon : Cat \longrightarrow String \\ begin : Cat \longrightarrow \mathbb{N} \\ end : Cat \longrightarrow \mathbb{N} \end{array}$$

- *phon*[4] se veut être la notation phonétique associée aux mots constituant une catégorie.
- *begin* représente la position de début de la catégorie, *end* sa position de fin.

2.2.3 Relations

La classe sentence est une sous-classe de la classe Cat, ainsi, toute phrase dispose des attributs begin et end représentant respectivement les positions de début et de fin d'une instance de cat. Nous définissons ci-dessous les relations *constituents* et *constituentsRev* qui représente les phrases contenues dans un texte [5]

[2] Le symbole $\mathbb{P}$ est le symbole de power-set, le symbole $==$ représente un raccourcis de notation

[3] Le symbole $\longrightarrow$ représente une fonction

[4] le type *String* n'est pas défini par défaut en Z, nous le déclarons : $[String]$

[5] $\nrightarrow$ représente une fonction partielle, l'opérateur infixe $\#$ est l'opérateur de cardinalité, $\Leftrightarrow$ est le symbole de l'équivalence logique.

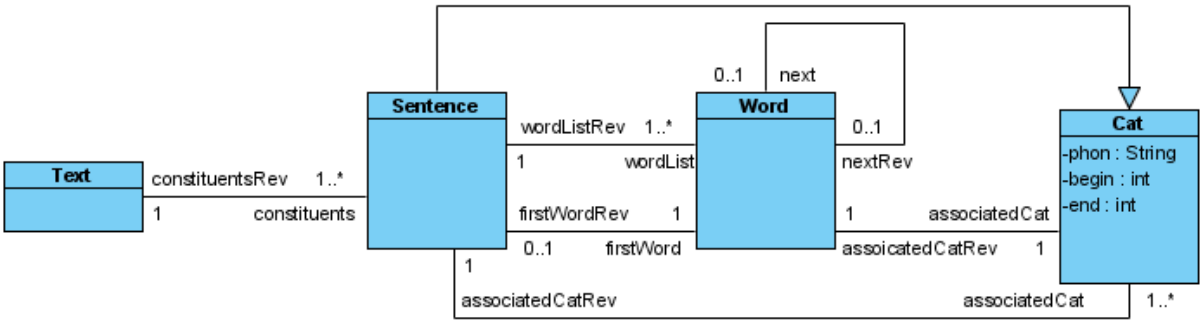


FIG. 2 – Modèle UML pour les grammaires de configuration

De plus, les phrases, dans un texte, sont ordonnées (par leurs attributs) :

$$
\begin{array}{|l}
constituents : Text \longrightarrow \mathbb{P}\ Sentence \\
constituentsRev : Sentence \rightarrowtail\!\!\!\!\!\!\rightarrow Text \\
\hline
\forall\, t : Text \bullet \#(constituents(t)) \geq 1 \\
\forall\, t : Text;\ s : Sentence \bullet s \in constituents(t) \Leftrightarrow \\
\qquad t = constituentsRev(s) \\
\forall\, t : Text \bullet \forall\, s_1, s_2 : constituents(t) \mid s_1 \neq s_2 \bullet \\
\qquad ((begin(s_1) < begin(s_2)) \vee \\
\qquad (begin(s_2) < begin(s_1)))
\end{array}
$$

Toute contrainte s'écrit en suivant le schéma suivant : *déclaration* | *conditions* • *relation entre les variables déclarées*. L'ensemble des phrases grammaticalement valides correspondent à un sous-ensemble de catégories (par exemple, une phrase en français peut être de type groupe nominal (SN) suivit d'un groupe verbal (SV)), c'est ce que traduit la fonction *associatedCat* ci-dessous :

$$
\begin{array}{|l}
associatedCat : Sentence \longrightarrow \mathbb{P}\ Cat \\
associatedCatRev : Cat \rightarrowtail\!\!\!\!\!\!\rightarrow Sentence \\
\hline
\forall\, s : Sentence \bullet \#(associatedCat(s)) \geq 1 \\
\forall\, s : Sentence;\ c : Cat \bullet c \in associatedCat(s) \Leftrightarrow \\
\qquad s = associatedCatRev(c)
\end{array}
$$

Une phrase contient une liste de mots, de plus, elle connait le premier mot de cette liste :

$$
\begin{array}{|l}
firstWord : Sentence \longrightarrow Word
\end{array}
$$

$$
\begin{array}{|l}
wordList : Sentence \longrightarrow \mathbb{P}\ Word \\
wordListRev : Word \rightarrowtail\!\!\!\!\!\!\rightarrow Sentence \\
\hline
\forall\, s : Sentence \bullet \#(wordList(s)) \geq 1 \\
\forall\, s : Sentence;\ w : Word \bullet w \in wordList(s) \Leftrightarrow \\
\qquad s = wordListRev(w) \\
\forall\, s : Sentence \bullet \\
\qquad \forall\, w : firstWord(s) \bullet w \in wordList(s)
\end{array}
$$

Dans une phrase, les mots sont ordonnés :

$$
\begin{array}{|l}
next : Word \rightarrowtail\!\!\!\!\!\!\rightarrow Word \\
nextRev : Word \rightarrowtail\!\!\!\!\!\!\rightarrow Word \\
\hline
\forall\, w_1 : Word;\ w_2 : Word \bullet w_2 \in next(w_1) \Leftrightarrow \\
\qquad w_1 = nextRev(w_2)
\end{array}
$$

Dans toute phrase non-ambigüe, tout mot est associé à exactement une seule catégorie. Dans une phrase ambigüe, les mots sont potentiellement associables à plusieurs étiquettes mais pour une analyse syntaxique donnée, une seule étiquette lui est asociée. La fonction *associatedCat* associe à chaque mot, pour une analyse donnée, une et une seule catégorie[6].

$$
\begin{array}{|l}
associatedCat : Word \rightarrowtail Cat \\
associatedCatRev == associatedCat^{\sim}
\end{array}
$$

Cet ensemble de fonctions et de types représente un modèle objet (contraint)[7] Il existe une bijection entre

[6] L'opérateur $^{\sim}$ est le symbole d'inverse relationnel, il ne peut être utilisé pour définir les relations inverses que dans le cas de fonctions (non partielles) ayant comme valeur un singleton (pas de $\mathbb{P}$)

[7] Constraint est ici entre parenthèses car aucune contrainte de compatibilité n'a été définie. Seules les contraintes relationnelles (définissant des contraintes sur les relations entre les éléments, telles que la cardinalité des relations) sont définies.

ces définitions de classes, objets, relations et les diagrammes de classe UML2[8] [10]. Le modèle objet ainsi défini peut donc être représenté par un diagramme de classes UML comme illustré sur la figure 2. Ce modèle objet est le modèle objet abstrait que nous utilisons pour implémenter les formalismes grammaticaux. Implémenter une grammaire dans notre formalisme, nécessite l'introduction dans le modèle de nouvelles sous-classes de la classe *Cat*. Nous allons illustrer ces créations par l'implémentation de deux grammaires : une théorie syntagmatique appelée les grammaires de propriétés et une théorie à base de dépendances, appelée les grammaires de configuration lexicalisées.

3 Formalisation d'une grammaire de propriétés

Les grammaires de propriétés[3, 4] est un formalisme issu des théories syntagmatiques. Les grammaires issues de cette théorie considèrent les phrases comme des ensembles ordonnés de syntagmes (syntagme nominal, syntagme verbal, ...) qui eux-mêmes sont composés de syntagmes ou de mots. Les analyses effectuées par ces grammaires produisent des arbres syntaxiques comme celui de la figure 1(a). Les grammaires de propriétés, comme de nombreuses théories linguistiques, utilisent des structures de traits pour représenter les étiquettes des arbres syntaxiques. Tout mot utilisé dans un texte est associé à une catégorie. Ce type de catégorie est appelé *catégorie terminale* car elles ne peuvent pas contenir d'autres catégories. Les catégories qui peuvent contenir d'autres catégories sont appelées des *catégories non-terminales*. Dans ce formalisme, les règles de bonne formation des catégories non-terminales sont nommées des *propriétés*.

3.1 Formalisation des catégories

Les catégories sont représentées par des structures de traits qui peuvent contenir des attributs et des propriétés. La figure 3 illustre la catégorie *NP* qui peut être utilisée pour représenter un groupe nominal (Noun Phrase). Dans notre approche, les catégories correspondent aux classes, les attributs à des attributs de classes, et les propriétés à des contraintes sur les classes. Le trait FEAT. représente les attributs de la catégorie et la partie PROPS. définit les propriétés. Dans la suite, XP représente un syntagme générique, $X_1, .., X_n$, $Y_1, .., Y_m$ et $Z_1, .., Z_p$ des catégories et X, Y et Z des ensembles de catégories ($X = \{X_1, .., X_n\}$, $Y = \{Y_1, .., Y_m\}$, $Z = \{Z_1, .., Z_p\}$). XP, en tant que syntagme générique, peut être remplacé par n'importe quel syntagme.

[8] Unified Modeling Language

$$\text{NP}\left[\begin{array}{ll}\text{FEAT.} & \left[\begin{array}{ll}\text{GENDER} & fem\,;masc\\ \text{NUMBER} & sing\,;plur\end{array}\right]\\ \text{PROPS.} & \left[\begin{array}{ll}\text{HEAD} & N \veebar Pro\\ \text{FACULT.} & Det\,;Adj\\ \text{UNIC.} & Det\,;Adj\\ \text{REQ.} & Det \Rightarrow N\\ \text{EXCL.} & Adj \nLeftrightarrow Pro\\ & Det \nLeftrightarrow Pro\\ \text{LIN.} & Det \prec N\\ \text{AGR.} & N.\text{GEN} \equiv Det.\text{GEN}\\ & N.\text{NB} \equiv Det.\text{NB}\end{array}\right]\end{array}\right]$$

FIG. 3 – La catégorie NP (Noun Phrase) représentant un groupe nominal

3.2 Formalisation des attributs

fem, *masc*, *sing* et *plur* sont des objets particuliers de l'univers et sont décrits ci-dessous :

$$fem, masc, sing, plur : Object$$

A partir de ces spécifications, les fonctions suivantes vont implémenter les attributs de classes :

$$XPGender : XP \longrightarrow \{fem, masc\}$$
$$XPNumber : XP \longrightarrow \{sing, plur\}$$

3.3 Formalisation des propriétés

Les propriétés sont des contraintes sur les catégories, elles sont utilisées pour représenter les règles de bonnes formation des phrases. Nous présentons les propriétés qui assurent une bonne représentation pour une grammaire générale. Cependant, il est toujours possible d'ajouter d'autres propriétés. Ces propriétés seront traductibles dans le formalisme choisi puisque ce dernier est plus expressif que celui des grammaires de propriétés. Les propriétés sont définies dans la littérature [3, 4, 8], nous rappellons ici les sept propriétés présentées dans [4] :

$$\text{XP}\left[\begin{array}{ll}\text{PROPS.} & \left[\begin{array}{ll}\text{HEAD} & \left\{X_1, .., X_n\right\}\\ \text{FACULT} & \left\{Y_1, .., Y_m\right\}\\ \text{UNIC} & \left\{Z_1, .., Z_p\right\}\\ \text{REQ.} & \left\{X_1, .., X_n\right\} \Rightarrow \left\{Y'_1, .., Y'_m\right\}\\ \text{EXCL.} & \left\{X_1, .., X_n\right\} \nLeftrightarrow \left\{Y_1, .., Y_m\right\}\\ \text{LIN.} & X \prec Y\\ \text{AGR.} & att(X_i) \equiv att(Y_j)\end{array}\right]\end{array}\right]$$

Ces septs propriétés représentent :

– head : propriété de tête. Une tête est obligatoire et unique dans tout syntagme. C'est l'élément central du syntagme (comme, par exemple, le verbe dans un groupe verbal), c'est notamment lui qui va porter l'accord du syntagme.
– facult : propriété de facultativité. Cet ensemble de catégories représente les éléments, non-tête, pouvant entrer dans la construction d'un syntagme.
– Unic : propriété d'unicité. Cette propriété définit qu'elles sont, parmis les catégories facultatives, celles qui ne peuvent pas apparaître plus d'une fois dans le syntagme.
– req. : propriété d'exigence. Cette contrainte définit les obligations de coocurrence entre catégories dans un syntagme.
– excl. : propriété d'exclusion. Cette contrainte définit les obligations de non-coocurrence entre catégories dans un syntagme.
– agr. : propriété d'accord. Cette propriété définit comment s'accordent les éléments constituant le syntagme.

Les propriétés de tête (*Head*) et de facultativité (*Facultative*) sont formalisées par des fonctions sur le modèle objet, par exemple, la tête d'un SN est un N ou un Pro. Les deux relations NPHN et NPHPro sur la figure 5 représentent ces relations. Cependant, une tête étant unique et obligatoire, il est nécessaire d'exprimer que la somme des cardinalités de ces deux relations doit être égale à 1, c'est ce que traduit la contrainte suivante :

$$\forall\, np : NP \bullet \#NPHN(np) + \#NPHPro(np) = 1$$

A partir des deux sous-ensembles de catégories *Head* et *Facultative*, il est possible de définir l'ensemble des catégories appartenant à un syntagme. Nous nommons cet ensemble *Const* (pour constituants) et le définissons par l'union des ensembles têtes et facultatifs. Les cinq autres propriétés sont formalisées par des contraintes. Nous présentons sur la figure 4 leur formalisation à l'aide de la catégorie générique XP.

4 Exemples

Tout comme le formalisme des modèles objets contraints, le formalisme de la configuration est correct et complet mais reste semi-décidable. Pour les deux exemples suivants, nous présentons dans un premier temps la traduction de la grammaire dans notre formalisme puis des résultats qualitatifs[9].

[9]Les implémentations présentées dans cet article ont été réalisées sur un pentium M 2GHz, avec 1024 Mo de RAM sous le système d'exploitation Microsoft Windows XP SP2 en utilisant le configurateur Ilog JConfigurator 2.19.

Unique :

$$\forall\, xp : XP;\ z : Z \bullet \#(XPConst(xp) \cap z) \leq 1$$

Requirement :

$$\forall\, xp : XP \bullet \\ \forall\, x : X \bullet (XPConst(xp) \cap x \neq \emptyset) \Rightarrow \\ (\exists\, y : Y' \bullet \forall\, y' : y \bullet XPConst(xp) \cap y' \neq \emptyset)$$

Exclusion :

$$\forall\, xp : XP \bullet \forall\, x : X;\ y : Y \bullet \\ ((XPConst(xp) \cap x \neq \emptyset) \Rightarrow \\ (XPConst(xp) \cap y = \emptyset)) \\ \wedge\ ((XPConst(xp) \cap y \neq \emptyset) \Rightarrow \\ (XPConst(xp) \cap x = \emptyset))$$

Linearity :

$$\forall\, xp : XP \bullet \forall\, x : X;\ y : Y \mid \\ ((x \cap XPConst(xp) \neq \emptyset) \\ \wedge\ (y \cap XPConst(xp) \neq \emptyset)) \bullet \\ \forall\, x' : x;\ y' : y \bullet end(x') \leq begin(y')$$

Agreement :

$$\forall\, xp : XP \bullet \forall\, x : X_1;\ y : Y_1 \mid \\ x \in XPConst(xp) \wedge y \in XPConst(xp) \bullet \\ Xatt(x) = Yatt(y)$$

FIG. 4 – Formalisation des propriétés d'unicité, d'exigence, d'exclusion de linéarité et d'accord

4.1 Une grammaire de propriété pour analyser syntaxiquement des syntagmes nominaux

Un syntagme nominal (NP pour noun phrase) est représenté dans le formalisme des grammaires de propriétés par une structure de traits comme celle définie sur la figure 3. A partir de cette description du NP et des règles de traduction vues en section 3, la figure 5 propose un modèle objet correspondant à cette structure de traits. Sur cette figure, les catégories apparaissent dans la définition du NP qui hérite de la classe cat. Les cardinalités des relations *NPAdj* et *NPDet* sont égales à 0..1, elles prennent ainsi en compte la propriété d'unicité. Nous avons implémenté un générateur de problèmes de configuration qui prend en entrée un fichier LaTeX contenant la description de la grammaire de propriétés (comme celle présentée sur

la figure 3). Cette première étape du générateur vérifie que la grammaire est bien formée. Par exemple, elle vérifie que toutes les catégories sont bien définies et qu'il n'y a pas d'appel à une catégorie inconnue. Ensuite, pour chaque phrase, le programme génère un ensemble de classes et de relations dans le modèle objet du problème de configuration en construction. Le

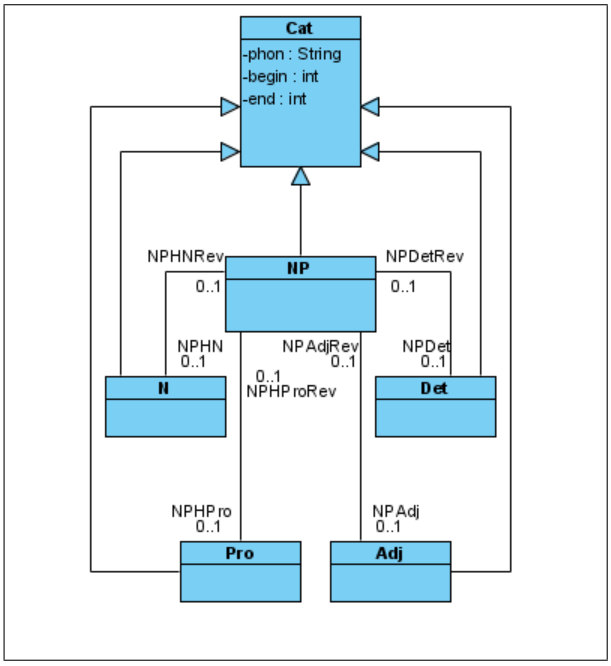


FIG. 5 – Diagramme UML pour l'analyse de syntagme nominal (NP)

Head :

$$\forall\, np : NP \bullet \#NPHN(np) + \#NPHPro(np) = 1$$

Requirement :

$$\forall\, np : NP \bullet \\ (\#NPDet(np) = 1) \Rightarrow (\#NPHN(np) = 1)$$

Exclusion :

$$\forall\, np : NP \bullet \\ ((\#NPAdj(np) = 1) \Rightarrow (\#NPHPro(np) = 0)) \wedge \\ ((\#NPHPro(np) = 1) \Rightarrow (\#NPHAdj(np) = 0)) \\ \forall\, np : NP \bullet \\ ((\#NPDet(np) = 1) \Rightarrow (\#NPHPro(np) = 0)) \wedge \\ ((\#NPHPro(np) = 1) \Rightarrow (\#NPHDet(np) = 0))$$

Linearity :

$$\forall\, np : NP \mid \\ (\#NPDet(np) = 1) \wedge (\#NPHN(np) = 1) \bullet \\ end(NPDet(np)) < begin(NPHN(np))$$

Agreement :

$$\forall\, np : NP \mid \\ (\#NPDet(np) = 1) \wedge (\#NPHN(np) = 1) \bullet \\ gen(NPDet(np)) = gen(NPHN(np)) \\ \forall\, np : NP \mid \\ (\#NPDet(np) = 1) \wedge (\#NPHN(np) = 1) \bullet \\ num(NPDet(np)) < num(NPHN(np))$$

FIG. 6 – Contraintes pour l'analyse d'un SN

modèle objet seul n'est pas assez expressif pour représenter toute la grammaire, les propriétés sont traduites en contraintes présentées sur la figure 6. Cette grammaire, comme toute autre, nécessite un lexique où, pour chaque mot rencontré, l'analyseur peut aller chercher les catégories associables. Par exemple ; la figure 7 présente une partie d'un lexique du français. Sur ce tableau, la première colonne représente le mot, la seconde la catégorie et dans les troisième, quatrième et cinquième colonnes, les valeurs pour les attributs de genre, de nombre et de personne. A partir de cette grammaire et de ce lexique, nous pouvons analyser des syntagmes nominaux simples comme : « la roue bleue ». Le processus d'analyse génère en premier lieu des instances des classes Text, Sentence et Word en rapport avec la phrase saisie (et lie ces ensembles entre-eux). Ensuite, les instances des classes issues de la classe cat sont générées en suivant les informations du lexique. Finallement, le parseur tente de trouver une solution avec l'ensemble de ces éléments.

Le résultat de ce processus d'analyse sur le syntagme nominal « la roue bleue » contient les objets suivant : {t :Text, s :Sentence, m_la :word, m_roue :word, m_bleue :word, la :Det, bleue :Adj, roue :N}[10].

L'exemple de grammaire de propriétés traité ici est volontairement simple pour faciliter la compréhension du mécanisme des grammaires de configuration. Notre approche a été testée avec une grammaire couvrant un sous-ensemble du français constitué de phrases acceptant :

- des syntagmes verbaux dont les compléments peuvent être des syntagmes nominaux, adjectivaux ou adverbiaux,
- des syntagmes nominaux plus complexes (avec, par exemple, des syntagmes prépositionnels im-

[10]La notation a :b signifie que l'objet nommé a est une instance de la classe (est de type) b. Les instances de mots sont précédées de « m_ » pour ne pas les confondre avec les instances des classes filles de cat.

mot	cat	genre	nombre	personne
...	...	...	...	..
bleue	Adj	fem	sing	-
...	...	...	...	
la	Det	fem	sing	3
la	Pro	fem	sing	3
...	...	...	...	
roue	N	fem	sing	3
...	...	...	...	

FIG. 7 – Partie d'un lexique du français

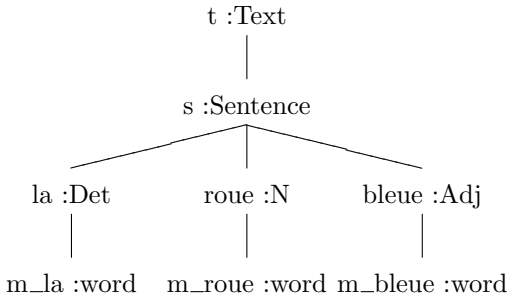


FIG. 8 – Arbre syntaxique pour le groupe nominal « la roue bleue »

briqués)
– ...

Un exemple de phrase acceptée par cette grammaire (en considérant le lexique adéquat[11]) peut être « la roue bleue de la charette de mon père a six rayons ». Des détails sur cette grammaire sont présentés dans [7].

4.2 Une grammaire de configuration lexicalisée pour analyser le scrambling language

L'intérêt de ce langage hors-contexte est la prise en compte des relations entre les mots dans un langage où l'ordre des mots est (plus ou moins) libre. Par exemple, en allemand, les noms arguments d'un syntagme verbal peuvent être permutés. Cette permutation est appelée *scrambling*. Il a été prouvé [2] qu'un formalisme peut prendre en compte le problème du scrambling si et seulement si il peut modéliser le langage suivant :

$$SCR = \{\pi(n^{[0]}, \ldots, n^{[k]})v^{[0]}, \ldots, v^{[k]}\}$$

with $k \geq 0$ *and* π *a permutation*

[11] Il est important de noter également, que la taille du lexique n'a pas d'importance sur les qualités des solutions trouvées.

Les grammaires de configuration lexicalisées [9] utilisent des contraintes, définies dans le lexique (Lex), pour construire l'analyseur. La grammaire est définie par $G_{SCR} = (\{n, v\}, \{n, v\}, Lex)$. Une entrée du lexique est représentée par le triplet $\langle A, B\,; C\rangle$, où A est l'ensemble des étiquettes d'arêtes entrantes possibles pour l'entrée considérée, B est l'ensemble des étiquettes d'arêtes sortantes et C l'ensemble des contraintes liant le nœud courant (associé à l'entrée du lexique) et les éléments de A et de B. Pour le scrambling language, les entrées lexicales sont :

– $\langle \{n\}, \varnothing;\ t\rangle$
– $\langle \varnothing, \{n, v\};\ n \prec \imath \wedge \imath \prec v \wedge \imath \bowtie v\rangle$
– $\langle \{v\}, \{n, v\};\ n \prec \imath \wedge \imath \prec v \wedge \imath \bowtie v\rangle$
– $\langle \varnothing, \{n\};\ n \prec \imath$

où $\imath$ représente le nœud courant, $\prec$ la relation de précédence linéaire et $\bowtie$ la relation d'adjacence. Ces entrées signifient respectivement :

– Tout nœud ayant une arête entrante étiqueté n et n'ayant pas d'arête sortante n'est soumis à aucune contrainte (t pour true, contrainte toujours vraie).
– Tout nœud n'ayant pas d'arête entrante et ayant une étiquette sortante étiquetée n ou v suit un nœud étiqueté n et précède directement un nœud étiqueté v ($\imath \prec v \wedge \imath \bowtie v$).
– Tout nœud ayant une arête entrante étiquetée v et ayant une étiquette sortante étiquetée n ou v suit un nœud étiqueté n et précède directement un nœud étiqueté v ($\imath \prec v \wedge \imath \bowtie v$).
– Tout nœud ne possédant pas d'arête entrante et ayant une arête sortante étiquetée n doit avoir au moins un nœud étiqueté n placé avant lui.

La figure 9 représente les classes introduites dans le modèle objet des grammaires de configuration pour prendre en compte le scrambling language. La figure 10

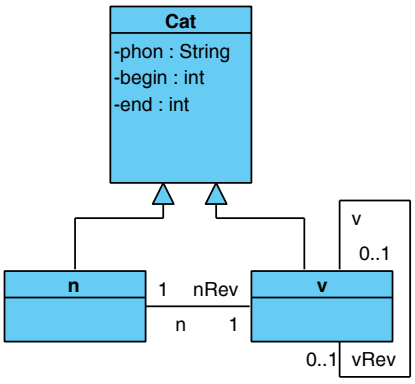


FIG. 9 – Modèle objet pour le scrambling language

quant à elle présente les contraintes associées au modèle objet et correspondant aux contraintes du lexique

des grammaires de configuration lexicalisées.

$$\forall\, v_i : v \bullet (v_i.n.fin \leq v_i.debut)$$

$$\forall\, v_i : v \bullet (v_i.v.debut = (v_i.fin) + 1)$$

$$\forall\, v_i : v \bullet \\ (\#(v_i.catAssociee^{\sim}.suivant) = 0) \vee \\ (v_i.v = v_i.catAssociee^{\sim}.suivant.catAssociee)$$

$$p.catAssociee \cap v \neq \varnothing \\ \forall\, v_i : v \mid v_i = p.catAssociee \bullet \#(v_i.v^{\sim}) = 0$$

FIG. 10 – Contraintes pour le scrambling language

Avec cette définition de grammaire, analyser une séquence de mots telle que « nnvv » produit :

- un ensemble d'objets : {t :Text, s :sentence, n_1 :word, n_2 :word, v_1 :word, v_2 :word, n_1 :n, n_2 :n, v_1 :v, v_2 :v} tels que les vs sont ordonnés et les ns ne le sont pas.
- les deux solutions (où $\forall\, i \in \{0,1\}, x \in \{n,v\}$ x_i : *word* est lié à $x_i : x$) :
  - l'une où v_1 est lié à n_1 et v_2 à n_2
  - l'autre où v_1 est lié à n_2 et v_2 à n_1
  - et seulement ces deux solutions

Cet analyseur peut être utilisé aussi bien de manière analytique ou générative.

5 Conclusion

Nous avons présenté les *Grammaires de Configuration*, un nouveau cadre de traitement pour les grammaires du langage naturel. Ce cadre est basé sur un modèle objet contraint abstrait. Ce modèle peut être étendu pour capturé des formalismes grammaticaux de divers horizons. L'utilisation du langage Z assure une sémantique forte au modèle. Plus spécifiquement, nous attachons une sémantique formelle aux propriétés des Grammaires de Propriétés, sémantique non présente dans le travail original.

Toute grammaire possède son propre langage d'expression et une question intéressante est de savoir quel est l'intérêt de ré-écrire les grammaires existantes dans un « autre nouveau » formalisme. Premièrement, cette ré-écriture permet de comparer les différentes grammaires en se basant sur un unique langage, en effet, ces grammaires ne pouvaient pas être comparées avant puisqu'elles ne partagent pas les mêmes notions. Deuxièmement, le parseur produit peut être utilisé aussi bien de manière analytique que générative. Cette propriété, triviale pour le scrambling language, reste vraie pour les grammaires appliquées au langage naturel. Ainsi, notre approche permet de tester les grammaires du point de vue de la surgénération et de la sous-génération d'une grammaire.

Certains principes et leur formalisation peuvent être développés pour limiter le nombre de contraintes. Par exemple, pour chaque relation réflexive (les attributs dom et ran de la fonction sont égaux), une contrainte telle que « $\forall\, c : \text{dom} \bullet index(relation(c))! = index(c)$ » doit être définie pour ordonner les éléments. Cette contrainte peut être replacée par un principe du type : « toute relation est ordonnée ». Ce principe peut être implémenté comme une relation universelle de Cat vers Cat qui représente toutes les autres relations et qui dispose des propriétés adéquates.

Il existe de nombreux travaux dans le domaine de la configuration pour améliorer les résultats de la recherche de modèles, comme les problèmes d'élimination de symétries [11] par exemple. Ces travaux, en améliorant la recherche de modèles finis, fourniront, sans nul doute, une aide supplémentaire à notre processus de résolution.

Références

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps—-application to configuration. *Artificial Intelligence*, 135(1-2) :199–234, 2002.

[2] Tilman Becker, Owen Rambow, and Michael Niv. The derivational generative power of formal systems, or, scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for research in cognitive science, Philadelphia, PA, 1992.

[3] Philippe Blache. Property grammars and the problem of constraint satisfaction. In *ESSLLI-2000 workshop on Linguistic Theory and Grammar Implementation*, pages 47–56, 2000.

[4] Philippe Blache. *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles.* Hermès Sciences, 2001.

[5] Noam Chomsky. *Aspects of the Theory of Syntax.* MIT Press, 1965.

[6] Denys Duchier and Ralph Debusmann. Topological dependency trees : A constraint-based account of linear precedence. In *proceedings of the 39th Annual Meeting of the Association for Compu-*

tational Linguistics (ACL 2001), pages 180–187, 2001.

[7] Mathieu Estratat. *Vers les grammaires de configuration*. thèse de doctorat, Université Paul Cézanne - Aix-Marseille III, Novembre 2006. directeur : Laurent Henocque.

[8] Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *proceedings of the European Conference for Artificial Intelligence ECAI 2004*, pages 591–595, Valencia, Spain, August 2004.

[9] Robert Grabowski, Marco Kuhlmann, and Mathias Möhl. Lexicalised configuration grammars. In *Second International Workshop on Constraint Solving and Language Processing (CSLP 2005)*, Sitges, Spain, 2005.

[10] Object Management Group. UML. Technical Report 2.0, OMG, 2004.

[11] Laurent Henocque, Mathias Kleiner, and Nicolas Prcovic. Advances in polytime isomorph elimination for configuration. In *proceedings of Principles and Practice of Constraint Programming - CP 2005,*, pages 301–313, Sitges Barcelona, Spain, 2005. Springer.

[12] Daniel Mailharro. A classification and constraint based framework for configuration. *AI-EDAM : Special issue on Configuration*, 12(4) :383 – 397, 1998.

[13] Geoffrey K. Pullum and Barbara C. Scholz. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retoré, editors, *'Logical Aspect of Computational Linguistics : 4th International Conference' (Lecture Notes in Artificial Intelligence - 2099)*, pages 17–43, 2001.

[14] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen. Unified configuration knowledge representation using weight constraint rules. In *Workshop Notes of the ECAI'2000 Configuration Workshop*, pages 79–84, Berlin, Germany, August 2000.

[15] J. M. Spivey. *The Z Notation : a reference manual*. Prentice Hall originally, now J.M. Spivey, 2001.

[16] Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. *Advances in Artificial Intelligence : Proceedings of the Third Congress of the Italian Association for Artificial Intelligence AI*IA'93*, pages 302–313, 1993.

[17] Lucien Tesnière. *Eléments de syntaxe structurale*. Klincksiek, Paris, 1959.