Traduction automatique de grammaires à base de traits en un problème de configuration syntaxique

Mathieu Estratat and Laurent Henocque

LSIS UMR CNRS 6168,
Université Paul Cézanne,
Avenue Escadrille Normandie Niemen,
Marseille, France

{mathieu.estratat, laurent.henocque}@lsis.org

Abstract

Nous présentons un outil effectif de traduction automatique d'une grammaire de propriété en un problème de configuration sous contraintes. Un template Latex (utilisant les AVMs[1] standards) est utilisé pour spécifier la grammaire et le générateur prend en entrée les fichiers Latex utilisés pour la documenter. Ainsi, notre contribution propose à la fois une validation syntaxique des grammaires de propriété (l'outil détecte les inconsistances dans la grammaire au niveau de la formulation) et une manière de vérifier la correction et la consistance de la grammaire (en détectant les erreurs et/ou les étiquetages manquant des phrases de test). L'approche pourrait se voir étendue à d'autres théories linguistiques basées sur les contraintes.

1 Introduction

Les *Grammaires de Propriété* [3, 4](GP) définissent un formalisme linguistique permettant l'analyse du langage naturel. Ce formalisme repose entièrement sur un fonctionnement à base de contraintes. Dans [5] nous proposons l'utilisation de la configuration à base de contraintes pour implémenter des analyseurs syntaxiques utilisant le formalisme des grammaires de propriété. Nous continuons ici ce travail en automatisant la traduction des grammaires en un problème de configuration exploitable directement. Pour ce faire, les grammaires sont représentées avec le formalisme des GP et écrites dans un fichier LaTeX. Ainsi, la contribution essentielle de ce travail est d'automatiser le processus de construction du modèle objet contraint utilisé par le configurateur pour analyser syntaxiquement(nous utiliserons indépendemment les termes *analyser syntaxiquement* ou *parser*) des phrases.
L'utilisation d'un générateur offre certains avantages, comme notamment, celui d'éviter le coût de création d'un modèle objet sous contraintes spécifique à chaque grammaire. Mais en plus, il offre la possibilité d'exploiter la structure du problème pour générer des heuristiques ainsi que des contraintes redondantes qui vont permettre d'améliorer la propagation et/ou de réduire la combinatoire des problèmes de parsing. Pour finir, le générateur est également capable d'utiliser des contraintes pour casser les symétries.
Dans les grammaires de propriété, chaque élément (mot ou groupe de mots) de la phrase est appellé *catégorie*. Par exemple, un nom (N), un pronom (Pro) ou un déterminant (Det) sont des catégories. De même un groupe nominal nommé syntagme nominal dans la littérature (SN), ou un syntagme verbal (SV) sont aussi des catégories. SN et SV sont appellés catégories non terminales car elles contiennent d'autres catégories (un SN contient généralement au moins un N ou un Pro) tandisque les catégories N, Det et Pro sont dites terminales. En GP les contraintes sont appellées *propriétés* et représentent les règles de bonne formation de chaque syntagme. Sept propriétés sont actuellement utilisées par les chercheurs du Labora-

[1] Attribute Value Matrices

toire Parole et Langage (LPL) où ont été dévelloppées et sont maintenues les GP. D'autres propriétés peuvent être définies mais selon les linguistes les septs présentées dans cet article sont suffisantes pour prendre en compte le type d'analyse que nous envisageons. Ces sept propriétés définissent les règles de bonne formation des syntagmes et sont présentées en section 3.1 ainsi que leur traduction en termes de contraintes de configuration.

Nous utilisons la configuration orientée objet sous contraintes [7, 8] pour résoudre le problème proposé. Problème qui s'articule non pas autour du parseur mais autour du programme qui permet de générer automatiquement un problème de configuration à partir d'une grammaire définie en Latex. Des parseurs implémentant la théorie des GP existe et nous pouvons nous référer à la thèse de Tristan VanRullen [11] qui propose un analyseur à granulaité variable mais également au parseur de Jean Marie Balfourier[1, 2]. Cependant ces analyseurs n'encapsulent pas les mêmes informations que le configurateur.

Configurer consiste à simuler la production d'un produit complexe à partir d'une ensemble d'objets choisis dans un catalogue de types. La configuration orientée objet représente les données à l'aide du paradigme orienté objet : les relations d'héritage, de composition, d'aggrégation, etc... mais également les classes et les instances. Les classes représentent les types d'objets tandisque les instances sont les objets d'un certain type. Les règles de constructions liées au modèle objet sont complétées par des contraintes pouvant porter sur les attributs, les cardinaliltés des relations, etc.... Le problème de configuration s'articule donc autour d'un modèle générique représentant de manière synthétique tous les objets réalisables et d'un ensemble de contraintes portant sur les classes et les relations de ce modèle. L'entrée d'un problème de configuration est un ensemble d'instances de classes plus ou moins complètes et plus ou moins interconnectées. Cette partie du modèle doit se retrouver impérativement dans la solution (si une solution existe). Dans la suite nous utilisons une combinaison entre les diagrammes de classes UML2[9] et la notation Z[10, 6] pour décrire la sémantique des modèles objets sous contraintes. La taille de la solution d'un problème de configuration est inconnue au départ. Et le problème est, dans le cas général, semi-décidable.

Le plan de l'article est le suivant : dans la section 2 nous intoduisons les commandes Latex définies pour que le parseur de Latex puisse générer le modèle objet sous contraintes. Dans la section 3 nous présentons le processus de traduction automatique des catégories et des propriétés. La section 4 décrit les contraintes nécessaires au problème de configuration. Dans la section 5 nous listons certains résultats expérimentaux et enfin la section 6 conlut.

2 Parser les définitions Latex des grammaires de propriété

La description de la grammaire se fait en Latex et doit utiliser le package que nous avons défini[2]. Le générateur prend en entrée un fichier Latex et construit le modèle objet sous contraintes correspondant. Durant cette première étape, la bonne formation de la grammaire est vérifiée. Par exemple, le programme vérifie que toutes les catégories sont déclarées convenablement. Ensuite le programme génère un ensemble de classes et de relations dans le modèle objet du problème de configuration en construction. Finallement une instance du problème de configuration est générée avec comme entrée les phrases à analyser.

2.1 Définitions Latex

Les catégories sont représentées par des AVM standards. Ces AVM sont divisés en deux parties, la première définit la liste des attributs de traits tandisque la seconde représente les propriétés (les contraintes). Seules les catégories non terminales possèdent des propriétés. Notre template inclut le package AVM[3]. Nous définissons des commandes supplémentaires spécifiques aux GPs qui seront reconnues comme des mots clé lors du processus de traduction. Ces commandes sont par exemple :

- `cat{}{}` : cette commande Latex attend deux arguments : le nom de la catégorie et ses spécifications(traits + propriétés). Cette commande est un mot clé qui signale au traducteur le début de construction d'une catégorie.
- `traits{}` : cette commande ne prend qu'un seul argument : la liste des traits de la catégorie sous la forme de couples (attribut, valeur).
- `\genre` : definit l'attribut *genre* de la catégorie.
- `\props` : liste des propriétés.
- `\tete`, `\facult`, ...: permettent de définir les propriétés comme décrit dans les GP.
- ...

[2] Ce style, nommé gp.sty est disponible electoniquement à l'adresse : http://www.lsis.org/estratatm/configuration/pgp/gp.sty

[3] avm.sty de Christopher Manning - disponible électroniquement à l'adresse http://nlp.standford.edu/manning/tex/avm.sty

La Figure 1 représente un source Latex qui utilise ces commandes pour décrire un SN[4]. La compilation de ce source produit l'AVM de la Figure 2. Le symbole "$*$" qui se trouve après certaines catégories permet de spécifier que la catégorie fait partie de l'ensemble des catégories pouvant être une tête du syntagme.

```
\Cat{SN}{
  \heritage{Cat}
  \traits{
    \genre{fem;masc}\\
    \nombre{sing;plur}
  }
  \props{
    \tete{N \stete Pro}
    \facult{Det \sfacult Adj}
    \unic{Det \sunic Adj}
    \exig{\sexig{Det}{N*}}
    \excl{    \sexcl{Adj}{Pro*}\\
         & \sexcl{Det}{Pro*}
    }
    \lin{ \slin{Det}{N*} }
    \acc{    \sacc{gen}{N*}{Det}\\
         & \sacc{nb}{N*}{Det}
    }
  }
}
```

Figure 1: La catégorie SN au format Latex

$$
\text{SN}\left[\begin{array}{ll}
\text{HERITAGE} & Cat \\
\text{TRAITS} & \left[\begin{array}{ll} \text{GENRE} & fem;masc \\ \text{NOMBRE} & sing;plur \end{array}\right] \\
\text{PROPS} & \left[\begin{array}{ll}
\text{TÊTE} & N \veebar Pro \\
\text{FACULTATIVITÉ} & Det\ ;Adj \\
\text{UNICITÉ} & Det\ ;Adj \\
\text{EXIGENCE} & Det \Rightarrow N* \\
\text{EXCLUSION} & Adj \not\Leftrightarrow Pro* \\
 & Det \not\Leftrightarrow Pro* \\
\text{LINÉARITE} & Det \prec N* \\
\text{ACCORD} & N*.\text{GEN} \equiv Det.\text{GEN} \\
 & N*.\text{NB} \equiv Det.\text{NB}
\end{array}\right]
\end{array}\right]
$$

Figure 2: La catégorie SN

3 Construction du modèle objet sous contraintes

Le source Latex représentant la grammaire de propriété est analysé (parsé) pour générer un modèle objet sous contraintes. Cette section détaille le traitement effectué pour chaque élément de cette grammaire. Le générateur présenté ici ne présuppose aucune information quand au genre de la grammaire analysée. Tous les éléments présentés ci-après (comme "genre", "sing", etc...) sont recueillis par le générateur lors de l'analyse du source Latex.

3.1 Traduction des catégories

Chaque catégorie présente dans le fichier Latex est traduite en une *classe* du modèle objet sous contraintes. Chaque trait est traduit en un *attribut de classe*. De plus, une liste de tous les traits disponibles pour une grammaire donnée est définie dans un autre fichier : *features.def*[5]. La Figure 3 présente un extrait de ce fichier. Chaque ligne de ce fichier est de la forme:

```
genre cat:{Det;N;SN;...}
    bool:false domain:{masc;fem;nd}
nombr cat:{Det;N;SN;...}
    bool:false domain:{sing;plur;nd}
principal cat:{V}
    bool:true domain:{true;false}
```

Figure 3: Extrait du fichier de traits

ident cat:$\{c_1, \ldots, c_n\}$ bool:T, F dom:$\{d_1, \ldots, d_n\}$ où *ident* est le nom du trait, $\{c_1, c_2, \ldots, c_n\}$ représente l'ensemble des catégories pouvant recevoir le trait *ident*, *bool* : *True*, *False* spécifie si la valeur de trait est booléenne et finallement $\{d_1, d_2, \ldots, d_n\}$ représente l'ensemble des valeurs non booléennes possibles le cas échéant. Pour une catégorie donnée, si il n'y a pas d'erreur dans la définition des traits, une classe correspondante est créée dans le modèle objet et des attributs sont définis pour traduire les traits. La Fig-

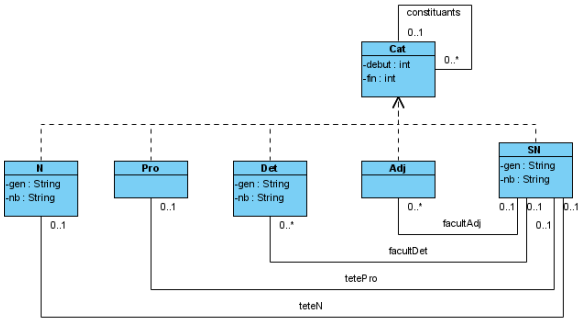

Figure 4: Classes utilisées pour représenter le *SN*

[4]Cette définition n'est pas unique et peut varier selon le type de grammaire que l'on cherche à représenter

[5]Ce fichier est disponible électroniquement à l'adresse http://www.lsis.org/ estratatm/configuration/pgp/features.def

ure 4 présente une partie du modèle objet sous contraintes construit à partir de l'exemple du SN[6].

3.2 Traduction des propriétés

Nous allons à présent décrire la manière de traduire les sept propriétés des GP. Nous illustrerons ces traductions par l'exemple du SN présenté plus haut, cependant ces propriétés seront traduites de la même manière pour tout syntagme. Lorsque les traductions ne sont pas possibles pour le SN, nous utilisons un syntagme générique "*Sx*" pour présenter la contrainte.

Nous définissons une relation supplémentaire "const" qui généralise les relations de tête et de facultativité. Cette contrainte est introduite pour des raisons techniques.

Dans cette section nous ferons appel à deux attributs *debut* et *fin* présents dans toutes les catégories, ces attributs représentent respectivement le début et la fin de chaque catégorie. Ces attributs sont définis dans la classe *Cat* dont chaque catégorie hérite.

3.2.1 Tête

Une tête est obligatoire et unique dans un syntagme. La propriété de tête définit l'ensemble des éléments pouvant être tête dans le syntagme traité. La tête est l'élément qui dirige le syntagme. Par exemple dans la définition du SN présentée sur la Figure 2, N et Pro peuvent être tête du SN. Cette propriété est traduite par des relations entre les classes SN, N et Pro comme nous pouvons le voir sur la Figure 4 : *teteN* et *tetePro*. Les cardinalités de ces relations sont 0..1, ce qui permet d'assurer au maximum un N et un Pro comme tête. Une autre contrainte est donc nécessaire pour décrire le fait que ces deux realtions sont mutuellement exclusives. Cette contrainte est une contrainte sur la somme des cardinalités des relations :

$$\forall\, sn : SN \bullet \#(sn.teteN \cup sn.tetePro) = 1$$

Il faut également préciser que l'élément tête doit avoir une valeur d'attribut de début (resp. fin) plus grande (resp. petite) que la valeur de l'attribut début (resp. fin) du syntagme dont il est la tête:

$$\begin{array}{l}\forall\, sn : SN \bullet \forall\, n : SN.teteN \bullet \\ \quad n.debut \geq sn.debut \wedge n.fin \leq sn.fin \\ \forall\, sn : SN \bullet \forall\, p : SN.tetePro \bullet \\ \quad p.debut \geq sn.debut \wedge p.fin \leq sn.fin\end{array}$$

[6]La définition de la Figure 2 n'est pas suffisante pour implémenter le modèle objet présenté mais pour des raison de clarté nous nous concentrons ici sur la classe SN et ses relations avec les autres classes.

3.2.2 Facultativité

Cette propriété définit l'ensemble des catégories pouvant être présentes dans un syntagme et n'étant pas une tête. Pour le SN, les deux catégories Det et Adj sont optionnelles. Cette propriété est traduite en utilisant des relations dans le diagramme de classes UML. Dans ce cas les cardinalités sont 0..*. Il faut bien entendu comme pour les têtes préciser les relations entre les attributs debut et fin.

$$\begin{array}{l}\forall\, sn : SN \bullet \forall\, d : sn.facDet \bullet \\ \quad d.debut \geq sn.debut \wedge d.fin \leq sn.fin \\ \forall\, sn : SN \bullet \forall\, a : sn.facAdj \bullet \\ \quad a.debut \geq sn.debut \wedge a.fin \leq sn.fin\end{array}$$

3.2.3 Constituants

Pour accéder rapidement à toutes les catégories dans une phrase sans avoir à passer par les relations tête et facultative, nous définissons une relation nommée "const" qui représente cette union. Dans l'exemple du SN, la relation "const" sera définit telle que :

$$\begin{array}{l}\forall\, sn : SN \bullet \langle sn.teteN, sn.tetePro, \\ \qquad sn.facDet, sn.facAdj\rangle \\ \quad \mathsf{partition}\ sn.const\end{array}$$

Il est alors nécessaire de spécifier que deux catégories ne peuvent pas avoir les mêmes constituants :

$$\begin{array}{l}\forall\, c1, c2 : Cat \bullet \\ (c1.const) \cap (c2.const) = \varnothing\end{array}$$

Une catégorie ne peut pas être incluse dans ses propres constituants :

$$\forall\, c : Cat \bullet c \notin (c.const)$$

Toute catégorie terminale utilisée dans la solution doit être liée à un mot de la phrase analysée :

$$\forall\, c : Cat \bullet c.const \neq \varnothing \Leftrightarrow c.motAssocie = \varnothing$$

3.2.4 Unicité

Cette propriété définit l'ensemble des catégories facultatives qui ne peuvent pas apparaître plus d'une fois. Pour l'exemple du SN, seulement un Det et seulement un Adj peuvent être présents :

$$\begin{array}{l}\forall\, sn : SN \bullet \#(sn.facDet) \leq 1 \\ \forall\, sn : SN \bullet \#(sn.facAdj) \leq 1\end{array}$$

3.2.5 Exigence

Cette contrainte est exprimé avec le symbole $\Rightarrow$: $\{A_1, A_2, ..., A_n\} \Rightarrow \{B_1, B_2, ..., B_m\}$ où les A_i sont des et les B_j des ensembles de categories. Cette propriété signifie que si tous les éléments A_i sont présents dans le syntagme alors au moins un sous-ensemble B_j doit être présent. Dans l'exemple du SN une contrainte simple, spécifiant que le nombre de teteN doit être plus grand ou égal que le nombre de Det, est suffisante :

$$\forall\, sn : SN \bullet \#(sn.teteN) \geq \#(sn.facDet)$$

Mais dans le cas général, le membre gauche de la propriété est un ensemble et le membre droit est un ensemble de sous-ensembles. Afin d'illustrer ceci, supposons que nous ayons la propriété d'exigence suivante pour un Sx (un syntagme x quelconque, comme SN ou SV par exemple) : $\{A_1, A_2^*, A_3\} \Rightarrow \{\{C_1, C_2\}, \{C_3^*, C_4\}\}$ où les A_i et les C_j sont des categories. Alors cette propriété doit être traduite par la contrainte :

$$\begin{aligned}&\forall\, sx : SX \bullet (sx.facA_1 \neq \varnothing \wedge \\ &\quad sx.teteA_2 \neq \varnothing \wedge sx.facA_3 \neq \varnothing) \\ &\Rightarrow ((sx.facC_1 \neq \varnothing \wedge sx.facC_2 \neq \varnothing) \vee \\ &\quad (sx.teteC_3 \neq \varnothing \wedge sx.facC_4 \neq \varnothing))\end{aligned}$$

3.2.6 Exclusion

Cette contrainte est notée $A \Leftrightarrow B$ où A et B sont des ensembles de catégories. Cette propriété définit l'obligation de non co-occurrence entre tous les éléments de A et tous ceux de B.

$$\begin{aligned}&\forall\, sx : SX \bullet (\forall\, a : A \mid a \in sxudfconst \bullet \\ &\neg\, (\exists\, b : B \bullet b \in sx.const)) \wedge \\ &(\forall\, b : B \mid b \in sx.const \bullet \\ &\neg\, (\exists\, a : A \bullet a \in sx.const))\end{aligned}$$

L'implémentation courante du programme utilise les cardinalités pour représenter cette propriété, de la même façon que pour la propriété d'exigence dans sa version simple (pour le SN exemple).

3.2.7 Linearité

Cette propriété est notée $A \prec B$ et permet de définir la précédence linéaire entre les éléments d'un syntagme. A et B représentent des ensembles de catégories. La propriété de linéarité signifie que chaque élément de l'ensemble A présent dans le syntagme doit apparaître avant tout élément de B également présent.

$$\begin{aligned}&\forall\, xp : xP \bullet \forall\, a : A; b : B \mid \\ &a \in xp.const \wedge b \in xp.const \bullet \\ &a.fin \leq b.debut\end{aligned}$$

3.2.8 Accord

Cette contrainte porte sur l'accord des catégories au sein d'un même syntagme. Les traits d'accord étant transcrits en attributs de classe, la propriété d'accord est traduite en tant que contrainte d'égalité sur les valeurs des attributs.

$$\begin{aligned}&\forall\, sn : SN \bullet \forall\, n : sn.teteN; det : sn.facDet \\ &\quad \bullet (n.genre) = (det.genre)\end{aligned}$$

4 Contraintes de configuration

La section précédente a présenté la traduction des sept propriétés des GP. Cependant, d'autres contraintes sont nécessaires pour construire un modèle objet sous contraintes "opérationnel". L'entrée du configurateur est un texte contenant des phrases, dont chacune contient une liste de mots. La Figure 5 présente le sous-modèle objet utilisé pour représenter les données d'entrée. Les relations entre les classes non présentes sur cette figure ne sont pas représentées.

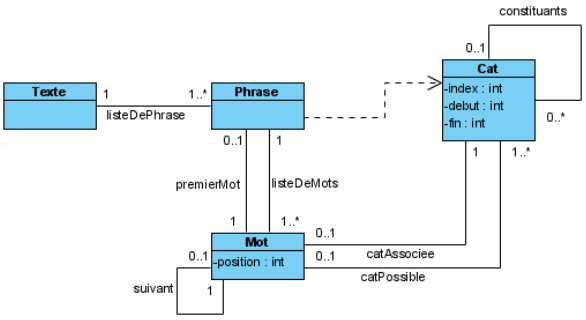

Figure 5: Classes utilisées pour représenter les données d'entrée du problème de configuration

Les contraintes nécessaires au configurateur sont les suivantes : Premièrement, il est important d'associer chaque mot avec ses étiquettes (catégories) possibles : Nous utilisons ici l'attribut *position* de la classe Mot. Cet attribut permet de représenter la position des mots et de les ordonner.

$$\begin{aligned}&\forall\, m : Mot; c : Cat \mid c \in m.catAssociee \\ &\quad \bullet m.position = ((c.debut) + 1)\end{aligned}$$

$$\begin{aligned}&\forall\, m : Mot; c : Cat \mid c \in m.catAssociee \\ &\quad \bullet m.position = (c.fin)\end{aligned}$$

Réciproquement, chaque catégorie est associée au mot pouvant porter son étiquette :

$$\begin{aligned}&\forall\, c : Cat; m : Mot \mid m \in c.motAssocie \\ &\quad \bullet (c.debut + 1) = (m.position)\end{aligned}$$

$$\forall\, c : Cat; m : Mot \mid m \in c.motAssocie \bullet (c.fin) = (m.position)$$

Pour finaliser cette relation, le mot associé à la catégorie associée à chaque mot est le mot lui même.

$$\forall\, m : Mot; c : Cat \mid c \in m.catAssociee \bullet \#(c.motAssocie) = 1$$

$$\forall\, m : Mot; c : Cat \mid c \in m.catAssociee \bullet m \in c.motAssocie$$

Une autre contrainte sur les catégories est nécessaire. Cette contrainte définit que chaque catégorie doit avoir une valeur d'attribut fin plus grande que sa valeur d'attribut debut :

$$\forall\, c : Cat \bullet c.fin > c.debut$$

La classe texte ne nécessite qu'une seule contrainte pour définir que tous les éléments présents dans la relation *listeDePhrase* sont uniques.

$$\forall\, texte : Texte; p : Phrase \mid p \in texte.listeDePhrase \bullet \exists_1\, p2 : Phrase \mid p2 \in texte.listeDePhrase \bullet p2 = p$$

Afin d'assurer la cohérence des valeurs des attributs debut et fin de chaque catégorie et des catégories entre elles les contraintes suivantes sont nécessaires :

Tout d'abord, chaque catégorie a une valeur d'attribut debut plus grande que sa valeur d'attribut fin.

$$\forall\, c : Cat \bullet c.fin > c.debut$$

Puis, pour chaque catégorie, la valeur minimale des attributs debut de ses constituants est égale à sa propre valeur d'attribut debut.

$$\forall\, c1 : Cat \bullet \neg\, (\exists\, c2 : Cat \mid c2 \in c1.const \bullet c2.debut < c1.debut)$$

De manière similaire, sa valeur d'attribut fin est égale à la valeur maximale des attributs fin de ses constituants.

$$\forall\, c1 : Cat \bullet \neg\, (\exists\, c2 : Cat \mid c2 \in c1.const \bullet c2.fin > c1.fin)$$

L'attribut *index* de la classe Cat est utilisé pour casser certaines symétries. La valeur de cet attribut n'est supérieure à 0 que lorsque la catégorie est non terminale. Lorsque la valeur est égale à 0 cela peut signifier deux choses. Soit que la catégorie concernée est une catégorie terminale, soit que la catégorie est non terminale mais qu'elle n'est pas utilisée dans la solution (elle n'a pas de constituants). Cet attribut est utilisé pour "ordonner" les instances d'une même classe.

$$\forall\, cat : Cat \bullet cat.index > 0 \Rightarrow \#(cat.motAssocie) = 0$$

Il est également interessant d'ordonner chaque paire de catégorie par leur attribut debut. Lors de la création d'une catégorie terminale, si il existe déjà une instance de ce type alors la nouvelle catégorie doit avoir une valeur d'attribut debut plus grande ou égale à la précédente.

L'attribut index va nous permettre de poser certaines contraintes sur la phrase elle même : Une phrase ne peut pas contenir de catégorie terminale :

$$\forall\, p : Phrase \bullet \forall\, c : Cat \mid c \in p.const \bullet c.index > 0$$

Toute phrase doit avoir au moins un constituant (un SV en général):

$$\forall\, p : Phrase \bullet \#(p.const) > 0$$

Nous présentons à présent deux contraintes utilisées pour casser les symétries pendant la résolution. A cet effet, il est nécessaire de définir une fonction qui retourne le type de chaque classe :

$$[U]$$
$$type : U \longrightarrow \mathbb{P}\, U$$
$$\forall\, u1 : U \bullet type(u1) = U$$

Dans ce problème, les ensembles de catégories potentielles sont crées au début de la recherche. Ce qui engendre un certain nombre de symétries comme par exemple, si deux SN (SN1 et SN2) sont présents dans l'ensemble de catégories initial, une première solution contiendra SN1 tandisqu'une deuxième contiendra SN2.

Les deux contraintes suivantes décrivent le fait que lorsque deux instances d'une même catégorie sont disponibles, celle avec la valeur d'index la plus petite sera choisie.

$$\forall\, c1, c2 : Cat \bullet ((c1.type = c2.type) \wedge (c1.index > 0) \wedge (c2.index > 0)) \Rightarrow ((c1.index < c2.index) \wedge (c1.debut < c2.debut))$$

$$\forall\, c1, c2 : Cat \bullet ((c1.type = c2.type) \wedge (c2.index > c1.index > 0)) \Rightarrow ((\#(c1.const) = 0) \Rightarrow (\#(c1.const) = 0))$$

5 Résultats expérimentaux

Nous présentons nos résulats sur une grammaire simplifiée du français présentée de la Figure 6 à la Figure 8. L'analyse se fera sur une phrase dont chaque mot aura plus d'une étiquette possible.

5.1 Une grammaire de propriété

Les catégories et leurs propriétés utilisées pour tester le programme sont listées ci-après. Il est important de noter que ce que le générateur analyse est exactement ce qui est présenté dans le fichier Latex ayant permis de générer ce document, et qui contient une série de déclaration de la forme de celle de la Figure 1.

$$\textsc{Phrase}\begin{bmatrix}\textsc{heritage} & Cat \\ \textsc{traits} & \begin{bmatrix}\textsc{catégorie} & phrase\end{bmatrix} \\ \textsc{props} & \begin{bmatrix}\textsc{tête} & SV \\ \textsc{facultativité} & SN \\ \textsc{unicité} & SN \\ \textsc{linéarite} & SN \prec SV*\end{bmatrix}\end{bmatrix}$$

Figure 6: Catégorie Phrase

$$\textsc{SN}\begin{bmatrix}\textsc{heritage} & Cat \\ \textsc{traits} & \begin{bmatrix}\textsc{catégorie} & Nominal\ Phrase\end{bmatrix} \\ \textsc{props} & \begin{bmatrix}\textsc{tête} & N \veebar Pro \\ \textsc{facultativité} & Det\ ;Adj\ ;PrepP \\ \textsc{unicité} & Det\ ;Adj\ ;PrepP \\ \textsc{exigence} & Det \Rightarrow N* \\ \textsc{exclusion} & Adj \nLeftrightarrow Pro* \\ & Det \nLeftrightarrow Pro* \\ \textsc{linéarite} & Det \prec N* \\ & N* \prec PrepP\end{bmatrix}\end{bmatrix}$$

Figure 7: Catégorie SN

$$\textsc{PrepP}\begin{bmatrix}\textsc{heritage} & Cat \\ \textsc{traits} & \begin{bmatrix}\textsc{catégorie} & Prepositionnal\ Phrase\end{bmatrix} \\ \textsc{props} & \begin{bmatrix}\textsc{tête} & Prep \\ \textsc{facultativité} & SN\ ;N \\ \textsc{unicité} & SN\ ;N \\ \textsc{exigence} & SN;N \Rightarrow Prep* \\ \textsc{exclusion} & SN \nLeftrightarrow N \\ \textsc{linéarite} & Prep* \prec SN;N\end{bmatrix}\end{bmatrix}$$

Figure 8: Catégorie SPrep

5.2 Lexique

Le lexique est écrit dans un fichier tabulé dont un extrait est présenté sur la Figure 11. (La taille du lexique n'a pas d'impact sur les performances du programme.)

$$\textsc{VP}\begin{bmatrix}\textsc{heritage} & Cat \\ \textsc{traits} & \begin{bmatrix}\textsc{catégorie} & Verbal\ Phrase\end{bmatrix} \\ \textsc{props} & \begin{bmatrix}\textsc{tête} & V \\ \textsc{facultativité} & SN\ ;SAdv \\ \textsc{unicité} & SN\ ;SAdv \\ \textsc{linéarite} & V* \prec SN;SAdv\end{bmatrix}\end{bmatrix}$$

Figure 9: Catégorie SV

$$\textsc{AdvP}\begin{bmatrix}\textsc{heritage} & Cat \\ \textsc{traits} & \begin{bmatrix}\textsc{catégorie} & Adverbial\ Phrase\end{bmatrix} \\ \textsc{props} & \begin{bmatrix}\textsc{tête} & Adv\end{bmatrix}\end{bmatrix}$$

Figure 10: Catégorie SAdv

5.3 Categories terminales

De même que les catégories non terminales, les catégories terminales sont définies à l'aide d'AVMs. Il est également possible de les générer automatiquement à partir des syntagmes. En effet si nous supposons une catégorie C et qu'aucun syntagme ne contient C alors C ne pourra pas apparaître dans une construction.

5.4 resultats

A partir de la phrase "*La porte ferme mal*" le configurateur trouve cinq étiquetages possible :

mot	categorie	genre	nombre	personne
⋮	⋮	⋮	⋮	⋮
ferme	N	fem	sing	3
ferme	Adj	indef	sing	0
ferme	V	indef	sing	3
la	Pro	fem	sing	3
la	Det	fem	sing	3
porte	N	fem	sing	3
porte	V	indef	sing	3
mal	N	masc	sing	3
mal	Adj	indef	indef	0
mal	Adv	indef	indef	0

Figure 11: Extrait du lexique

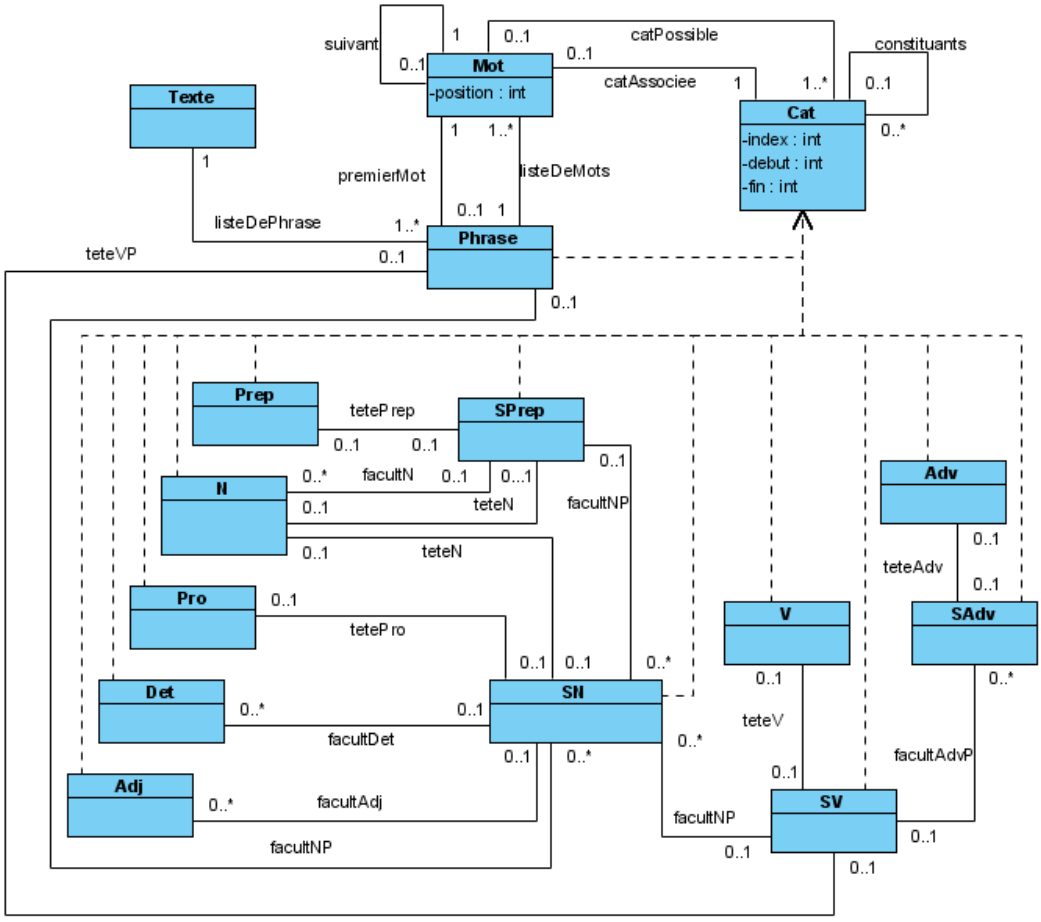

Figure 13: Constrained object model automatically generated

$$\textsc{Adj}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{N}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{Adv}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{Pro}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{Det}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{V}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{Conj}\left[\textsc{heritage} \quad Cat\right]$$

$$\textsc{Prep}\left[\textsc{heritage} \quad Cat\right]$$

Figure 12: avm des catégories terminales

SN1 [Pro ["La"]]
SV1 [V ["porte"] SN2 [N["ferme"]] SAdv1 [Adv["mal"]]]

SN1 [Pro ["La"]]
SV1 [V["porte"] SN2 [N["ferme"] Adj["mal"]]]

SN1 [Pro ["La"]] SV1 [V["porte"]
SN2 [Adj["ferme"] N["mal"]]]

SN1 [Det ["La"] N["porte"]]
SV1 [V["ferme"] SN2 [N["mal"]]]

SN1 [Det ["La"] N["porte"]]
SV1 [V["ferme"] SAdv1 [Adv["mal"]]]

Il serait possible d'objecter le fait que l'étiquetage SN SV SN où "La" est Pro, "porte" est verbe, "ferme" est Adj et "mal" est N, est un étiquetage valide car il ne correspond pas au sens que l'on se fait de cette phrase. Cependant, ce résultat n'est pas interdit par la grammaire définit plus haut. L'arbre syntaxique associé à la dernière analyse est présenté en Figure 14.

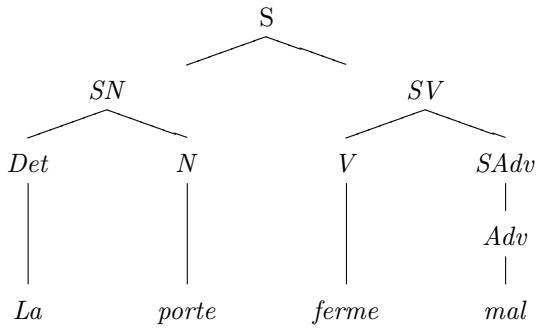

Figure 14: Etiquetage attendu pour la phrase "La porte ferme mal"

Cette expérimentation montre en outre que notre programme est correct et complet : toutes les interprétations sont trouvées et aucune mauvais interprétation (n'étant pas autorisée par la grammaire) n'est générée.

5.5 Temps d'exécution

Nous présentons dans cette section les temps d'exécution du programme. La figure 15 présente dans la première colonne le temps de génération du fichier de contraintes ainsi que du modèle objet, dans la deuxième colonne le temps de chargement de ces fichiers dans le configurateur et enfin dans la troisième colonne le temps d'exécution du configurateur sur l'instance donnée. La première (resp. dernière) solution est trouvée au bout de 2550 ms (resp. 2750 ms).

generation	chargement	résoltution
79 ms	2812 ms	5765 ms

Figure 15: Temps d'exécution du programme

6 Conclusion

Ce papier propose d'une part un style Latex pour représenter les grammaires de propriété et d'autre part un outil pour traduire automatiquement une grammaire définie avec le formalisme des GP dans le problème de configuration associé. Le configurateur peut ensuite être utilisé directement pour analyser des phrases. La source même de cet article a été parsée par notre outil pour obtenir les résultats présentés dans la section 5.4. Cette recherche ouvre de nombreuses perspectives dans le domaine des grammaires à base de contraintes. Etant automatique, le processus de construction de l'analyseur permet d'implémenter de nombreuses optimisations, comme par exemple le développement de contraintes redondantes ou d'élimination de symétries. L'approche générale peut être vue comme une évolution des complilateurs de compilateur de langages réguliers existant (comme lex/yacc) vers des générateurs d'analyseurs pour des langages hors contexte. Le compilateur/analyseur résultat peut alors travailler de manière analytique mais également générative, donnée d'importance dans le domaine de la traduction.

References

[1] Jean Marie Balfourier, Philippe Blache, Marie-Laure Guénot, and Tristan VanRullen. Comparaison de trois analyseurs symboliques pour

une tâche dŠannotation syntaxique. In *TALN'05*, pages 41–48, 2005.

[2] Jean Marie Balfourier, Philippe Blache, and Tristan VanRullen. From shallow to deep parsing using constraint satisfaction. In *COLING'02*, pages 36–42, 2002.

[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] Mathieu Estratat and Laurent Henocque. An intuitive tool for constraint based grammars. In Henning Christiansen, Peter Rossen Skadhauge, and Jørgen Villadsen, editors, *Revised Selected and Invited Papers, Constraint Solving and Language Processing, First International Workshop, CSLP 2004, Roskilde, Denmark, September 1–3*, volume 3438 of *Lecture Notes in Computer Science*, pages 121–139. CSLP, Springer, 2005. isbn 3–540-26165–6.

[6] Laurent Henocque. Modeling object oriented constraint programs in z. *RACSAM (Revista de la Real Academia De Ciencias serie A Mathematicas)*, special issue on Symbolic Computation in Logic and Artificial Intelligence(98 (1–2)):127–152, 2004.

[7] Ulrich Junker and Daniel Mailharro. The logic of ilog (j)configurator: Combining constraint programming with a description logic. In *IJAI 2003 Workshop On Configuration*, Mexico, 2003.

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

[9] Object Management Group OMG. Uml 2 superstructure. Technical Report 2.0, OMG, 2004.

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

[11] Tristan VanRullen. *Vers une analyse syntaxique à granularité variable*. Thèse de doctorat, Université de Provence (Aix-Marseille I), 12 décembre 2005.