

## Application des programmes de contraintes orientés objet à l'analyse du langage naturel

Mathieu Estratat (1), Laurent Henocque (2)

LSIS

Université d'Aix-Marseille III

Avenue Escadrille Normandie-Niemen

13397 Marseille cedex 20

(1) mathieu.estratat@lsis.org

(2) henocque@esil.univ-mrs.fr

### Résumé - Abstract

Les évolutions récentes des formalismes et théories linguistiques font largement appel au concept de *contrainte*. De plus, les caractéristiques générales des grammaires de traits ont conduit plusieurs auteurs à pointer la ressemblance existant entre ces notions et les *objets* ou *frames*. Une évolution récente de la programmation par contraintes vers les programmes de contraintes orientés objet (OOC) possède une application possible au traitement des langages naturels. Nous proposons une traduction systématique des concepts et contraintes décrits par les grammaires de propriétés sous forme d'un OOC. Nous détaillons l'application de cette traduction au langage "context free" archétypal  $a^n b^n$ , en montrant que cette approche permet aussi bien l'analyse que la génération de phrases, de prendre en compte la sémantique au sein du même modèle et ne requiert pas l'utilisation d'algorithmes ad hoc pour le passage.

Recent evolutions of linguistic theories heavily rely upon the concept of constraint. Also, several authors have pointed the similitude existing between the categories of feature based theories and the notions of objects or frames. A recent evolution of constraint programming to object oriented constraint programs (OOC) can be applied to natural language parsing. We propose here a systematic translation of the concepts and constraints introduced by property grammars to an OOC. We apply this translation to the archetypal context free language  $a^n b^n$ , and show that this approach allows to both parse and generate, to account for the semantics in the same formalism, and also that it does not require the use of ad hoc algorithms.

### Mots-clefs – Keywords

Grammaires de propriétés, traitement du langage naturel, contraintes, configuration  
Property grammars, natural language processing, constraints, configuration

## Introduction

Les évolutions récentes des formalismes et théories linguistiques font largement appel au concept de *contrainte* (Pollard & Sag, 1994; Blache, 2000; Blache, 2001). Selon ces formalismes, la validité d'une construction syntaxique est définie par le respect de contraintes portant sur les traits. De plus, les caractéristiques générales des grammaires de traits (Gazdar *et al.*, 1985) ont conduit plusieurs auteurs à pointer la ressemblance existant entre ces notions et les *objets* ou *frames* et l'héritage même multiple (Pollard & Sag, 1994). Dans le même temps, une évolution de la programmation par contraintes vers les programmes de contraintes orientés objet (OOC) apporte une solution technique à un problème d'un grand intérêt industriel : la configuration<sup>1</sup> (voir par exemple (Fargier & Henocque, 2002) pour une présentation générale et une large bibliographie). Au delà de la configuration, les OOCs sont des CSP quantifiés ayant des applications générales à l'IA. Nous en présentons ici une application au traitement de langage context free, travail qui a débuté avec (Estratat, 2003) et qui a pour objectif le traitement de langages naturels.

### programmes de contraintes orientés objet

Un OOC est un CSP quantifié, dans lequel peuvent apparaître arbitrairement quantifications existentielles et universelles qui s'apparente à un problème de la logique des prédicats, indécidable dans le cas général. C'est donc un problème considérablement plus général que les CSP, formulé explicitement comme un problème de recherche de modèles finis au premier ordre<sup>2</sup>. Un tel programme est décrit par un modèle orienté objet (comme illustré par la figure 3), assorti de contraintes de bonne formation. Une présentation formelle des OOC figure dans (Henocque, 2003). Résoudre techniquement le problème d'énumération associé à une requête peut être fait en utilisant divers formalismes et approches techniques: extensions des CSP (Mittal & Falkenhainer, 1990; Fleischanderl *et al.*, 1998; Sabin & Freuder, 1996), approches basées sur la connaissance (Stumptner, 1997) ou sur les logiques terminologiques (Nebel, 1990), programmation logique étendue (chaînage avant et arrière, sémantiques non standard) (Colmerauer, 1990; Fromherz *et al.*, 1997; Soinen *et al.*, 2000; Gelfond & Lifschitz, 1988), approches "pur objet". Nos expérimentations sont conduites avec un outil d'énumération de ce dernier type: Ilog Jconfigurator (Mailharro, 1998). Nous avons choisis cet outil pour deux raisons majeures : tout d'abord c'est un produit industriel aisément acquiescent (bonne diffusion), installable facilement et proposant un large support technique. D'un autre côté, ce configurateur propose une énumération orientée objet des solutions. Son comportement lors de la recherche de solutions est aisé à comprendre et facilite l'implémentation d'heuristiques. En effet, son comportement est celui d'un CSP étendu dans le paradigme orienté objet. De plus, la génération de modèle est intuitive. Lors de la recherche de solution, la complexité d'un tel système est liée d'un côté au nombre de points de choix empruntés par le solveur et de l'autre au temps de calcul dépendant du coût de propagation des contraintes dans le système.

---

<sup>1</sup>Configurer consiste à simuler la réalisation d'un produit complexe à partir de composants choisis dans un catalogue de types. Les composants sont soumis à des relations (cette information est appelée "partonomique"), et leurs types sont soumis à des relations d'héritage (information "taxonomique"). Des contraintes définissent les produits réalisables. Un outil de configuration prend en entrée un fragment du produit final, et complète ces données de façon à satisfaire à la fois les contraintes générales et les contraintes et objectifs particuliers à une requête.

<sup>2</sup>Il est impossible de connaître au départ le nombre et le type des composants qui interviendront dans une solution d'un OOC.

## grammaires de propriétés

Les grammaires de propriétés (GP) (Blache, 2001) constituent une théorie basée exclusivement sur les contraintes qui propose d'une part une classification des contraintes de traits (appelées *propriétés*) et d'autre part un algorithme de passage spécialisé exploitant la propagation de ces contraintes. Indépendamment des algorithmes, les grammaires de propriétés s'articulent autour de deux notions : les *catégories* qui représentent toute unité syntaxique du texte, associée à un mot isolé ou à un groupe de mots (voir figure 1), et les *propriétés* (ou contraintes) portant sur ces catégories, qui spécifient les règles de bonne formation des syntagmes et les règles de cohésion de la phrase.

## catégories et programmation par contraintes

Les catégories sont des structures de traits : des fonctions partielles définies sur l'ensemble des expressions linguistiques et à valeur dans un certain ensemble de valeurs de traits. Cette définition est récursive : la valeur d'un trait peut être un trait, ou une liste de traits. Une structure de traits est un ensemble de couples (attribut, valeur), permettant d'étiqueter toute unité linguistique, comme illustré figure 1, qui décrit *Livre* comme un *nom commun masculin*, employé à la *troisième personne du singulier*. D'un point de vue fonctionnel, un trait est assimilable à une

[	<i>Cat:</i>	N	]
	<i>Phon:</i>	Livre	
	<i>Accord:</i>	[	
		<i>Genre:</i>	masc
		<i>Nombre:</i>	sing
		<i>Pers:</i>	3
		]	
	<i>Type :</i>	commun	]

Figure 1: La catégorie N

variable de CSP et une structure de traits peut être vue comme une affectation de variables à des valeurs prises dans le domaine adéquat (par exemple une énumération comme  $\{Sing, Plur\}$ , ou un entier comme  $\{1, 2, 3\}$ ). En pratique, la valeur d'un trait pouvant être un autre trait, ou un ensemble (ordonné) de traits, les variables des CSP classiques (à domaines finis) ne permettent pas d'en rendre compte facilement. Il est nécessaire de disposer de variables ensemblistes, comme celles introduites par (Mailharro, 1998).

## propriétés et programmation par contraintes

Les *propriétés* sont des contraintes qui portent sur les catégories, et spécifient les règles de bonne formation des syntagmes et de cohésion de la phrase. Il y a sept sortes de propriétés : *constitution* (décrit les composants possibles d'un syntagme), *noyau* (décrit les constituants noyaux), *unicité* (contraint un constituant à n'apparaître qu'une fois au plus), *exigence* (contraint la cooccurrence de groupes de catégories), *exclusion* (idem pour la non cooccurrence), *linéarité* (ordre de précedence entre catégories), et *dépendance* (contraintes entre catégories distantes). Par exemple  $Const(SN) = \{Det, N, Pro, \dots\}$  est une propriété du français. Ces propriétés correspondent directement à des contraintes posées, soit sur le modèle objet sous forme de contraintes de cardinalité, soit en complément de ce dernier en spécifiant quelles relations doivent entretenir les instances de classes entre elles. La section 1 propose une traduction automatique de ces propriétés.

## plan

La section 1 présente une traduction systématique des grammaires de propriétés sous forme de programmes de contraintes orientés objet. La section 2 décrit une application de cette approche à la grammaire du langage context free  $a^n b^n$ . La section 3 décrit l'utilisation du programme de contraintes pour le parsing ou la génération, et illustre la prise en charge de la sémantique. La section 4 présente les résultats obtenus. La section 5 fournit une conclusion et des perspectives.

# 1 Traduire les GP en OOC

## 1.1 Un modèle objet de la syntaxe

Les éléments structurels des formalismes linguistiques modernes correspondent naturellement à des concepts de programmation par contraintes orientée objet. Un *trait* est un attribut dont le domaine de définition est fini, qui correspond à une variable de CSP, ensembliste ou non, de même domaine. Ce domaine est défini comme l'ensemble des valeurs possibles du trait associé. Une *structure de traits* est un agrégat de traits, dont les affectations sont des ensembles de couples (attribut,valeur). De tels agrégats sont bien modélisés par des classes d'un modèle objet. Une *catégorie* est une structure de traits nommée, qui correspond également à une classe dans un modèle objet, insérée dans une hiérarchie de classes utilisant l'héritage.

De nombreux traits ont pour valeur des structures de traits, ou des ensembles de telles structures. Cette situation est adéquatement traduite par des relations entre les classes du modèle objet correspondant. Notamment, un *syntagme* est un ensemble de mots en relation avec un élément central appelé *noyau*. Par exemple, un syntagme nominal est un ensemble de mots dont le *noyau* est un nom (ou un pronom). Cette relation entre un syntagme et sa catégorie noyau peut être formulée explicitement par une relation d'un modèle objet.

La figure 1, décrit une catégorie de GP représentant un Nom. Le modèle objet associé, illustré par la figure 2 comporte trois classes. La classe *catTerminale* est une abstraction intégrant les traits communs à toutes les catégories terminales, par des attributs propres mais également par des relations de compositions avec notamment la classe *Accord* qui implémente les traits de *genre*, *nombre* et *personne* pour toute catégorie terminale. Ainsi, la classe *N* hérite des attributs de *catTerminale*, l'attribut *type* lui étant spécifique. Disposer d'un modèle objet permet d'introduire des classes qui réalisent des abstractions utiles pour la simplification du modèle. La figure 2 illustre cette possibilité par l'insertion de l'abstraction *catTerminale*. L'exemple  $a_n b_n$  développé en section 2 illustrera plus en détail cette possibilité.

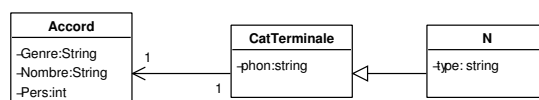


Figure 2: Un modèle orienté objet simplifié de la catégorie N

## 1.2 Traduction des propriétés en contraintes du modèle objet

Les propriétés peuvent être vues comme des contraintes et des relations adjointes du modèle objet associé aux traits d'une grammaire de propriétés. Les symboles dénotant des catégories sont des lettres majuscules (e.g.  $S, A, B, C \dots$ ). Lorsqu'il existe une relation entre deux catégories

$S$  et  $A$ , on note  $s.A$  l'ensemble des  $A$  liés à une instance  $s$  de  $S$  donnée et  $|s.A|$  sa cardinalité. Lorsque cela est possible, à des fins de simplicité, on utilise la notation  $\forall S F$  plutôt que  $\forall s \in S F$ . La notation pointée habituelle est utilisée pour désigner les attributs d'une classe (ex :  $a.begin$  désigne l'attribut  $begin$  de l'objet  $a$ ). Chaque catégorie peut être discriminée par des attributs de ses éléments: par exemple  $Det_{[art]}$  dénote l'ensemble des éléments appartenant à la classe  $Det$  et de type  $art$ . Pour des raisons de place mais sans perte de généralité, nous ne décrivons pas ce cas général dans la suite. Enfin,  $S$  dénote un syntagme quelconque.

Sur la base de ces définitions, nous proposons pour chaque type de propriété les traductions qui suivent :

- **Constituants (Const) :**

La propriété  $Const(S) = \{A_m\}_{m \in I_A}$  spécifie qu'un  $S$  ne peut contenir que des éléments appartenant à  $\{A_m\}_{m \in I_A}$ . Cette propriété est adéquatement décrite dans un modèle objet par des relations de cardinalité  $[0, n]$  entre la catégorie  $S$  et chacune des catégories de  $\{A_m\}_{m \in I_A}$ , comme illustré dans le modèle objet sur la figure 3.

- **Noyaux (Noyaux) :**

La propriété  $Noyaux(S) = \{A_m\}_{m \in I_A}$  spécifie que tout  $S$  doit avoir pour noyau (un représentant d') une des catégories de l'ensemble  $\{A_m\}_{m \in I_A}$ . Le *noyau* est unique et obligatoire dans tout syntagme. La relation  $Noyaux$  est un sous-ensemble de la relation  $Const$ . Une telle propriété est traduite en utilisant des relations comme pour les *Constituants* et des contraintes de cardinalité adéquates.

- **Unicité (Unic) :**

La propriété  $Unic(S) = \{A_m\}_{m \in I_A}$  spécifie que la catégorie  $S$  ne peut contenir qu'au plus une instance de chaque  $A_m$ . Les cardinalités des relations codant la propriété de constitution permettent l'expression de ces contraintes. Traduire l'unicité revient à contraindre chaque relation  $S.A_m$  à avoir une cardinalité inférieure ou égale à 1 (ce que dans toute la suite en l'absence d'ambiguïté nous noterons  $|S.A_m| \leq 1$ ) :

$$|\{x : S.Const \mid x \in A_m\}| \leq 1$$

- **Exigence ( $\Rightarrow$ ) :**

La propriété  $\{A_m\}_{m \in I_A} \Rightarrow_S \{\{B_n\}_{n \in I_B}, \{C_o\}_{o \in I_C}\}$  signifie que toute occurrence de la totalité des  $A_m$  dans un  $S$  implique nécessairement au moins une occurrence d'un des ensembles :  $\{B_n\}$  ou  $\{C_o\}$ . Cette propriété correspond à la contrainte :

$$\forall S (\forall m \in I_A |S.A_m| \geq 1) \Rightarrow ((\forall n \in I_B |S.B_n| \geq 1) \vee (\forall o \in I_C |S.C_o| \geq 1))$$

- **Exclusion ( $\nLeftarrow$ ) :**

La propriété  $\{A_m\}_{m \in I_A} \nLeftarrow_S \{B_n\}_{n \in I_B}$  spécifie que deux ensembles de catégories s'excluent mutuellement ( $\Rightarrow$  désigne ici l'implication logique et non la propriété d'exigence):

$$\forall S, \left\{ \begin{array}{l} (\forall m \in I_A |S.A_m| \geq 1) \Rightarrow (\forall n \in I_B |S.B_n| = 0) \wedge \\ (\forall n \in I_B |S.B_n| \geq 1) \Rightarrow (\forall m \in I_A |S.A_m| = 0) \end{array} \right.$$

- **Linéarité ( $\prec$ ) :**

La propriété  $\{A_m\}_{m \in I_A} \prec_S \{B_n\}_{n \in I_B}$  signifie que si des catégories de  $\{A_m\}_{m \in I_A}$  coexistent avec des catégories de  $\{B_n\}_{n \in I_B}$  alors elles les précèdent dans le syntagme. A cette

fin, la représentation des catégories dans le modèle objet comporte deux attributs *debut* et *fin* (des entiers). Cette propriété est traduite par la contrainte <sup>3</sup> :

$$\forall S \forall m \in I_A \forall n \in I_B, \max(\{i \in S.A_m \bullet i.fin\}) \leq \min(\{i \in S.B_n \bullet i.debut\})$$

- **Dépendance** ( $\rightsquigarrow$ ) :

Cette propriété permet d'établir des relations spécifiques entre des catégories distantes, en rapport avec la sémantique (pour traduire par exemple le lien existant entre un pronom et son référent dans une phrase précédente). Chaque cas étant spécifique, la prise en compte des dépendances ne peut être présentée de façon générique.

Les sept propriétés ci-dessus sont transcrites sous forme de contraintes indépendantes. Il est cependant possible de décrire par une seule contrainte plusieurs propriétés (existence et unicité par exemple), notamment grâce au contrôle fin des cardinalités. L'exemple détaillé suivant (figure 3) illustrera cette possibilité. Les expressions logiques codant les propriétés présentées ci-dessus se traduisent immédiatement en terme de contrainte. Par exemple, la formule logique  $|S.A_m| \leq 1$  pour la propriété d'unicité se traduit par : *forall*(*S*, *leq*(*S.getCardinality*("A<sub>m</sub>"), 1)) de manière très intuitive, en effet, l'opérateur d'ingalité est postfixe dans la traduction et infixé dans la formule logique, de même l'opérateur de calcul de cardinalité est remplacé par la méthode *getCardinality* de la classe *S*.

## 2 Application au langage $a^n b^n$

Nous présentons maintenant une application à la représentation et au parsing du langage  $a^n b^n$ , archétype des grammaires context free. Cet exemple illustre de façon simple l'adéquation des OOCF au problème du parsing de grammaires arbitrairement complexes et récursives<sup>4</sup>, ce qui est requis pour l'analyse du langage naturel et permet de réaliser concrètement un couplage syntaxe/sémantique<sup>5</sup>. (Blache, 2001) décrit le langage  $a^n b^n$  par les propriétés suivantes :

$$\left\{ \begin{array}{l} \text{Constituants : } Const(S) = \{S, a, b\}; \\ \text{Noyaux : } Noyaux(S) = \{a\}; \\ \text{Unicité : } Unic(S) = \{S, a, b\}; \\ \text{Exigence : } a \Rightarrow_S b; \\ \text{Linarité : } a \prec_S b; a \prec_S S; S \prec_S b; \end{array} \right.$$

Cet exemple introduit trois catégories : *S* (non terminal), *a* et *b* (terminales). Cette grammaire est récursive. *S* est un syntagme dont le noyau est un *a*. Un *S* contient exactement un *a* et un *b* et optionnellement un autre *S* tels que *a* précède *b* ou *a* précède *S* et *S* précède *b* selon le cas. Nous proposons le modèle objet illustré par la figure 3 et les contraintes de configuration associées pour décrire et parser ce langage. Les classes *S*, *A*, et *B* correspondent aux trois catégories précédentes. La classe *Cat* est une abstraction pour toutes les catégories. Elle décrit notamment les deux attributs *debut* et *fin* requis pour l'expression des propriétés de linéarité, cela simplifie le modèle, l'héritage permettant de ne pas répéter ces attributs dans les sous-classes. La classe *Mot* est une abstraction pour les catégories terminales *A* et *B* qui héritent au passage de la relation suivant. La catégorie *Cat* factorise des attributs et la classe *Mot* factorise des relations.

<sup>3</sup> $\{i \in S.A_m \bullet i.fin\}$  dénote l'ensemble des valeurs de l'attribut *i.fin* pour *i* variant dans l'ensemble *S.A<sub>m</sub>*

<sup>4</sup>La récursivité est très présente dans le langage naturel. Notamment la syntaxe du syntagme nominal (*SN*) est récursive, par exemple : "[la selle [du vélo [de ma grand-mère]]]".

<sup>5</sup>Dans l'exemple traité la sémantique est simple, mais l'efficacité du couplage syntaxe/sémantique laisse présager de bonnes dispositions de la part des OOCF à effectuer cette tâche sur des grammaires plus complexes.

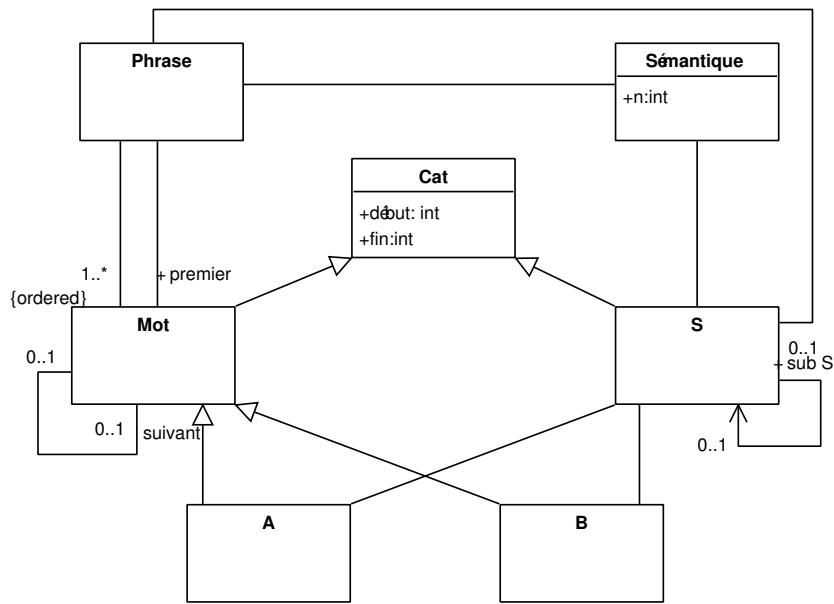


Figure 3: Modèle objet pour le langage  $a^n b^n$

La classe *Phrase* permet de décrire une liste de mots. Elle est également en relation avec le premier mot de cette liste. Chaque *Mot* hors le dernier possède un mot suivant. Chaque *S* est en relation avec un *A*, un *B* et optionnellement un autre *S*. La classe *Sémantique* modélise la sémantique d'une entrée (ici le nombre  $n$  de répétitions de *A*). Les classes *S* et *Phrase* sont en relation entre elles (cela associe une phrase à sa représentation syntaxique), et avec la classe *Sémantique* : chaque unité syntaxique a une sémantique associée.

Dans cet exemple, les propriétés de *constitution*, de *noyau*, d'*unicité* et d'*exigence* sont implicitement décrites par les relations du modèle et leurs multiplicités, ce qui illustre la remarque finale de la section précédente.

$$\left\{ \begin{array}{l} |S.A| = 1; \text{ (tout } S \text{ contient un } A) \\ |S.B| = 1; \text{ (tout } S \text{ contient un } B) \\ |S.S| = \{0, 1\}; \text{ (tout } S \text{ contient éventuellement un autre } S) \end{array} \right.$$

Les propriétés de *linéarité*, sont décrites par des contraintes complémentaires à ce modèle :

$$\left\{ \begin{array}{l} \forall S, S.A.debut < S.B.debut; \\ \forall S, (|S.S| == 1) \Rightarrow (S.A.fin \leq S.S.debut); \\ \forall S, (|S.S| == 1) \Rightarrow (S.B.debut \geq S.S.fin); \end{array} \right.$$

## 2.1 Sémantique

Dans ce cas particulièrement simple, la sémantique associée à une phrase de  $a^n b^n$  est naturellement le nombre  $n$ . La classe *Sémantique* du modèle comprend donc un attribut de type entier :  $n$ . Le nombre  $S.Sémantique.n$  représente le nombre total de *A* dans un *S*. Les contraintes qui lient la sémantique à la syntaxe sont les suivantes :

$$\left\{ \begin{array}{l} \forall S (|S.S| == 1) \Rightarrow S.Sémantique.n = 1 + S.S.Sémantique.n \\ \forall S (|S.S| == 0) \Rightarrow S.Sémantique.n = 1 \\ \forall Phrase \text{ Phrase}.Sémantique = Phrase.S.Sémantique; \end{array} \right.$$

Ces contraintes définissent récursivement la sémantique associée à un  $S$  comme le nombre de  $A$  qu'il contient, et la sémantique d'une phrase comme la sémantique associée à son syntagme de plus haut niveau.

### 3 Parsage

Un programme de contraintes orienté objet est utilisé de la manière suivante : il reçoit en entrée un groupe d'objets partiellement connus et partiellement inter connectés (par exemple des instances de *Mot* liés par la relation *suivant*). Les éléments ainsi fournis forment une partie de la solution (s'il en existe). Un programme spécialisé (nous avons utilisé le configurateur orienté objet d'Ilog (Mailharro, 1998)) peut alors compléter les données entrées par de nouveaux objets, par l'établissement des relations manquantes et par la détermination exacte des types et des attributs. Dans notre cas, ce mode de fonctionnement permet d'utiliser la grammaire traduite sous forme d'un programme de contraintes orienté objet soit en mode analyse, soit en mode génération, voire de manière hybride, tout en prenant en compte la sémantique. Si une suite de mots est fournie en entrée, le programme produit l'arbre syntaxique et la sémantique. Si un fragment de l'arbre syntaxique, ou la sémantique, ou une suite de mots incomplète, est donnée, le programme génère si elle existe la suite de mots valide correspondante, tout en complétant ce qui doit l'être. La figure 4 illustre ce comportement. Les états du système  $y$  sont décrits par des tuples  $\langle mots, syntaxe, sémantique \rangle$  (le caractère ? dénote un objet inconnu et le caractère  $\diamond$  un mot inconnu) et son comportement par des règles  $état\ entrée \mapsto état\ sortie$ . Les deux

$$\left\{ \begin{array}{l} \langle aaabbb, ?, ? \rangle \mapsto \langle aaabbb, S(A, S(A, S(A, null, B), B), B), 3 \rangle \\ \langle abbb, ?, ? \rangle \mapsto false \\ \langle \diamond a \diamond b, ?, ? \rangle \mapsto \langle aabb, S(A, S(A, null, B), B), 2 \rangle \\ \langle ?, ?, 2 \rangle \mapsto \langle aabb, S(A, S(A, null, B), B), 2 \rangle \end{array} \right.$$

Figure 4: Exemples de parsage

premiers cas sont des exemples de parsage pur. Dans le deuxième cas, la phrase fournie en entrée n'appartient pas au langage. Les autres exemples illustrent le fonctionnement génératif ou hybride du programme.

### 4 Résultats

La figure 5 présente les résultats obtenus<sup>6</sup> pour des entrées (consistantes et inconsistantes) de différentes tailles. Les résultats (nombre de backtracks, nombre de points de choix, temps en secondes) sont donnés d'une part pour la production de la première solution si elle existe, d'autre part pour le parcours de l'espace de recherche complet. Une entrée est définie par une suite de *Mot* liés entre eux par la relation *suivant* (c.f. figure 3) et n'incorpore aucune autre information susceptible de guider le parsage. La totalité du travail d'analyse reste donc à faire par le configurateur. Les temps de calcul qui peuvent paraître élevés pour des "phrases" de 100 mots sont fortement influencés par l'utilisation du langage Java et par la complexité de la propagation des contraintes ensemblistes (qui est corrélée polynomialement au nombre d'objets). De façon plus remarquable, on observe que ce problème n'est pas combinatoire. On montrerait aisément que la profondeur de l'espace de recherche est en  $O(5n)$ ,  $n$  étant le nombre de mots. On observe que le nombre de points de choix (appelés ici "noeuds") comme le nombre de backtracks (ou "fails") sont bornés par cette valeur. L'espace de recherche est

<sup>6</sup>Pentium4 2,4GHz - 512 Mo DDR - Windows XP Professional SP1 - Java 2 V.1.4.2 - Ilog Jconfigurator 2.08

<i>mot</i>	<i>Nb Cont</i>	<i>Nb Var</i>	<i>1<sup>ère</sup> solution</i>			<i>terminaison</i>		
			<i>Fails</i>	<i>Noeuds</i>	<i>Tps</i>	<i>Fails</i>	<i>Noeuds</i>	<i>Tps</i>
<i>aaabbb</i>	105	74	0	20	0,38 s	20	20	0,41 s
$\diamond a \diamond b$	145	104	0	29	0,5 s	29	29	0,53 s
<i>a(10) b(10)</i>	383	300	0	92	0,8 s	92	92	0,84 s
<i>a(20) b(20)</i>	723	580	0	182	1,44 s	182	182	1,66 s
<i>a(50) b(50)</i>	1743	1420	0	452	5,53 s	452	452	5,69 s
<i>a(51) b(49)</i>	1743	1420	-	-	-	1	0	4,62 s
$\diamond a(50) b(49)$	1740	1419	-	-	-	1	0	4,61 s

Figure 5: Résultats expérimentaux

de taille linéaire, donc le problème est polynomial si l'on tient compte des contraintes. De plus, le fait que lorsque l'entrée est inconsistante (*a(51) b(49)* et  $\diamond a(50) b(49)$ ) le programme termine sans point de choix montre que la propagation des contraintes suffit à détecter cette inconsistance. Notons que notre modèle implémente une contrainte redondante très générale pour briser les symétries existant entre les instances des catégories non-terminales (les valeurs des attributs *begin* de toutes les instances de *S* sont ordonnées) et la sémantique.

## 5 Conclusion

Nous avons décrit une traduction des grammaires de propriétés sous forme d'un programme de contraintes orienté objet, et montré que ce programme peut être utilisé aussi bien de manière analytique (construire l'arbre syntaxique d'une phrase bien formée) que de manière générative (produire une phrase bien formée sous contraintes). Cela place cette approche au niveau des grammaires de clauses définies (Definite Clause Grammars en Prolog (Colmerauer, 1978)) et au dessus des formalismes reposant sur l'utilisation de règles exploitées en chaînage avant comme les constraint grammars (Karlsson, 1990). La complétion d'entrée incomplète, ou la génération de phrases, a beaucoup d'applications pratiques.

(Blache, 2001) présente un formalisme reposant exclusivement sur des contraintes, mais emploie un algorithme ad hoc pour la propagation et le parsage. De plus, la coopération de la sémantique avec la syntaxe lors de l'analyse n'est pas abordée. Modéliser la syntaxe sous forme d'un programme de contraintes orienté objet donne accès à la grande gamme de contraintes disponibles dans ce cadre, et permet l'utilisation d'algorithmes généraux.

La sémantique véhiculée par des textes descriptifs simples peut être décrite comme un OOCF (de nombreux articles sur la configuration utilisent un exemple de configuration de PC (Soininen *et al.*, 2000)). Dans une telle situation nos expérimentations montrent que le modèle sémantique peut être combiné avec le modèle syntaxique. Ainsi, syntaxe et sémantique coopèrent étroitement lors de l'analyse ou de la génération. Nos recherches s'orientent actuellement vers l'implémentation d'un parseur du langage naturel pour un sous-ensemble du français, dans le cadre de la description de scènes 3D (La Greca & Daniel, 2002).

## Remerciements

Ces travaux de recherche ont pu être réalisés grâce à un financement JemSTIC du CNRS.

## Références

- BLACHE P. (2000). Property grammars and the problem of constraint satisfaction. In *ESSLLI-2000 workshop on Linguistic Theory and Grammar Implementation*.
- BLACHE P. (2001). *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles*. Hermès Sciences.
- COLMERAUER A. (1978). Metamorphosis grammars. *Lecture Notes on Computer Science*, 63, 133–189.
- COLMERAUER A. (1990). An introduction to prolog 3. *communication of the ACM*, 33(7), 69–90.
- ESTRATAT M. (2003). Application de la configuration à l'analyse syntaxico sémantique de descriptions. Master's thesis, Faculté des Sciences et Techniques de Saint Jérôme, LSIS équipe InCA, Marseille, France, submitted for the obtention of the DEA degree.
- FARGIER H. & HENOCQUE H. (2002). Configuration à base de contraintes. *Information Interaction Intelligence, Actes des 2ièmes Assises nationales du GdR I3*, p. 141–159.
- FLEISCHANDERL G., FRIEDRICH G., HASELBÖCK A., SCHREINER H. & STUMPTNER M. (1998). Configuring large-scale systems with generative constraint satisfaction. *IEEE Intelligent Systems - Special issue on Configuration*, 13(7).
- FROMHERZ M., GUPTA V. & SARASWAT V. (1997). cc - a generic framework for domain specific languages. *POPL Workshop on Domain Specific Languages*, p. 89–96.
- GAZDAR G., KLEIN E., PULLUM G. & SAG I. (1985). *Generalized Phrase Structure Grammar*. Oxford: Blackwell.
- GELFOND M. & LIFSCHITZ V. (1988). The stable model semantics for logic programming. In *5th International Conference and Symposium on Logic Programming*, p. 1070–1080, Cambridge: MIT Press.
- HENOCQUE L. (2003). *Modeling Object Oriented Constraint Programs in Z*. Rapport interne, LSIS Research Report, available at <http://arXiv.org/abs/cs/0312020>.
- KARLSSON J. (1990). Constraint grammar as a framework for parsing running text. In *COLING-90, 13th International Conference on Computational Linguistics*, volume 3, p. 168–173: H. Karlgren.
- LA GRECA R. & DANIEL M. (2002). Modélisation déclarative : De la description vers les modèles. In *Proceedings of Journées Des Doctorants du LSIS*, p. 3–12.
- MAILHARRO D. (1998). A classification and constraint based framework for configuration. *AI-EDAM : Special issue on Configuration*, 12(4), 383 – 397.
- MITTAL S. & FALKENHAINER B. (1990). Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, p. 25–32.
- NEBEL B. (1990). Reasoning and revision in hybrid representation systems. *Lecture Notes in Artificial Intelligence*, 422.
- POLLARD C. & SAG I. (1994). *Head-Driven Phrase Structure Grammar*. Chicago: The University of Chicago Press.
- SABIN S. & FREUDER E. (1996). Configuration as composite constraint satisfaction. *Artificial Intelligence and Manufacturing Research Planning Workshop*, p. 153–161.
- SOININEN T., NIEMELÖ I., TIIHONEN J. & SULONEN R. (2000). Unified configuration knowledge representation using weight constraint rules. In *ECAI 2000 Configuration Workshop*.
- STUMPTNER M. (1997). An overview of knowledge-based configuration. *AI Communications*, 10(2), 111–125.