Implantation des Grammaires de Propriétés en CHR

Veronica Dahl, Philippe Blache

Implantation de Grammaires de Propriétés en CHR

Veronica Dahl[1&2] **& Philippe Blache**[2]

(1) LPG, Simon Fraser University
Vancouver, Canada
veronica@cs.sfu.ca
(2) xLPL, Université de Provence
Aix-en-Provence, France
pb@lpl.univ-aix.fr

RÉSUMÉ. *Nous proposons dans cet article une interprétation directe des formalismes linguistiques basés sur les contraintes dans laquelle les notions de dérivation et de hiérarchie laissent la place à la notion plus flexible de statisfaisabilité. De telles approches définissent l'acceptabilité de phrases en termes de propriétés devant être satisfaites par un groupe de catégories : par exemple le SN peut être décrit dans les grammaires de propriétés proposées dans [Blache01a] par un ensemble de propriétés comme la précédence (un déterminant précède un nom), unicité (il ne peut y avoir qu'un déterminant), d'exclusion (un pronom ne peut apparaître avec un déterminant), etc. Le résultat est alors proposé en termes de listes de propriétés qu'une entrée satisfait ou ne satisfait pas, une même structure pouvant comporter des descriptions avec des granularités différentes. La possibilité de définir des conditions sur les propriétés permet d'analyser des inputs incomplets ou incorrects de façon adaptable et efficace.*

ABSTRACT. *We propose a direct interpretation for constraint-based linguistic formalisms in which the notions of derivation and hierarchy give way to the more flexible notion of property satisfaction between categories. Such frameworks define sentence acceptability in terms of the properties that must be satisfied by groups of categories (e.g. English noun phrases can be described in Property Grammar terms [Blache01a] through a few properties such as precedence (a determiner must precede a noun); uniqueness (there must be only one determiner); exclusion (an adjective phrase must not coexist with a superlative); and so on. Rather than resulting in either a parse tree or failure, such frameworks characterize a sentence through the list of the properties it satisfies and the list of properties it violates. This allows us to parse incomplete and incorrect input in a very modular and adaptable, while efficient, manner.*

MOTS-CLÉS : *Grammaires de propriétés, CHR, CHRG, contraintes, interprétation directe*

KEYWORDS: *Property Grammars, CHR, CHRG, constraints, direct interpretation*

1. Introduction

Un des problèmes actuels pour le traitement automatique des langues naturelles est la réutilisabilité. Pour ce qui concerne l'analyse syntaxique, la question porte plus précisément sur le développement de systèmes à granularité variable. Par exemple certaines applications telles que les systèmes de synthèse de la parole nécessitent simplement des analyseurs superficiels tandis que d'autres comme la traduction automatique reposent sur des analyseurs approfondis. Dans certains cas, il est quelquefois nécessaire d'utiliser différents niveaux d'analyse au sein d'une même application (par exemple, le composant sémantique peut être utilisé pendant l'analyse syntaxique pour désambiguïser des parties de textes avant d'être synthétisées).

Les techniques d'analyse flexibles offrent de grands avantages dans cette perspective. En particulier, les techniques basées sur les contraintes permettent d'utiliser un même mécanisme, la satisfaction de contraintes, y compris en cas de grammaire incomplète, hétérogène, etc. De plus, une même structure (ou description syntaxique) peut comporter des sous parties à granularités différentes. De plus, à condition que toute l'information soit représentées sous forme de contraintes et qu'aucun autre mécanisme que la satisfaction ne soit utilisé, les contraintes peuvent être relâchée en fonction de critères définis par l'utilisateur.

Plusieurs théories linguistiques font une utilisation intensive de la notion de contrainte, en particulier HPSG (cf. [Pollard94], [Sag99]) ou la théorie de l'optimalité (cf. [Prince93]). Les modèles linguistiques basés sur les contraintes [Bès99, Blache00] considèrent les contraintes comme des propriétés entre des ensembles de catégories. Cette perspective présente plusieurs avantages, notamment la possibilité pour les constructions non standard de recevoir malgré tout des descriptions là où les approches classiques rejetteraient l'input sans pouvoir le traiter.

Ainsi que cela avait été souligné dans [Morawietz00], les règles CHR (*Constraint Handling Rules*, cf. [Frühwirth98]) permettent de décrire efficacement de tels modèles. Dans cet article nous décrivons une méthodologie d'implantation des grammaires de propriétés (cf. [Blache00]) s'appuyant exclusivement sur les contraintes, contrôlant l'analyse par un mécanisme basé sur les têtes et permettant une interprétation directe préservant les caractéristiques théoriques de cette approche. Nous utilisons pour cela un langage de programmation par contraintes appelé CHRG (CHR Grammars) décrit dans [Christiansen01] reposant sur CHR (cf. [Frühwirth98]). La méthodologie présentée ici a inspiré un modèle cognitiviste de formation de concept (cf. [Dahl04]) qui est utilisé dans des application de traitement d'information médicales (voir [Barranco04]). Pour des raisons de clarté et de complétude, nous commençons par une présentation rapide de ce modèle (section 2) , que nous spécialisons ensuite à l'analyse syntaxique des grammaires de propriétés (section 3). Dans la section 4 nous décrivons l'analyseur, qui repose essentiellement sur une simple règle de réécriture combinant deux catégories avec une troisième à condition que le résultat vérifie les propriétés impératives. La représentation de l'information syntaxique se fait à l'aide de graphe (ou réseaux de contraintes) qui peuvent se réduire à des arbres. Nous pré-

sentons également une propriété d'héritage permettant d'hériter à chaque étape des propriétés des niveaux précédents en calculant seulement les propriétés nécessaires et mettant à jour les autres. Cet analyse permet de réduire le surcoût entraîné par le calcul des propriétés face a la liberté donne a l'utilisateur pour les relaxer. La section 5 présente une discussion ainsi qu'une relation a d'autres travaux, et des tests pour l'analyseur sur un fragment de grammaire de propriété pour l'analyse du SN sont montres dans l'appendice. Si bien l'on se concentre ici sur l'analyse syntaxique, il faut dire que notre méthodologie est applicable a des propriétés de n'importe quel type.

2. Modélisation de connaissance constructive

2.1. *Présentation*

Le constructivisme repose sur l'idée que nous construisons notre monde plutôt que celui-ci ne soit déterminé par la réalité extérieure. Dans la mesure où l'idée de construction (solutions, preuves) est au coeur à la fois de l'informatique et de la programmation logique, le constructivisme peut constituer un pont naturel entre les sciences cognitives et l'informatique. Ces théories considèrent l'apprentissage comme la construction de nouveaux concepts à partir de concepts précédemment acquis. Pendant ce processus, l'apprenant sélectionne et transforme des informations, construit des hypothèses, prend des décisions en s'appuyant sur une structure cognitive. La formation de nouveaux concepts à partir de connus est également central à la logique cognitive ainsi qu'à la programmation logique (notamment dans le nouveau paradigme de Constraint Handling Rules (CHR).

[Dahl04] tire parti de ces connections naturelles pour développer un modèle cognitif de la construction de connaissances pouvant être directement exécuté en CHR. Dans ce modèle, l'information est sélectionnée comme un effet de bord de la recherche de règles CHR applicables et transformée automatiquement (ou simplement augmentée) lorsqu'une règle se déclenche. Des hypothèses peuvent être faites sous la forme d'assomptions (cf. [Dahl97]). Des décisions découlent de l'application normale des règles et la structure cognitive est donnée par les propriétés que les concepts provenant d'une règle donnée doivent satisfaire. De plus une marge de manoeuvre est possible et l'utilisateur peut déclarer les circonstances selon lesquelles une propriété peut être relâchée. Les concepts provenant du relâchement de propriété contienne une information indiquant quel concept a été formé, ainsi que les propriétés associées ayant été satisfaites et celles ayant été enfreintes. Cette caractéristique assure une exécutabilité directe tout en fournissant une flexibilité naturelle.

2.2. *Les propriétés et leur rôle dans la formation de concepts*

Considérons la règle CHR suivante, qui pourrait être rencontrée dans un système expert de design :

```
(1) couleur(A,bleu,Pb), couleur(B,jaune,Py) =>
        Pb> Py, Pg is Pb+Py | couleur(A-B,vert,Pg).
```

Une telle règle pourrait être utilisée pour indiquer qu'un matériel `A` est bleu à `Pb%` et le matériel `B` est jaune à `Py%`. Le pourcentage de bleu domine et les caractéristiques peuvent être combinées dans le matériel`A-B` qui est vert à `Pb+Py%`.

Notons la différence qualitative entre les deux tests dans la garde de la règle (1) : alors que le second test est utilitaire (i.e. un simple calcul de quelques valeurs devant entrer dans le nouveau concept), le premier test représente une propriété qui les attributs des deux concepts participants doivent satisfaire pour permettre la dérivation du nouveau concept. Nous identifions de tels tests sous le nom de *propriétés*. Elles doivent être utilisées par les trois prédicats primitifs décrits plus loin. Prendre en compte ces propriétés par ces primitives plutôt que par des tests Prolog permet au système de garder trace implicitement du degré de satisfaction de ces propriétés pour chaque nouveau concept.

2.2.1. *Les propriétés : définition*

Commençons par nommer les propriétés. Par exemple, appelons `être_vert` la propriété de la règle (1). Soit P la spécification prolog d'une propriété N, soit A la liste des attributs intervenant dans P, et soit L la liste dont la tête est N et la queue A. La forme générale d'une définition de propriété est donnée en (1), un exemple est fournit en (2).

```
(2) prop(L):- P.
(3) prop([etre_vert,Pb,Pj]):- Pb>Pj.
```

2.2.2. *Relaxation de propriétés*

La flexibilité dans la dérivation de concepts est obtenue par la possibilité de relâcher la satisfaction de certaines propriétés en fonction de critères fournis par l'utilisateur. Une liste de propriétés satisfaites et non-satisfaites sera produite dans le cas de concepts pour lesquels des propriétés ont été relâchées. Dans certains cas, le degré "d'incorrection" peut être retourné de cette façon. Le relâchement d'une propriété N, de façon à permettre la dérivation de concepts malgré tout, se note comme indiqué en (4).

```
(4) relax(N).
(5) relax(etre_vert).
(6) relax(L,D).
(7) relax([etre_vert,Pb,Pj],2):- Pj > 2 * Pb, Pj<3*Pb.
```

Dans le cas de notre exemple, la propriété *être_vert*, qui est satisfaite quand le pourcentage de bleu est plus grand que le jaune, peut être relâchée comme indiqué en (5). Les degrés d'acceptabilité sont définis en (6) à travers une version binaire des primitives de relâchement, où L est décrit comme supra et D est une mesure d'acceptabilité. Ainsi, l'exemple (7) indique comment garder trace de la façon dont la propriété *être_vert* a été enfreinte.

Ceci exprime le degré de violation de la propriété : le niveau de jaune est supérieur à deux fois mais inférieur à trois fois le niveau de bleu. Notre système recueille automatiquement les degrés de violation de la propriété et en rend compte à l'utilisateur.

2.2.3. *L'usage des propriétés dans les règles*

Soit *prop* le nom de ma propriété, soit L décrit comme supra, soit D le degré d'acceptation et $L1$ la concaténation de L et D. Il suffit d'appeler `acceptable(prop(L1))` dans la garde de la règle. Pour résultat, la propriété concernée sera évaluée et D unifié avec vrai, faux ou une autre valeur de degré définie par l'utilisateur. La règle (1) peut alors être exprimée comme suit :

```
(8) couleur(A,bleu,Pb), couleur(B,jaune,Pj) =>
          acceptable(etre_vert(Pb,Pj,D), Pg is Pb+Pj |
          colour(A-B,vert,Pg).
```

Une liste de propriétés satisfaites et enfreintes ainsi que le degré de non-satisfaction si nécessaire, est fourni pour chaque propriété définie dans un programme CFG donné.

2.3. *CFG : Concept Formation Grammars*

Les CFG (Concept Formation Grammars) sont construites sur la base de CHRG (cf. [Christiansen01]), qui sont à CHR ce que les DCG sont à Prolog. De la même façon que les règles DCG sont compilées en Prolog, les règles CHRG sont compilées en règles CHR. Par exemple, la règle CHRG (3) se compile en règle CHR (4) :

```
(9)   determinant, nom, verbe ::> phrase.
(10)  determinant(P1,P2), nom(P2,P3), verbe(P3,P) =>
           phrase(P1,P).
```

Les CFG sont des CHRG plus des propriétés et l'acceptabilité définie de la même façon que pour des règles CF normales. Dans la mesure où les propriétés appartiennent à la garde de la règle qui contient les appels à prolog plutôt que les symboles de grammaire, aucun dispositif extérieur n'est nécessaire pour aller des programmes CF aux grammaires CF. Pour les applications dans lesquelles les frontières de phrases doivent être manipulées explicitement, CHRG inclut des facilités pour rendre explicite les frontières de mots. De plus, elles incluent le contexte droit et gauche de même que les règles CHRG incluent également les notations pour les manques et le matching parallèle, non utilisés ici. Nous pouvons définir plus formellement le sous-ensemble de CFG utile pour nos besoins comme suit :

Définition : une CFG est un ensemble fini de règles CHRG de la forme :

```
(11) Head => Guard | Body
```

où Head et Body sont des conjonctions d'atomes et Guard est un test construit de primitives Prolog plus le prédicat spécifique à CF property/2 ; les variables dans Guard et Body peuvent aussi paraître dans Head ; si Guard est la constante "true" alors

elle est omise, ainsi que la barre verticale. Sa signification logique est la formule : $\forall(Guard \Rightarrow (Head \Rightarrow Body))$, et la signification d'un programme ou grammaire est donnée par conjonction. Les règles CFG sont interprétées comme des règles de réécriture sur des réseaux de symboles des grammaires logiques. Par exemple, la grammaire CHRG qui suit [1] :

```
[un] ::> det(singulier).          [rit] ::>  verb(singulier).
[garcon] ::>  nom(singulier).     [garcons] ::> nom(pluriel).

determinant(Number),noun(Number),v(Number)::> phrase(Number).
```

peut servir pour analyser des phrases correctes, telles que “un garçon rit". Pour accepter aussi bien des phrases qui ne s'accordent pas en nombre, tout en signalant l'erreur comme effet de bord de l'analyse, l'on peut remplacer la dernière règle par les règles CFG suivantes :

```
determinant(Ndet), nom(Nn), v(Nv) =>
      acceptable(accord,Ndet,Nn,Nv,N,_)x | phrase(N).
prop(accord,[Ndet,Nn,Nv,N]):- Ndet=Nn, Nn=Nv, !, N=Nv.
prop(accord,[Ndet,Nn,Nv],mismatch).
relax(accord).
```

La propriété d'accord en nombre apparaîtra dans la liste de propriétés violées, qui est construite automatiquement comme résultat de l'analyse. Dans le cas de notre exemple, l'on n'a pas besoin d'utiliser la valeur booléenne calculée par `acceptable/2` dans son deuxième argument, donc on le laisse anonyme. Jusqu'à présent, l'on est resté dans le cadre traditionnel de règles de réécriture qui implicitement définissent un arbre d'analyse. Ensuite l'on utilisera CFGs pour implanter le modèle linguistique plus flexible des grammaires de propriétés.

3. CFG pour les Grammaires de Propriétés

3.1. *Préliminaires : Grammaires de Propriétés*

L'idée de base des *Grammaires de Propriétés* est de représenter des types différents d'information syntaxique séparément. Dans cette approche, la structure syntaxique n'est pas exprimée en termes de hiérarchie, mais seulement par des relations entre catégories. De telles relations n'ont pas de contraintes topologiques, par exemple elles peuvent être croisées. En plus, seulement des relations entre objets sont utilisées pour décrire une catégorie. En conséquence, la notion de constituant n'est plus cruciale pour le processus de description : une catégorie est spécifiée par un ensemble de propriétés, non par un ensemble de constituants. Autrement dit, le fait que plusieurs catégories appartiennent à un réseau de relations indique qu'elles caractérisent une catégorie de haut niveau. Une catégorie syntaxique se décrit alors par un ensemble

1. Les symboles terminaux sont en notation liste, style DCGs

de propriétés qui représentent des relations entre d'autres catégories (lexiques ou syntaxiques). Dans cette approche, l'objectif est d'expliciter toutes les différentes relation qui peuvent exister. L'on distingue dans cette perspective les types suivants d'information :

– précédence linéaire, qui est une relation d'ordre entre des catégories,

– sous-catégorisation, qui indique des relations de co-ocurrence entre catégories ou entre ensembles de catégories,

– l'impossibilité de co-occurrence entre catégories,

– l'impossibilité de répéter une catégorie,

– l'ensemble minimal de constituants obligatoires (normalement, un seul constituant) qui constituent la tête,

– des relations sémantiques entre catégories, en termes de dépendance.

Ces types différents d'information correspondent a des propriétés différentes, respectivement : *linéarité, exigence, exclusion, unicité, obligation, dépendance*. Telle information peut toujours être exprimée en termes de relations entre catégories, comme les exemples suivant le montrent :

– Précédence linéaire : $Det \prec N$ (un déterminant précède le nom)

– Dépendance : $AP \rightsquigarrow N$ (une phrase adjectivale dépend du nom)

– Requirement : $V[inf] \Rightarrow to$ (une infinitive se réalise avec *to*)

– Exclusion : $seems \not\Rightarrow ThatClause[subj]$ (le verbe *seems* ne peut pas avoir *That* sujets de clause)

Toutes les catégories syntaxiques catégories se caractérisent par un ensemble de relations qui forment un graphe connexe. La description syntaxique d'un langage consiste de toutes les différentes relations qui peuvent être exprimées entre catégories. Une relation (aussi appelée une propriété) peut être conçue comme une contrainte sure l'ensemble de catégories. Une grammaire est alors l'ensemble de contraintes, et la satisfaisabilité devient le noyau du processus d'analyse (voir [Blache01b]). Ce qui est intéressant dans cette approche est qu'il n'y a pas besoin d'information implicite, par exemple sous la forme d'un mécanisme spécifique. En particulier, il n'y a pas besoin de construire une structure avant de pouvoir vérifier ses propriétés, comme il arrive dans des approches génératifs classiques. En plus, l'utilisation de seule la satisfaisabilité a des conséquences importantes dans la conception de la structure syntaxique. L'évaluation du système de contraintes pour un ensemble donne de catégories nous permet de spécifier précisément l'ensemble de propriétés qui sont vérifiées. De la même façon, dans le cas de données mal formées, une telle évaluation identifie précisément l'ensemble de propriétés satisfaites et violées. Un tel résultat a alors un intérêt profond, dans le sens qu'il identifie précisément toutes les spécificités des données. Dans les *Grammaires de Propriétés* cette information constitue le résultat d'une analyse, qui n'est autre que l'état du système de contraintes après évaluation.

3.2. *Schéma d'analyse*

Le mécanisme de base dans les problèmes de satisfaction de contraintes est de trouver, pour un ensemble donne de variables, une affectation qui satisfasse le système de contraintes. Dans le problème qui nous occupe, les variables prennent leurs valeurs dans l'ensemble de catégories. Une affectation est faite a partir des données (i.e. La phrase a analyser). En partant de l'ensemble de catégories lexicales correspondant aux mots de la phrase, on évalue toutes les affectations possibles (i.e. sous-ensembles de catégories). Quand une catégorie syntaxique est caractérisée, on l'ajoute à l'ensemble de catégories à évaluer. Cette approche est basiquement incrémentale dans le sens que n'importe quel sous-ensemble de catégories peut être évalué. Cela signifie qu'une affectation $\mathcal{A}$ peut être complétée par addition d'autres catégories. Quand l'on déduit après le premier pas du processus, il est alors possible de compléter les premières affectations (faites avec des catégories lexicales) avec de nouvelles, syntaxiques.

On a vu que dans les Grammaires de Propriétés, une catégorie se décrit par un ensemble de contraintes. Mais réciproquement, il est possible d'identifier une catégorie a partir d'un propriété donnée. Ceci est typiquement le cas avec des propriétés qui expriment des relations entre catégories, telles que linéarité, exigence, obligation et dépendance. L'évaluation de telles propriétés permet de déduire que la catégorie syntaxique a laquelle s'attache la propriété est en train d'être caractérisée. On a fait référence à de telles propriétés sous le nom collectif de contraintes de *sélection*.

Le rôle de contraintes de sélection est central dans notre approche. La raison pour laquelle de telles contraintes nous permettent de sélectionner la catégorie caractérisée est qu'elles sont locales a cette catégorie. En plus, dans certains cas elles ont un rang global sur la catégorie : leur valeur de satisfaisabilite (i.e. satisfaite or violée) ne peut pas changer pour une catégorie donnée, n'importe quel est le sous-ensemble de constituants. Aussitôt la contrainte en mesure d'être évaluée, sa valeur est permanente. Par exemple, quand une contrainte de linéarité ou de dépendance est satisfaite, l'addition de constituants nouveaux à la catégorie ne peut pas changer ce fait. D'autres types de contraintes doivent être ré-évaluées à chaque pas. Par exemple, quand l'on ajoute une nouvelle catégorie, on doit vérifier que l'unicité et l'exclusion se satisfont encore. Dans la suite, on appelle ce dernier type des *contraintes de filtrage*. Contrairement aux *contraintes de sélection*, on ne peut pas déduire la matérialisation d'une catégorie syntaxique a partir de leur évaluation. Elles jouent un rôle de filtrage dans le sens qu'elles excluent quelque construction.

Un autre type de contrainte, que l'on appellera, *contraintes récupérables* peut être satisfaite par l'incorporation d'une catégorie de plus dans une phrase donnée pour laquelle, sans cette catégorie ajoutée, la contrainte ferait échec.

Examinons les conséquences de tout cela sur l'évaluation des contraintes. Comme expliqué auparavant, le principe repose sue la complétion des affectations originales par des novelles catégories quand elles sont déduites. Dans la mesure où les contraintes de sélection (dès qu'elles peuvent être évaluées) sont valides pour une affectation complète, quels que soient ses constituants, il n'est pas nécessaire de la recalculer.

En d'autres termes, quant une affectation $\mathcal{A}$ est construite par complétion d'une autre affectation $\mathcal{A}'$, l'ensemble de contraintes de sélection de $\mathcal{A}'$ est hérité par $\mathcal{A}$.

La section 3.7 étudie les types différents de propriétés et leur conséquence pour de nouvelles affectations, par rapport à une instance spécifique du schéma général d'analyse présenté ici. Dans ce qui suit, on note les contraintes de sélection et filtrage comme $\mathcal{R}_{select}(\mathcal{C}, XP)$ et $\mathcal{R}_{filter}(\mathcal{C}, XP)$ où $\mathcal{C}$ est la contrainte et XP la catégorie syntaxique à laquelle la contrainte est associée. Pour une affectation $\mathcal{A}$, une contrainte est pertinente (ou peut être évaluée) quand les catégories de $\mathcal{A}$ sont un sous-ensemble des catégories participant à C. Nous remarquons le fait que $\mathcal{A}$ peut être évaluée pour une contrainte comme suit : $\mathcal{A}/\mathcal{R}_{select}(\mathcal{C}, XP)$. Nous remarquons l'ensemble de contraintes de filtrage et sélection pour une catégorie donnée *XP* par $\mathcal{R}(C, XP) = \mathcal{R}_{select}(\mathcal{C}, XP) \cup \mathcal{R}_{filter}(\mathcal{C}, XP)$. Finalement, nous notons l'état du système de contraintes Σ pour une affectation $\mathcal{A}$ après évaluation par $SAT(\mathcal{A}, \Sigma)$. Chaque catégorie est indexée par ses bornes, notées $c_{(i,j)}$.

Soit $\mathcal{K}$ l'ensemble de catégories, soit $c_i, c_{i+1} \in \mathcal{K}$

1. $\mathcal{A} \leftarrow \{c_{(i,j)} c_{(j+1,k)}\}$
2. if $\exists \mathcal{A}/\mathcal{R}_{select}(\mathcal{C}, XP)$
3. instantiate $XP_{(i,k)}$
4. $\Sigma_{XP} = \bigcup \mathcal{R}(\mathcal{C}, XP)$
5. $Char(\mathcal{A}) \leftarrow SAT(\mathcal{A}, \Sigma)$
6. $\mathcal{A}' \leftarrow \mathcal{A} \cup \{c_{k+1,l}\}$
7. while $SAT(\mathcal{A}', \Sigma)$ acceptable
8. $\mathcal{A} \leftarrow \mathcal{A}' ; Char(\mathcal{A}) \leftarrow SAT(\mathcal{A}, \Sigma)$
9. $\mathcal{A}' \leftarrow \mathcal{A} \cup \{c_{(l+1,m)}\}$

Dans ce schéma d'algorithme, le mécanisme consiste à évaluer la caractérisation de toutes les séquences de catégories. La spécificité des contraintes de sélection s'utilise comme un mécanisme de contrôle : quand une contrainte de sélection est satisfaite, la catégorie décrite XP est instanciée et l'ensemble relié de contraintes Σ est actif. Il est intéressant de noter qu'une catégorie syntaxique peut être projetée par n'importe quelle contrainte de sélection, indépendamment des informations de constituants (en particulier la tête). Chaque nouvelle affectation, construite par addition de nouvelles catégories juxtaposées à l'ensemble initial, est alors évaluée.

Une telle complétion de l'affectation initiale est possible quand la satisfaisabilite de Σ pour cette nouvelle affectation est *acceptable* (cf. ligne 7). Cette notion implante la flexibilité de l'analyseur. Quand on a besoin de construire seulement des structures grammaticales, l'*acceptabilité* se réduit a la satisfaisabilite. Mais pour des besoins plus flexibles de l'analyse (e.g. langage parlé), on peut relâcher des contraintes. L'ensemble de contraintes relâchées, leur nombre, etc. est indiqué à ce point. Finalement, le processus général est répété jusqu'à ce qu'on ne puisse plus ajouter de nouvelles catégories.

Ce schéma d'analyse propose un cadre général dans lequel des contraintes peuvent être intégrées. Chaque propriété est implémentée par un solveur de contraintes. Le mé-

canisme consiste à construire une caractérisation pour chaque affectation possible (i.e. n'importe quel sous-ensemble de catégories). La particularité des contraintes de sélection joue un rôle important dans ce schéma. Dans une technique ascendante classique, le mécanisme consiste à trouver une *poignée* qui lie un ensemble de catégories avec un non terminal. Une telle relation dans notre approche s'établit entre un ensemble de propriétés et une catégorie. A la différence des techniques de structure de phrase, la notion de constituant ne joue aucun rôle particulier. Dès qu'une contrainte de sélection est évaluée, la catégorie syntaxique correspondante est ajoutée à l'ensemble des catégories et toutes les contraintes participant dans sa description sont activées. Concretètement, toutes les contraintes de sélection peuvent être évaluées sans avoir a connaître la catégorie du niveau plus haut, en contraste avec des contraintes de filtrage qui doivent être activées. Des stratégies différentes peuvent s'appliquer suivant les besoins de l'analyse. Une application restreinte stipule que toutes les contraintes doivent être satisfaites. Dans ce cas, seulement des caractérisations grammaticales sont construites, toutes les structures mal formées sont éliminées. Pour des applications plus flexibles, typiquement dans le cas de l'analyse de matériel de langue parlée, on doit relâcher des contraintes. Dans ce cas, les caractérisations peuvent violer des contraintes. Nous verrons comment utiliser les CFGs dans notre approche pour arriver a une interprétation directe dans ces types d'applications flexibles.

3.3. *Interprétation directe*

Notre méthodologie pour Grammaires de Propriétés a été conçue pour fournir une interprétation directe des règles de Grammaires de Propriétés. Cela constitue une contribution intéressante par rapport aux Grammaires de Propriétés elles-mêmes, mais aussi une preuve de concept nouveau et important qui peut montrer le chemin pour n'importe quel formalisme d'analyse basé sur des contraintes qui relie des catégories contextuellement a travers leur propriétés.

A notre connaissance, notre méthodologie est la première permettant l'expression de telles contraintes d'analyse directement exécutable de façon efficace. Nous pouvons ainsi dire que notre approche est aux grammaires basées sur les propriétés contextuelles ce que les DCGs sont aux grammaires indépendantes du contexte : elle constitue un formalisme descriptif directement exécutable au même titre que les DCGs. [2]. Nous décrivons dans cette section, les différents composants de notre méthodologie : la notion de catégorie étendue, qui inclut non seulement l'information traditionnelle comme les types de catégories ou les traits, mais également la caractérisation de la catégorie en termes de contraintes satisfaites ou non satisfaites ainsi que la notation modulaire avec laquelle un utilisateur définit les propriétés dans une grammaire donnée, la règle unique d'analyse et enfin la propriété d'héritage.

2. Bien entendu, les DCG permettent également une analyse dépendante du contexte, mais la sensibilité au contexte ne peut être directement représentée par la contiguïté de symboles, elle doit être indirectement exprimée dans des arguments ou mécanismes additionnels comme l'implication linéaire.

Précédence linéaire	*Dépendance*	*Constituance*
prec(det,n,sn). prec(det,sa,sn). prec(n,sa,sn).	dep(det,n,sn). dep(n,sa,sn).	cons(sn,[det,adj,sa,n]). cons(sa,adj]).

Unicité	*Exclusion*	*Exigence*	*Syntagmes*
one(det,sn).	exclude(sa,sup,sn).	req(n,det,sn).	xp(sn). xp(sa).

Figure 1. *Propriétés utilisateur*

3.4. *Catégories étendues*

Les catégories étendues sont de la forme : *cat(Name, Features, Graph, Sat, Unsat)* où *Name* est le nom de la catégorie, *Features* une liste de traits associés à la catégorie, *Graph* est un graphe d'analyse qui est obtenu en effet de bord de l'analyse (construit y compris dans des cas d'input mal formé). *Sat* et *Unsat* sont respectivement la liste des propriété satisfaites et non satisfaites que les constituants de *Name* dans *Graph* vérifient. Dans le cas de catégories lexicales, les listes *Sat* et *Unsat* sont vides. Ces catégories sont créées automatiquement à partir des définitions lexicales qui sont faites en termes de CHRG comme dans l'exemple (12) qui se compile en (13) :

```
(12) [le] ::> cat(det,[singulier,masculin]).
(13) [le] ::> cat(det,[singulier,masculin],det(le),[],[]).
```

Dans la mesure où il s'agit de règles CHRG (i.e. des règles de grammaires par opposition à des règles CHR), les frontières de mots ne sont pas exprimées. En cas de nécessité, on peut y accéder en ajoutant les arguments `:(Start,End)` Après la catégorie qui unifiera `Start` au point de départ de la catégorie et `End` à son point terminal. De même, il est possible d'écrire une règle CHR qui considère `cat/5` non pas en tant que règle de grammaire, mais en tant que contrainte CHR. Dans ce cas, `Start` et `End` sont accessibles en tant que premiers arguments de la contrainte correspondante, `cat/7`.

3.5. *Propriétés utilisateur*

Notre système permet à l'utilisateur d'entrer les propriétés de précédence linéaire, dépendance, exigence, exclusion, constituance et unicité décrites dans la grammaire en tant que prédicats : `prec/3, dep/3, req/3, exclude/3, cons/2`, et `one/2`. La figure 3.5 présente l'exemple simplifié du SN.

Il est intéressant de noter que la propriété de constituance n'est pas véritablement indispensable dans la mesure où lorsque les propriétés de sélection sont vérifiées, la constituance est déduite par effet de bord. De la même façon, les définitions de syntagmes peuvent être inférées de n'importe quelle autre propriété. Cependant, les définir explicitement rend le système plus simple et lisible. Les définitions des propriétés

```
cat(Cat,Features1,Graph1,Sat1,Unsat1) :(Start1,End1),
cat(Cat2,Features2,Graph2,Sat2,Unsat2) :(End1,End2) : :>
    xp_or_obli(Cat2,XP), ok_in(XP,Cat),
    acceptable(precedence(XP,Start1,End1,End2,Cat,Cat2,Sat1,
                Unsat1,SP,UP,BP),
    acceptable(dependency(XP,Start1,End1,End2,Cat,Features1,Cat2,
                Features2,SP,UP,SD,UD,BD),
    build_tree(XP,Graph1,Graph2,Graph,ImmDaughters),
    acceptable(unicity(Start,End2,Cat,XP,ImmDaughters,SD,UD,SU,UU,BU),
    acceptable(requirement(Start,End2,Cat,XP,ImmDaughters,SU,UU,SR,UR,BR),
    acceptable(exclusion(Start,End2,Cat,XP,ImmDaughters,SR,UR,Sat,
                Unsat,BE) | cat(XP,Features2,Graph,Sat,Unsat).
```

Figure 2. *Inférence de nouvelles catégories*

données par l'utilisateur sont appelées par les prédicats système qui vérifient chacune de ces propriétés pour un ensemble donné de catégories.

3.6. *Inférence de nouvelles catégories :* one rule fits all

3.6.1. *Constituants adjacents*

Une simple règle (cf. figure 2) est suffisante pour une analyse ascendante complète dans le cas de constituants adjacents. Cette règle combine deux catégories consécutives (l'une étant de type `XP`) dans une troisième en testant chacune des propriétés de la paire et en créant la nouvelle liste de propriété par héritage de propriétés (cf. section suivante).

Cette règle teste tout d'abord qu'une des catégories est de type `XP` ou la tête de `XP` et que l'autre catégorie est un constituant acceptable de `XP`. Elle vérifie ensuite les propriétés de la grammaire en construisant les listes de propriétés satisfaites et non satisfaites. Enfin, elle infère une nouvelle catégorie de type `XP` en fournissant sa caractérisation. En pratique, cette règle se déplie en deux parties symétriques pour s'adapter au cas où la catégorie XP apparaît avant la catégorie Cat qui doit lui être incorporée.

3.6.2. *Obligation*

La propriété d'obligation, qui indique quelles catégories (les têtes) sont obligatoires pour un syntagme, n'est pas représentée explicitement comme un appel, à la différence des autres propriétés. L'obligation est assurée par des règles qui projettent un noyau de syntagme (ses catégories obligatoires) au niveau syntagmatique. Dans la mesure où les règles contextuelles incorporent plus de catégories aux syntagmes déjà trouvés, et dans la mesure où la plus petites catégories syntagmatique inclut le noyau syntagmatique, il n'y a pas de possibilité pour une catégorie obligatoire de ne pas être réalisée.

3.7. *Héritage de propriétés*

Toutes les propriétés ne sont pas héritées du syntagme dans lequel une catégorie doit être incorporée. Nous examinons dans cette section les propriétés de précédence, dépendance, exigence, exclusion et unicité d'une catégorie donnée pour savoir si elles sont héritées ou pas du syntagme formé par cette catégorie à laquelle une autre s'adjoint.

Soit `XP` un syntagme dans lequel on cherche à incorporer une catégorie `Cat`. Soit `P` l'une des propriétés mentionnées ci-dessus.

– `Fp(XP)` = ensemble des propriétés `P` non satisfaites pour `XP`

– `Sp(XP)`= ensemble des propriétés `P` satisfaites pour `XP`

– `fp(Cat,XP)` = ensemble des propriétés `P` non satisfaites pour `Cat` et `XP`

– `sp(Cat,XP)`= ensemble des propriétés `P` satisfaites pour `Cat` et `XP`

Désignons par `XP+Cat` le nouveau constituant (de type `XP`) formé par incorporation de `Cat` dans `XP`.

Définition : On dit qu'une propriété `P` est monotone pour le succès (resp. pour l'échec) si toutes les propriétés `P` qui sont satisfaites (resp. non satisfaites) pour `XP` sont aussi satisfaites (resp. non satisfaites) pour `XP+Cat`.

Nous considérons trois types de propriétés : sélection, filtrage et and récupérable.

3.7.1. *Propriétés de sélection*

Les propriétés de sélection sont à la fois monotones pour le succès et pour l'échec. En GP, ce sont les propriétés de précédence linéaire et de dépendance : les constituants qui précèdent ou dépendent d'autres dans un syntagmes plus petit continuent de précéder ou dépendre de ces constituants dans des syntagmes étendus. De même, ceux qui ne satisfont pas les contraintes de dépendance ou de linéarité dans un syntagme ne les satisfont pas plus dans son extension.

3.7.2. *Propriétés de filtrage*

Les propriétés de filtrage sont monotones pour l'échec mais pas pour le succès. C'est le cas par exemple des propriétés d'unicité ou d'exclusion qui, si elles ne sont pas satisfaites pour un ensemble de constituants, ne le sont toujours pas pour un ensemble plus grand. Cependant, leur satisfaction pour un sous-ensemble ne garantit pas la satisfaction pour un ensemble plus grand qui peut introduire par exemple une duplication ou un défaut d'exclusion.

3.7.3. *Propriétés de récupération*

Les propriétés telle l'exigence sont récupérables car si elles échouent pour un `XP` donné, leur extension par une catégorie `Cat` peut récupérer cette erreur en incorporant l'élément requis. Les propriétés récupérables sont monotones pour le succès (par

Filtrage	Unicité	FU(XP+Cat)= FU(XP) ∪ {fu(Cat,XP)}
		SU(XP+Cat)= SU(XP) ∪ {su(Cat,XP)} - {fu(Cat,XP)}
	Exclusion	FE(XP+Cat)= FE(XP) ∪ {fe(Cat,XP)}
		SE(XP+Cat)= SE(XP) ∪ {se(Cat,XP)}-{fe(Cat,XP)}
Sélection	Précédence	FP(SK+Cat)= FP(XP) ∪ {fp(Cat,SK}
		SP(XP+Cat)=SP(XP) ∪ {sp(Cat,XP)}
	Dépendance :	FD(XP+Cat)= FD(XP) ∪ {fd(Cat,XP)}
		SD(XP+Cat)= SD(XP) ∪ {sd(Cat,XP)}
Récupérable	Exigence	FR(XP+Cat)= FR(XP) ∪ {fr(Cat,XP)}
		SR(XP+Cat)= SR(XP) ∪ {sr(Cat,XP)}-{fr(Cat,XP)}

Figure 3. *Classification des propriétés*

exemple la satisfaction d'une exigence pour un ensemble de catégories est toujours valable pour une de ses extensions), mais pas pour l'échec.

La figure 3 présente le calcul des ensembles de propriétés satisfaites ou non satisfaites pour un `XP` incorporant une catégorie supplémentaire `Cat`.

4. Raffinement de la notion de caractérisation

La flexibilité est une propriété essentielle des systèmes d'analyse, en particulier pour la possibilité de traiter du tout-venant, y compris de la langue parlée. Cependant, augmenter le seuil de tolérance aux fautes diminue l'efficacité. Pour permettre une flexibilité maximale, ont propose d'indiquer quelles propriétés doivent être strictement respectée et lesquelles peuvent être relâchées. Par exemple, dans un système d'apprentissage d'une langue seconde, on peut avoir besoin de relâcher la dépendance de façon à être tolérant aux fautes d'accord. Cette caractéristique peut être exploitées dans une perspective de correction d'erreur : autoriser la formation d'input mal formés puis examiner les réparations possibles des erreurs une fois celles-ci localisées. Le niveau d'acceptabilité attendu est indiqué dans la définition utilisateur :

```
(14) tolerate_violations_of(Prec, Dep, Uni, Req, Excl).
```

dans laquelle une propriété est indiquée nécessaire ou pas par l'utilisateur dans l'argument correspondant. Cette approche se prête bien à l'expérimentation. Tout ce qu'il est nécessaire de faire pour tester différents niveaux de tolérance d'erreur est de faire varier la valeur (booléenne dans notre implantation) de l'argument. Du point de vue de l'implantation, il suffit d'assurer que la génération d'une catégorie n'est effectuée que pour si ses propriétés requises sont satisfaites. La caractérisation d'une catégorie devient donc une liste de propriétés satisfaites ainsi qu'une liste de propriétés non satisfaites, mais n'étant pas requises.

Il est bien entendu possible de raffiner la caractérisation en partitionnant l'ensemble des instances de chaque propriété dans un ensemble qui peut être relâché et un autre qui doit être satisfait. Il est également possible d'envisager l'implantation d'un degré d'acceptabilité.

5. Discussion et travaux connexes

Dans notre approche, les catégories syntaxiques sont inférées à partir de l'évaluation des propriétés sans avoir besoin d'information de constituance. Cet aspect a des conséquences importantes sur le rôle des contraintes dans le processus d'analyse. Un des problèmes avec les approches basées sur les contraintes est que celles-ci sont généralement exprimées sur des objets ou structures de haut niveau. C'est le cas par exemple en HPSG : les structures des traits complexes doivent d'abord être construites avant de pouvoir évaluer les contraintes. De la même façon, la théorie de l'optimalité génère également un ensemble de structures (ou structures candidates) et utilise ensuite les contraintes pour filtrer cet ensemble en ordonnant les candidats. Dans notre approche, toutes les contraintes peuvent être évaluées, à tout moment et pour n'importe quel ensemble de catégorie. Une telle évaluation permet d'ajouter dynamiquement de nouvelles informations. La satisfaction d'une contrainte de *sélection* instancie la catégorie syntaxique associée. Une telle instanciation est conçue comme un effet de bord de l'évaluation : la satisfaction d'une contrainte n'est pas dépendante de la connaissance d'une catégorie de niveau supérieur. En d'autres termes, l'information hiérarchique n'est plus prépondérante dans le processus d'analyse. Il devient donc possible d'évaluer séparément des sous-ensembles de contraintes, par exemple pour des applications ne nécessitant que la reconnaissance de certaines catégories (comme les SN pour la recherche d'information). Dans cette approche, la conception de la relation entre langage et grammaire est très différente de celle du paradigme génératif. Dans ce dernier, le langage est conçu comme étant généré par la grammaire. Dans les grammaires de propriétés, une grammaire est seulement utilisée comme un mécanisme de caractérisation des propriétés du langage.

Ainsi, plutôt que de réduire le rôle de l'analyse à la seule évaluation de la grammaticalité de l'input, nous pouvons proposer une vision plus flexible dans laquelle la sortie de l'analyseur est la description de toutes les propriétés de l'input. Concrètement, une telle description est constituée par l'état du système de contrainte après évaluation (en d'autres termes, l'ensemble des contraintes satisfaites ou enfreintes). Dans certains cas, une caractérisation peut ne contenir que des propriétés satisfaites. Dans d'autres, une ou plusieurs contraintes peuvent ne pas être satisfaites sans affecter véritablement l'acceptabilité de l'input correspondant.

Les GP partagent avec les Grammaires de Dépendance (cf. [Tesnière59]) un certain nombre de points communs. Elles sont en particulier équationnelles (voir sur ce point [Mel'čuk88]) et la notion de génération ou de dérivation entre une structure abstraite et une chaîne n'est pas utilisée. Cependant, alors que la propriété de dépendance joue bien entendu un rôle fondamental dans cette théorie, elle ne constitue en GP qu'un élément d'information parmi d'autres.

Une analyse précise de la complexité de notre approche reste à faire. Cependant, on peut voir que l'analyse part des noyaux de phrase et active les solveurs de contraintes en fonction de ces noyaux et des catégories adjacentes, convergeant ainsi vers la solution. L'interprétation des résultats est directe : on n'a qu'à regarder les résultats pour la

phrase complète (les résultats intermédiaires pouvant s'imprimer ou non, au besoin). Bien sûr, il reste le problème de la discontinuité des constituants, auquel on s'attaquera prochainement, peut-être en adoptant les principes de [Blache01a]. Bien sur, plus les contraintes seront relâchées, plus la vérification de contraintes introduira un peu de surcharge. Mais comme on évite de recalculer les propriétés qui sont héritées d'un niveau au prochain, cette surcharge (déjà pas trop lourde) est minimisée.

L'idée de dépasser le paradigme traditionnel du schéma hiérarchique en faveur d'une vision de l'analyse impliquant des propriétés sur des catégories plutôt que des schémas de réécriture a été décrite tout d'abord dans le paradigme 5P (cf.[Bès99]).

Des travaux préliminaires concernant la faisabilité d'une interprétation directe de ce type d'approches utilisant les contraintes avaient conduit à des résultats pessimistes : [Blache95] avait montré que le mécanisme de vérification de contraintes pour l'analyse syntaxique pouvait être très coûteux dans la mesure où dans une vision classique, le système de contrainte doit être vérifié à chaque étape. Nous avons montré cependant dans cet article comment contourner l'obstacle grâce à l'héritage de propriétés qui évite la nécessité de recalculer toutes les propriétés à chaque étape. Notre travail valide le modèle d'une analyse centrée sur les propriétés en préservant à la fois l'efficacité et le niveau de généralité de la théorie. De plus, une interprétation directe garantit une meilleure évolution du système initial : il est susceptible de s'adapter facilement aux modification de la théorie ainsi qu'aux étapes d'expérimentation.

Nous avons montré dans cet article comment les approches basées sur les contraintes peuvent être à la fois flexibles et efficaces, tout en préservant une correspondance entre les niveaux conceptuels et représentationnels, y compris pour des approches non standard comme les grammaires de propriétés dans lesquelles l'inférence de catégorie ne dépend pas de la notion de hiérarchie ou de constituant.

Les représentations autorisées par notre méthodologie sont directement exécutables et non déterministes. C'est une caractéristique intéressante dans la perspective de la programmation logique qui vise la déclarativité et l'expressivité. De plus, cette approche permet de décrire les caractéristiques de n'importe quelle catégorie, y compris incomplète ou mal formée. Un mécanisme simple d'analyse permet à l'utilisateur de raffiner la notion de caractérisation de catégorie en relâchant ou pas certaines contraintes. Les résultats obtenus permettent d'envisager des implantations à plus grande échelle.

Appendice – Quelques exemples

Toutes les contraintes sont ici potentiellement relâchées, les caractérisations sont indiquées dans les deux derniers arguments. Pour des raisons d'espace, une sortie complète est seulement fournie pour le premier exemple.

```
| ?- s3.
 <0> le <1> livre <2> jaune <3>
```

```
all(0,3),
begin(-1,0),
end(3,4),
token(0,1,le),
token(1,2,livre),
token(2,3,jaune),
cat(0,1,det,[sing,masc],det(le),[],[]),
cat(1,2,n,[sing,masc],n(livre),[],[]),
cat(1,2,sn,[sing,masc],sn(n(livre)),[],[]),
cat(0,2,sn,[sing,masc],sn(det(le),n(livre)),
        [prec(0,det,1,n,2),dep(0,det,1,n,2),
unicity(det,0,2),exige(n,det,0,2)],[]),
cat(2,3,adj,[sing,masc],adj(jaune),[],[]),
cat(2,3,sa,[sing,masc],sa(adj(jaune)),[],[]),
cat(1,3,sn,[sing,masc],sn(n(livre),sa(adj(jaune))),
        [prec(1,n,2,sa,3),dep(1,n,2,sa,3),unicity(n,1,3)],
        [exige(n,det,1,3)]),
cat(0,3,sn,[sing,masc],sn(det(le),n(livre),sa(adj(jaune))),
        [prec(1,n,2,sa,3),dep(1,n,2,sa,3),unicity(n,1,3),
          prec(0,det,1,n,3),dep(0,det,1,n,3),unicity(det,0,3),
          exige(n,det,0,3)],[])?
yes

| ?- s4.
 <0> le <1> le <2> livre <3>

cat(0,3,sn,[sing,masc],sn(det(le),det(le),n(livre)),
        [prec(1,det,2,n,3),dep(1,det,2,n,3),unicity(det,1,3),
          exige(n,det,1,3),prec(0,det,1,n,3),dep(0,det,1,n,3),
          exige(n,det,0,3)],[unicity(det,0,3)]) ?
yes

| ?- s5.
 <0> livre <1> le <2>

cat(0,2,sn,[sing,masc],sn(n(livre),det(le)),
        [dep(0,n,1,det,2),unicity(det,0,2),exige(n,det,0,2)],
        [prec(0,det,1,n,2)]) ?

yes
```

6. Bibliographie

[Abdennadher98] Abdennadher S. and Schütz H. (1998) “CHR : A flexible query language”, in proceedings of *Int. Conference on Flexible Query Answering Systems*, volume 1495 of LNCS, Springer-Verlag.

[Barranco04] Barranco-Mendoza, A., Persaoud, D.R. and Dahl, V. (2004) “A property-based model for lung cancer diagnosis”, in proceedings of *8th Annual Int. Conf. on Computational Molecular Biology, RECOMB 2004*, San Diego, California (accepted poster).

[Bès99] Bès G & P. Blache (1999) “Propriétés et analyse d’un langage, in proceedings of *TALN’99*.

[Blache95] Blache P. & N. Hathout (1995) "Constraint Logic Programming for Natural Language Processing",in proceedings of *NLULP’95*.

[Blache00] Blache P. (2000) “Constraints, Linguistic Theories and Natural Language Processing”, in *Natural Language Processing*, D. Christodoulakis (ed.), Lecture Notes in Artificial Intelligence, Springer.

[Blache01a] Blache P. (2001) *Les Grammaires de Propriétés : Des contraintes pour le traitement automatique des langues naturelles*, Hermès.

[Blache01b] Blache P. & J.-M. Balfourier (2001) “Property Grammars : a Flexible Constraint-Based Approach to Parsing”, in proceedings of *IWPT-2001*.

[Christiansen01] Christiansen, H. (2001) “CHR as grammar formalism, a first report”, Sixth Annual Workshop of the ERCIM Working Group on Constraints.

[Christiansen02] Christiansen, H. (2002) *CHR Grammar web site*, http ://www.ruc.dk/˜henning/chrg

[Christiansen02] Christiansen H. & V. Dahl (2002), “Logic grammars for diagnosis and repair”, in proceedings of *ICTAI’02*.

[Dahl04] Dahl V. and Voll K. (2004) “Concept Formation Rules : an executable cognitive model of knowledge construction”, in proceedings of *First International Workshop on Natural Language Understanding and Cognitive Sciences*, INSTICC Press.

[Dahl97] Dahl, V., Tarau, P. and Li, R. (1997) “Assumption Grammars for Natural Language Processing”. in *Lee Naish (ed.) Proc. Fourteenth International Conference on Logic Programming*, pages 256-270, MIT Press, 1997.

[Frühwirth98] Frühwirth T. (1998) “Theory and Practice of Constraint Handling Rules”, in *Journal of Logic Programming*, 37 :1-3.

[Gazdar85] Gazdar G., E. Klein, G. Pullum, I. Sag (1985), *Generalized Phrase Structure Grammar*, Blackwell.

[Hecksher02] Hecksher T., S. Nielsen & A. Pigeon (2002) *A CHRG model of the ancient Egyptian grammar*, Project report, Roskilde University, Denmark.

[Mel’čuk88] Igor Mel’čuk (1988) “*Dependency Syntax*", SUNY Press.

[Morawietz00] Morawietz F. (2000) “Chart parsing as contraint propagation”, in proceedings of *COLING-2000*.

[Pollard94] Pollard C. & I. Sag (1994), *Head-driven Phrase Structure Grammars*, CSLI, Chicago University Press.

[Prince93] Prince A. & Smolensky P. (1993), *Optimality Theory : Constraint Interaction in Generative Grammars*, Technical Report RUCCS TR-2, Rutgers Center for Cognitive Science.

[Sag99] Sag I. & T. Wasow (1999), *Syntactic Theory. A Formal Introduction*, CSLI.

[Shieber95] Shieber S., Y. Schabes & F. Pereira (1995) “Principles and implementation of deductive parsing”, in *Journal of Logic Programming*, 24(1-2) :3-36, 1995.

[SICSTUS03] Swedish Institute of Computer Science (2003) *SICStus Prolog user’s manual, Version 3.10*, Most recent version available at http ://www.sics.se/isl, 2003.

[Tesnière59] Tesniŕe L. (1959) *El/’ements de syntaxe structurale*, Klincksieck.