TALN 2004, Fès, 19–21 avril 2004

Vers un statut de l'arbre de dérivation : exemples de construction de representations sémantiques pour les Grammaires d'Arbres Adjoints

Sylvain Pogodalla
LORIA – Campus Scientifique
BP239
F-54602 Vandoeuvre-lès-Nancy
sylvain.pogodalla@loria.fr

Résumé – Abstract

Cet article propose une définition des arbres de dérivation pour les Grammaires d'Arbres Adjoints, étendant la notion habituelle. Elle est construite sur l'utilisation des Grammaires Catégorielles Abstraites et permet de manière symétrique le calcul de la représentation syntaxique (arbre dérivé) et le calcul de la représentation sémantique.

This paper suggests a definition for Tree Adjoining Grammar derivation trees, extending the usual notion. It results from using Abstract Categorial Grammars and enables, in a symmetric way, the computation of both the syntactic representation (derived tree) and the semantic representation.

Mots-clefs – Keywords

Sémantique, grammaires d'arbre adjoints, grammaires catégorielles, λ-calcul
Semantics, tree adjoining grammars, categorial grammars, λ-calculus

1 Introduction

Dans le processus de construction de la représentation sémantique d'une expression de la langue naturelle, une approche importante consiste à associer les règles de combinaison syntaxique et celles de combinaison sémantique. Si dans (Montague, 1974) la définition du calcul syntaxique se prête mal à l'implantation, les principes sous-jacents sont à la base de nombreux autres travaux menés sur des formalismes syntaxiques plus élaborés. Nous mentionnerons en premier lieu les grammaires catégorielles (Lambek, 1958; Moortgat, 1996), par le lien basé sur l'homomorphisme de Curry-Howard qu'elles établissent entre un calcul syntaxique logique et un calcul sémantique basé sur le λ-calcul. Puis, les grammaires HPSG, avec les travaux de (Copestake *et al.*, 1999), ou LFG (Dalrymple, 1999).

Nous nous intéressons ici plus particulièrement aux travaux réalisés dans ce cadre pour les grammaires d'arbres adjoint (TAG) (Joshi *et al.*, 1975; Abeillé, 1993). Le principe de ces grammaires est de proposer un calcul d'arbres présentant deux opérations : l'adjonction et la substitution. Ainsi, lors de l'analyse syntaxique d'une phrase, se construit *l'arbre dérivé*, résultat du calcul de toutes ces opérations sur des arbres issus du lexique, les arbres *initiaux* et les arbres *auxiliaires*. Cet arbre s'accompagne d'un autre arbre, *l'arbre de dérivation* qui indique les opérations entre les arbres du lexiques par des flèches orientées vers le bas s'il s'agit de substitution et vers le haut s'il s'agit d'adjonction (exemple de la figure 1).

Ce dernier arbre, en ce qu'il décrit les opérations de combinaison entre les arbres, a généralement été pris comme base dans les propositions de calcul de représentation sémantique pour les TAG (Schabes & Shieber, 1994; Candito & Kahane, 1998; Kallmeyer, 2002; Joshi *et al.*, 2003). Néanmoins, pour répondre à diverses contraintes, telle celle de la représentation en Français des quantificateurs par des arbres auxiliaires, ces structures nécessitent d'être enrichie, sans que cela ne règle tous les problèmes.

Figure 1: Arbre de dérivation

(Gardent & Kallmeyer, 2003) propose de résoudre ces problèmes en construisant directement sur l'arbre dérivé les représentations sémantiques, sans plus tenir compte de l'arbre de dérivation. Néanmoins se pose alors le problème de l'attachement et de la réutilisation des représentations ainsi construites.

Dans cet article nous nous proposons de revenir sur la notion d'arbre de dérivation. S'inspirant d'autres formalismes grammaticaux qui n'ont pas non plus de règles syntagmatiques, la notion que nous définissons permet de considérer de manière équivalente la représentation syntaxique et la représentation sémantique en tant que résultat d'un processus spécifié par l'arbre de dérivation. La différence reflète alors simplement le choix de langages formels différents pour la partie syntaxique (arbres) et la partie sémantique (langage de représentation sous-spécifié), ainsi que les définitions lexicales.

Pour ce faire, nous faisons appel au formalisme, introduit par (de Groote, 2001), des grammaires catégorielles abstraites (ACG). Puis nous montrons comment y modéliser les TAG, puis la sémantique $\mathbf{L_U}$ (Bos, 1995). Enfin nous explorons sur quelques exemples les résultats de cette approche.

2 Grammaires catégorielles abstraites

Les ACG, formalisme grammatical basé sur la logique linéaire (Girard, 1987), correspondent plus à un cadre grammatical dans lequel modéliser d'autres formalismes grammaticaux et leur apporter ses outils que comme un nouveau formalisme grammatical. Ceci en raison des caractéristiques suivantes :

- chaque ACG engendre deux langages : un *langage abstrait*, qui peut être vu comme un ensemble abstrait de structures grammaticales, et un *langage objet* représentant les formes réalisées des structures abstraites. Ici, le langage abstrait correspond à la structure grammaticale que l'on veut manipuler : l'arbre de dérivation. Il sera associé, par deux

ACG différentes, d'une part à la réalisation *syntaxique* sous forme d'arbres de TAG et d'autre part à la réalisation *sémantique* sous forme de formules de $\mathbf{L_U}$;

- les ACG reposent sur un ensemble réduit de primitives mathématiques, en particulier le connecteur implicatif de la logique linéaire $\multimap$. Ce connecteur permet de représenter l'implication comme consommateur de ressources : lorsqu'une formule A est utilisée avec une formule $A \multimap B$ pour produire une formule B, les deux premières formules ne sont plus utilisables par la suite dans la preuve, contrairement à la logique classique. La conséquence de ce comportement est que les λ-termes associés via l'isomorphisme de Curry-Howard (Howard, 1980) sont dits *linéaires*, c'est-à-dire que toute λ-abstraction lie une et une seule variable.

Dans (de Groote, 2001), les langages engendrés par les ACG sont des ensembles de λ-termes *linéaires*. Ces derniers généralisent aussi bien les langages de chaînes de caractères que ceux d'arbres. Cela suffit pour modéliser les TAG, c'est en revanche insuffisant pour la partie sémantique (par exemple la variable x est utilisée deux fois pour dire que l'individu qu'elle dénote est un chien $\mathbf{chien}(x)$ et qu'il est noir $\mathbf{noir}(x)$ dans l'expression *chien noir*). La motivation principale pour se limiter au λ-calcul linéaire tient dans le principe sous-jacent d'analyse dans ce cadre : il consiste en la résolution de problèmes de filtrage de λ-termes d'ordre supérieur. Or ce problème pris dans toute sa généralité est difficile. Suivant les cas, il est indécidable (Loader, 2003), de complexité encore inconnue (Dowek, 2001), ou décidable. Pour le filtrage de λ-termes linéaires, le problème est NP-complet (de Groote, 2000), ce qui en fait un bon candidat pour les langages des ACG.

Toutefois, si l'on considère pour le langage objet des termes du λ-calcul simplement typé, avec certaines conditions sur le lexique, la propriété de décidabilité est conservée (le problème est ramené à un problème de recherche de preuve dans le fragment multiplicatif de la logique linéaire, qui est décidable). Nous n'approfondirons pas ce point, mais c'est le cas pour les ACG que nous considérons ici. Par ailleurs, le problème ne se poserait que pour passer de la représentation sémantique à la représentation syntaxique (génération), l'analyse sémantique ne posant pas ces problàmes de décidabilité.

Pour les définitions précises des ACG, nous renvoyons le lecteur à (de Groote, 2001). Toutefois, certaines notions sont un peu différentes et nous les redéfinissons ici.

Definition 1 (Types). *Soit A un ensemble de types atomiques. L'ensemble $\mathcal{T}(A)$ des* types *construits sur A est défini par : $\mathcal{T}(A) ::= A | A \to \mathcal{T}(A) | \mathcal{T}(A) \multimap \mathcal{T}(A)$*

L'ensemble des types $t \in \mathcal{T}(A)$ tels que le symbole $\to$ n'apparaît pas dans t est l'ensemble des types linéaires implicatifs*, noté $\mathcal{T}_L(A)$. Les autres renvoie au λ-calcul simplement typé habituel.*

Definition 2 (Signature d'ordre supérieur). *Une* signature d'ordre supérieure *est un triplet $\Sigma = \langle A, C, \tau \rangle$ où :*

- *A est un ensemble de types atomiques ;*
- *C est un ensemble fini de constantes ;*
- *$\tau : C \to \mathcal{T}(A)$ qui assigne à chaque constante de C un type.*

Si $\tau : C \to \mathcal{T}_L(A)$, la signature est dite linéaire.

L'ensemble des termes typés $\Lambda(\Sigma)$ est défini de manière habituelle, avec l'aide de la logique linéaire pour manipuler aussi bien des termes linéaires que non linéaires. Nous utilisons ici les deux implications linéaire $\multimap$ et intuitionniste $\rightarrow$. Ceci est permis par le codage de la logique intuitionniste en logique linéaire par la traduction $A \rightarrow B \equiv (!A) \multimap B$ (Girard, 1987; Danos & Cosmo, 1992). Cela signifie qu'une variable de type α dans un terme de type $\alpha \rightarrow \beta$ pourra être abstraite plusieurs fois (mais, ici, au moins une), alors qu'elles ne l'est qu'une et une seule fois dans un terme de type $\alpha \multimap \beta$.

Definition 3 (Lexique). *Étant donnés une signature* linéaire *d'ordre supérieur* $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *et une signature d'ordre supérieur* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$*, un* lexique $\mathcal{L}$ de Σ_1 vers Σ_2 *est une paire* $\mathcal{L} = \langle F, G \rangle$ *telle que :*

- $F : A_1 \rightarrow \mathcal{T}(A_2)$ *est une fonction d'interprétation des types atomiques de* Σ_1 *comme des types implicatifs construits à partir de* A_2*. On appellera* F *également l'extension homomorphique de* F *à tous les types de* $\mathcal{T}(A_1)$ *;*
- $G : C_1 \rightarrow \Lambda(\Sigma_2)$ *est une fonction d'interprétation des constantes de* Σ_1 *comme des* λ*-termes construits à partir de* Σ_2*. On appellera* G *son extension homomorphique ;*
- *les fonctions d'interprétation sont compatibles avec la relation de typage, c'est-à-dire que pour tout* $c \in C_1$,$G(c) : F(\tau_1(c))$ *(le type de l'image de* c *est l'image du type de* c*).*

Dans la suite, on utilisera $\mathcal{L}(a)$ *pour* $F(a)$ *ou* $G(a)$ *suivant le contexte.*

Definition 4 (Grammaire catégorielle abstraite). *Une* grammaire catégorielle abstraite *est un quadruplet* $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ *où :*

- Σ_1 *est une signature* linéaire *d'ordre supérieure, et* Σ_2 *une signature d'ordre supérieure. Ils sont appelés* vocabulaire abstrait *et* vocabulaire objet *;*
- $\mathcal{L} : \Sigma_1 \rightarrow \Sigma_2$ *est un lexique ;*
- s *est un type atomique du vocabulaire abstrait, appelé le* type distingué *de la grammaire.*

Definition 5 (Langages abstrait et objet). *Soit* $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ *une grammaire catégorielle abstraite.*

1. *Le* langage abstrait $\mathcal{A}(\mathcal{G})$ *engendré par* $\mathcal{G}$ *est défini par* $\mathcal{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) | t : s\}$
2. *Le* langage objet $\mathcal{O}(\mathcal{G})$ *engendré par* $\mathcal{G}$ *est défini par* $\mathcal{O}(\mathcal{G}) = \{t \in \Lambda(\Sigma_2) | \exists u \in \mathcal{A}(\mathcal{G}) \text{ avec } t = \mathcal{L}(u)\}$

3 Les grammaires d'arbres adjoints comme une ACG

Dans cette section, nous allons montrer comment une grammaire TAG peut être modélisée avec une ACG. Nous reprenons cette modélisation de (de Groote, 2002).

Soit $G = \langle \Sigma, N, I, A, S \rangle$ une TAG. Σ, N, I, A et S sont respectivement l'ensemble des symboles terminaux, des symboles non terminaux, l'ensemble des arbres initiaux et auxiliaires et le symbole non terminal distingué de la grammaire. $\mathcal{G}^G = \langle \Sigma^G_{\text{DT}}, \Sigma^G_{\text{TAG}}, \mathcal{L}^G, s^G \rangle$ la grammaire catégorielle abstraite associée est construite de la manière suivante :

Σ^G_{DT} : à partir de G, on définit :

- $A^G_{\text{DT}} = \{\alpha_S, \alpha_A | \alpha \in N\}$
- $C^G_{\text{DT}} = \{c_T | T \in I\} \cup \{c_T | T \in A\} \cup \{I_\alpha | \alpha \in N\}$ avec
  - si $T \in I$ alors $c_T : \gamma_{1A} \multimap \cdots \gamma_{mA} \multimap \beta_{1S} \multimap \cdots \beta_{nS} \multimap \alpha_S$ où α est la racine de T, les γ_i ses nœuds intérieurs et les β_i ses nœuds où les substitutions ont lieu
  - si $T \in A$ alors $c_T : \gamma_{1A} \multimap \cdots \gamma_{mA} \multimap \beta_{1S} \multimap \cdots \beta_{nS} \multimap \alpha_A \multimap \alpha_A$ où α est la racine de T, les γ_i ses nœuds intérieurs et les β_i ses nœuds où les substitutions ont lieu
  - I_α est de type α_A

$\Sigma^G_{\textbf{TAG}}$ **:** à partir de G, on définit :

- $A^G_{\text{TAG}} = \{\tau\}$ où τ représentera le type arbre
- C^G_{TAG} est constitué de
  - pour chaque terminal u de Σ, une constante u de type τ
  - pour chaque non terminal α, des constantes $\alpha_i : \underbrace{\tau \multimap \cdots \multimap \tau}_{i \text{ fois}} \multimap \tau$ pour $1 \leq i \leq k_\alpha$, avec k_α le nombre maximum de fils que peut avoir un nœud étiqueté α dans G

Il reste maintenant à définir le lexique. Plutôt que de le définir dans sa généralité, nous donnerons les principes de construction à partir des arbres du tableau 1. On a alors :

- aux types α_S sont associés le type τ, c'est-à-dire qu'ils correspondent à des arbres ;
- aux types α_A sont associés le type $\tau \multimap \tau$, c'est-à-dire qu'ils correspondent à des fonctions transformant un arbre en arbre ;
- aux constantes I_α correspondent à l'identité $\lambda x.x$.

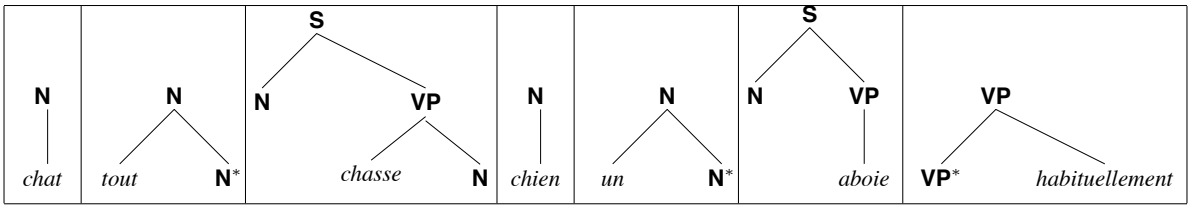


Table 1: Exemple de lexique TAG

Un arbre tel que celui de *aboie* est appelé à être défini de la manière suivante : $\mathbf{S}_2 N(\mathbf{VP}_1\ \textit{aboie})$ c'est-à-dire que $\mathbf{S}_2$ a deux fils : N et un deuxième qui est lui-même un arbre de racine $\mathbf{VP}_1$ qui a un seul fils *aboie*. Compte tenu de la substitution possible en N, mais aussi des adjonctions, en particulier en **VP**, le lexique lui associe la forme $\lambda V \lambda N.\mathbf{S}_2 N(V(\mathbf{VP}_1\ \textit{aboie}))$. En effet, si un arbre T, de *habituellement*, est adjoint en $\mathbf{VP}_1$, T prend l'arbre de racine $\mathbf{VP}_1$ et l'ajoute à lui-même, au niveau de son fils gauche, pour former un nouvel arbre. Autrement dit, T est de la forme $\lambda x.\mathbf{VP}_2 x\ \textit{habituellement}$, un terme d'ordre supérieur qui transforme un arbre en un autre arbre.

Cela nous conduit par exemple au lexique d'ACG du tableau 2[1]. On laissera le lecteur vérifier que le typage des termes est correct.

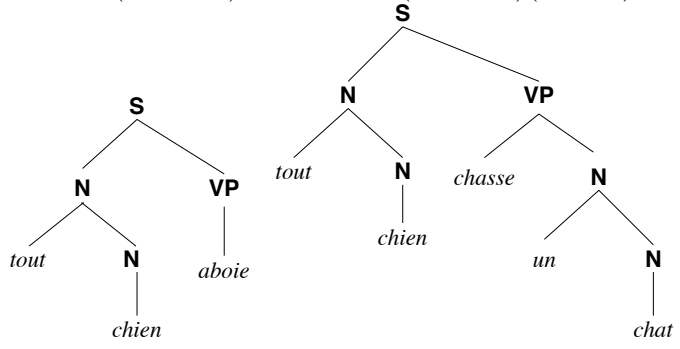
[1]Pour des raisons de concision, on omet les nœuds habituels *Det*, V, etc. Pour les mêmes raisons, on supprime comme paramètres les nœuds sur lesquels il n'y aura pas d'adjonction dans les exemple (on pourrait sinon les neutraliser en leur appliquant I_α).

Constante c	son type
c_{chien}	: $\mathbf{N}_A \multimap \mathbf{N}_S$
c_{chat}	: $\mathbf{N}_A \multimap \mathbf{N}_S$
c_{tout}	: $\mathbf{N}_A$
c_{un}	: $\mathbf{N}_A$
c_{aboie}	: $\mathbf{VP}_A \multimap \mathbf{N}_S \multimap \mathbf{S}_S$
c_{chasse}	: $\mathbf{VP}_A \multimap \mathbf{N}_S \multimap \mathbf{N}_S \multimap \mathbf{S}_S$
$c_{habituellement}$	: $\mathbf{VP}_A \multimap \mathbf{VP}_A$
$c' = \mathcal{L}(c) \in \Lambda(\Sigma_{TAG})$	
$c'_{chien} = \lambda N.N(\mathbf{N}_1\ \textit{chien})$	: $(\tau \multimap \tau) \multimap \tau$
$c'_{chat} = \lambda N.N(\mathbf{N}_1\ \textit{chat})$	: $(\tau \multimap \tau) \multimap \tau$
$c'_{tout} = \lambda x.\mathbf{N}_2\ \textit{tout}\ x$	: $\tau \multimap \tau$
$c'_{un} = \lambda x.\mathbf{N}_2\ \textit{un}\ x$	: $\tau \multimap \tau$
$c'_{aboie} = \lambda V.\lambda x.\mathbf{S}_2 x(V(\mathbf{VP}_1 \textit{aboie}))$	: $(\tau \multimap \tau) \multimap \tau \multimap \tau$
$c'_{chasse} = \lambda V.\lambda x.\lambda y.\mathbf{S}_2 x(V(\mathbf{VP}_2 \textit{chasse}\ y))$	: $(\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau$
$c'_{habituellement} = \lambda V.\lambda x.V(\mathbf{VP}_2 x\ \textit{habituellement}\ )$	: $(\tau \multimap \tau) \multimap (\tau \multimap \tau)$

Table 2: Lexique de $\mathcal{G}^G$

En neutralisant les nœuds sur lesquels aucune adjonction n'est faite par la constante I_α correspondante, on a alors par exemple la correspondance entre les termes du langage abstrait de $\Lambda(\Sigma_{DT})$ et les termes du langage objet de $\Lambda(\Sigma_{TAG})$ du tableau 3.

$\Lambda(\Sigma_{\text{DT}})$: $c_{aboie} I_{VP}(c_{chien} c_{tout})$ $c_{chasse} I_{VP}(c_{chien} c_{tout})(c_{chat} c_{un})$



$\Lambda(\Sigma_{\text{TAG}})$:

Table 3: Exemple de termes abstraits et des arbres dérivés correspondants dans G

Remarquons que le terme $c_{aboie} I_{VP}(c_{chien} c_{tout})$ peut s'écrire sous forme d'arbre foncteurs-arguments. Sur cet arbre, si l'on supprime les liens vers les arguments I_α qui indiquent simplement que certains nœuds n'ont fait l'objet d'aucune adjonction, et si *l'on indique le typage de l'argument par l'orientation de la flèche*, avec une flèche orientée vers le bas si l'argument est de type α_S et vers le haut si l'argument est de type α_A, on obtient bien l'arbre de dérivation traditionnel de la figure 1.

Avec une telle traduction, on a le théorème suivant (de Groote, 2002) : soit G une TAG. Le langage d'arbre engendré par G est isomorphe aux langages objet de l'ACG $\mathcal{G}^G$.

Alors que l'arbre de dérivation n'est généralement utilisé que dans le cadre sémantique, nous voyons ici que l'on peut lui donner un statut utile au calcul de l'arbre dérivé, c'est-à-dire au cadre syntaxique. Dans la section suivante, nous allons montrer comment, en utilisant

comme notion d'arbre de dérivation les termes du langage abstrait $\Lambda(\Sigma_{DT})$, peut se construire la représentation sémantique associée aux arbres dérivés. En particulier, nous reprenons le principe d'élévation de type utilisé pour les arbres adjoints, qui leur donne un type d'ordre supérieur.

4 La sémantique $\mathbf{L_U}$ comme une ACG

Le choix de la représentation sémantique pour notre expérimentation se rapproche de celui fait dans différents articles sur la représentation sémantique en TAG, en particulier (Kallmeyer, 2002; Gardent & Kallmeyer, 2003). Il s'agit de la sémantique sous-spécifiée $\mathbf{L_U}$ dite « à trou » (Bos, 1995; Blackburn & Bos, 2003). Nous renvoyons le lecteur à ces deux références pour les détails.

Le principe de $\mathbf{L_U}$ est de spécifier entre différentes formules les relations de sous-formule. Ainsi, la formule de $\mathbf{L_U}$ associée à l'expression *tout chien chasse un chat* :

$$\begin{array}{l}\lambda hl.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1) \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \mathbf{chien}(v_1) \\ \wedge \exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \wedge l'_2 : \mathbf{Ex}(v'_1, l'_3) \wedge l'_3 : \mathbf{And}(l'_1, h'_1) \wedge h'_1 \geq l \wedge h \geq l'_1 \wedge l'_1 : \mathbf{chat}(v'_1) \\ \wedge h \geq l \wedge l : \mathbf{chasse}(v_1, v'_1)))\end{array}$$

spécifie que la formule étiquetée par l ($l : \mathbf{chasse}(v_1, v'_1)$) soit sous-formule de celle étiquetée par l_2 (car $h_1 \geq l$, et h_1 sous-formule de $\mathbf{Imp}(l_1, h_1)$ étiquetée par l_3, elle-même sous-formule de $\mathbf{All}(v_1, l_3)$ d'étiquette l_2). De la même manière, elle doit être sous-formule de $\mathbf{Ex}(v'_1, l'_3)$ étiquetée par l'_2 (via $h'_1 \geq l$ et l'_3).

Cela donne deux formules de la logique de prédicats possibles qui vérifient cette spécification :

$$\begin{array}{c}\mathbf{All}(x, \mathbf{Imp}(\mathbf{man}(x), \mathbf{Ex}(y, \mathbf{And}(\mathbf{woman}(y), \mathbf{love}(x, y))))) \\ \mathbf{Ex}(y, \mathbf{And}(\mathbf{woman}(y), \mathbf{All}(x, \mathbf{Imp}(\mathbf{man}(x), \mathbf{love}(x, y)))))\end{array}$$

ou, exprimées dans la logique des prédicats usuelle :

$$\begin{array}{c}\forall x(\mathbf{man}(x) \Rightarrow \exists y(\mathbf{woman}(y) \wedge \mathbf{love}(x, y))) \\ \exists y(\mathbf{woman}(y) \wedge \forall x(\mathbf{man}(x) \Rightarrow \mathbf{love}(x, y)))\end{array}$$

Ici, nous ne nous intéressons pas à trouver les formules de la logiques des prédicats qui satisfont les formule de $\mathbf{L_U}$. Nous ne faisons que construire des formules de $\mathbf{L_U}$. Pour ce faire, nous utiliserons une signature d'ordre supérieure $\Sigma_{\mathbf{L_U}} = \langle A_{\mathbf{L_U}}, C_{\mathbf{L_U}}, \tau_{\mathbf{L_U}} \rangle$ définie de la manière suivante[2] :

- $A_{\mathbf{L_U}} = \{h, l, t, p, e\}$ sont les types représentant respectivement les trous, les étiquettes, les descriptions d'arbre, les prédicats et les entités (t et e sont choisis pour l'analogie avec la sémantique de Montague, en particulier pour l'élévation de type) ;
- $C_{\mathbf{L_U}}$ et $\tau_{\mathbf{L_U}}$ sont définis par les tableau 4 et trois quantificateurs existentiels $\exists_l : (l \to t) \multimap t$, $\exists_h : (h \to t) \multimap t$ et $\exists_e : (e \to t) \multimap t$, tous notés par abus de langage $\exists$, et dont l'interprétation intuitive est que $\exists P$ est une description d'arbre satisfiable s'il existe x respectivement étiquette, trou ou individu, $P\,x$ est satisfiable. Par abus de notation toujours, on notera $\exists x P$ pour $\exists(\lambda x.P)$ où x est une variable libre de P.

$\geq$	$: h \to l \to t$	pour spécifier les relations de dominance large *entre trous et étiquettes*
:	$: l \to p \multimap t$	permet d'étiqueter des prédicats
$\wedge$	$: t \multimap t \multimap t$	permet la conjonction de descriptions
All, **Ex**	$: e \to l \to t$	destiné à être interprété comme les quantifications universelle et existentielle
And, **Imp**	$: l \to h \to p$	destiné à être interprété comme la conjonction et l'implication
chien, **chat**, **aboie**	$: e \to p$	Prédicats habituels
chasse	$: e \to e \to p$	
habituellement	$: h \to p$	

Table 4: Définition de la signature pour le langage de sous-spécification

Pour utiliser $\Sigma_{\mathbf{L_U}}$, il reste à définir une ACG $\langle \Sigma_{DT}, \Sigma_{\mathbf{L_U}}, \mathcal{L} \rangle$, avec $\mathcal{L}$ un lexique qui lie les termes abstraits des arbres de dérivation aux termes des descriptions d'arbre. On aura bien entendu pour tout c, $\mathcal{L}(c) = c''$ dans le tableau 5, avec pour la transformation des types atomiques :

$\mathcal{L}(N_S) = (e \to h \to l \to t) \multimap (h \to l \to t)$ $\quad$ $\mathcal{L}(VP_A) = (h \to l \to t) \multimap (h \to l \to t)$

$\mathcal{L}(S_S) = h \to l \to t$ $\quad$ $\mathcal{L}(N_A) = (e \to h \to l \to t) \multimap (e \to h \to l \to t) \multimap (h \to l \to t)$

$$
\begin{aligned}
c''_{\textit{chien}} &= \lambda q.q(\lambda x h l.h \geq l \wedge l : \mathbf{chien}(x)) \\
c''_{\textit{chat}} &= \lambda q.q(\lambda x h l.h \geq l \wedge l : \mathbf{chat}(x)) \\
c''_{\textit{aboie}} &= \lambda a_{dv} \lambda s.s(\lambda x.a_{dv}(\lambda h l.h \geq l \wedge l : \mathbf{aboie}(x))) \\
c''_{\textit{chasse}} &= \lambda a_{dv} \lambda s \lambda o.s(\lambda x.a_{dv}(o(\lambda y h' l'.h' \geq l' \wedge l' : \mathbf{chasse}(x,y)))) \\
c''_{\textit{tout}} &= \lambda r p.\lambda h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1) \\
&\quad \wedge h_1 \geq l \wedge r\, v_1\, h\, l_1 \wedge p\, v_1\, h\, l) \\
c''_{\textit{un}} &= \lambda r p.\lambda h' l'.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h' \geq l'_2 \wedge l'_2 : \mathbf{Ex}(v'_1, l'_3) \wedge l'_3 : \mathbf{And}(l'_1, h'_1) \\
&\quad \wedge h'_1 \geq l' \wedge r\, v'_1\, h'\, l'_1 \wedge p\, v'_1\, h'\, l') \\
c''_{\textit{habituellement}} &= \lambda a_{dv}.\lambda r.\lambda h l.\exists h_1 l_1 (r\, h\, l \wedge h \geq l_1 \wedge l_1 : \mathbf{habituellement}(h_1) \\
&\quad \wedge h_1 \geq l \wedge a_{dv}(\lambda h' l'.h' \geq l') h\, l_1)
\end{aligned}
$$

Table 5: Quelques représentations sémantiques

Nous ne pouvons pas détailler ici les calculs, mais ce lexique rend compte de l'ambiguïté de portée des quantificateurs, en calculant par exemple

$$
\begin{aligned}
c''_{\textit{chien}} c''_{\textit{tout}} &= \lambda p h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1) \\
&\quad \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \mathbf{chien}(v_1) \wedge p\, v_1\, h\, l) \\
c''_{\textit{chat}} c''_{\textit{un}} &= \lambda p h' l'.\exists h'_1 l'_1 l'_2 l'_3 v'_1 (h' \geq l'_2 \wedge l'_2 : \mathbf{Ex}(v'_1, l'_3) \wedge l'_3 : \mathbf{And}(l'_1, h'_1) \\
&\quad \wedge h'_1 \geq l' \wedge h' \geq l'_1 \wedge l'_1 : \mathbf{chat}(v'_1) \wedge p\, v'_1\, h'\, l') \\
\mathcal{L}(c_{\textit{chasse}} I_{VP}(c_{\textit{chien}} c_{\textit{tout}})(c_{\textit{chat}} c_{\textit{un}})) &= c''_{\textit{chasse}} I_{VP}(c''_{\textit{chien}} c''_{\textit{tout}})(c''_{\textit{chat}} c''_{\textit{un}}) \\
&= \lambda h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1) \\
&\quad \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \mathbf{chien}(v_1) \wedge \exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \geq l'_2 \\
&\quad \wedge l'_2 : \mathbf{Ex}(v'_1, l'_3) \wedge l'_3 : \mathbf{And}(l'_1, h'_1) \wedge h'_1 \geq l \wedge h \geq l'_1 \\
&\quad \wedge l'_1 : \mathbf{chat}(v'_1) \wedge h \geq l \wedge l : \mathbf{chasse}(v_1, v'_1)))
\end{aligned}
$$

où le prédicat $\mathbf{chasse}(v_1, v'_1)$, étiqueté par l, est dominé à la fois par l_2 et l'_2, ces deux quantifi-

[2]Nous remarquons que cette fois nous utilisons des termes non linéaires : si $\lambda x.P : \alpha \to \beta$ alors x est une variable libre de P qui y apparaît *une ou plusieurs fois*.

cateurs n'étant pas ordonnés l'un par rapport à l'autre.

On peut de la même manière tenir compte de l'ambiguïté introduite par les adverbes. Ainsi, on remarquera que le terme associé à $c''_{habituellement}$ agit comme un adverbe opaque : si un autre adverbe lui est adjoint (le paramètre a_{dv}), son champ dominera l_1 : **habituellement**(h_1). Cela rend compte de ce que si un autre adverbe est ajouté, *habituellement* restera toujours dans son champs. Pour lever cette opacité, il suffirait que l'adverbe a_{dv} ne domine que le label du verbe, soit l.

5 Conclusion

Revenant sur les diverses approches pour construire une représentation sémantique en TAG, nous avons proposé de donner un rôle central à l'arbre de dérivation. Ce dernier est pris au sens de terme du langage abstrait de l'ACG associée à la grammaire TAG, plus large que le sens traditionnel. Dès lors, la représentation syntaxique (arbre dérivé) comme la représentation sémantique s'obtiennent au bout d'un processus de calcul décrit par l'arbre de dérivation.

Si la partie syntaxique a été montrée comme complète vis-à-vis du langage d'arbre engendré (de Groote, 2002), nous n'avons donné ici que quelques exemples de la pertinence de notre approche pour le calcul sémantique en TAG. Il s'agit de poursuivre cet effort et de vérifier comment modéliser les différents phénomènes identifiés comme problématiques dans (Kallmeyer, 2002; Frank & van Genabith, 2001; Gardent & Kallmeyer, 2003), en particulier pour les verbes à contrôle. Néanmoins nous avons montré que les arbres de dérivation suffisent pour rendre compte du rôle sémantique des quantificateurs, même en les représentant par des arbres adjoints.

Enfin, nous n'avons pas tenu compte des contrôles induits sur la dérivation par la présence des traits. Pour ce faire, il sera sans doute nécessaire d'élargir encore le langage de λ-termes en faisant intervenir les types additifs de la logique linéaire, de la même manière qu'ils sont utilisés pour modéliser les traits dans les grammaires de types logique (Morrill, 1994).

Références

ABEILLÉ A. (1993), *Les nouvelles syntaxes*, Paris, Armand Colin Éditeur.

BLACKBURN P., BOS J. (2003), Computational semantics for natural language, `http://www.iccs.informatics.ed.ac.uk/~jbos/comsem/book1.html`, Course Notes for NASSLLI 2003.

BOS J. (1995), Predicate logic unplugged, *Proceedings of the Tenth Amsterdam Colloquium*.

CANDITO M.-H., KAHANE S. (1998), Can the tag derivation tree represent a semantic graph? an answer in the light of meaning-text theory, *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Framework (TAG+4)*, volume 98-12 of *IRCS Technical Report Series*.

COPESTAKE A., FLICKINGER D., SAG I., POLLARD C. (1999), Minimal recursion semantics: An introduction, `http://www-csli.stanford.edu/~aac/papers/newmrs.pdf`.

M. DALRYMPLE, Ed. (1999), *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*, MIT Press.

DANOS V., COSMO R. D. (1992), The linear logic primer, `http://www.pps.jussieu.fr/~dicosmo/CourseNotes/LinLog/`, An introductory course on Linear Logic.

DE GROOTE P. (2000), Linear higher-order matching is np-complete, dans L. BACHMAIR, Ed., *Rewriting Techniques and Applications, RTA'00*, volume 1833 of *LNCS*, p. 127–140, Springer.

DE GROOTE P. (2001), Towards abstract categorial grammars, *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, p. 148–155.

DE GROOTE P. (2002), Tree-adjoining grammars as abstract categorial grammars, *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, p. 145–150, Università di Venezia.

DOWEK G. (2001), Higher-order unification and matching, dans A. ROBINSON & A. VORONKOV, Eds., *Handbook of Automated Reasoning*, volume 2, chapter 16, Elsevier Science.

FRANK A., VAN GENABITH J. (2001), Glue tag: Linear logic based semantics construction for ltag - and what it teaches us about the relation between lfg and ltag, dans M. BUTT & T. H. KING, Eds., *Proceedings of the LFG '01 Conference*, Online Proceedings, CSLI Publications, `http://cslipublications.stanford.edu/LFG/6/lfg01.html`.

GARDENT C., KALLMEYER L. (2003), Semantic construction in feature-based tag, *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*.

GIRARD J.-Y. (1987), Linear logic, *Theoretical Computer Science*, Vol. 50, 1–102.

J. R. HINDLEY & J. P. SELDIN, Eds. (1980), *To H. B. Curry: Essays on combinatory logic, Lambda Calculus and Formalism*, Academic Press.

HOWARD W. A. (1980), *The Formulæ-as-Types Notion of Construction*, p. 479–490, dans (Hindley & Seldin, 1980).

JOSHI A. K., KALLMEYER L., ROMERO M. (2003), Flexible composition in ltag: Quantifier scope and inverse linking, dans H. BUNT, I. VAN DER SLUIS & R. MORANTE, Eds., *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*.

JOSHI A. K., LEVY L. S., TAKAHASHI M. (1975), Tree adjunct grammars, *Journal of Computer and System Sciences*, Vol. 10(1), 136–163.

KALLMEYER L. (2002), Using an enriched tag derivation structure as basis for semantics, *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*.

LAMBEK J. (1958), The mathematics of sentence structure, *American Mathematical Monthly*, Vol. 65(3), 154–170.

LOADER R. (2003), Higher order β matching is undecidable, *Logic Journal of the IGPL*, Vol. 11(1), 51–68.

MONTAGUE R. (1974), The proper treatment of quantification in ordinary english, dans P. PORTNER & B. H. PARTEE, Eds., *Formal Semantics: The Essential Readings*, chapter 1, Blackwell Publishers, 2002 edition.

MOORTGAT M. (1996), Categorial type logics, dans J. VAN BENTHEM & A. TER MEULEN, Eds., *Handbook of Logic and Language*, p. 93–177, Amsterdam, Elsevier Science Publishers.

MORRILL G. V. (1994), *Type Logical Grammar Categorial Logic of Signs*, Kluwer Academin Publishers.

SCHABES Y., SHIEBER S. M. (1994), An alternative conception of tree-adjoining derivation, *Computational Linguistics*, Vol. 20(1), 91–124.