SemTAG : une plate-forme pour le calcul sémantique à partir de Grammaires d'Arbres Adjoints

Yannick Parmentier

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

SemTAG : une plate-forme pour le calcul sémantique à partir de Grammaires d'Arbres Adjoints

THÈSE

présentée et soutenue publiquement le 6 avril 2007

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Yannick PARMENTIER

Composition du jury

Présidente :	Dr Noëlle Carbonell	Professeur, Université Henri Poincaré, LORIA Nancy
Rapporteurs :	Dr Philippe Blache	Directeur de Recherches CNRS, LPL Aix-en-Provence
	Dr Laura Kallmeyer	Projektleiterin, SFB 441 Universität Tübingen
Examinateurs :	Dr Eric De La Clergerie	Chargé de Recherches, INRIA Rocquencourt
	Dr Denys Duchier	Professeur, Université d'Orléans, LIFO Orléans
	Dr Claire Gardent	Directrice de Recherches CNRS, LORIA Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Remerciements

Vous vous apprêtez à lire un document retraçant plus de trois années de travaux de thèse. Une thèse, c'est à la fois une aventure scientifique et humaine. Dans les 200 pages qui suivent, l'aspect scientifique de ce travail est présenté en détail. Si vous le permettez, j'aimerais prendre quelques lignes pour revenir sur son aspect Humain.

Le travail présenté ici n'aurait pu être réalisé sans l'aide d'autres personnes qui se sont avérées être bien plus que de simples collègues. J'aimerais remercier tout particulièrement Claire Gardent pour ses qualités scientifiques et humaines, pour sa capacité à créer une réflexion, à faire apparaître les zones d'ombres, à proposer des idées. J'aimerais remercier Denys Duchier, Benoît Crabbé et Joseph Le Roux avec qui j'ai pris grand plaisir à travailler. Merci pour tout ce que vous m'avez appris. Un grand merci également à l'ensemble du bureau B224, Sébastien Hinderer, Eric Kow, Jacqueline Lai et Dmitry Sustretov, pour avoir fait de mon quotidien une histoire digne d'un vidéo-blog, et surtout pour votre amitié. Merci aux doctorants passés (et présents) de l'équipe Langue Et Dialogue, pour votre écoute, vos conseils et bien plus, merci beaucoup à Hélène Manuélian, Joseph Roumier, Djamé Seddah, Frédéric Landragin, Huyen N'Guyen, Alexandre Denis, Paul Bédaride et Phuong Le Hong. Merci à l'ensemble des membres de l'équipe, tout particulièrement à Bertrand Gaiffe, que j'ai eu la chance d'avoir comme co-encadrant de DEA, pour ses conseils et son attention, et à Laurent Romary pour son professionnalisme. Merci à Patrick Blackburn et Carlos Areces pour leur enthousiasme communicatif dans leur travail, merci beaucoup à Laurence Kbida, Matthieu Quignard, Guillaume Pitel, Jesse Tseng, Anna Kupść, Azim Roussanaly, Isabelle Kramer, Karën Fort pour les échanges enrichissants (quel que soit le domaine, du TAL à l'enseignement en passant par le rugby). Merci beaucoup à Isabelle Blanchard et Nadine Beurné, pour leur aide inestimable dans toutes mes démarches administratives.

Je tiens également à remercier chaleureusement les membres du jury, Philippe Blache, Noëlle Carbonell, Eric De La Clergerie, Denys Duchier, Claire Gardent et Laura Kallmeyer, pour m'avoir fait l'honneur d'accepter de participer à l'évaluation de ce travail, merci beaucoup pour votre attention et vos commentaires.

Je voudrais également remercier les personnes extérieures au monde de la Recherche et sans qui ce travail serait encore plus difficile. Merci à Frédéric Ham, Julien, Fanny et Rémi Massignan, Alexandre Ney, Christelle Zerr, Alexandre Campo, Stéphanie Grzelak, Karim Saddem, Aline et Francis Bonacina, Estelle et Jean-François Douce, Mélanie et Cyril Gérard pour votre amitié si précieuse et pour votre soutien de tous les instants. Merci. Merci également à la famille Milko et à Sveta Maire, merci pour vos encouragements.

Merci aussi aux personnes que j'ai rencontrées durant cette thèse et dont le contact m'a beaucoup appris et aidé. Merci à Paula Chesley, Nadiya Yampolska, Ustun Yildiz, Nicolas Padoy, Jérôme Simonin, Colin Riba, François Klein, Dana Tec, Sarah Maarek, Joseph Razik, Emmanuel Didiot, Benoît Sagot, Lucas Champollion, Delphine Bernhard, Davy Weissenbacher, Yael Cohen-Sygal, Mathieu Morey, Wolfgang Maier, Timm Lichte, et

les French Connexions d'ESSLLI 2003, 2004 et 2005. Merci également à Owen Rambow, Tatjana Scheffler, SinWon Yoon, Guy Perrier, Bruno Guillaume, Sylvain Pogodalla, Sylvain Salvati, et l'ensemble des membres de l'équipe Calligramme pour les interactions bénéfiques.

Merci aux membres de l'UFR STMIA et de l'ESSTIN pour les échanges enrichissants, en particulier Frédéric Suter, Monique Grandbastien, Fiametta Namer, Yannick Toussaint, Malika Smaïl, Vincent Chevrier et Jean-Marie David.

Un merci tout particulier à Emilie Balland et Jacques Henry, merci pour votre amitié, merci d'avoir participé à faire de cette thèse un moment inoubliable.

Pour finir, j'aimerais remercier ma famille et comme le dit si justement (Koller, 2004), si cette thèse vous donne quelque raison d'être fier de moi, alors cela en valait la peine.

Pardon à tous ceux que j'oublie.

Nancy, le 4 décembre 2006.

PS : merci à Emmanuel Beffara, que je ne connais pas, mais qui a développé la police cursive qui me permet de mettre en relief ces remerciements.

A mes parents Anne-Marie et René,
A mon frère Damien,

Table des matières

Partie III Construction sémantique pour Grammaires d'Arbres Adjoints : implantation et évaluation

Table des figures

Introduction

Comme son titre l'indique, le thème de cette thèse est la *construction sémantique*. Mais qu'entendons nous exactement par construction sémantique ? Une réponse à cette question est le calcul d'une représentation du *sens* d'un énoncé.

Etre capable de construire *automatiquement* le sens d'une phrase représente un intérêt pour de nombreuses applications parmi lesquelles le dialogue Homme / machine, les systèmes de Questions / Réponses ou encore la traduction automatique. De manière plus générale, disposer d'une représentation du sens est indispensable pour pouvoir effectuer la tâche de déduction, appelée aussi inférence. Prenons l'exemple de la traduction automatique. Comment nous assurer que deux énoncés dans deux langues différentes ont le même sens ? Une possibilité est de considérer que deux énoncés ont le même sens s'ils peuvent être déduits l'un de l'autre. Cette déduction correspond à la notion d'*implication textuelle* (*Textual Entailment*) au centre de nombreuses recherches menées actuellement (Bar-Haim et al., 2006).

Avant de pouvoir réaliser de telles déductions, plusieurs questions se posent à nous : quelle forme doit avoir une représentation du sens ? Comment est-elle définie ? Depuis les travaux de (Montague, 1974b), nous avons de bonnes raisons de penser que le sens d'un énoncé peut être représenté par une formule logique, qui est définie à partir de la structure syntaxique de la phrase (ordre et catégorie des mots).

Pour construire une représentation du sens d'un énoncé, nous devons donc tout d'abord être capable de construire une représentation de sa structure syntaxique. Cela passe par l'utilisation d'une grammaire. Dans notre cas, nous nous sommes concentrés sur un type de grammaires particulier : les *Grammaires d'Arbres Adjoints* (*Tree Adjoining Grammars* – TAG). Ces grammaires présentent de nombreux intérêts formels et informatiques (Joshi et Schabes, 1997). Parmi les plus importants, nous pouvons citer leur capacité à représenter les dépendances à longue distance au sein d'une même structure et leur complexité d'analyse polynomiale ($\mathcal{O}(n^6)$). Enfin, à la différence d'autres formalismes grammaticaux tels que les grammaires lexicales fonctionnelles ou les grammaires syntagmatiques guidées par les têtes, il n'existe pas de consensus sur la définition d'une interface entre syntaxe et sémantique pour les Grammaires d'Arbres Adjoints.

Dans notre travail, nous nous sommes fixés pour objectif de définir une architecture logicielle permettant la construction sémantique à partir de grammaires TAG. Plus précisément,

nous nous sommes intéressés à deux tâches principales :

- la définition d'un formalisme de haut niveau permettant de décrire des grammaires TAG incluant une interface syntaxe / sémantique, et l'implantation d'un compilateur pour ce formalisme,
- le développement d'un prototype de calculateur de représentations sémantiques pour grammaires TAG annotées sémantiquement.

Le résultat de ce travail correspond à la plate-forme SemTAG[1] [2].

Enfin, nous avons procédé à une évaluation de cette plate-forme par rapport à un corpus de test couvrant différents phénomènes linguistiques.

Ce document est divisé en trois parties : un état de l'art, une proposition, et une évaluation.

La première partie introduit le domaine de la sémantique en linguistique informatique (chapitre 1) et présente l'état de l'art de la construction sémantique au moyen de Grammaires d'Arbres Adjoints (chapitre 2). Dans ce chapitre nous présentons en détail les différents procédés de calcul sémantique qui ont été proposés pour les grammaires TAG. A partir des différentes approches existantes mises en relations les unes aux autres, nous présentons l'interface syntaxe / sémantique utilisée dans notre plate-forme.

La seconde partie introduit notre proposition pour la production de grammaires TAG à portée sémantique. Dans un premier temps (chapitre 3), nous revenons sur les approches métagrammaticales existantes pour la génération de grammaires TAG de taille réelle, puis nous présentons en détail notre proposition de formalisme métagrammatical, *eXtensible MetaGrammar* (chapitre 4). Cette introduction présente chacune des fonctionnalités offertes par notre formalisme. Ensuite (chapitre 5), nous présentons l'implantation d'un compilateur pour ce formalisme. Enfin (chapitre 6), nous montrons comment, en pratique, définir une grammaire TAG de taille réaliste pour le français, et disposant à la fois d'une dimension syntaxique et sémantique.

En troisième partie, nous présentons deux procédés de calcul sémantique pour TAG. Ces procédés utilisent une grammaire produite en utilisant le formalisme XMG et utilisent deux types d'analyseurs syntaxiques pour grammaires TAG (chapitre 7). L'un de ces procédés (le système SemConst) a été évalué sur un corpus de référence afin d'évaluer la pertinence de notre proposition (chapitre 8).

Finalement, nous concluons par une récapitulation des apports de ce travail, les questions soulevées, et les perspectives de poursuite des recherches entamées.

[1] Voir `http://trac.loria.fr/~semtag`.

[2] Pour être plus précis, notons que cette plate-forme SemTAG comporte différents outils, permettant non seulement la production semi-automatique de grammaires (compilateur XMG), la construction sémantique (module SemConst), mais également la génération (réalisateur GenI), cette dernière tâche correspond aux travaux en cours dans le cadre de la thèse d'Eric Kow.

Première partie

Construction sémantique pour Grammaires d'Arbres Adjoints : état de l'art

La vie atteint sa plénitude à l'instant où les choses semblent avoir perdu leur signification.

Hermann Hesse, Klein et Wagner

Chapitre 1

Le calcul sémantique en linguistique informatique

Sommaire

Ce chapitre donne un aperçu historique de la sémantique formelle et de la sémantique en linguistique informatique. Plus précisément, nous introduisons dans un premier temps les travaux de Frege, considérés comme l'origine d'une sémantique formelle pour la langue naturelle. Ensuite, nous étudions l'apport central de Montague dans ce domaine. En effet, Montague est le premier à avoir proposé un système de calcul sémantique automatisable pour la langue naturelle. Enfin, nous présentons deux aspects centraux en sémantique pour le *Traitement Automatique des Langues* (TAL), les sémantiques sous-spécifiées d'une part, et le calcul sémantique par unification d'autre part.

1.1 Une sémantique formelle pour la langue naturelle

1.1.1 Les travaux de Frege

La première question que nous nous posons est « Qu'entend-on par sens d'un énoncé ? » Une première réponse à cette question, proposée par le mathématicien allemand *Gottlob Frege* à la fin du XIX^{e} siècle, est : l'identification des conditions dans lesquelles cet énoncé est vrai.

En outre, à partir du constat que la langue naturelle permet de construire un nombre potentiellement infini d'énoncés ayant un sens à partir d'un nombre fini de mots, Frege émit l'hypothèse que le sens d'un énoncé est construit à partir du sens de ses constituants. Cette hypothèse est plus connue sous le nom de *principe de compositionnalité du sens* :

> *Le sens d'une expression composée est une fonction du sens de ses parties.*

A partir de ces deux hypothèses (sens donné par une valeur de vérité et compositionnalité du sens), Frege a proposé une définition formelle d'une sémantique pour la langue naturelle au moyen d'un *langage logique*. Ainsi, en évaluant une expression de ce langage, on obtient une valeur de vérité dénotant le sens de la phrase.

Dans ce contexte, le principe de compositionnalité fournit une méthodologie pour la construction sémantique. Cette méthodologie est complétée en faisant l'hypothèse que cette construction sémantique se base sur la structure syntaxique de l'énoncé[3] :

> *Le sens d'une expression composée est une fonction du sens de ses parties et de la règle syntaxique par laquelle elles sont combinées.*

Dans ce qui suit, nous présentons un procédé de construction sémantique inspiré des travaux de Frege, et utilisant la logique propositionnelle pour représenter le sens d'un énoncé. Nous verrons également comment déduire de cette représentation (sous forme de formule logique propositionnelle) une valeur de vérité dénotant le sens de la phrase (*i.e.*, comment *interpréter* la formule logique).

1.1.2 Interprétation de la langue naturelle au moyen de la logique propositionnelle

Dans une première approximation, nous utilisons un langage de la logique propositionnelle pour représenter le sens d'une phrase.

1.1.2.1 Logique propositionnelle : définitions

Définition 1 (Logique propositionnelle)**.** *Un langage de la logique propositionnelle est défini inductivement comme suit :*

1. *Toute lettre propositionnelle notée p,q,r,... est une formule bien formée.*

[3] Reformulation du principe de compositionnalité telle que donnée dans (Gamut, 1991).

2. *Si α est une formule bien formée, alors $\neg\alpha$ est une formule bien formée.*
3. *Si α et β sont des formules bien formées, alors $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$ sont des formules bien formées.*
4. *Rien d'autre n'est une formule bien formée.*

Définition 2 (Interprétation d'une formule de logique propositionnelle)**.** *L'interprétation d'une formule propositionnelle correspond à une fonction V calculant la valeur de vérité de la formule en fonction des valeurs de vérité des propositions, en suivant les règles ci-dessous :*

$$\begin{array}{lll} V(\neg\alpha) = 1 & ssi & V(\alpha) = 0 \\ V(\alpha \wedge \beta) = 1 & ssi & V(\alpha) = 1 \text{ et } V(\beta) = 1 \\ V(\alpha \vee \beta) = 1 & ssi & V(\alpha) = 1 \text{ ou } V(\beta) = 1 \\ V(\alpha \rightarrow \beta) = 1 & ssi & V(\alpha) = 0 \text{ ou } V(\beta) = 1 \end{array}$$

où α et β représentent des formules bien formées pour la logique propositionnelle.

1.1.2.2 Interface syntaxe / sémantique

Pour réaliser la composition sémantique, nous nous basons sur le lien entre les constituants syntaxiques de la phrase. Ce lien est exprimé au moyen de règles hors-contextes. Ces règles sont de la forme :

$$\alpha \rightarrow \beta \star$$

où α est un symbole non-terminal de notre vocabulaire, β appartient à l'union des ensembles des symboles terminaux et non-terminaux, et $\star$ l'étoile de Kleene.

La grammaire hors-contexte que nous utilisons, dans notre approximation, pour représenter la structure syntaxique de la phrase comprend les symboles non-terminaux P (pour phrase) et $Conj$ (pour conjonction). Les symboles terminaux correspondent à des phrases atomiques (à valeur propositionnelle). Notons que cette représentation syntaxique est très limitée, puisqu'elle n'exprime pas de manière fine les relations entre constituants de la phrase.

Un exemple de grammaire hors-contexte très minimale pour le Français est donné ci-dessous :

$$\begin{array}{rl} P & \rightarrow P\,Conj\,P \\ P & \rightarrow Jean\_aime\_Marie \\ P & \rightarrow Marie\_dort \\ Conj & \rightarrow et \mid ou \end{array}$$

A partir de ces règles, nous associons à chaque symbole terminal un élément du langage logique. Ainsi, nous associons au symbole $Jean\_aime\_Marie$ la lettre propositionnelle p et

au symbole $Marie\_dort$ la lettre propositionnelle q. De même les conjonctions *et* et *ou* se voient associées respectivement aux connecteurs logiques $\wedge$ et $\vee$.

Nous pouvons ainsi traduire de manière immédiate la représentation syntaxique en formule logique pour la logique propositionnelle, tel qu'illustré sur la figure 1.1.

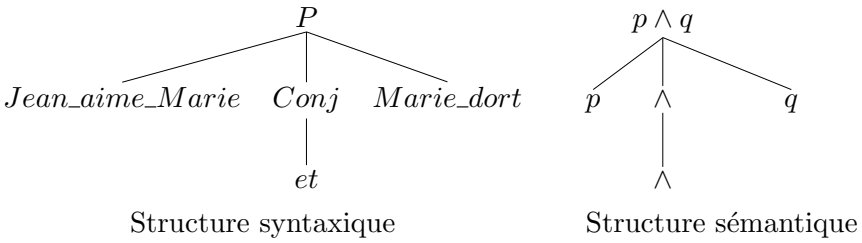


FIG. 1.1 – Calcul sémantique au moyen de la logique propositionnelle.

Ainsi, la phrase « Jean aime Marie et Marie dort » sera associée à la représentation sémantique $p \wedge q$.

On remarque que ce premier procédé de construction sémantique, au moyen d'une logique propositionnelle, ne permet pas de réaliser une composition tenant compte de la structure syntaxique interne à la phrase. Nous allons donc étendre ce procédé en utilisant, en seconde approximation, la logique des prédicats.

1.1.3 Interprétation de la langue naturelle au moyen de la logique des prédicats

1.1.3.1 Logique des prédicats : définitions

Définition 3 (Logique des prédicats)**.** *Un langage de la logique des prédicats est constitué de formules manipulant des termes. Un terme peut être :*

- *une constante individuelle (notée généralement $a, b, c, \ldots$),*
- *une variable individuelle (notée généralement $x, y, z, \ldots$),*
- *si P^n est un symbole de prédicat d'arité n et pour tout i tel que $1 \leq i \leq n$, τ_i est une constante ou variable individuelle, alors $P^n(\tau_1, \ldots, \tau_n)$ est un terme.*

Dans ce contexte, une formule de la logique des prédicats est définie inductivement comme suit :

1. *Si ϕ est une formule bien formée, alors $\neg\phi$ est une formule bien formée.*
2. *Si ϕ et ψ sont des formules bien formées, alors $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ sont des formules bien formées.*
3. *Si ϕ est une formule bien formée, et que x est une variable, alors $\exists x\phi$ et $\forall x\phi$ sont des formules bien formées (ϕ est appelée la matrice de la formule, x est dite variable liée).*
4. *Rien d'autre n'est une formule bien formée.*

A présent, pour calculer le sens de la phrase, nous ne pouvons plus utiliser une fonction d'évaluation comme dans le cas de la logique propositionnelle. En effet, l'évaluation de la matrice d'une expression quantifiée est impossible (comment évaluer $dort^1(x)$ par exemple ?).

La notion de vérité que nous utilisons a été définie dans le cadre de la *théorie des modèles* proposée par (Tarski, 1939). Dans cette théorie, les phrases d'un langage sont vraies par rapport à un modèle.

Dans le cas de la logique des prédicats, un modèle M est un couple (D, I) où D est un domaine d'interprétation des entités que nous manipulons, et I est une fonction d'interprétation qui assigne à chaque symbole du vocabulaire (prédicats ou constantes) un élément du domaine.

Dans ce contexte, la valeur de vérité d'une phrase est donnée par une notion de *satisfaction*. Une satisfaction est une relation ternaire entre une formule logique ϕ, un modèle $M = (D, I)$ et une fonction g d'assignation de valeurs dans M aux variables de ϕ. Avant de définir cette fonction d'assignation, revenons sur les variables que nous manipulons dans nos formules. Les variables utilisées sont de deux types, celles apparaissant dans des formules quantifiées (existentiellement ou universellement) sont dites liées, les autres variables sont dites libres. Une fonction d'assignation nous permet d'associer une valeur à une variable libre, et ainsi de lui donner une dénotation. De manière intuitive, nous pouvons voir les formules comme des descriptions, les modèles comme des situations et les assignations comme des contextes. Ainsi, une description est vraie dans une situation donnée s'il n'y a pas de contexte contradictoire à cette description.

Quel est le lien entre fonction d'interprétation I et fonction d'assignation g ? Comme nous l'avons vu, g permet d'associer des valeurs aux variables. Ainsi, si nous considérons un terme τ (constante ou variable individuelle), l'interprétation de τ est donnée par :

$$\|\tau\| = \begin{cases} I(\tau) & \text{si } \tau \text{ est une constante} \\ g(\tau) & \text{si } \tau \text{ est une variable} \end{cases}$$

De plus, si deux fonctions d'assignation g et g' sont telles que, pour toute variable y distincte de x, $g(y) = g'(y)$ et $g(x) = g'(x)$, alors g' est appelée un *x-variant* de g.

Définition 4 (Satisfaction)**.** *A présent, nous pouvons définir par induction la satisfaction d'une formule ϕ par rapport à un modèle $M = (D, I)$ et une fonction d'assignation g donnés, ce que nous notons $M, g \models \phi$.*

$$\begin{array}{lll}
M, g \models P^n(\tau_1, \ldots, \tau_n) & ssi & (\|\tau_1\|, \ldots, \|\tau_n\|) \in I(P^n) \\
M, g \models \neg\phi & ssi & M, g \not\models \phi \\
M, g \models \phi \wedge \psi & ssi & M, g \models \phi \text{ et } M, g \models \psi \\
M, g \models \phi \vee \psi & ssi & M, g \models \phi \text{ ou } M, g \models \psi \\
M, g \models \phi \rightarrow \psi & ssi & M, g \not\models \phi \text{ ou } M, g \models \psi
\end{array}$$

$M, g \models \exists x\phi$ *ssi* $M, g' \models \phi$ *pour un x-variant* g' *de* g

$M, g \models \forall x\phi$ *ssi* $M, g' \models \phi$ *pour tout x-variant* g' *de* g

Les deux dernières propriétés traduisent respectivement le fait que (1) une formule quantifiée existentiellement est satisfaite dans un modèle par rapport à une assignation g si et seulement si il existe une assignation g' x-variant de g qui satisfait cette formule dans un modèle, et (2) une formule quantifiée universellement est satisfaite dans un modèle par rapport à une assignation g si et seulement si toute assignation g' x-variant de g satisfait la formule dans le modèle.

A partir de cette définition de satisfaction d'une formule logique prédicative bien formée, nous pouvons définir la valeur de vérité d'une *phrase* (par phrase, nous entendons une composition de formules bien formées) :

Définition 5 (Valeur de vérité d'une phrase dans un modèle)**.** *Une phrase ϕ est vraie dans un modèle M si et seulement si, pour toute assignation g de valeurs aux variables libres, dans M, $M, g \models \phi$. Si ϕ est vraie dans M, nous écrivons $M \models \phi$.*

1.1.3.2 Interface syntaxe / sémantique

De manière intuitive, nous souhaitons réaliser la construction sémantique, comme précédemment, c'est-à-dire en nous basant sur la structure syntaxique de la phrase exprimée au moyen de règles hors-contextes. Nous souhaitons en outre, que ces règles décrivent les constituants internes à la phrase.

Prenons, l'exemple suivant :

$$
\begin{aligned}
P &\rightarrow GN\ GV_1 \\
P &\rightarrow GN\ GV_2\ GN \\
GN &\rightarrow NP \\
GN &\rightarrow un\_homme \\
GN &\rightarrow une\_femme \\
NP &\rightarrow Jean \\
NP &\rightarrow Marie \\
GV_1 &\rightarrow dort \\
GV_2 &\rightarrow aime
\end{aligned}
$$

Si nous analysons la phrase « Un homme aime Marie », nous obtenons la structure syntaxique donnée en figure 1.2.

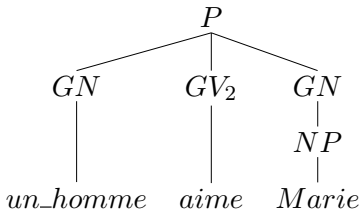


FIG. 1.2 – Analyse syntaxique de la phrase « Un homme aime Marie ».

Nous souhaiterions associer aux symboles terminaux de la grammaire hors-contexte décrivant les liens entre constituants syntaxiques, des symboles de notre langage logique. Nous définissons ainsi un ensemble de règles similaires aux règles syntaxiques, en suffixant les symboles non-terminaux par $'$ et en utilisant les symboles de notre langage logique comme symboles terminaux. Les règles introduites précédemment sont traduites comme indiqué ci-dessous :

$$
\begin{aligned}
P' &\rightarrow GN'\, GV_1' \\
P' &\rightarrow GN'\, GV_2'\, GN' \\
GN' &\rightarrow NP' \\
GN' &\rightarrow \exists\, z homme(z) \\
GN' &\rightarrow \exists\, z femme(z) \\
NP' &\rightarrow Jean_j \\
NP' &\rightarrow Marie_m \\
GV_1' &\rightarrow dormir(x) \\
GV_2' &\rightarrow aimer(x, y)
\end{aligned}
$$

où $Jean_j$ et $Marie_m$ sont des constantes, x, y des variables, et $homme$, $femme$, $dormir$ et $aimer$ représentent des prédicats.

A présent, nous souhaiterions construire une représentation sémantique de manière identique à l'analyse syntaxique, comme illustré figure 1.3.

Cependant, il nous manque un moyen de spécifier que l'argument x du prédicat $aimer$ est le même que la variable quantifiée existentiellement z, de même, nous ne pouvons pas spécifier que la variable y et la constante $Marie_m$ doivent être interprétées comme dénotant le même individu. Dans ce contexte, comment produire la représentation sémantique attendue $\exists\, x\, homme(x) \wedge\ aimer(x, Marie_m)$?

En utilisant la logique des prédicats, nous augmentons l'expressivité de notre langage de représentation sémantique au moyen de foncteurs d'arité n et de quantificateurs (universel et existentiel). Cependant, il nous manque un mécanisme permettant de combiner deux

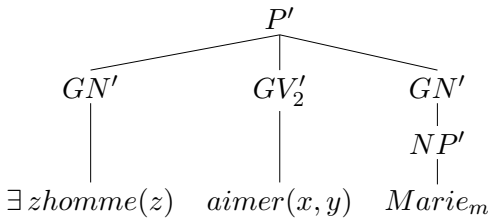


FIG. 1.3 – Analyse sémantique de la phrase « Un homme aime Marie ».

représentations sémantiques en substituant correctement les variables.

Dans la section qui suit, nous présentons les travaux de Richard Montague, qui le premier, a proposé un mécanisme de composition sémantique pour la logique des prédicats. Ce mécanisme permet en l'occurrence de gérer les instanciations de variables, et d'ainsi utiliser la structure syntaxique interne de la phrase comme guide pour la construction sémantique.

1.1.4 L'héritage de Montague : un calcul sémantique fondé sur le λ-calcul typé

(Montague, 1974a) propose de calculer le sens d'un énoncé en fonction du sens de ses constituants et de sa structure syntaxique (ordre et catégorie des constituants). Plus précisément, il propose un calcul sémantique compositionnel pour l'anglais, dans lequel :

1. à chaque mot du lexique est associée une représentation sémantique,
2. les règles syntaxiques de la grammaire sont couplées avec des règles sémantiques indiquant comment calculer le sens de la phrase en fonction du sens de ses constituants.

Concernant le point 1, Montague définit les représentations sémantiques du lexique au moyen d'un langage correspondant au λ-calcul typé. En d'autres termes, les mots du lexique se voient attribués un λ-terme typé.
Concernant le point 2, les règles sémantiques définissent les applications fonctionnelles utilisées par le λ-calcul pour construire une représentation sémantique de la phrase à partir des λ-termes des mots la composant.

Dans un premier temps, nous définissons le langage utilisé pour définir les λ-termes des représentations sémantiques, puis nous introduisons le λ-calcul et illustrons la construction sémantique de Montague au moyen d'exemples.

1.1.4.1 Définition de représentations sémantiques à base de λ-termes typés

La syntaxe du langage L utilisé par Montague pour définir les représentations sémantiques est fondée sur la *théorie des types* (Church, 1940). L manipule des constantes et des variables (éventuellement quantifiées). A la différence des langages du premier ordre, ces variables ne réfèrent pas uniquement à des individus, mais peuvent référer à des prédicats

(comme dans les langages du second ordre) ou à tout objet défini par la théorie des types, L est donc appelé langage d'ordre supérieur.

Définition 6 (Types sémantiques)**.** *Les types utilisés par Montague sont définis inductivement comme suit :*

1. *e est un type,*
2. *t est un type,*
3. *si a et b sont des types, alors $\langle a, b\rangle$ sont des types,*
4. *rien d'autre n'est un type.*

Dans le langage L, les variables et constantes individuelles sont interprétées par des entités du domaine D du modèle utilisé. C'est pourquoi ces objets sont de type $\langle e\rangle$. Les phrases sont interprétées par une valeur de vérité (en anglais *truth-value*), elles sont donc de type $\langle t\rangle$. Dans ce contexte, un prédicat unaire est interprété comme un sous-ensemble de l'ensemble des éléments du domaine D (plus précisément l'ensemble des éléments vérifiant une condition donnée). Une autre représentation de cette interprétation est d'utiliser la fonction caractéristique du sous-ensemble en question.

Définition 7 (Fonction caractéristique)**.** *La fonction caractéristique f d'un ensemble E est définie comme suit :*

$$f(x) = \begin{cases} 1 & si\ x \in E \\ 0 & si\ x \notin E \end{cases}$$

L'utilisation de cette fonction caractéristique nous montre que l'on peut interpréter un prédicat unaire comme une fonction qui, à une variable ou constante individuelle, associe une valeur de vérité. Le type d'un prédicat unaire est donc $\langle e, t\rangle$.

De manière similaire, un prédicat binaire peut être interprété comme une fonction qui à un élément de $D \times D$ associe une valeur de vérité. Son type est donc $\langle e, \langle e, t\rangle\rangle$.

Définition 8 (Langage de représentation sémantique)**.** *A partir de ce système de types, la syntaxe du langage L est définie inductivement comme suit :*

1. *Si ϕ est une variable ou une constante individuelle de type a alors ϕ est une formule bien formée de type a.*
2. *Si ϕ est une formule bien formée de type $\langle a, b\rangle$ et ψ une formule bien formée de type a, alors $\phi(\psi)$[4] est une formule bien formée de type b (ce que nous appelons le principe d'application fonctionelle).*
3. *Si ϕ est une formule bien formée de type a, alors $\neg\phi$ est une formule bien formée de type a.*
4. *Si ϕ et ψ sont des formules bien formées de type a, alors $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ sont des formules bien formées de type a.*

[4]Dans ce qui suit, nous utiliserons de manière équivalente la notation $\phi@\psi$.

5. *Si ϕ est une formule bien formée de type b, et que x est une variable de type a, alors $\exists x\phi$ et $\forall x\phi$ sont des formules bien formées de type b.*
6. *Si ϕ est une formule bien formée de type b, et que x est une variable de type a, alors $\lambda x.\phi$ est une formule bien formée de type $\langle a, b\rangle$ (la formule $\lambda x.\phi$ est appelée une λ-abstraction, ou encore un λ-terme).*
7. *Rien d'autre n'est une formule bien formée.*

Comme pour la logique des prédicats, le sens d'une représentation sémantique ϕ est donnée par sa satisfaction par rapport à un modèle $M = (D, I)$. D représente le domaine d'interprétation de ϕ, et I sa fonction d'interprétation.

Définition 9 (domaine d'interprétation)**.** *Dans notre cas, les domaines d'interprétation des objets manipulées sont définis en fonction du type sémantique de l'objet comme suit :*
- $D_e = D$, $D_t = \{0, 1\}$
- $D_{\langle a,b\rangle} = (D_b)^{D_a}$

Définition 10 (satisfaction)**.** *La satisfaction de ϕ correspond à la notion de satisfaction d'une formule de logique des prédicats vue précédemment (définition 4 page 11), étendue par la définition de l'interprétation d'une application fonctionnelle et d'une λ-abstraction comme suit :*

(application fonctionnelle) *Si ϕ est une formule bien formée de type $\langle a, b\rangle$ et ψ une formule bien formée de type a, alors la formule $\phi(\psi)$ est interprétée par rapport au modèle M et à l'assignation g via $\|\phi(\psi)\|_{M,g} = \|\phi\|_{M,g}(\|\psi\|_{M,g})$ (ainsi l'interprétation de $\phi(\psi)$ correspond à l'interprétation de ϕ appliquée à l'interprétation de ψ).*

(λ-abstraction) *Si ϕ est une formule bien formée de type b, et que x est une variable de type a, alors $\|\lambda x.\phi\|_{M,g} = h \in \{0, 1\}^D$ telle que $\forall d \in D, h(d) = \|\phi\|_{M,g[v/d]}$ (où $[v/d]$ représente la substitution de v par d).*

Notons que l'opérateur d'abstraction λ permet de référer à un ensemble de fonctions caractéristiques (*i.e.* à un ensemble d'interprétations).
Dans ce contexte, (Montague, 1974a) associe directement aux catégories syntaxiques du langage, un type sémantique permettant de contrôler l'interprétation sémantique d'un mot. Les principaux types sémantiques utilisés sont donnés en figure 1.4.
Enfin, les mots du lexique se voient attribués une formule du langage L (généralement une λ-abstraction), comme dans les exemples ci-dessous :

$$\begin{array}{llcll} \text{(a)} & \text{dort} & \rightarrow & \lambda x.dormir(x) & : \quad \langle e, t\rangle \\ \text{(b)} & \text{aiment} & \rightarrow & \lambda u\lambda v.aimer(u, v) & : \quad \langle e, \langle e, t\rangle\rangle \end{array}$$

Dans l'exemple (a), nous associons au mot *dort* une abstraction par rapport à une variable x désignant l'argument du prédicat *dormir*. Dans l'exemple (b), nous associons

Catégorie syntaxique	Type sémantique
Nom commun, Verbe intransitif	$\langle e, t\rangle$
Verbe transitif	$\langle e, \langle e, t\rangle\rangle$
Verbe ditransitif	$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$
Modificateur de phrase	$\langle t, t\rangle$
Modificateur de nom ou de verbe	$\langle\langle e, t\rangle, \langle e, t\rangle\rangle$
Quantificateur, prédicat du second ordre	$\langle\langle e, t\rangle, t\rangle$

FIG. 1.4 – Associations catégories syntaxiques – types sémantiques.

au mot *aiment* une double abstraction, tout d'abord par rapport à la variable v, puis par rapport à la variable u, désignant respectivement le premier et le second argument du prédicat *aimer*.

Les λ-abstractions vont nous permettre de définir la façon dont les variables d'une formule sont substituées lors de la composition sémantique. Elles marquent la présence d'une information sémantique manquante dans l'expression liée à un constituant de la phrase.

1.1.4.2 Composition sémantique au moyen du λ-calcul

A partir du langage logique introduit au paragraphe précédent, nous pouvons à présent automatiser le calcul sémantique au moyen de règles de composition fondées sur les règles syntaxiques décrivant les relations entre constituants de la phrase.

Prenons l'exemple de la règle syntaxique (hors-contexte) suivante :

$$P \rightarrow GN\, GV_1$$

Cette règle décrit le lien entre un groupe nominal sujet et un verbe intransitif. La règle sémantique correspondante est la suivante :

$$P' \rightarrow GN' \,@\, GV_1'$$

où P',GN' et GV_1' correspondent aux représentations sémantiques associées respectivement aux catégories syntaxiques P, GN et GV_1, et @ représente l'application fonctionnelle.
Ainsi, si notre grammaire contient également les règles suivantes :

$$GN \rightarrow Jean$$

$$GV_1 \rightarrow dort$$

dont les règles sémantiques correspondantes sont les suivantes (les mots ont été substitués par leur représentation sémantique) :

$$GN' \rightarrow \lambda P.P@Jean$$

$$GV_1' \rightarrow \lambda x.dormir(x)$$

La phrase « Jean dort », de catégorie syntaxique P, est donc interprétée sémantiquement comme suit :

$$P' \rightarrow \lambda P.P@Jean \,@\, \lambda x.dormir(x)$$

cette interprétation correspond à l'application fonctionnelle de l'expression $\lambda P.P@Jean$ à l'expression $\lambda x.dort(x)$ (nous désignons par le terme d'argument l'expression située à droite du symbole @, et par celui de foncteur celle située à gauche de ce même symbole). Cette interprétation peut être simplifiée par β-réduction. La β-réduction correspond à la substitution des occurrences de la variable abstraite d'une λ-abstraction, par un argument. Dans le cas présent, la β-réduction a pour effet de remplacer toutes les occurrences de la variable abstraite (ici P) par l'argument ($\lambda x.dormir(x)$ dans notre exemple), ce qui produit la formule suivante :

$$P' \rightarrow \lambda x.dormir(x) \,@\, Jean$$

après une nouvelle β-réduction, nous obtenons finalement l'expression :

$$P' \rightarrow dormir(Jean)$$

Pour récapituler, le procédé de construction sémantique proposé par (Montague, 1974a; Montague, 1974b) se fait en trois étapes :

1. Définition d'un ensemble de règles permettant de décrire la structure syntaxique des phrases couvertes[5]
2. Association à chaque mot du lexique, d'une λ-abstraction typée sémantiquement.
3. Conversion des règles syntaxiques en règles sémantiques guidant le calcul sémantique (réalisé par applications fonctionnelles).

1.1.4.3 Exemples

Pour illustrer ce calcul sémantique, nous allons considérer la grammaire hors-contexte ci-dessous[6] :

$$\begin{array}{lcl} P & \rightarrow & (GN\ GV_1 \mid GN\ GV_2) \\ GN & \rightarrow & Det\, NC \\ GN & \rightarrow & NP \\ NP & \rightarrow & Jean \mid Marie \mid Pierre \\ GV_1 & \rightarrow & V_1 \\ V_1 & \rightarrow & dort \end{array}$$

[5]Ici, nous utilisons des règles d'une grammaire hors-contexte, dans (Montague, 1974a), l'auteur utilise des règles d'une grammaire catégorielle.

[6]Ici, nous étendons les règles hors-contextes utilisées dans la section précédentes pour autoriser les disjonctions dans les membres de droite des règles (symbole |).

$$
\begin{array}{lcl}
GV_2 & \rightarrow & V_2\, GN \\
V_2 & \rightarrow & aime \mid\ parle\_a \\
NC & \rightarrow & homme \mid\ femme \\
Det & \rightarrow & un \mid\ tout
\end{array}
$$

Celle-ci permet de décrire des phrases telles que « Jean aime Marie », « un homme dort », ou encore « Marie aime un homme ». Pour pouvoir construire les représentations sémantiques de ces phrases, nous allons associer aux symboles terminaux de cette grammaire (les mots du lexique) des λ-abstractions cohérentes par rapport au type sémantique. Ainsi, nous définissons les associations mot – abstraction de la figure 1.5.

Mot du lexique	λ-abstraction	type sémantique
Jean (catégorie NP)	$\lambda P.P@Jean$	$\langle\langle e,t\rangle,t\rangle$
Marie (catégorie NP)	$\lambda P.P@Marie$	$\langle\langle e,t\rangle,t\rangle$
dort (catégorie GV_1)	$\lambda x.dormir(x)$	$\langle e,t\rangle$
aime (catégorie GV_2)	$\lambda U.\lambda x.(U@\lambda y.aimer(x,y))$	$\langle e,\langle e,t\rangle\rangle$
un (catégorie Det)	$\lambda P.\lambda Q.(\exists u.P@u \wedge Q@u)$	$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$
homme (catégorie NC)	$\lambda z.homme(z)$	$\langle e,t\rangle$

FIG. 1.5 – Associations entre mots du lexique et λ-abstractions.

Remarque : d'où viennent ces abstractions ? Une première interrogation, aux vues de ces abstractions, concerne les noms propres. Pourquoi ceux-ci sont-ils représentés par une abstraction et non par une constante ? Les règles de notre grammaire nous indiquent que les groupes nominaux et les noms propres apparaissent aux mêmes positions de la phrase. Ce qui suggère qu'ils aient le même type sémantique, à savoir $\langle\langle e,t\rangle,t\rangle$ ou $\langle e\rangle$ (puisque s'ils apparaissent comme sujet d'un verbe intransitif, et que ce dernier est de type $\langle e,t\rangle$ et la phrase de type $\langle t\rangle$, le type du verbe appliqué à celui du sujet doit donner $\langle t\rangle$). Dans notre cas, nous avons choisi d'utiliser le type $\langle\langle e,t\rangle,t\rangle$, en associant le nom propre $Jean$ à l'abstraction $\lambda P.P@Jean$.

Une seconde interrogation concerne les verbes transitifs. Pourquoi ne pas les abstraire directement via $\lambda x.\lambda y.aimer(y,x)$[7] ? Nous souhaitons construire une représentation sémantique en logique du premier ordre, or un argument d'un verbe transitif peut très bien être représenté par un prédicat (cas d'un nom commun par exemple). En utilisant la représentation $\lambda x.\lambda y.aimer(y,x)$, nous substituerions x par une λ-abstraction que nous ne pourrions plus réduire par la suite.

[7] Aux vues des règles hors-contextes introduites précédemment, le patient est appliqué avant l'agent d'où l'ordre des arguments du prédicat.

Une troisième interrogation enfin concerne l'abstraction liée au déterminant *un* (représenté au moyen du quantificateur existentiel). Un déterminant correspond à une fonction du type $\langle e, t\rangle$ (représentant un nom commun) vers le type $\langle\langle e, t\rangle, t\rangle$ (représentant un groupe nominal). De manière intuitive, un déterminant permet de référer à un objet d'un certain type (la *restriction* du déterminant) qui a une propriété donnée (la *portée* du déterminant). L'abstraction que nous utilisons doit donc permettre de référer à ces deux informations via des variables qui seront substituées correctement par application du nom commun, puis du verbe. Cette abstraction est $\lambda P.\lambda Q.(\exists u.P@x \wedge Q@x)$.

Construction sémantique de la phrase « Jean aime Marie » Une fois notre grammaire et notre lexique sémantique définis, nous pouvons construire la représentation sémantique de la phrase. Sous réserve que nous ayons mis la grammaire hors-contexte sous forme normale, nous pouvons représenter la structure syntaxique de la phrase sous la forme d'un arbre binaire. Ainsi, la figure 1.6 nous donne la structure syntaxique de la phrase « Jean aime Marie ».

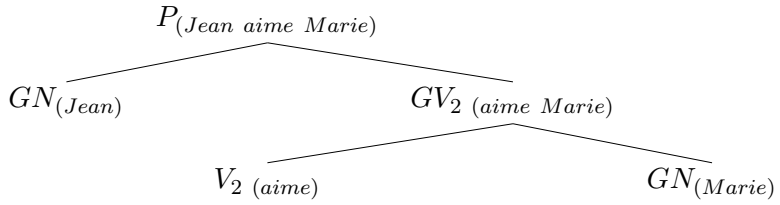
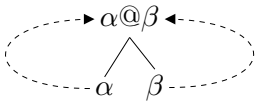


FIG. 1.6 – Structure syntaxique de la phrase « Jean aime Marie ».

Nous allons utiliser cette structure syntaxique pour réaliser les applications fonctionnelles entre nœuds frères de l'arbre syntaxique. Plus précisément, nous appliquons les formules comme illustré ci-dessous :



La formule associée au nœud racine correspond à l'application $\alpha@\beta$ (qui sera soumise à β-réduction).

Si nous revenons à notre exemple « Jean aime Marie », les applications fonctionnelles réalisées dans le cadre du calcul sémantique sont représentées en figure 1.7.

Pour obtenir la représentation sémantique associée à la phrase, il reste à β-réduire la

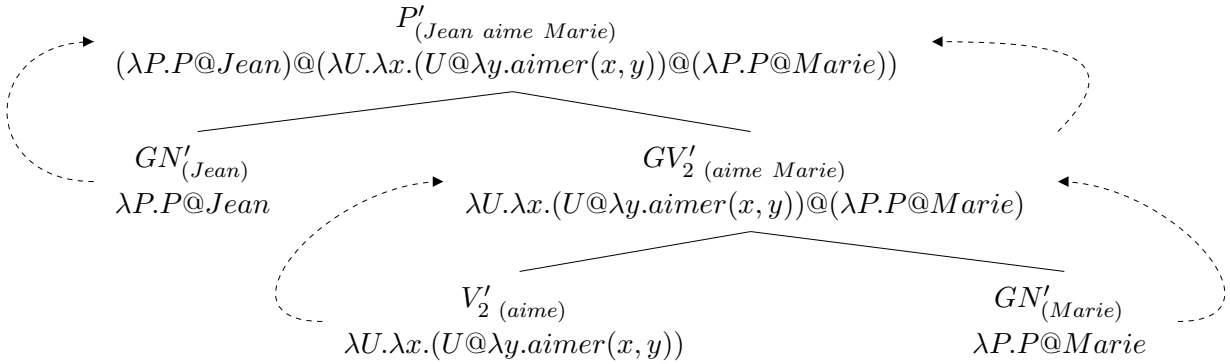


FIG. 1.7 – Calcul sémantique pour la phrase « Jean aime Marie ».

formule associée au nœud racine de l'arbre :

$$\begin{aligned}
&(\lambda P.P@Jean)@((\lambda U.\lambda x.(U@\lambda y.aimer(x,y))@(\lambda P.P@Marie))\\
&(\lambda U.\lambda x.(U@\lambda y.aimer(x,y))@(\lambda P.P@Marie))@Jean\\
&\lambda x.((\lambda P.P@Marie)@\lambda y.aimer(x,y))@Jean\\
&(\lambda P.P@Marie)@\lambda y.aimer(Jean,y)\\
&\lambda y.aimer(Jean,y)@Marie\\
&aimer(Jean,Marie)
\end{aligned}$$

Notons enfin que lors de la β-réduction, il peut arriver que l'on ait le choix de l'ordre des abstractions à réduire. Quelque soit l'ordre choisi, le calcul donnera toujours le même résultat (par confluence du λ-calcul).

Construction sémantique de la phrase « un homme aime Marie » Nous abordons ici un second exemple, légèrement plus complexe, de calcul sémantique au moyen du λ-calcul, en considérant la phrase « un homme aime Marie » (voir figure 1.8).

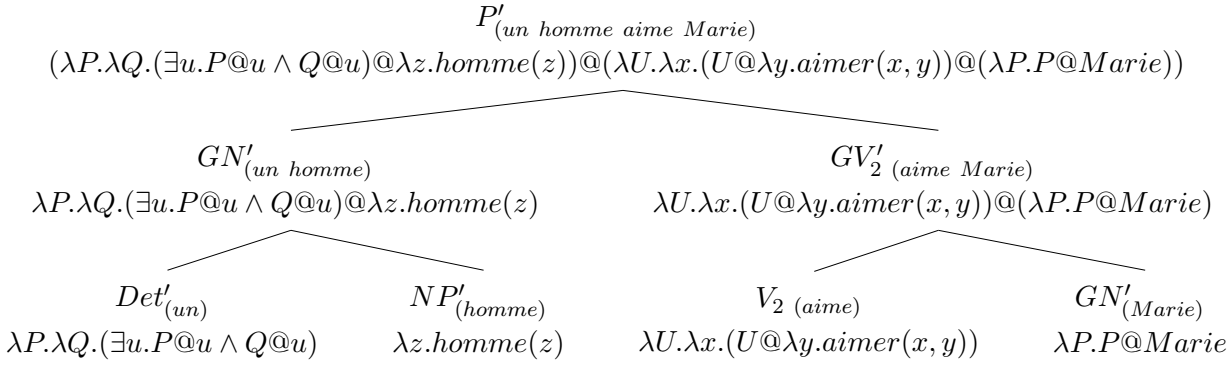


FIG. 1.8 – Calcul sémantique pour la phrase « un homme aime Marie ».

Après β-réduction, nous obtenons :

$$(\lambda P.\lambda Q.(\exists u.P@u \wedge Q@u)@\lambda z.homme(z))@(\lambda U.\lambda x.(U@\lambda y.aimer(x,y))@(\lambda P.P@Marie))$$
$$(\lambda Q.(\exists u.(\lambda z.homme(z))@u \wedge Q@u)@(\lambda x.(\lambda P.P@Marie)@\lambda y.aimer(x,y))$$
$$(\lambda Q.(\exists u.(\lambda z.homme(z))@u \wedge Q@u)@(\lambda x.(\lambda y.aimer(x,y)@Marie))$$
$$(\lambda Q.(\exists u.(\lambda z.homme(z))@u \wedge Q@u)@(\lambda x.aimer(x,Marie))$$
$$(\exists u.(\lambda z.homme(z))@u \wedge (\lambda x.aimer(x,Marie))@u)$$
$$(\exists u.homme(u) \wedge (\lambda x.aimer(x,Marie))@u)$$
$$(\exists u.homme(u) \wedge aimer(u,Marie))$$

Remarque : les noms de variables sont gérés par α-conversion Dans chacun des exemples précédent, nous avons pu construire une représentation sémantique par application fonctionnelle et β-réduction. A aucun moment, nous n'avons eu un même nom de variable apparaissant dans deux abstractions distinctes. Cela est du au fait que, dans notre lexique sémantique (figure 1.5 page 19), les entrées utilisent des abstractions à noms de variable distincts. L'utilisation de noms distincts n'est pas gérable dans un lexique de taille réelle. Afin d'éviter un conflit de nom lors de la β-réduction (conflit dont la conséquence serait le calcul de formules non-valides), nous utilisons une opération de renommage des variables appelée α-conversion.

Définition 11 (α-équivalence)**.** *Deux λ-abstractions qui ne diffèrent que par le nom de variables abstraites sont dites α-équivalentes.*

Définition 12 (α-conversion)**.** *Une α-conversion est une opération consistant à remplacer une λ-abstraction par une λ-abstraction α-équivalente.*

1.2 Une sémantique pour la linguistique informatique

Dans la section précédente, nous avons introduit le domaine de la sémantique formelle, nous avons entre autres présenté un procédé de construction sémantique pour la langue naturelle, utilisant le λ-calcul. Ce procédé, proposé par (Montague, 1974a), a ouvert la voie à une automatisation du traitement sémantique du langage. Dans cette optique de traitement automatique du sens, (Copestake et al., 2005) définit les caractéristiques d'une sémantique pour la linguistique informatique :

L'expressivité – le formalisme sémantique doit être suffisamment expressif.

La compatibilité grammaticale – la description sémantique doit pouvoir s'intégrer à différents types de description syntaxique.

La compatibilité avec un traitement informatique – la description sémantique doit permettre un traitement informatique efficace, notamment lors de la comparaison de

classes d'équivalences sémantiques (dans un contexte de génération de langue naturelle ou traduction automatique par exemple).

Le caractère sous-spécifié – le formalisme sémantique doit être suffisamment flexible pour permettre de manipuler des représentations sémantiques partielles.

Parmi ces desiderata, nous nous intéressons ici à deux points que nous considérons comme centraux, (i) la flexibilité du formalisme via sous-spécification, et (ii) la compatibilité avec différents types de grammaires. Le premier point nous amène à l'introduction des sémantiques sous-spécifiées, le second point à l'introduction du calcul sémantique par unification.

1.2.1 Les sémantiques sous-spécifiées

Nous introduisons les sémantiques sous-spécifiées en nous intéressant à un phénomène linguistique particulier, pour lesquelles ces sémantiques s'avèrent particulièrement adaptées, à savoir les ambiguïtés de portée.

1.2.1.1 Les ambiguïtés de portée

Une ambiguïté de portée correspond à une situation où deux quantificateurs (ou expressions au comportement similaire) peuvent être interprétés comme ayant une portée supérieure l'un par rapport à l'autre. Un exemple célèbre d'ambiguïté de portée correspond à la phrase :

Tout homme aime une femme.

L'ambiguïté de cette phrase provient de la présence de deux quantificateurs dont les portées sont interprétables de deux façons :

1. une première interprétation de cette phrase serait que, pour chaque homme, on puisse déterminer une femme, pour laquelle, l'homme est amoureux. Ce cas correspondrait à la formule logique

$$\forall x.homme(x) \rightarrow (\exists y.femme(y) \wedge aimer(x, y))$$

2. une seconde interprétation correspondrait au cas où tous les hommes aiment la même femme, ce qui serait représenté par la formule logique

$$\exists y.femme(y) \wedge (\forall x.homme(x) \rightarrow aimer(x, y))$$

Un autre cas d'ambiguïté de portée correspond à la phrase :

Tous les boxeurs n'ont pas le nez cassé.

Dans ce cas, l'ambiguïté provient de la présence d'un quantificateur et d'une négation, dont les portées respectives sont interprétables de deux manières :

1. dans une première interprétation, nous pouvons considérer qu'il existe des boxeurs qui n'ont pas le nez cassé, ce qui est représenté par la formule logique

$$\exists x.(\, boxeur(x) \wedge \neg\text{nez-cassé}(x)\,)$$

ou encore, de manière équivalente,

$$\neg(\, \forall x.boxeur(x) \rightarrow \text{nez-cassé}(x)\,)$$

2. dans une seconde interprétation, nous pouvons considérer qu'aucun boxeur n'a le nez cassé. Nous représentons cela au moyen de la formule logique

$$\forall x.(\, boxeur(x) \rightarrow \neg\text{nez-cassé}(x)\,)$$

Dans chacun de ces exemples, la méthode compositionnelle de construction sémantique proposée par Montague ne permet pas de calculer les deux interprétations possibles, en effet, son procédé de calcul sémantique à base de λ-calcul ne peut construire qu'une seule représentation sémantique pour une analyse syntaxique non-ambigüe, puisque l'arbre d'analyse guide les applications fonctionnelles du λ-calcul.

Diverses extensions du formalisme de Montague ont été proposées pour permettre de produire toutes les représentations sémantiques dans les cas d'ambiguïté de portée des quantificateurs. Parmi celles-ci, nous pouvons citer celle de Montague lui-même (Montague, 1988), qui proposa de postuler plusieurs analyses syntaxiques possibles en présence de quantificateurs, ou encore celle de (Barwise et Cooper, 1981), revue et modifiée par (Keller, 1988). Ces extensions proposent des algorithmes permettant une construction des différentes interprétations possibles. Nous n'allons pas introduire ici ces extensions, nous renvoyons le lecteur intéressé vers (Blackburn et Bos, 2005) pour une présentation détaillée.

Deux limites importantes de ces méthodes de construction des interprétations correspondent (1) à leur manque d'expressivité, puisqu'elles ne permettent pas de gérer les ambiguïtés causées par les négations, et (2) à leur efficacité. Le problème du calcul des interprétations est exponentiel par rapport au nombre de quantificateurs (ou constituants similaires). Ainsi, la célèbre phrase due à (Hobbs, 1983) :

Un homme politique peut tromper la plupart des votants sur la plupart des problèmes la plupart du temps, mais aucun homme politique ne peut tromper tous les votants sur chaque problème tout le temps.

est considérée comme ayant plus d'une centaine d'interprétations[8].

Une autre approche pour permettre de gérer les ambiguïtés de portée correspond aux sémantiques sous-spécifiées. L'idée de ce type de représentation sémantique est de ne pas

[8]Bien que l'on exclut en pratique de nombreuses interprétations directement à partir du contexte de l'énoncé.

associer à une phrase plusieurs représentations sémantiques, mais plutôt une unique représentation sémantique sous-spécifiée, qui sera résolue pour produire toutes les interprétations possibles pour la phrase. Ainsi, nous économisons le coût de la résolution des ambiguïtés lors du calcul sémantique. Le « dépliage » de la formule sous-spécifiée (*i.e.*, le calcul des diverses interprétations) sera réalisé *a posteriori* au moyen de techniques efficaces comme par exemple la satisfaction de contraintes de (Koller et al., 2004)[9].

L'idée sous-jacente aux représentations sémantiques sous-spécifiées est représentée sur la figure 1.9 (issue de (Koller et al., 2004)).

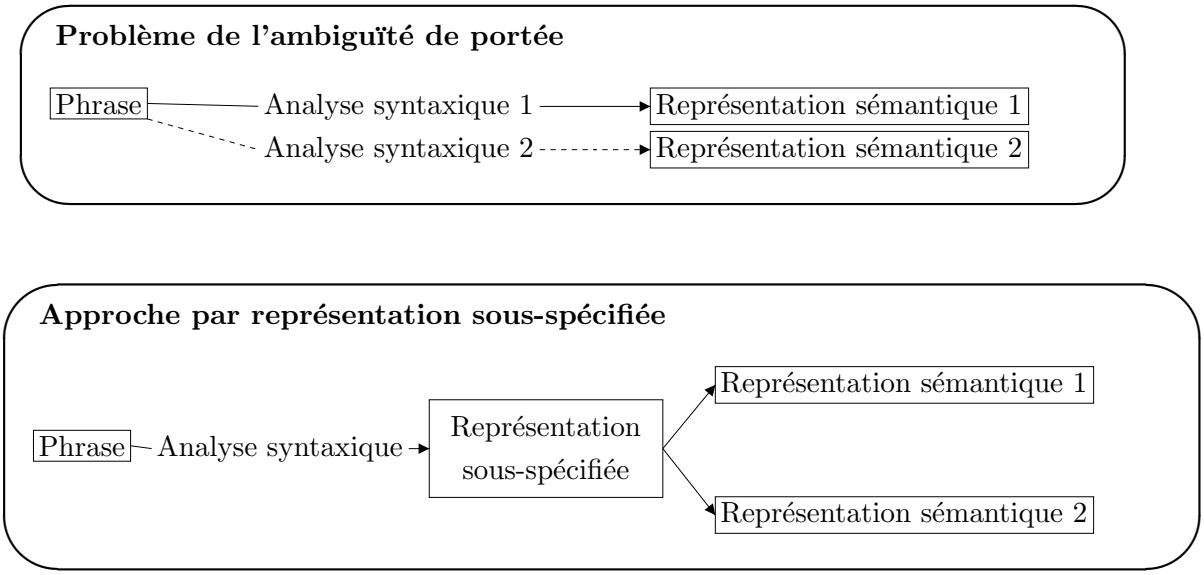


FIG. 1.9 – Construction sémantique et sous-spécification.

Plusieurs formalismes à base de représentation sémantique sous-spécifiée ont vu le jour. Parmi les plus importants, nous pouvons citer la *forme quasi-logique (Quasi-Logical Form – QLF)* (Alshawi et Crouch, 1992), la *sémantique à trous (Hole Semantics)* (Bos, 1995), la *sémantique à récursion minimale (Minimal Recursion Semantics – MRS)* (Copestake et al., 2005), et enfin le *Langage de Contraintes pour Structures Lambda (Constraint Language for Lambda Structures – CLLS)* (Egg et al., 2001) (et de manière plus générale les *Contraintes de Dominance* (Koller, 2004)).

Nous n'allons pas introduire ici l'ensemble de ces formalismes, mais plutôt nous concentrer sur la sémantique à trous, qui nous servira dans la suite de notre présentation (voir chapitre 2). Une autre raison pour laquelle nous axons notre présentation sur la sémantique à trous est que certaines des autres approches peuvent être considérées comme très proches (cas des Contraintes de Dominance, de la CLLS et de la MRS, *cf* (Koller et al., 2003; Fuchss et al., 2004)).

[9]Nous pouvons également citer (**?**) qui ont appliqué la technique de satisfaction de contraintes au problème des ambiguïtés référentielles dans le traitement du discours.

Enfin, pour une présentation plus détaillée des représentations sémantiques sous-spécifiées, le lecteur est renvoyé à (Player, 2004; Ebert, 2005).

1.2.1.2 La sémantique à trous (*Hole Semantics*)

La sémantique à trous a été définie par (Bos, 1995) comme un langage de description de représentations sémantiques sous-spécifiées. En d'autres termes, cette sémantique définit un méta-langage permettant de décrire toutes les interprétations possibles d'une représentation sous-spécifiée.

Un point fort de ce formalisme est qu'il peut être appliqué à divers langages logiques (dans (Bos, 1995), l'auteur l'applique à la logique des prédicats et à la théorie de représentation du discours, *Discourse Representation Theory – DRT*).

Ce formalisme manipule les objets suivants :

- des constantes d'étiquette, appelées *labels* et notées l_i ;
- des variables d'étiquette, appelées *trous (holes)* et notées h_j ;
- des contraintes de portée, représentées par l'opérateur infixe $\leq$.

Dans ce contexte, l'intuition du formalisme est la suivante. Les formules du langage logique auquel s'applique la sémantique à trous[10] sont étiquetées au moyen de labels. Chaque prédicat référant à un constituant syntaxique ayant une portée (quantificateur, négation, etc) utilise une variable d'étiquette comme argument pour la portée. Enfin, des contraintes de portée entre constantes d'étiquette l_i et variables d'étiquette h_j sont spécifiées : $l_i \leq h_j$ (cette contrainte oblige la formule étiquetée par l_i à être incluse directement ou indirectement dans la portée représentée par la variable h_j). Finalement l'interprétation de la formule sous-spécifiée représentée au moyen de ce méta-langage se fait en définissant un ensemble de « branchements », qui correspondent à des assignations entre constantes et variables d'étiquette, assignations respectant l'ensemble des contraintes de portée. On remarque que ce méta-langage à base de contraintes de portée d'étiquette nous permet de ne manipuler que des termes non-récursifs (*i.e.*, prédicats dont les arguments sont soit des constantes soit des variables). Cette non-récursivité présente certains avantages, notamment pour la sélection lexicale en génération ((Copestake et al., 2005)). Cette caractéristique justifie le fait que la sémantique de (Bos, 1995) fait partie de ce que l'on appelle communément les sémantiques plates.

A. Formalisation A présent, nous allons formaliser le méta-langage correspondant au formalisme de la sémantique à trous.

Définition 13 (Représentation sous-spécifiée)**.** *En sémantique à trous, une représentation sous-spécifiée (Underspecified Representation – UR) correspond à un triplet* $U = \langle H, L, C \rangle$ *où* H *est un ensemble de variables d'étiquette,* L *un ensemble de formules étiquetées du langage cible, et* C *un ensemble de contraintes sur* $H \cup L$*.*

[10] Nous appellerons ce langage, langage cible.

Dans la suite, nous noterons l'ensemble H de la représentation U H_U, de même pour les ensembles L et C notés respectivement L_U et C_U.

Définition 14 (Subordination)**.** *Soient k et k' tels que $k, k' \in H_U \cup L_U$, soient $l \in L_U$ et $h \in H_U$. Nous définissons la relation de subordination SUB par rapport à une UR U comme suit :*

1. $SUB_U(k, k)$.
2. $SUB_U(k, k')$ *ssi* $k \leq k' \in C_U$.
3. $SUB_U(h, l)$ *ssi (a)* $\exists \phi$ *telle que* $l : \phi \in L_U$*, (b)* h *est un argument de* $l : \phi$ *et (c)* $\neg SUB_U(l, h)$.
4. $SUB_U(k, k')$ *ssi* $\exists k'' \in H_U \cup L_U$ *telle que* $SUB_U(k, k'')$ *et* $SUB_U(k'', k')$.
5. *aucune autre expression n'est une subordination.*

On en déduit que la relation de subordination est (i) réflexive (via 1) et transitive (via 4), de plus elle est antisymétrique dans certains cas (via 3).

A partir de cette relation, il est possible de définir ce qu'est une représentation *appropriée*.

Définition 15 (Représentation appropriée)**.** *Une UR U est dite appropriée ssi* $\forall k, k' \in H_U \cup L_U$, $\exists k'' \in H_U \cup L_U$ *tel que* $SUB_U(k, k'')$ *et* $SUB_U(k', k'')$.

On remarque que la relation de contrainte $\leq$ est réflexive ($\forall k \in H_U \cup L_U, k \leq k$), antisymétrique ($\forall k, k' \in H_U \cup L_U, [k \leq k' \wedge k' \leq k] \Rightarrow k = k'$) et transitive ($\forall k, k', k'' \in H_U \cup L_U, [k \leq k' \wedge k' \leq k''] \Rightarrow k \leq k''$), elle définit donc un ordre partiel sur $H_U \cup L_U$. De plus, $\forall k_i, k_j \in H_U \cup L_U$, $\exists k \in H_U \cup L_U$ tel que $k \leq k_i \wedge k \leq k_j$. Ainsi, $\langle H_U \cup L_U, \leq \rangle$ définit un *inf-demi-treillis*.

De manière similaire, $\langle H_U \cup L_U, SUB_U \rangle$ définit un inf-demi-treillis ssi U est une UR appropriée.

Définition 16 (Branchement)**.** *Un branchement P (pour Plugging) est une assignation totale de H_U vers L_U. En d'autres termes, un branchement associe de manière bijective, à chaque variable de H_U une constante de L_U.*

Définition 17 (Fonction d'interprétation de branchement)**.** *Une fonction d'interprétation de branchement par rapport à un branchement P est définie par :*

$$I_P(k) = \begin{cases} P(k) & si\ k \in H_U \\ k & si\ k \in L_U \end{cases}$$

Définition 18 (Représentation consistente)**.** *Une UR U est dite consistente par rapport à un branchement P, ce qui est noté $CONS_{U,P}$, ssi $\forall k, k'$ tels que $SUB_U(k, k')$, nous avons :*

$$\begin{array}{ll} soit & I_P(k) = I_P(k') \\ soit & I_P(k) \neq I_P(k') \quad \wedge \neg SUB_U(k', k) \end{array}$$

Ainsi, une UR consistente est une UR dont l'assignation de branchement respecte la caractéristique d'inf-demi-treillis de $\langle H_U \cup L_U, SUB_U \rangle$. A partir de cette définition d'UR consistente, il est possible de définir les branchements acceptables.

Définition 19 (Branchement acceptable)**.** *On appelle branchement acceptable un élément de l'ensemble PP_U des branchements P pour lesquels l'UR U est consistente par rapport à P :*

$$PP_U = \{P \mid CONS_{P,U}\}$$

Enfin, nous allons définir l'étiquette de l'opérateur de portée supérieure, que nous appelons TOP, pour une UR donnée U par rapport à un branchement P.

Définition 20 (Etiquette TOP)**.** *Pour une UR U consistente, l'équivalence suivante est vraie :*
$TOP_{U,P} = I(k)$ *avec* $k \in H_U \cup L_U$ *ssi* $\nexists\ k' \in H_U \cup L_U$ *tel que* $k \neq k' \wedge SUB_U(k, k')$.

Dans ce contexte, l'interprétation d'une UR U par rapport à un modèle M correspond à l'interprétation de la formule étiquetée par TOP, et ce, pour chaque branchement acceptable.

Définition 21 (Interprétation d'une UR)**.** *Une UR U est interprétée par rapport à un modèle M comme suit*[11] *:*

$$\|U\|_M = \{\|TOP_{P,U}\|_{M,P} \mid P \in PP_U\}$$

Finalement, nous allons modifier cette définition pour refléter le fait que suivant le branchement considéré, l'interprétation de l'UR n'est pas la même. Ainsi, si le langage cible est interprété par une valeur de vérité, il sera possible de distinguer la valeur renvoyée pour chaque branchement.

Définition 22 (Interprétation d'une UR (bis))**.** *Une UR est interprétée par rapport à un modèle M comme suit :*

$$\|U\|^*_M = \{\langle P, \|TOP_{P,U}\|_{M,P}\rangle \mid P \in PP_U\}$$

Ainsi, nous avons défini formellement la sémantique à trous, ainsi que son interprétation par rapport à un modèle. Nous allons à présent appliquer cette sémantique à la logique des prédicats et calculer la représentation sous-spécifiée d'une phrase à ambiguïté de portée.

[11] Nous utilisons le même symbole $\|\|$ à la fois pour la fonction d'interprétation du méta-langage et pour la fonction d'interprétation du langage cible, ainsi $\|U\|_M$ représente la fonction d'interprétation du langage de sémantique à trous et $\|TOP_{P,U}\|_{M,P}$ la fonction d'interprétation du langage cible.

B. Application à la logique des prédicats L'application de la sémantique à trous à la logique des prédicats a donné lieu à la logique des prédicats débranchée (*Predicate Logic Unplugged – PLU*).

Définition 23 (Formule de PLU)**.** *Une formule de PLU est définie de manière inductive comme suit :*

1. *Si* $h_i, h_j \in H_U$ *alors* $\neg h_i, h_i \rightarrow h_j, h_i \wedge h_j, h_i \vee h_j$ *sont des formules de PLU.*
2. *Si* x *est une variable de la logique de prédicats,* $h \in H_U$ *alors* $\forall x h, \exists x h$ *sont des formules de PLU.*
3. *Si* P *est un symbole de prédicat n-aire, et* $t_1, \ldots, t_n$ *sont des variables ou constantes de la logique des prédicats, alors* $P(t_1, \ldots, t_n)$ *est une formule de PLU.*
4. *Toute formule qui ne peut être formée à partir des règles précédentes en un nombre fini d'étapes n'est pas une formule de PLU.*

Si l'on considère un modèle M défini par $M = (D, I)$ où D est le domaine d'interprétation et I la fonction d'interprétation, et soient g et g' des fonctions d'assignation, et $\|t\|_{M,g}$ l'interprétation du terme t par rapport à $\{M, g\}$. Alors, la définition de l'interprétation d'une formule de PLU est la suivante.

Définition 24 (Fonction d'interprétation d'une formule de PLU)**.** *Nous définissons la fonction d'interprétation d'une formule de PLU de manière inductive comme suit :*

1. $\|h_i \rightarrow h_j\|_{M,P,g} = 1$ *ssi* $\|h_i\|_{M,P,g} = 0$ *ou* $\|h_j\|_{M,P,g} = 1$
2. $\|h_i \vee h_j\|_{M,P,g} = 1$ *ssi* $\|h_i\|_{M,P,g} = 1$ *ou* $\|h_j\|_{M,P,g} = 1$
3. $\|h_i \wedge h_j\|_{M,P,g} = 1$ *ssi* $\|h_i\|_{M,P,g} = 1$ *et* $\|h_j\|_{M,P,g} = 1$
4. $\|\neg h_i\|_{M,P,g} = 1$ *ssi* $\|h_i\|_{M,P,g} = 0$
5. $\|\forall x h_i\|_{M,P,g} = 1$ *ssi* $\forall d \in D \ \|h_i\|_{M,P,g[d/x]} = 1$
6. $\|\exists x h_i\|_{M,P,g} = 1$ *ssi* $\exists d \in D \ \|h_i\|_{M,P,g[d/x]} = 1$
7. $\|P(t_1, \ldots, t_n)\|_{M,P,g} = 1$ *ssi* $(\|t_1\|_{M,g}, \ldots, \|t_n\|_{M,g}) \in I(P)$

C. Un exemple Observons à présent le calcul sémantique d'une phrase présentant une ambiguïté de portée au moyen de la sémantique à trous. Nous allons considérer la phrase :

Tout homme aime une femme.

Supposons qu'à partir de l'analyse syntaxique de cette phrase, nous extrayons la représentation sous-spécifiée suivante :

$$\langle \left\{ \begin{array}{l} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{array} \right\}, \left\{ \begin{array}{lcl} l_0 & : & \forall x\, h_1 \\ l_1 & : & h_2 \rightarrow h_3 \\ l_2 & : & homme(x) \\ l_3 & : & aimer(x,y) \\ l_4 & : & \exists y\, h_4 \\ l_5 & : & h_5 \wedge h_6 \\ l_6 & : & femme(y) \end{array} \right\}, \left\{ \begin{array}{lcl} l_0 & \leq & h_0 \\ l_1 & \leq & h_1 \\ h_1 & \leq & l_1 \\ l_2 & \leq & h_2 \\ h_2 & \leq & l_2 \\ l_3 & \leq & h_3 \\ l_3 & \leq & h_6 \\ l_4 & \leq & h_0 \\ l_5 & \leq & h_4 \\ h_4 & \leq & l_5 \\ l_6 & \leq & h_5 \\ h_5 & \leq & l_6 \end{array} \right\} \rangle \tag{1.1}$$

Nous pouvons également représenter ces contraintes sous forme graphique comme illustré à la figure 1.10 (les flèches représentent la relation $\leq$).

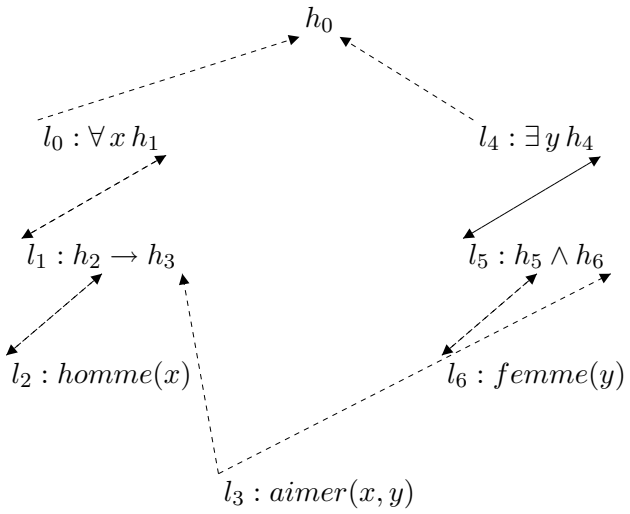


FIG. 1.10 – Représentation sous-spécifiée de « Tout homme aime une femme ».

La variable de trou h_0 sera interprétée comme le TOP de notre représentation, en d'autres termes, l'étiquette de la formule qui sera « branchée » sur cette variable aura la portée la plus étendue. Les contraintes de cette représentation traduisent l'ambiguïté à deux niveaux : (a) quel quantificateur a la portée la plus étendue ($l_0 \leq h_0$, $l_4 \leq h_0$) et (b) dans la portée de quel quantificateur apparaît le prédicat *aimer* ($l_3 \leq h_3$, $l_3 \leq h_6$). Les portées immédiates sont représentées au moyen des contraintes « duales » telles que $l_2 \leq h_2$ et $h_2 \leq l_2$.

Dans ce contexte, calculer les représentations sémantiques pour l'UR (1.1) revient à définir les assignations (« branchements ») entre variables d'étiquette et variables de trous consistents avec les contraintes de l'UR.

Un premier branchement possible est le suivant :

$$P_1 = \{l_0 = h_0; l_1 = h_1; l_2 = h_2; l_3 = h_6; l_4 = h_3; l_5 = h_4; l_6 = h_5\} \tag{1.2}$$

Ce branchement définit une assignation bijective de chaque variable d'étiquette vers une variable de trou. En l'occurrence, la représentation sémantique (totalement spécifiée) correspondante est la suivante :

$$\forall x.homme(x) \rightarrow (\exists y.femme(y) \wedge aimer(x, y))$$

Un autre branchement possible pour cette UR est :

$$P_1 = \{l_0 = h_6; l_1 = h_1; l_2 = h_2; l_3 = h_3; l_4 = h_0; l_5 = h_4; l_6 = h_5\} \tag{1.3}$$

Ce branchement permet la construction de la seconde représentation sémantique :

$$\exists y.femme(y) \wedge (\forall x.homme(x) \rightarrow aimer(x, y))$$

Cet exemple clôt la présentation des sémantiques sous-spécifiées. Nous allons a présent nous intéresser au second point mentionné en introduction de cette section, à savoir l'intégration du calcul sémantique avec différents formalismes grammaticaux.

1.2.2 Un calcul sémantique basé sur l'unification

Depuis les années 80, les grammaires d'unification ont pris un essor important (Shieber et al., 1986). Dans ce contexte, la possibilité d'utiliser l'unification comme opération de composition sémantique est devenue de plus en plus d'actualité. Utiliser l'unification permet en effet d'intégrer les représentations sémantiques dans la grammaire, pour par exemple disposer d'une grammaire utilisable à la fois en analyse et en génération (cas des grammaires syntagmatiques guidées par les têtes – HPSG– (Copestake et al., 2005)).

Intuitivement, il est possible d'émuler le λ-calcul au moyen de variables d'unification. L'idée est de représenter les prédicats par des termes, et de définir une interface syntaxe / sémantique permettant aux variables d'unification contenues dans les termes d'être unifiées correctement lors de la composition sémantique.

Nous pouvons donner un exemple simplifié de construction sémantique à base d'unification, à partir de la grammaire hors-contexte jouet suivante :

$$\begin{aligned} P &\rightarrow GN\ GV_2\ GN \\ GN &\rightarrow NP \\ NP &\rightarrow Jean \\ NP &\rightarrow Marie \\ GV_2 &\rightarrow aime \end{aligned}$$

En utilisant le formalisme des Grammaires de Clauses Définies (DCG) (Pereira et Warren, 1980), il est possible d'ajouter aux symboles utilisés dans ces règles des expressions dans un langage logique (termes par exemple), qui seront accumulées (modulo unification) lors de la dérivation des règles.

Ainsi, nous pouvons réécrire les règles précédentes sous forme de DCG comme ci-dessous (moyennant une mise sous forme normale pour avoir des règles binaires) :

```
p(Sem) --> gn(X,SemGV,Sem) , gv2(X,SemGV).
gv2(X,Sem) --> gv(X,Y,SemGV) , gn(Y,SemGV,Sem).
gn(X,Sem,Sem) --> np(X).

np(j) --> [jean].
np(m) --> [marie].

gv(X,Y,aimer(X,Y)) --> [aime].
```

Dans cet exemple, la sémantique est un terme calculé par accumulation des termes fournis par les constituants syntaxiques. Ainsi la sémantique (variable `Sem`) est calculée par composition de la sémantique du groupe nominal sujet et de celle du groupe verbal. On remarque que le lien entre prédicat verbal et arguments est défini dans la règle `gv(X,Y,aimer(X,Y))`.

Ainsi, l'analyse de la phrase « Jean aime Marie » au moyen de cette grammaire produit la représentation sémantique `aimer(j,m)`.

1.3 Conclusion

Dans ce chapitre, nous avons présenté les origines de la sémantique formelle en linguistique, et les travaux qui en ont découlé, notamment le calcul sémantique de Montague, qui est à la base de la sémantique informatique (*Computational Semantics*). Nous avons également présenté en détail la sémantique plate de (Bos, 1995) fournissant un cadre flexible pour représenter le sens d'énoncés.

Le fait de pouvoir associer à un énoncé une représentation sémantique sous forme de formule logique, présente plusieurs avantages, dont le principal est de pouvoir raisonner à partir de ces représentations sémantiques (*i.e.*, faire de l'inférence).

Cependant, les représentations sémantiques basées sur une logique du premier ordre telle que celles présentées ici sont sujettes à discussion de plusieurs points de vue.

D'un point de vue de l'expressivité du langage, les prédicats et quantificateurs ne peuvent s'appliquer qu'à des constantes ou variables individuelles (par définition de premier ordre). Comment alors construire la représentation sémantique de phrases telles que « Rouge est une couleur » ou « Jean est tout ce qu'est Marie » ?[12]

[12] Exemples issus de (Gregory, 2000).

D'un point de vue intentionnel, le traitement de certains types d'adjectifs est impossible au moyen de la logique des prédicats. Ainsi les adjectifs relatifs (par exemple *grande* dans « une grande souris ») ne peuvent être représentés (« une grande souris » ne peut être représenté par $grand(x) \wedge souris(x)$ puisque le caractère *grand* est lié à l'objet *souris*, sa valeur est relative). De même, les adjectifs intentionnels (comme par exemple *ancien*) ne peuvent être traités puisqu'ils ne réfèrent pas directement au nom auquel ils se rattachent syntaxiquement (notion de temporalité).

Ces deux limitations de l'approche basée sur la logique du premier ordre ont été traitées par Montague au moyen de la logique intentionnelle d'ordre supérieur. La présentation de ce formalisme n'entre pas dans le cadre de nos travaux, le lecteur intéressé pourra consulter (Montague, 1970; Van Benthem, 1988) pour plus de détails sur cette logique.

De plus, dans l'approche que nous avons présenté ici, nous avons considéré que le sens d'un énoncé correspond à sa valeur de vérité. Cette approche, proposée par Tarski, est souvent appelée sémantique statique, par opposition à l'approche de la sémantique dynamique (Kamp, 1981; Groenendijk et Stokhof, 1991; Stalnaker, 1998). Dans cette dernière, le sens d'un énoncé est vu comme son potentiel de modification du contexte. Le contexte est représenté par l'ensemble des croyances des participants à la conversation. De plus, un locuteur fait des assertions ayant un contenu, lequel est dépendant du contexte de locution. Dans notre cas, nous cherchons à construire une représentation du sens d'un énoncé, indépendemment de sa localisation dans le discours. Nous ne détaillons donc pas plus la sémantique dynamique et renvoyons le lecteur intéressé vers les articles sus-cités.

Dans le chapitre qui suit, nous introduisons le formalisme grammatical des Grammaires d'Arbres Adjoints, ainsi que les différents procédés de construction sémantique pour ce formalisme. Notre objectif est de parvenir à la définition d'une plate-forme permettant le développement de grammaires d'arbres adjoints avec interface syntaxe / sémantique qui soient utilisables à la fois en analyse et en génération.

Chapitre 2

Le calcul sémantique pour Grammaires d'Arbres Adjoints

Sommaire

Dans le chapitre précédent, nous avons étudié un procédé de construction sémantique basé sur la structure interne de la phrase. Cette structure a été représentée au moyen de règles hors-contextes. A présent, nous allons utiliser un autre formalisme syntaxique permettant de rendre compte des liens entre constituants de la phrase : les Grammaires d'Arbres Adjoints (*Tree Adjoining Grammars – TAG*). Dans un premier temps, nous présentons ce formalisme syntaxique, nous introduisons notamment ses propriétés linguistiques et informatiques. Dans un second temps, nous présentons différents procédés de construction sémantique pour ce formalisme.

2.1 Les Grammaires d'Arbres Adjoints

Les Grammaires d'Arbres Adjoints (*Tree Adjoining Grammars* – TAG) ont été introduites par (Joshi et al., 1975)[13]. A la différence des grammaires hors-contextes qui constituent un système de réécriture de chaînes, les grammaires TAG sont un système de réécriture d'arbres. Ainsi une grammaire TAG est composée d'arbres pouvant être combinés au moyen d'opérations de réécriture définies dans le formalisme TAG.

2.1.1 Définitions

Définition 25 (Grammaire TAG)**.** *Formellement, une grammaire d'arbres adjoints G est définie par le quintuplet suivant :*

$$G = (NT, T, S, I, A)$$

où :

- *NT représente l'ensemble fini des symboles non-terminaux (en pratique, ce sont les catégories syntaxiques),*
- *T l'ensemble fini des symboles terminaux, $NT \cap T = \emptyset$,*
- *S est le symbole distingué ou axiome*[14]*,*
- *I correspond à l'ensemble fini des arbres initiaux,*
- *A correspond à l'ensemble fini des arbres auxiliaires.*

L'union des arbres initiaux et des arbres auxiliaires ($I \cup A$) forme l'ensemble des arbres élémentaires.

Les arbres contenus dans I et A se distinguent (a) par leur forme et (b) par les opération de réécriture dans lesquelles ils peuvent apparaître.

Concernant (a), les arbres initiaux et auxiliaires se définissent comme suit :

Les arbres initiaux. Ils se caractérisent par des nœuds intérieurs étiquetés par des symboles non-terminaux, et des nœuds feuilles qui sont soit étiquetés par des symboles terminaux, soit étiquetés par des symboles non-terminaux et marqués pour une substitution (symbole $\downarrow$).

Les arbres auxiliaires. Ils disposent parmi leurs nœuds feuilles d'un nœud étiqueté par le même symbole que la racine de l'arbre, ce nœud est alors appelé *nœud pied* et est marqué par le symbole $\star$.

Concernant (b), les opérations de réécriture autorisées par le formalisme TAG sont de deux types : **substitution** et **adjonction**.

[13] Voir aussi (Joshi et Schabes, 1997)

[14] Dans ce qui suit nous prendrons comme symbole distingué p, correspondant à la catégorie syntaxique *phrase*.

Définition 26 (Opération de substitution)**.** *L'opération de substitution, représentée figure 2.1, consiste à substituer un nœud feuille étiqueté X d'un arbre γ, par un arbre (initial ou dérivé d'un arbre initial) α, dont la racine est étiquetée X également.*

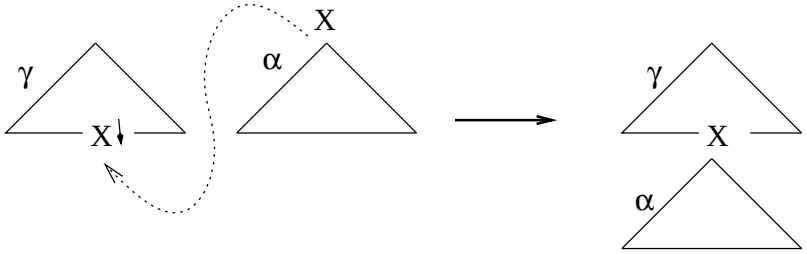


FIG. 2.1 – Opération de substitution.

Définition 27 (Opération d'adjonction)**.** *L'opération d'adjonction, représentée figure 2.2, consiste à insérer un arbre* auxiliaire *à l'intérieur d'un arbre. Ainsi, sur la figure 2.2, l'arbre auxiliaire β de racine X et ayant un nœud feuille d'étiquette identique, est inséré à l'intérieur de l'arbre γ (sur un nœud interne étiqueté X), ce qui produit l'arbre θ.*

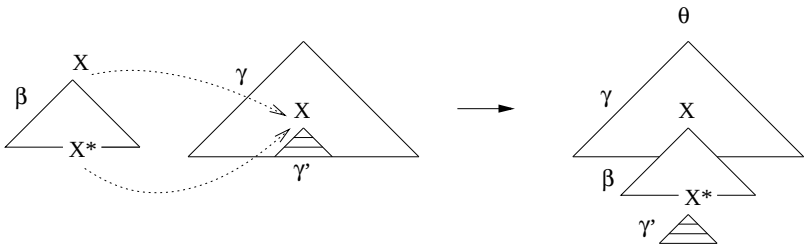


FIG. 2.2 – Opération d'adjonction.

On note qu'il est interdit d'adjoindre un arbre sur un nœud marqué pour substitution (symbole ↓).

Lorsque l'on applique une opération de réécriture (adjonction ou substitution) sur deux arbres élémentaires, un nouvel arbre est produit, cet arbre est appelé arbre *dérivé*. Après réécritures successives, on obtient un arbre dérivé dont tous les nœuds feuilles sont des items lexicaux (plus aucun nœud feuille marqué pour substitution), on parle alors d'arbre dérivé *complet*. On appelle généralement *dérivation*, la réécriture d'un arbre dérivé complet à partir d'un arbre élémentaire dont la racine est étiquetée par le symbole distingué. Un exemple de dérivation est donné figure 2.3 (l'arbre dérivé complet ainsi produit est représenté à droite de la flèche). Si nous désignons par τ_{Jean} (respectivement τ_{dort} et $\tau_{beaucoup}$) l'arbre associé au mot *Jean* (respectivement *dort* et *beaucoup*), cette dérivation utilise les deux règles de réécriture du formalisme TAG comme suit[15] :

[15] On note que l'ordre d'application de ces opérations de réécritures est indifférent (voir remarque page 38).

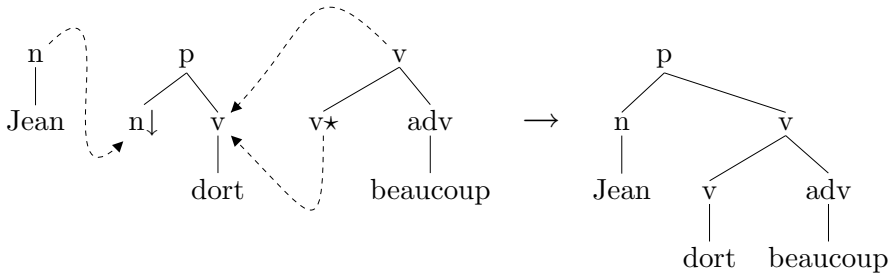


FIG. 2.3 – Dérivation pour « Jean dort beaucoup ».

- l'arbre τ_{Jean} est *substitué* sur le nœud feuille d'étiquette n de l'arbre τ_{dort}, ce qui produit un arbre que nous appelons τ'_{dort} (non-représenté sur la figure 2.3) ;
- l'arbre (auxiliaire) $\tau_{beaucoup}$ est *adjoint* sur le nœud interne d'étiquette v de l'arbre τ'_{dort}, ce qui produit finalement l'arbre *dérivé* τ''_{dort} (à droite sur la figure 2.3).

Lors d'une dérivation, il est possible de définir, en plus de l'arbre dérivé complet, une autre structure, qui décrit les opérations de réécriture utilisées lors de la construction de l'arbre dérivé. Cette structure est appelée l'*arbre de dérivation.*

Définition 28 (Arbre de dérivation)**.** *L'arbre de dérivation est une structure où (i) les nœuds correspondent aux arbres élémentaires impliqués dans la dérivation, et (ii) chaque arc représente une opération de réécriture entre les deux arbres situés à ses extrémités. De plus, chaque arc est étiqueté par l'adresse de Gorn*[16] *du nœud où a lieu l'opération de réécriture.*

L'arbre de dérivation décrivant l'analyse de la phrase « Jean dort beaucoup » est représenté figure 2.4 (les opérations d'adjonction sont représentées en pointillés, celles de substitution en trait plein, les adresses de Gorn des nœuds sur lesquels s'applique l'opération de réécriture sont notées entre parenthèses).

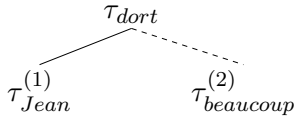


FIG. 2.4 – Arbre de dérivation pour la phrase « Jean dort beaucoup ».

Remarques

1. Dans le formalisme TAG, il est interdit d'adjoindre deux arbres auxiliaires sur le même nœud d'un arbre donné. La conséquence de cela est que l'ordre entre deux adjonctions appliquées *à un même arbre* est indifférent. Quelle que soit l'adjonction réalisée

[16] L'adresse de Gorn est définie comme suit : le nœud racine a pour adresse 0, ses fils $1, \ldots, n$ et le k^e fils d'un nœud d'adresse j a pour adresse $j.k$.

en premier, l'arbre dérivé produit sera le même. On remarque donc que, via cette restriction, les nœuds frères d'un arbre de dérivation ne sont pas ordonnés.

2. L'opération d'adjonction peut être contrainte de manière à limiter les arbres auxiliaires pouvant s'adjoindre sur un nœud donné. Une contrainte d'adjonction concerne un nœud d'un arbre élémentaire. Il existe trois types de contraintes d'adjonction :

   Adjonction obligatoire. Le nœud en question doit obligatoirement être le site d'une adjonction au cours de la dérivation.

   Adjonction interdite. Le nœud en question ne peut pas être site d'une adjonction.

   Adjonction sélective. Le nœud en question ne peut être le site d'une adjonction que pour certains arbres auxiliaires.

A partir de ces définitions d'une grammaire TAG, et des opérations permises, il est possible de définir le langage généré par une grammaire TAG :

Définition 29 (Langage généré par une grammaire TAG)**.** *Soit G une grammaire* TAG*. On note T_G l'ensemble des arbres dérivés complets engendrés par G. On appelle langage généré (ou encore langage couvert) par G, l'ensemble des chaînes de symboles terminaux situées sur les feuilles des éléments de T_G.*

A présent, nous introduisons deux extensions du formalisme TAG, qui sont utilisées couramment en linguistique informatique, à savoir les grammaires TAG à structures de traits, et les grammaires TAG lexicalisées.

2.1.2 Les grammaires TAG à structures de traits *(Feature-Based Tree Adjoining Grammars – FBTAG)*

Cette extension du formalisme TAG a été proposée par (Vijay-Shanker et Joshi, 1988). Dans cette version des grammaires TAG, chaque nœud d'un arbre élémentaire se voit associé deux structures de traits[17] nommées *top* et *bot*[18]. La structure *top* contient des traits référant aux relations d'un nœud avec ceux qui le domine, et la structure *bot* aux relations de ce nœud avec ses nœuds fils.

Ces structures de traits contraignent les dérivations des manières suivantes :

(substitution) La structure de traits *top* du nœud de substitution est unifiée avec la structure de traits *top* du nœud racine de l'arbre substitué (voir figure 2.5).

(adjonction) Si l'on appelle N le nœud site de l'adjonction, et R et P le nœud racine, respectivement pied, de l'arbre auxiliaire adjoint, les structures de traits *top* des nœuds N et R sont unifiées, ainsi que les structures de traits *bot* des nœuds N et P (voir figure 2.6).

En fin de dérivation (lorsque l'on obtient un arbre dérivé complet), les structures de traits *top* et *bot* de chacun des nœuds de l'arbre sont unifiées.

[17] Appelées également matrices attributs-valeurs.

[18] Sauf les nœuds marqués pour substitution qui ne contiennent qu'une structure *top*.

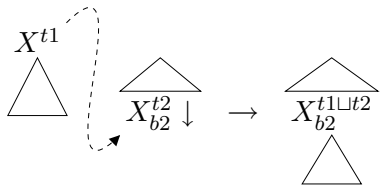


FIG. 2.5 – Substitution pour FB-TAG.

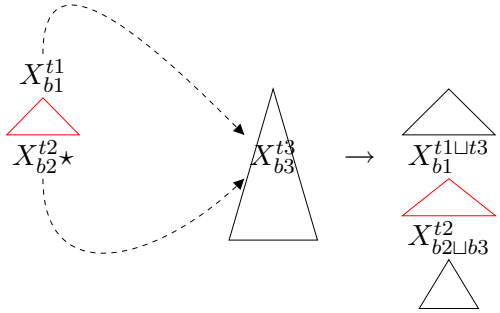


FIG. 2.6 – Adjonction pour FB-TAG.

Remarque Nous avons vu précédemment qu'il est possible de contraindre les adjonctions sur un nœud donné. Dans le formalisme FB-TAG, les contraintes d'adjonctions peuvent être représentées au moyen des structures de traits *top* et *bot* :

- dans le cas d'une adjonction obligatoire, il suffit d'associer au nœud en question des structures *top* et *bot* qui ne sont pas unifiables ;
- dans le cas d'une adjonction interdite, il faut associer à l'une des structures *top* ou *bot* un trait qui n'est unifiable avec aucun autre trait de la grammaire ;
- dans le cas d'une adjonction sélective, il faut associer aux structures *top* et *bot* des traits unifiables uniquement avec ceux des arbres autorisés à s'adjoindre.

2.1.3 Les grammaires TAG lexicalisées *(Lexicalized Tree Adjoining Grammars – LTAG)*

Définition 30 (Grammaire TAG lexicalisée)**.** *Une grammaire* TAG *lexicalisée est une grammaire* TAG *dont chaque arbre élémentaire contient au moins un nœud feuille étiqueté par un symbole terminal.*

Ainsi une grammaire lexicalisée peut être vue comme une fonction associant à chaque mot du lexique, un ensemble de structures syntaxiques (arbres) représentant l'usage de ce mot dans les différentes phrases de la langue. Notons que les grammaires lexicalisées présentent un avantage pratique : lors de l'analyse syntaxique, l'analyseur peut sélectionner une sous-grammaire suivant les mots de la chaîne à analyser, ce qui facilite la complexité en temps de l'analyse.

Dorénavant, nous désignerons par grammaire TAG, une grammaire TAG à structures de

traits et lexicalisée (LFB-TAG). Comme nous allons le voir dans les sections suivantes, ce type de grammaires présente de nombreux intérêts dans un contexte de traitement automatique de la langue.

2.1.4 Propriétés formelles

Classe de langage des grammaires TAG En considérant la classification des langages de (Chomsky, 1956), les grammaires TAG appartiennent à la famille des grammaires permettant d'engendrer tous les langages hors-contextes, ainsi que certains langages contextuels, comme par exemple le langage $a^n b^n e c^n d^n$ (voir figure 2.7[19]).

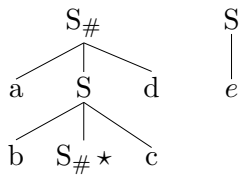


FIG. 2.7 – Grammaire TAG engendrant le langage $a^n b^n e c^n d^n$.

Cependant, une grammaire TAG ne peut engendrer le langage contextuel $a^n b^n e^n c^n d^n$, de ce fait, les grammaires TAG sont considérées comme appartenant à la classe des grammaires légèrement sensibles au contexte (*Mildly Context Sensitive Grammars*).

Lexicalisation d'une grammaire hors-contexte Une grammaire TAG permet de lexicaliser une grammaire hors-contexte finiment ambigüe[20] en conservant son pouvoir génératif fort[21]. La preuve de cette propriété est donnée dans (Joshi et Schabes, 1997). Il convient de noter que les grammaires TAG ne constituent pas la plus petite classe de grammaires permettant de lexicaliser une grammaire hors-contexte, en effet (Schabes et Waters, 1995) montre que les *Grammaires d'Arbres d'Insertion*[22] permettent de lexicaliser une grammaire hors-contexte. (Rogers, 1994) définit également une sous-classe des grammaires TAG permettant de lexicaliser une grammaire hors-contexte, tout en étant moins contrainte que celle de (Schabes et Waters, 1995).

Un domaine de localité étendu A la différence des grammaires hors-contextes, représentables sous forme d'arbres de profondeur un (voir figure 2.8), les grammaires TAG peuvent avoir une profondeur quelconque.

En conséquence, lors de la définition des arbres élémentaires de notre grammaire, il est possible d'associer à un prédicat (verbe, adjectif prédicatif, etc) une structure disposant

[19] le symbole # indique une adjonction interdite.

[20] Une grammaire est dite finiment ambigüe si une phrase de longueur finie ne peut pas être engendrée par un nombre infini de dérivations.

[21] Par pouvoir génératif fort, nous désignons l'ensemble des chaînes générées ainsi que leur structure dérivée (lors que l'on ne considère que les chaînes dérivées, *i.e.*, le langage engendré, on parle de pouvoir génératif

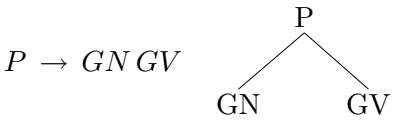


FIG. 2.8 – Règle hors-contexte représentée sous forme d'arbre.

d'un nœud représentant chacun des arguments que ce prédicat sous-catégorise[23] (ce qui est appelé généralement *Principe de Coocurrence Prédicat-Arguments* (Abeillé, 1993; Frank, 2002)). Cette capacité des grammaires TAG à manipuler des structures dont le domaine de localité contient un nœud pour chacun des arguments du prédicat quelle que soit sa réalisation est appelée *domaine de localité étendu*.

Un intérêt important de ce domaine de localité étendu est qu'il est possible de représenter en TAG certains phénomènes linguistiques tels que l'accord sur une seule et même structure, sans avoir à propager de trait supplémentaire (voir figure 2.9[24]).

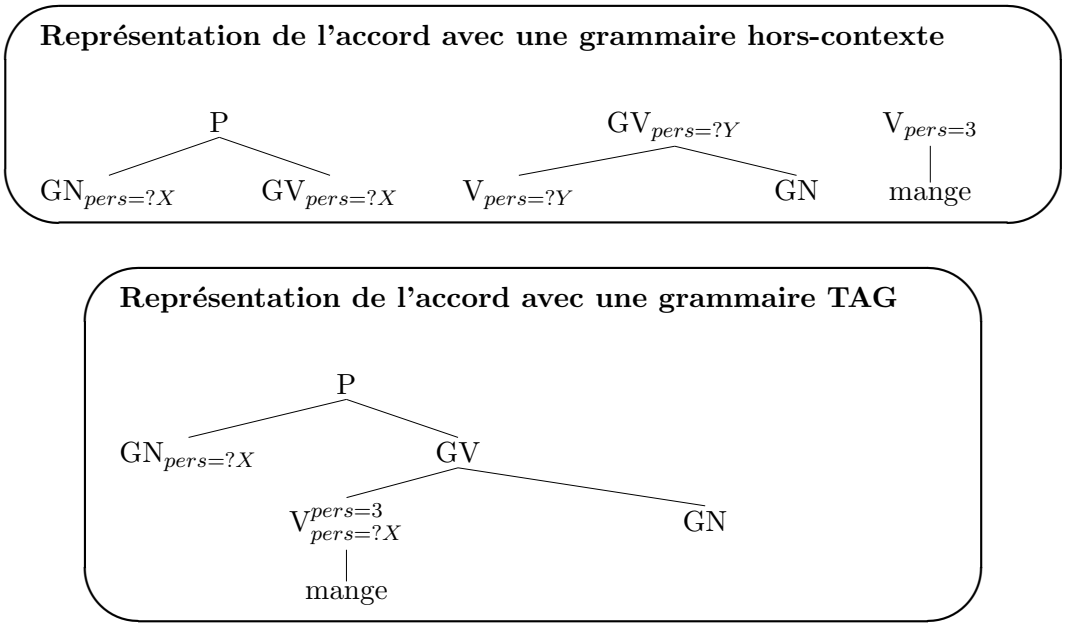


FIG. 2.9 – Représentation de l'accord avec une grammaire hors-contexte et une grammaire TAG.

Des dépendances longue distance et une factorisation des composantes grammaticales récursives hors du domaine de localité Par *dépendance longue distance*,

faible).

[22]Une grammaire d'Arbres d'Insertion correspond à une grammaire TAG dont l'opération d'adjonction est contrainte à être non-englobante, *i.e.*, à n'apparaître que d'un côté de l'épine dorsale de l'arbre.

[23]Par sous-catégorisation, nous entendons arité d'un prédicat et catégorie syntaxique de ses arguments.

[24]Exemple issu de (Crabbé, 2005b).

on réfère à la situation où un argument est réalisé à une distance éloignée du prédicat dont il dépend comme dans les exemples (1c), (1d) ci-dessous :

(1) a. Jean regarde un film.

b. Le film [que Jean regarde].

c. Le film [que Pierre pense [que Jean regarde]].

d. Le film [que Marie croit [que Pierre pense [que Jean regarde]]].

On remarque qu'il peut apparaître un nombre infini d'occurrences de composantes du type *Pierre pense que*. Une représentation élégante de ce phénomène a été proposée pour TAG par (Kroch et Joshi, 1985). L'idée est de décrire la position des arguments du prédicat dans l'arbre élémentaire qui lui est associé, et d'utiliser l'opération d'adjonction pour insérer les composantes récursives dans cet arbre élémentaire. Ainsi l'exemple énuméré précédemment se traite en TAG au moyen de la grammaire représentée figure 2.10.

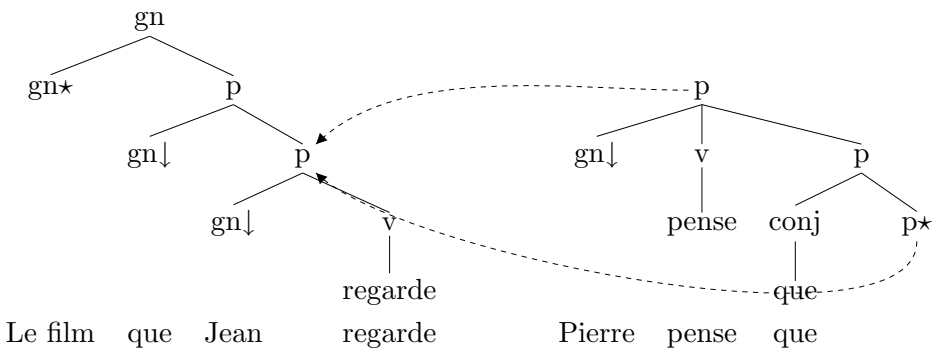


FIG. 2.10 – Factorisation des composantes récursives hors du domaine de localité.

Expressions semi-figées et mots composés. Enfin, comme l'a montrée (Abeillé, 1990), le formalisme TAG permet d'associer à un même arbre plusieurs symboles terminaux, ce qui permet de traiter les expressions semi-figées et mots-composés (voir figure 2.11).

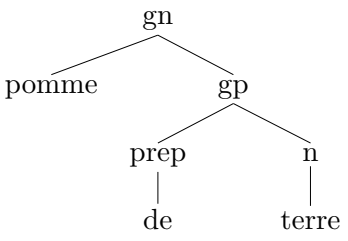


FIG. 2.11 – Représentation du mot-composé « pomme de terre » en TAG.

Les propriétés formelles du formalisme TAG que nous venons de voir en font un formalisme intéressant pour la description de la langue naturelle. Dans la section qui suit, nous

allons voir que ce formalisme présente également des propriétés informatiques qui justifient son utilisation dans le cadre du traitement automatique des langues.

2.1.5 Propriétés informatiques

Algorithmes et temps d'analyse pour grammaires TAG Une propriété des TAG très importante d'un point de vue du traitement automatique est que les grammaires TAG sont analysables en un temps polynomial, avec des temps d'analyse en $\mathcal{O}n^6$ dans le pire des cas.

Les algorithmes d'analyse développés pour les grammaires hors-contextes, à savoir *Cocke Kasami Younger* et *Earley*, ont été adpatés aux grammaires TAG (Schabes et Joshi, 1988). De nouveaux algorithmes basés sur les automates à deux piles ont également été proposés (Becker, 1994; Villemonte de la Clergerie et Alonso, 1998; Alonso et al., 2000).

Analyseurs syntaxiques pour grammaires TAG Plusieurs analyseurs syntaxiques ont été développés pour les TAG. Pour l'anglais, le système de référence est XTAG (XTAG-Research-Group, 2001), développé à l'Université de Pennsylvanie. Ce système comprend un module de représentation du lexique pour une TAG lexicalisée et un analyseur syntaxique à base de *chart*, décrit dans (Sarkar, 2000). Pour le français, on distingue principalement l'analyseur LLP2 (Lopez, 1999), et le générateur d'analyseurs DyALog (Villemonte de la Clergerie, 1993) (voir également chapitre 7).

Ressources linguistiques TAG Enfin, des ressources (grammaires et lexiques au format électronique) de taille importante ont été développées pour le formalisme TAG. Nous avons cité précédemment le système XTAG incluant une grammaire de l'anglais, ainsi qu'une grammaire du coréen (hye Han et al., 2000). Pour le français, l'Université Paris 7 a également développé une grammaire de couverture importante (Abeillé et al., 1999; Abeillé, 2002). Enfin, pour l'allemand, une grammaire utilisant une extension du formalisme TAG, à savoir les TAG à composants multiples, est en cours de création à l'Université de Tübingen.

2.2 L'interface syntaxe / sémantique pour Grammaires d'Arbres Adjoints

Bien que les grammaires TAG aient été au centre de nombreuses recherches depuis leur découverte dans les années 70, leur utilisation s'est concentrée sur la description de la syntaxe de la langue naturelle. Aucun consensus ne s'est dégagé au sujet de la construction sémantique, et en particulier au sujet de la définition d'une interface entre syntaxe et sémantique. Dans cette section, nous donnons tout d'abord une vue intuitive (et introductive) de la question de la définition d'une interface syntaxe / sémantique en TAG, puis nous introduisons les principales approches concernant la construction sémantique dans le formalisme

des TAG, approches que nous pouvons classer en quatre catégories :

1. L'approche basée sur les TAG synchrones de Shieber et Schabes.
2. Les approches basées sur l'arbre de dérivation.
3. Les approches basées sur l'arbre dérivé.
4. Les approches hybrides.

2.2.1 L'intuition

Comme nous l'avons vu, les arbres élémentaires d'une grammaire TAG ont un domaine de localité étendu. Cela permet, en respectant le principe de coocurrence prédicat-arguments, de définir des arbres élémentaires encodant le cadre de sous-catégorisation[25] du prédicat qui leur est associé. Les arguments du prédicat se retrouvent donc dans l'arbre élémentaire, soit sous forme de nœud de substitution, soit sous forme de nœud pied. Cette propriété des arbres élémentaires a longtemps laissé penser que l'arbre de dérivation, qui enregistre comment ces nœuds sont réécrits lors de la dérivation d'une phrase, pouvait être interprété comme un graphe de dépendances sémantiques (*cf* (Candito et Kahane, 1998)). L'idée sous-jacente est qu'une substitution doit être interprétée comme une dépendance du prédicat lié à l'arbre inséré *par rapport au* prédicat lié au nœud site de la substitution , alors qu'une adjonction doit être interprétée comme une dépendance du prédicat lié au nœud site de l'adjonction *par rapport au* prédicat lié à l'arbre auxiliaire adjoint. Ainsi, il est possible d'extraire un graphe de dépendance à partir de l'arbre de dérivation de phrases telles que « Jean pense souvent à Marie » (voir figure 2.12).

Cependant, cette technique ne peut être généralisée, comme l'a observé (Rambow et al., 1995), en raison du *problème du lien manquant*, que nous illustrons au moyen de l'exemple suivant[26] :

(2) a. Jean prétend que Marie semble adorer les plats épicés.

L'arbre de dérivation et le graphe de dépendances de cette phrase sont donnés figure 2.13.

On voit clairement que l'interprétation de l'arbre de dérivation présentée précédemment ne permet pas de construire un graphe de dépendances correct. La raison à cela provient du fait que les arbres associés à *prétendre* et *sembler* s'adjoignent sur des nœuds distincts de l'arbre associé à *adorer*. Le lien de dépendance entre ces deux prédicats ne peut donc être construit.

Dans ce qui suit, nous allons étudier les différentes propositions de calcul sémantique compositionnel en TAG, dont le point de départ correspond à l'approche de Shieber et Schabes en 1990.

[25] Par *cadre de sous-catégorisation*, nous entendons le nombre d'arguments, la position et la catégorie des arguments du prédicat.

[26] Exemple issu de (Rambow et al., 1995).

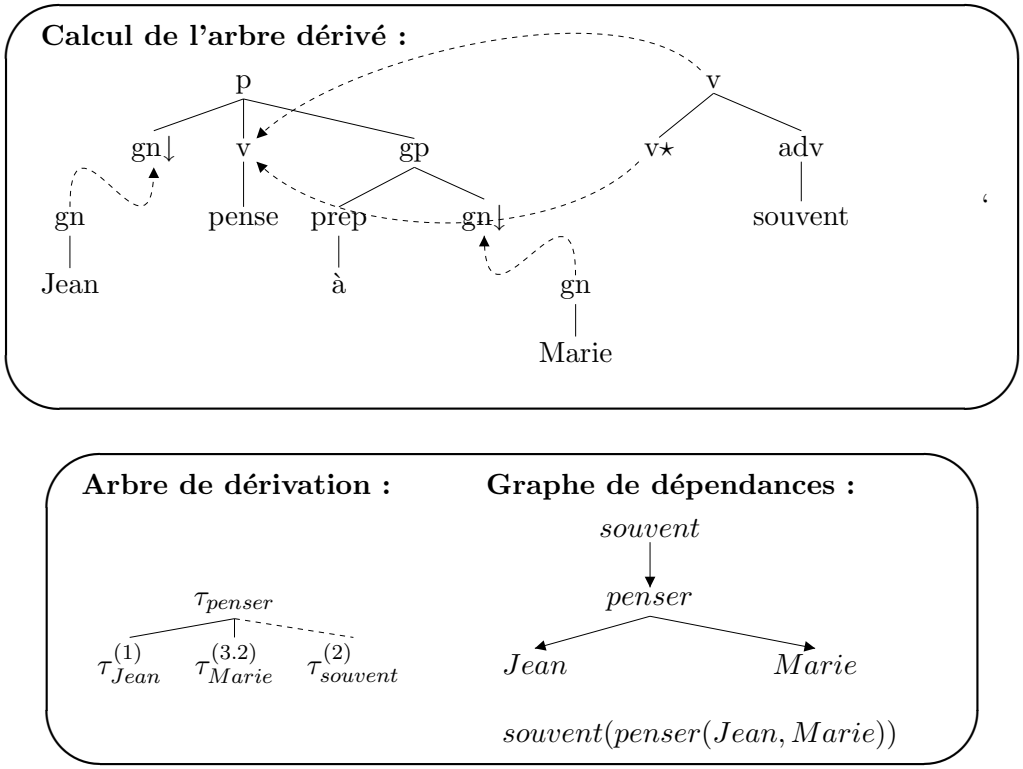


FIG. 2.12 – Arbre de dérivation et graphe de dépendances.

2.2.2 L'approche de Shieber et Schabes (1990) : utilisation de TAG synchrones

(Shieber et Schabes, 1990) propose d'utiliser les TAG synchrones pour définir une interface syntaxe / sémantique. Les TAG synchrones sont une extension des TAG, dans laquelle on ne manipule plus des arbres élémentaires, mais des paires d'arbres élémentaires. Certains nœuds de ces arbres sont liés entre eux et synchronisés par rapport aux opérations (adjonction ou substitution) qui y sont opérés. Ainsi si l'on considère deux paires d'arbres $\langle A_1, A_2 \rangle$ et $\langle B_1, B_2 \rangle$, et que B_1 est substitué sur un nœud lié de A_1 alors une substitution de B_2 sur le nœud correspondant de A_2 est déclenchée.

Dans ce contexte, l'idée de (Shieber et Schabes, 1990) est de définir une TAG synchrone dans laquelle chaque paire d'arbres élémentaires est constituée d'un arbre *syntaxique*, lié à un arbre *sémantique*. L'interface syntaxe / sémantique correspond alors à la définition des liens entre nœuds des arguments du prédicat dans chacun de ces deux arbres (voir figure 2.14).

Si nous reprenons l'exemple (2a), nous pouvons, au moyen de la grammaire synchrone présentée figures 2.14 et 2.15[27], construire un graphe de dépendances correct, comme illustré figure 2.16[28].

[27] Pour gagner de la place, nous ne représentons pas les paires d'arbres associées aux groupes nominaux.

[28] Pour mettre en relief le lien entre *prétendre* et *sembler*, seuls les liens entre nœuds des arbres sémantique

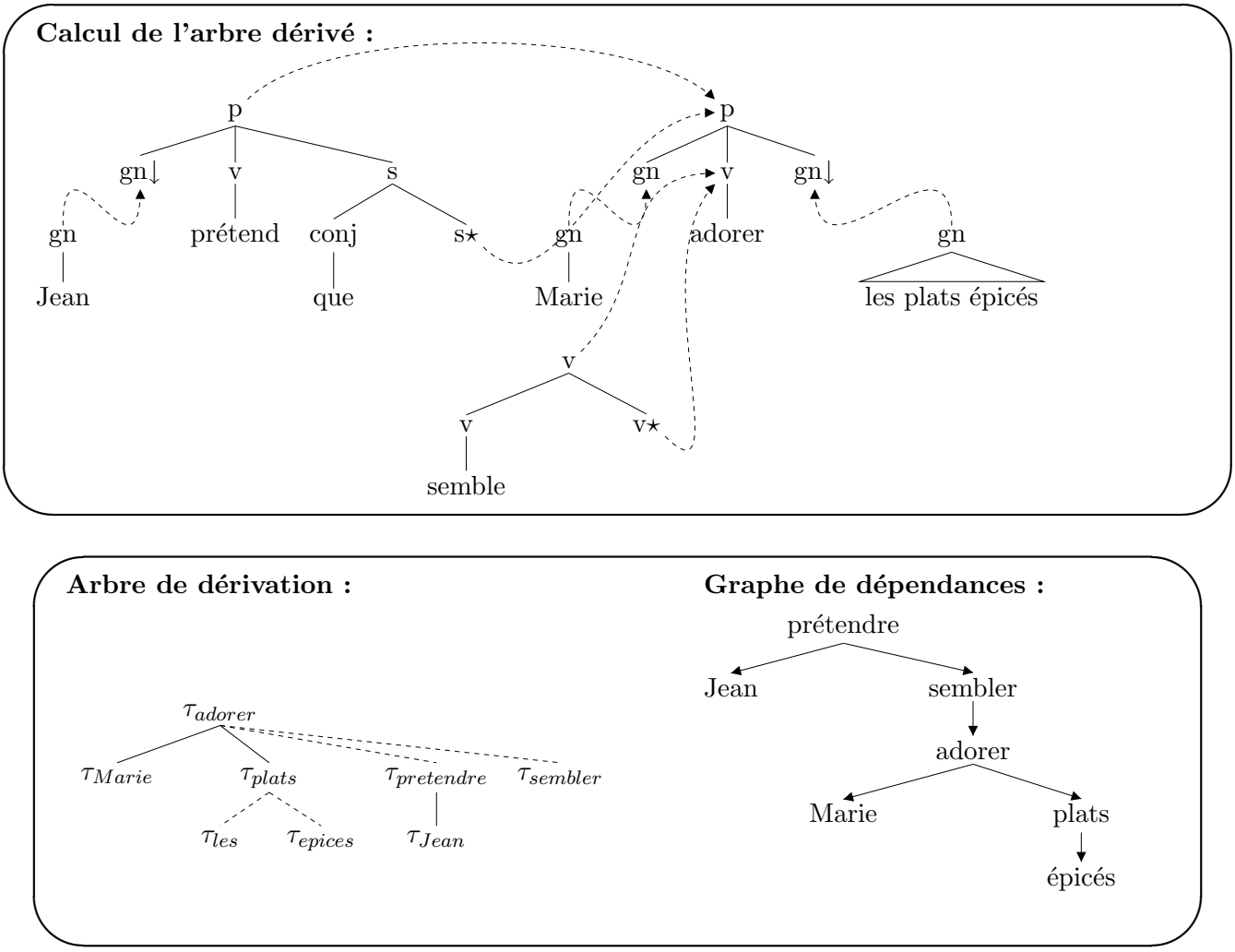


FIG. 2.13 – Arbre de dérivation et graphe de dépendances (suite).

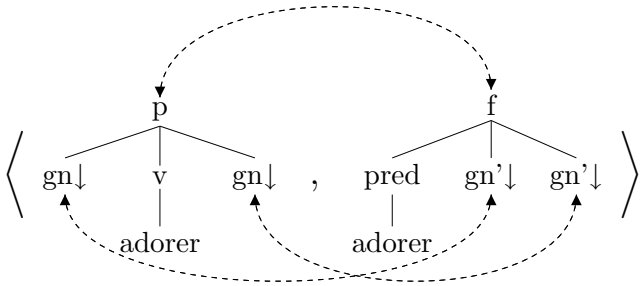


FIG. 2.14 – Interface syntaxe / sémantique dans une TAG synchrone.

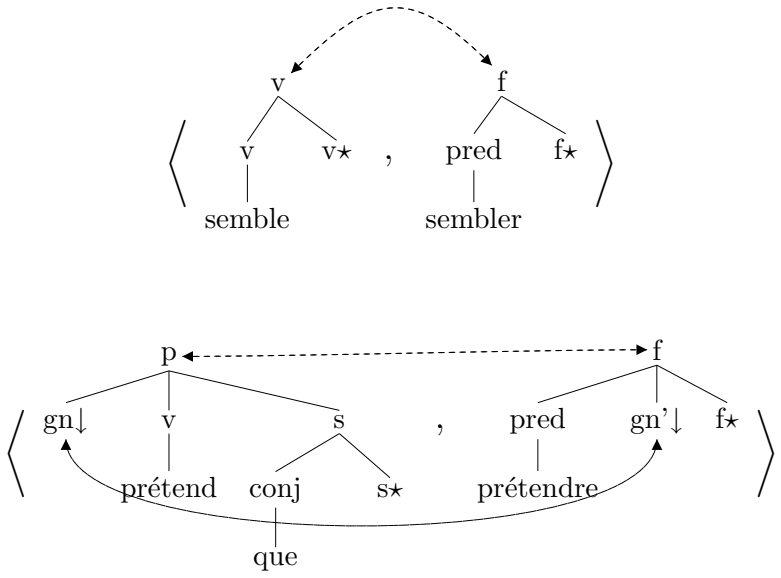


FIG. 2.15 – Grammaire TAG synchrone permettant l'analyse de « Jean prétend que Marie semble adorer les plats épicés ».

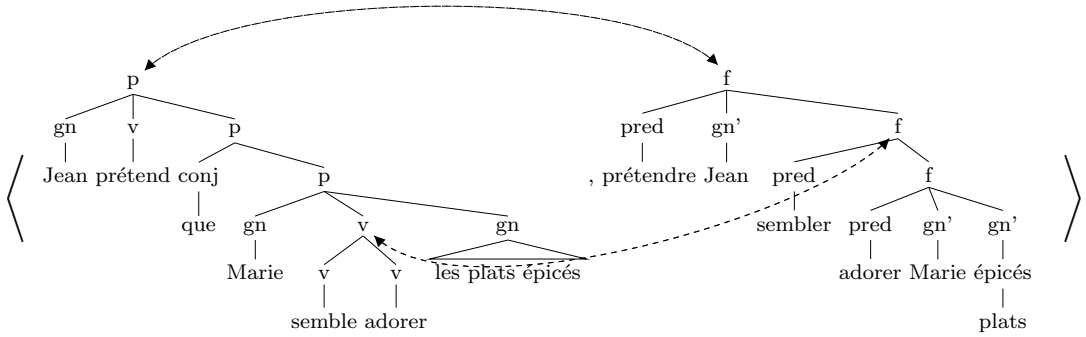


FIG. 2.16 – Composition sémantique pour « Jean prétend que Marie semble adorer les plats épicés » au moyen d'une TAG synchrone.

Ce type d'interface syntaxe / sémantique permet donc de traiter certains cas problématiques dans une approche basée exclusivement sur une interprétation de l'arbre de dérivation. La raison à cela réside dans le fait que les TAG synchrones permettent de décrire les liens manquants dans l'arbre de dérivation.

Cependant comme le notent (Shieber et Schabes, 1990), cette technique est dépendante de l'ordre des opérations de dérivation. Plus précisément, certaines TAG synchrones sont sensibles à l'ordre des dérivations, c'est-à-dire que des ordres différents entre substitutions et adjonctions peuvent produire des représentations sémantiques différentes. Cette différence peut s'interpréter sous forme de portée des constituants.

La conséquence de cela est que (a) il faut calculer toutes les dérivations possibles pour produire toutes les représentations sémantiques (puisqu'il n'y a pas de sous-spécification dans les représentations utilisées) et (b) plusieurs dérivations peuvent produire des représentations sémantiques identiques.

Notons enfin que cette technique a été revisitée dernièrement par (Nesson et Shieber, 2006) où les auteurs traitent certains cas reconnus difficiles pour TAG (portée des quantificateurs, interactions entre verbes à montée et adverbes, clauses relatives, etc). Les auteurs comparent également leur approche avec certaines des approches ci-dessous et se fixent trois objectifs : n'utiliser que les relations exprimées dans l'arbre de dérivation, représenter toutes les analyses sémantiques possibles, et ne pas modifier l'expressivité du formalisme TAG. Dans ce contexte, l'adjonction multiple sur un même nœud est utilisée pour produire les différentes représentations sémantiques.

2.2.3 Les méthodes basées sur l'arbre de dérivation

2.2.3.1 L'approche de Joshi et Vijay-Shanker (1999) : définition d'une sémantique compositionnelle basée sur l'arbre de dérivation

La première formulation d'une méthodologie pour un calcul sémantique compositionnel en TAG date de (Joshi et Vijay-Shanker, 1999). Dans cet article, les auteurs se concentrent sur les relations de type prédicat-arguments et tentent de déterminer la quantité de sous-spécification nécessaire dans ce contexte.

Le calcul sémantique proposé repose sur (a) l'association de représentations sémantiques à chaque arbre élémentaire, et (b) sur une composition de ces représentations à partir d'un parcours dirigé de l'arbre de dérivation.

(a) Représentations sémantiques élémentaires Les représentations sémantiques associées aux arbres élémentaires comportent trois parties :

1. un ensemble de variables distinguées, destinées à être partagées avec les autres représentations sémantiques élémentaires (selon la logique utilisée, ces variables peuvent dénoter des individus, des événements, etc) ;

et syntaxique pour ces prédicats sont représentés

2. une formule de logique des prédicat ;
3. des boîtes contenant les variables partagées avec des nœuds de l'arbre (ces boîtes peuvent être considérées comme des interfaces entre représentations sémantiques) .

Considérons à nouveau l'exemple (2a). Les représentations sémantiques élémentaires utilisées sont données figure 2.17.

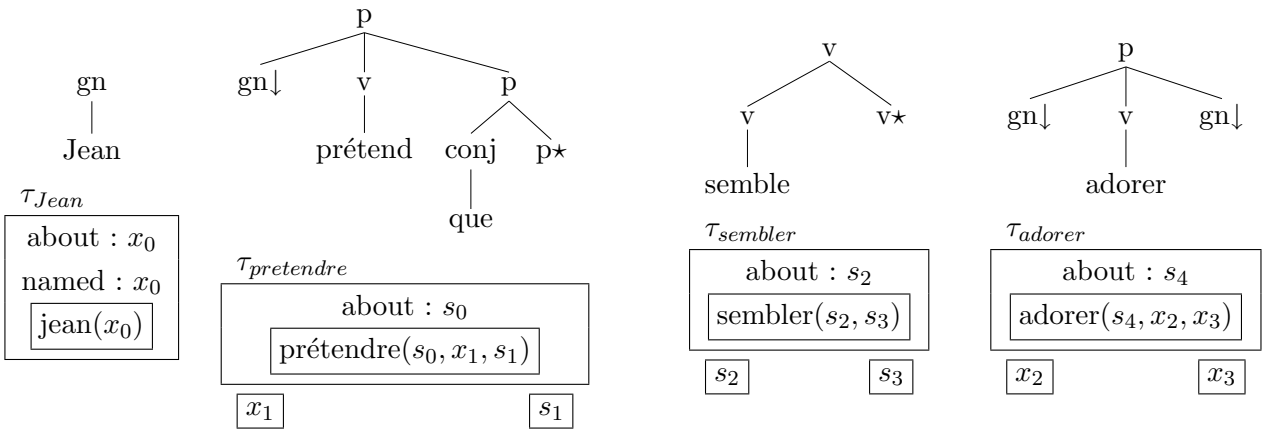


FIG. 2.17 – Représentations sémantiques élémentaires pour « Jean prétend que Marie semble adorer les plats épicés ».

Par exemple, sur la figure 2.17, la représentation sémantique associée à l'arbre de *prétendre* contient les informations suivantes :

– (variable distinguée) cette représentation traite d'une situation s_0 (champ *about*) ;
– (formule logique) cette situation est réalisée par l'individu x_1 et porte sur une situation s_1 pour le prédicat *prétendre* ;
– (variables partagées) les variables x_1 et s_1 sont partagées avec des nœuds de l'arbre élémentaire.

(b) Composition sémantique Les représentations sémantiques introduites précédemment sont composées en suivant un parcours dirigé de l'arbre de dérivation.

Lorsqu'un arc désignant une opération de substitution est traversé, les représentations sémantiques associées aux arbres sont composées comme suit : les variables distinguées de la représentation sémantique associée à l'arbre substitué sont unifiées avec les variables partagées avec le nœud site de la substitution. Ce qui entraîne la mise à jour de la représentation sémantique associée à l'arbre site de la substitution (en particulier, les variables partagées unifiées sont retirées). L'exemple de la substitution de l'arbre de *Jean* sur celui de *prétendre* est donné figure 2.18.

Lorsqu'un arc désignant une opération d'adjonction est traversé, une composition similaire a lieu, à ceci près qu'elle est ordonnée de manière inverse, *i.e.*, l'unification concerne

les variables distinguées de la représentation de l'arbre site de l'adjonction avec les variables partagées avec le nœud pied de l'arbre adjoint.

A l'issue de la dérivation, la représentation sémantique finale correspond à l'union des représentations sémantiques élémentaires (dans lesquelles les variables ont été unifiées).

Notons que le parcours de l'arbre de dérivation est un parcours ascendant avec les extensions suivantes :

- l'adjonction d'un arbre auxiliaire prédicatif est traité comme une substitution,
- en cas d'adjonction d'arbres auxiliaires prédicatifs sur des nœuds distincts d'un même arbre, ces adjonctions sont traitées de manière ordonnée : la composition des représentations liées à l'arbre s'adjoignant sur le nœud le plus bas de l'arbre dérivé est réalisée en premier.

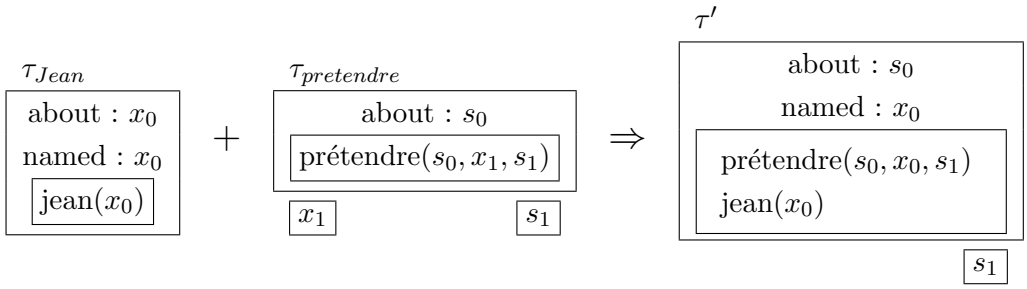


FIG. 2.18 – Composition des représentations sémantiques élémentaires.

Cette méthodologie pour la construction sémantique compositionnelle en TAG présente certains inconvénients. Tout d'abord, elle repose sur un parcours ordonné de l'arbre de dérivation et n'est donc pas facilement généralisable (cas des langues dont les structures syntaxiques diffèrent ?). De plus, cette approche s'appuie sur une composition sémantique par application d'une représentation liée à un *foncteur* sémantique sur une représentation liée à un *argument* sémantique. Cependant, les différents constituants syntaxiques ne jouent pas toujours le rôle de foncteur ou[29] d'argument. Prenons par exemple le cas des quantificateurs. Un quantificateur joue à la fois le rôle de foncteur et d'argument sémantique via sa portée et sa restriction (*cf* paragraphe suivant). Une phrase telle que « tout homme court » ne peut donc être analysée sémantiquement via ce procédé. Pour pallier à cela, une extension du procédé au moyen d'une sémantique sous-spécifiée et de grammaires TAG multi-composants a été proposée par (Kallmeyer et Joshi, 1999).

2.2.3.2 L'approche de Kallmeyer et Joshi (1999, 2003) : sémantique sous-spécifiée pour grammaires LTAG

L'approche de (Kallmeyer et Joshi, 1999; Kallmeyer et Joshi, 2003) étend celle de (Joshi et Vijay-Shanker, 1999) comme suit :

[29] ou exclusif

1. les représentations sémantiques associées aux arbres élémentaires n'utilisent plus une logique des prédicats mais une logique sous-spécifiée (en l'occurrence, la *logique des prédicats débranchés* introduite à la section 1.2.1.2 page 26) ;
2. les quantificateurs sont traités au moyen du formalisme des TAG mutli-composants[30][31]

Concernant le point 1, les représentations sémantiques manipulées ici sont constituées (i) d'un ensemble de formules logiques sous-spécifiées (utilisant des variables d'étiquettes et de trous, et de contraintes de portée), et (ii) d'un ensemble (éventuellement vide) de variables argumentales (désignant soit des arguments individuels – notés x_i – soit des arguments propositionnels – notés s_j – soit des arguments de portée – notés g_k)[32].

A titre d'exemple, les arbres élémentaires et représentations sémantiques associées permettant de couvrir la phrase « Jean pense souvent à Marie » sont donnés figure 2.19[33]. Ces représentations associent à chaque arbre une formule prédicative (pour cette exemple, aucune sous-spécification n'est nécessaire), ainsi que des arguments sémantiques (qui peuvent apparaître dans la seconde partie de la représentation s'ils doivent être renseignés par composition sémantique).

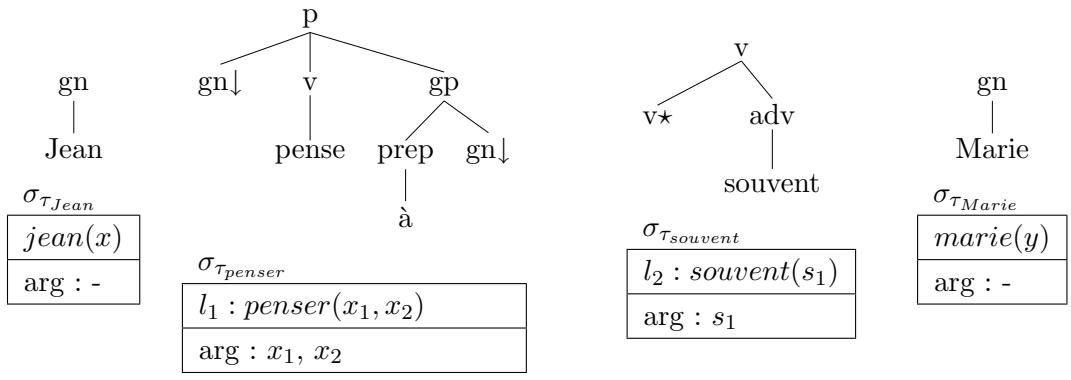


FIG. 2.19 – Grammaire TAG avec représentations sémantiques sous-spécifiées.

Concernant le point 2, comme nous l'avons annoncé, les quantificateurs ayant un double rôle sémantique, ceux-ci sont représentés par un ensemble de deux arbres syntaxiques ayant chacun une représentation sémantique associée. L'un de ces arbres représente la relation avec un argument verbal (la *portée* du quantificateur) et l'autre la relation avec un argument nominal (la *restriction* du quantificateur). L'ensemble d'arbres associé au quantificateur

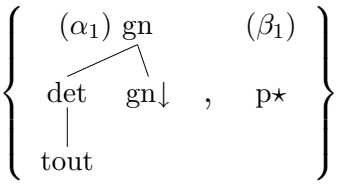
[30] Une grammaire TAG multi-composants (MC-TAG) est une grammaire dont les unités de base sont des ensembles d'arbres élémentaires TAG. De plus on parle de MC-TAG locale lorsque tous les arbres d'un ensemble se réécrivent avec un et un seul arbre d'un autre ensemble. Les MC-TAG locales ont un pouvoir génératif (faible et fort) équivalent aux TAGs, voir (Schuler et al., 2000).

[31] Cette extension était déjà évoquée dans (Joshi et Vijay-Shanker, 1999).

[32] Ces variables argumentales sont partagées avec des nœuds des arbres, bien que ce lien n'est pas représenté ici, il correspond aux adresses de Gorn des nœuds.

[33] On note σ_t la représentation sémantique associée à l'arbre t.

tout est donné figure 2.20. On note l'emploi d'un arbre auxiliaire syntaxiquement vide pour la portée du quantificateur.



σ_{α_1}

$l_3 : tout(x, h_1, h_2)$
$s_2 \leq h_2$
arg : s_2

σ_{β_1}

$l_4 : p_1(x)$
$l_4 \leq h_1$
arg : p_1

FIG. 2.20 – Ensemble d'arbres et représentations sémantiques sous-spécifiée pour les quantificateurs.

La représentation sémantique associée à l'arbre de la portée du quantificateur (arbre β_1) utilise une formule sous-spécifiée, celle associée à la restriction (arbre α_1) utilise un argument dénotant le prédicat inclus dans la portée de la variable de trou h_1 (ce prédicat correspond au nom associée au quantificateur).

Notons enfin que cet exemple utilise un arbre initial (avec nœud de substitution) pour représenter la restriction du quantificateur, ce qui ne respecte pas l'usage décrit dans (Abeillé, 2002), nous reviendrons sur ce point dans le paragraphe suivant, dans lequel nous présentons l'approche de (Kallmeyer, 2002a).

Composition sémantique Dans ce contexte, (Kallmeyer et Joshi, 1999) définit un procédé de construction sémantique compositionnelle à base d'application de représentations sémantiques.

Cette application entre représentations est guidée par l'arbre de dérivation, et plus particulièrement par le type d'opération (adjonction ou substitution) décrite par une branche de cet arbre :

- Dans le cas d'une substitution d'un arbre t_1 sur un arbre t_2, on applique la représentation σ_{t_2} à σ_{t_1}.
- Dans le cas d'une adjonction d'un arbre t_1 sur un arbre t_2, on applique la représentation σ_{t_1} à σ_{t_2}.

L'application d'une représentation σ_A sur une représentation σ_B correspond à l'assignation des variables (individuelles ou propositionnelles) de σ_B aux arguments de σ_A (l'assignation entre arguments et variables utilise l'adresse de Gorn pour déterminer quelle variable réfère à quel argument).

Si, nous prenons l'exemple « Jean pense souvent à Marie » (l'arbre de dérivation cor-

respondant est donné figure 2.21), la construction sémantique fait intervenir trois applications[34] :

- la représentation $\sigma_{\tau_{penser}}$ est appliquée à la représentation $\sigma_{\tau_{Jean}}$, ce qui résulte en l'assignation de la variable individuelle x à l'argument x_1 (x_1 étant liée au nœud site de la substitution et x correspondant à la variable introduite par l'arbre substitué),
- la représentation $\sigma_{\tau_{penser}}$ est appliquée à $\sigma_{\tau_{Marie}}$, ce qui provoque l'assignation de la variable individuelle y à l'argument x_2 (pour les mêmes raisons),
- enfin, la représentation $\sigma_{\tau_{souvent}}$ est appliquée à $\sigma_{\tau_{penser}}$, au cours de cette opération, la variable d'étiquette l_1 est assignée à l'argument s_1, ce qui produit la représentation finale suivante :

$l_1 : penser(x, y), jean(x), marie(y), l_2 : souvent(l_1)$
arg : -

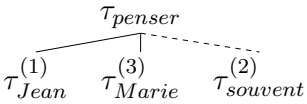


FIG. 2.21 – Arbre de dérivation de la phrase « Jean pense souvent à Marie ».

En ce qui concerne les quantificateurs, prenons l'exemple de la phrase « tout homme aime une femme ». La dérivation d'un groupe nominal quantifié tel que « tout homme aime » (dont la dérivation est représentée figure 2.22) provoque la composition des représentations sémantiques associées comme suit :

- la représentation σ_{β_1} est appliquée à $\sigma_{\tau_{aimer}}$ (adjonction de l'arbre auxiliaire vide), ce qui provoque l'assignation de la variable d'étiquette l_5 à l'argument s_2,
- la représentation σ_{α_1} est appliquée à $\sigma_{\tau_{homme}}$ (substitution du groupe nominal), ce qui complète la représentation σ_{α_1} de la façon suivante (le prédicat *homme* est assigné à l'argument p_1) :

$l_4 : homme(x), l_4 \leq h_1$
arg : -

- la représentation $\sigma_{\tau_{aimer}}$ est appliquée à σ_{α_1}, ce qui provoque l'assignation de la variable du prédicat *homme*, en l'occurrence x, à l'argument du prédicat *aimer*, $\sigma_{\tau_{aimer}}$ devient :

$l_3 : tout(x, h_1, h_2), l_4 : homme(x), l_5 : aimer(x, x_2), l_4 \leq h_1, l_5 \leq h_2$
arg : x_2

[34]Notons que l'ordre entre ces applications n'a pas d'effet sur la représentation construite.

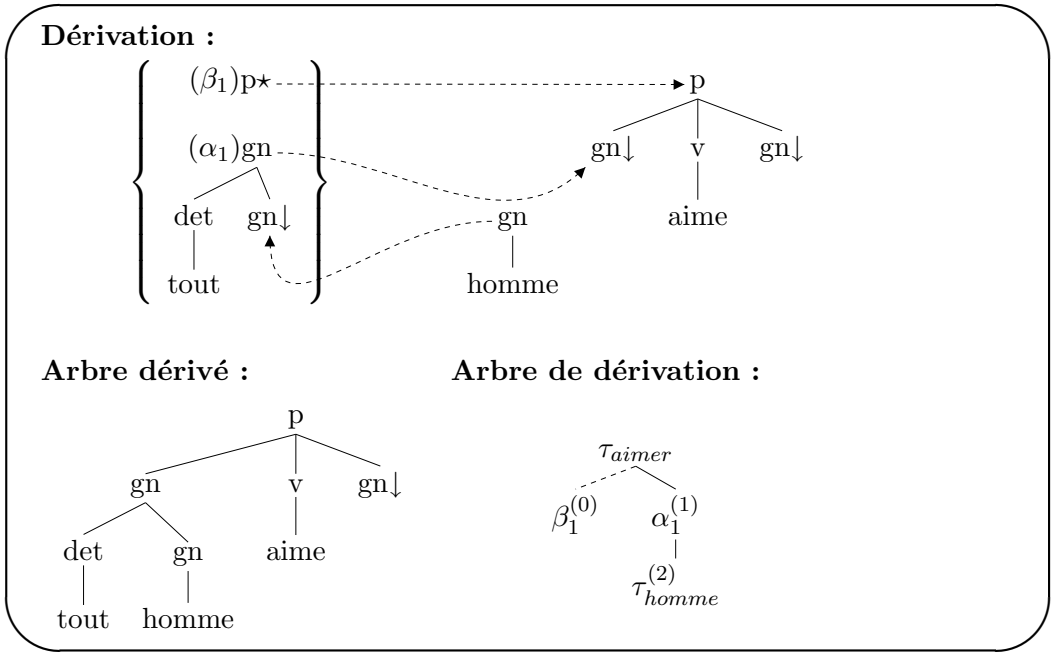


FIG. 2.22 – Dérivation de « tout homme aime ».

Le traitement du groupe nominal objet, à savoir « une femme », est réalisée de manière similaire à celui du groupe nominal sujet, c'est-à-dire que l'arbre lié à la *portée* du quantificateur *une* est adjoint à la racine de l'arbre τ_{aimer}[35], et l'arbre lié à la *restriction* y est substitué (cet arbre lié à la *restriction* aura reçu la substitution de l'arbre τ_{femme}). A l'issue de la composition sémantique (*i.e.* des applications de représentations), la représentation finalement construite est la suivante :

$l_3 : tout(x, h_1, h_2), l_4 : homme(x), l_5 : aimer(x, x_2), l_6 : une(y, h_3, h_4), l_7 : femme(y),$ $l_4 \leq h_1, l_5 \leq h_2, l_7 \leq h_3, l_5 \leq h_4$
arg : -

A partir de la représentation sous-spécifiée ainsi calculée, il est possible de déterminer toutes les représentations sémantiques, comme nous l'avons vu en section 1.2.1.2.

L'approche que nous venons de présenter distingue les relations de type prédicat-arguments des relations de portée au moyen de grammaires TAG multi-composants, ce qui lui permet de n'introduire qu'un niveau limité de sous-spécification dans les représentations sémantiques élémentaires. Ce niveau de sous-spécification permet de représenter les ambiguïtés de portée, et de traiter sémantiquement certains phénomènes linguistiques complexes tels que les modificateurs multiples et contraintes d'îlots[36].

[35]Ce qui signifie que ce procédé de construction sémantique autorise l'adjonction multiple sur un nœud donné, or nous avons vu que ce n'est pas le cas habituellement en TAG.

[36]Nous ne détaillons pas le traitement de ces phénomènes ici, le lecteur est renvoyé à (Kallmeyer et Joshi,

Cependant, comme nous allons le voir en section 2.2.5.1, cette approche se révèle inadaptée à l'analyse de la quantification par adjonction.

2.2.3.3 L'approche de Pogodalla (2004) : interprétation de l'arbre de dérivation au moyen d'une Grammaire Catégorielle Abstraite

Dans l'approche de (Pogodalla, 2004a), l'auteur se place dans la tradition consistant à utiliser l'arbre de dérivation comme structure de base pour la construction sémantique. Plus précisément, l'auteur propose d'utiliser les Grammaires Catégorielles Abstraites (*Abstract Categorial Grammars – ACG*) pour définir un lien entre arbre de dérivation TAG et formule sémantique sous-spécifiée.

Notre introduction à cette approche se fera en trois temps :

(A) dans un premier temps, nous définissons formellement les ACG (*cf* (de Groote, 2001)),

(B) ensuite, nous montrons comment représenter le lien entre arbre dérivé et arbre de dérivation TAG au moyen d'une ACG (*cf* (de Groote, 2002)), cette présentation a pour but de nous familiariser avec les ACG dans le contexte des TAG, avant de les appliquer à la construction sémantique,

(C) enfin nous appliquons les ACG à la sémantique à trous de (Bos, 1995) (*cf* (Pogodalla, 2004a)).

A. Les Grammaires Catégorielles Abstraites Initialement introduites par (de Groote, 2001), les Grammaires Catégorielles Abstraites sont un formalisme basé sur les grammaires logiques de types (*Type Logical Grammars*). L'intérêt des ACG par rapport à ces grammaires, est qu'elles permettent de représenter deux langages manipulés au moyen des mêmes primitives, on peut donc passer d'un langage à l'autre (le formalisme est réversible). Ces langages peuvent, par exemple, représenter une description syntaxique et une description sémantique. Dans la version de (de Groote, 2001), ces deux langages, que l'on appelle *langage abstrait* et *langage objet*, prennent la forme d'ensembles de λ-termes linéaires typés[37].

Dans un premier temps, nous introduisons les objets mathématiques manipulés par les ACG, ce qui nous permet par la suite de donner une définition des langages abstrait et concret d'une ACG.

Définition 31 (Type linéaire implicatif)**.** *Soit A un ensemble de types atomiques. L'ensemble $\mathscr{T}(A)$ des types linéaires implicatifs construit à partir de A sont définis inductivement comme suit :*

1. *si $a \in A$, alors $a \in \mathscr{T}(A)$;*
2. *si $\alpha, \beta \in \mathscr{T}(A)$, alors $(\alpha \multimap \beta) \in \mathscr{T}(A)$.*

1999) pour une présentation complète.

[37] Voir (Girard, 1987) pour une introduction à la logique linéaire.

Définition 32 (Signature linéaire d'ordre supérieur)**.** *Une signature linéaire d'ordre supérieur consiste en un triplet $\Sigma = \langle A, C, \tau \rangle$ tel que :*

1. *A est un ensemble fini de types atomiques ;*
2. *C est un ensemble fini de constantes ;*
3. *$\tau : C \to \mathscr{T}(A)$ est une fonction assignant à chaque constante de C un type linéaire implicatif de $\mathscr{T}(A)$.*

Définition 33 (Ensemble des λ-termes linéaires)**.** *L'ensemble des λ-termes linéaires $\Lambda(\Sigma)$ construits à partir d'une signature $\Sigma = \langle A, C, \tau \rangle$ est défini inductivement comme suit (on note X un ensemble infini dénombrable de λ-variables) :*

1. *si $c \in C$, alors $c \in \Lambda(\Sigma)$;*
2. *si $x \in X$, alors $x \in \Lambda(\Sigma)$;*
3. *si $x \in X, t \in \Lambda(\Sigma)$, et si x apparaît comme variable libre exactement une fois dans t, alors $(\lambda x.t) \in \Lambda(\Sigma)$;*
4. *si $t, u \in \Lambda(\sigma)$, et que les ensembles de variables libres de t et u sont disjoints, alors $(tu) \in \Lambda(\Sigma)$.*

Définition 34 (Typage de λ-termes linéaires)**.** *Etant donné une signature linéaire d'ordre supérieur Σ, il est possible de typer chaque λ-terme linéaire de $\Lambda(\Sigma)$ au moyen d'un type linéaire implicatif. Ce typage obéit à un système d'inférence dont les règles utilisent des séquents de la forme :*

$$\Gamma \vdash_\Sigma t : \alpha$$

où :

1. *Γ est un ensemble fini de déclarations de typage de λ-variables, de la forme $x : \beta$ (avec $x \in X$ et $\beta \in \mathscr{T}(A)$), tel que chaque variable est déclarée au plus une fois ;*
2. *$t \in \Lambda(\Sigma)$;*
3. *$\alpha \in \mathscr{T}(A)$.*

Les axiomes et règles d'inférence pour la signature Σ sont :

$$\vdash_\Sigma c : \tau(c) \ \textit{(cons)}$$

$$x : \alpha \vdash_\Sigma x : \alpha \ \textit{(var)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.t) : (\alpha \multimap \beta)} \ \textit{(abs)}$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \multimap \beta) \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (tu) : \beta} \ \textit{(app)}$$

Définition 35 (Lexique)**.** *Etant donnés deux signatures linéaires d'ordre supérieur* $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *et* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$*, on appelle lexique* $\mathscr{L} : \Sigma_1 \rightarrow \Sigma_2$ *une réalisation de* Σ_1 *dans* Σ_2*. Cette réalisation correspond à une fonction d'interprétation (a) des types atomiques de* Σ_1 *comme types de* Σ_2*, et (b) des constantes de* Σ_1 *comme* λ*-termes linéaires de* Σ_2*. Formellement,* $\mathscr{L}$ *est défini comme la paire* $\langle F, G \rangle$ *telle que :*

1. $F : A_1 \rightarrow \mathscr{T}(A_2)$ *est une fonction interprétant les types atomiques de* Σ_1 *comme types linéaires implicatifs construits sur* A_2 *;*
2. $G : C_1 \rightarrow \Lambda(\Sigma_2)$ *est une fonction interprétant les constantes de* Σ_1 *comme* λ*-termes linéaires de* Σ_2 *;*
3. *ces fonctions d'interprétation respectent les relations de typage, i.e., pour tout* $c \in C_1$*, nous pouvons inférer :*

$$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c))$$

où $\hat{F}$ *est l'unique extension homomorphe de* F*.*

A présent, nous sommes en mesure de définir ce qu'est une ACG :

Définition 36 (ACG)**.** *Une Grammaire Catégorielle Abstraite est un quadruplet* $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, s \rangle$ *où :*

1. Σ_1 *et* Σ_2 *sont deux signatures linéaires d'ordre supérieur, appelées respectivement vocabulaire abstrait et vocabulaire objet ;*
2. $\mathscr{L} : \Sigma_1 \rightarrow \Sigma_2$ *est un lexique du vocabulaire abstrait vers le vocabulaire objet ;*
3. s *est un type atomique du vocabulaire abstrait, appelé type distingué.*

Définition 37 (Langage abstrait)**.** *Le langage abstrait généré par une* ACG $\mathscr{G}$*, noté* $\mathcal{A}(\mathscr{G})$*, est défini comme suit :*

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \| \vdash_{\Sigma_1} t : s \text{ peut être inféré}\}$$

($\mathcal{A}(\mathscr{G})$ *est l'ensemble des* λ*-termes linéaires clos construits à partir du vocabulaire abstrait* Σ_1*, et de type* s*).*

Définition 38 (Langage objet)**.** *Le langage langage objet généré par une* ACG $\mathscr{G}$*, noté* $\mathcal{O}(\mathscr{G})$*, est défini comme suit :*

$$\mathcal{O}(\mathscr{G}) = \{t \in \Lambda(\Sigma_2) \mid \exists u \in \mathcal{A}(\mathscr{G}) \text{ tel que } t = \mathscr{L}(u)\}$$

A partir de la définition d'une ACG donnée ici, nous allons à présent voir comment encoder le lien entre arbre de dérivation et arbre dérivé TAG dans le formalisme des ACG.

B. Représentation du lien entre arbre de dérivation et arbre dérivé TAG au moyen d'une ACG L'encodage d'une grammaire TAG via une ACG présenté ici provient de (de Groote, 2002). L'idée sous-jacente à cet encodage est de définir une ACG $\mathscr{G}$ dont le langage abstrait permet de représenter des arbres de dérivations d'une grammaire TAG, et dont l'interprétation au moyen du lexique $\mathscr{L}$ de cette ACG permet de représenter les arbres dérivés dans le langage objet. Notons que (de Groote, 2002) ne s'arrête pas là, puisqu'il propose d'utiliser le langage objet utilisé pour représenter les arbres dérivés comme langage abstrait d'une seconde ACG $\mathscr{G}'$, dont le langage objet permettrait de représenter les phrases du langage couvert par la grammaire TAG (voir figure 2.23). Nous ne présentons pas cette seconde ACG $\mathscr{G}'$ ici.

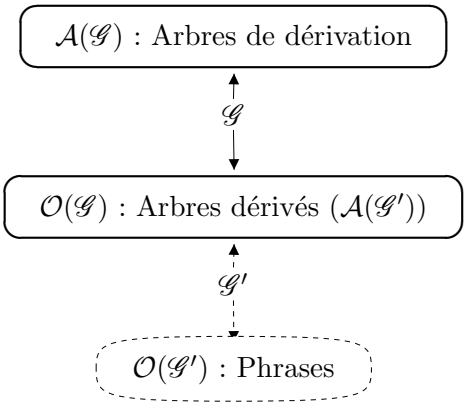


FIG. 2.23 – Encodage d'une grammaire TAG sous forme d'ACG.

La méthodologie d'encodage d'une grammaire TAG $G = \langle NT, T, p, I, A\rangle$ (où NT désigne l'ensemble des symboles non-terminaux, T l'ensemble des symboles terminaux, p le symbole distingué, I l'ensemble des arbres initiaux, et A celui des arbres auxiliaires) en une ACG $\mathscr{G}^G = \langle \Sigma_1^G, \Sigma_2^G, \mathscr{L}^G, s^G\rangle$ est la suivante :

(i) Définition de Σ_1. On définit la signature $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ à partir de laquelle les arbres de dérivation de G seront construits (voir paragraphe Exemple *infra*).
Tout d'abord, nous définissons, pour chaque symbole $X \in NT$, deux types atomiques X_S et X_A (où S et A représentent respectivement *Substitution* et *Adjonction*). Ainsi A_1 correspond aux types linéaires implicatifs construits à partir des types atomiques de $\bigcup_{X\in NT}\{X_S, X_A\}$. Puis, nous associons à chaque arbre élémentaire t de G une constante c_t. De plus, pour chaque symbole $X \in NT$, nous définissons également une constante I_X.

$$C_1 = \{c_t \mid t \in I \cup A\} \cup \{I_X \mid X \in NT\}$$

Enfin les constantes de C_1 sont typées par la fonction τ_1 de la manière suivante :

a) Soit t un arbre *initial* de G et dont le nœud racine est étiqueté par le symbole non-terminal α, les nœuds internes (nœuds d'adjonction) par les symboles non-terminaux

$\gamma_1, \ldots, \gamma_m$, et les nœuds de substitution par $\beta_1, \ldots, \beta_n$, on a alors[38] :

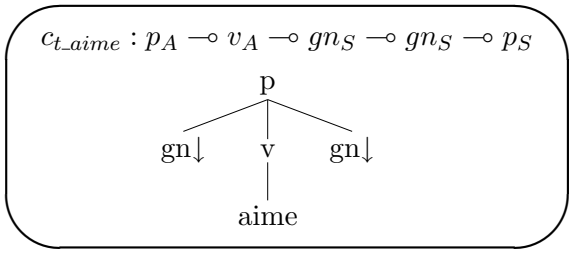
$$c_t : \gamma_{1A} \multimap \ldots \multimap \gamma_{mA} \multimap \beta_{1S} \multimap \ldots \multimap \beta_{nS} \multimap \alpha_S$$

b) Soit t est un arbre *auxiliaire* de G de nœud racine d'étiquette α, de nœuds internes d'étiquettes $\gamma_1, \ldots, \gamma_m$, et de nœuds de substitution d'étiquettes $\beta_1, \ldots, \beta_n$, nous avons alors :

$$c_t : \gamma_{1A} \multimap \ldots \multimap \gamma_{mA} \multimap \beta_{1S} \multimap \ldots \multimap \beta_{nS} \multimap \alpha_A \multimap \alpha_A$$

c) Soit X est un symbole non-terminal de G, alors nous avons $I_X : X_A$.

Ainsi, nous avons défini la signature Σ_1 utilisé par le langage abstrait de $\mathscr{G}^G$.
Pour illustrer cette définition de Σ_1, voici un exemple de constante de C_1, à savoir celle associée à l'arbre *aime* avec arguments sous forme canonique, et son type associé :



(ii) Définition de Σ_2. On définit la signature $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ à partir de laquelle nous représenterons les arbres dérivés de la grammaire G.

Nous définissons un unique type atomique τ (désignant un arbre). A_2 est alors l'ensemble des types linéaires implicatifs construits sur le singleton $\{\tau\}$.
L'ensemble des constantes C_2 est constitué comme suit :

1. Pour chaque symbole X de T (*i.e.*, X est un symbole terminal de G), nous définissons une constante c_X ;
2. Pour chaque symbole α de NT (*i.e.*, α est un symbole non-terminal de G), nous définissons les constantes $\alpha_1, \ldots, \alpha_k$ où k est le nombre maximal de fils d'un nœud étiqueté α dans G.

La fonction de typage τ_2 est alors définie comme suit :

1. Pour chaque symbole X de T, nous avons : $X : \tau$;
2. Pour chaque symbole α_i tel que $\alpha \in NT$ et i représente le nombre maximum de fils d'un nœud étiqueté par α, nous avons :

$$\alpha_i : \underbrace{\tau \multimap \ldots \multimap \tau}_{ifois} \multimap \tau$$

A présent, nous avons défini la signature Σ_2 utilisée par le langage objet de $\mathscr{G}^G$.

[38] L'ordre entre ces nœuds correspond à un parcours ordonné de l'arbre, *e.g.* parcours en largeur.

(iii) Définition de $\mathscr{L}^G$. Nous pouvons alors définir le lexique $\mathscr{L}^G$ permettant d'interpréter (a) chaque type atomique de A_1 comme un type linéaire implicatif de $\mathscr{T}(A_2)$, et (b) chaque constante de C_1 comme un λ-terme linéaire de $\Lambda(\Sigma_2)$.
Concernant (a), dans notre cas, le lexique réalise l'interprétation de types suivante :

1. $X_S : \tau$
2. $X_A : \tau \multimap \tau$

pour tout $X \in NT$. Ce qui nous permet d'interpréter chaque constante de C_1 en fonction de son type.
Concernant (b), l'interprétation des constantes de C_1 (associées aux arbres élémentaires de G) correspond à un λ-terme linéaire représentant la structure de l'arbre élémentaire. Plus précisément :

- chaque nœud interne est associé à une λ-variable de second ordre, ce qui permet de représenter les adjonctions possible sur un tel nœud,
- chaque nœud feuille par une λ-variable de premier ordre (ou par une constante s'il s'agit d'un symbole terminal).
- le λ-terme linéaire est alors construit à partir d'un parcours de l'arbre élémentaire correspondant à la constante (la dominance entre nœuds étant interprétée en termes d'application fonctionnelle).

En reprenant l'exemple de la constante c_{t_aime} lié à l'arbre *aime*, nous obtenons l'interprétation donnée figure 2.24.

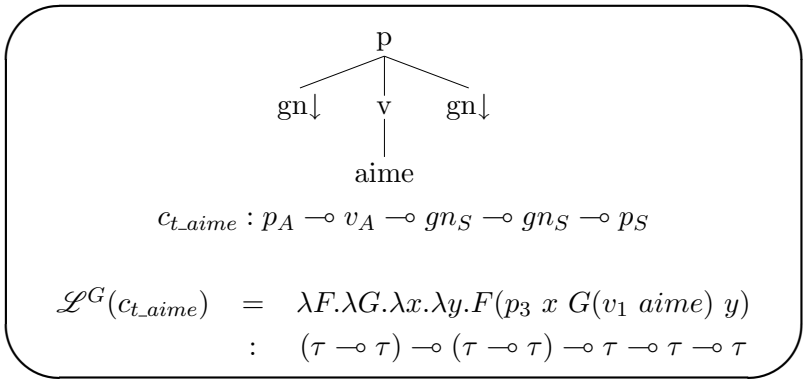


FIG. 2.24 – Exemple d'interprétation d'une constante du langage abstrait par le lexique $\mathscr{L}^G$.

Notons enfin que les variables de second ordre introduite ici devront être éliminées lors de la production des arbres dérivés, pour réaliser cela, nous utilisons les constantes I_X introduit dans le langage abstrait, qui sont interprétées par la fonction d'identité (*cf* exemple de la figure 2.27 page 2.27) :

$$\mathscr{L}^G(I_X) = \lambda x.x$$

(iv) Définition de s^G. Le type distingué s^G correspond au type atomique p_S (p étant le symbole distingué de la grammaire TAG G).

Via **(i)**, **(ii)**, **(iii)** et **(iv)**, nous avons défini une ACG $\mathscr{G}^G$ permettant d'encoder une grammaire TAG G.

Exemple. A présent, nous allons voir un exemple d'interprétation d'un arbre de dérivation exprimé dans le langage abstrait $\mathcal{A}(\mathscr{G}^G)$ en un arbre dérivé représenté dans le langage objet $\mathcal{O}(\mathscr{G}^G)$.

Considérons la grammaire TAG représentée figure 2.25. Cette grammaire permet d'analyser la phrase « tout homme aime une femme ».

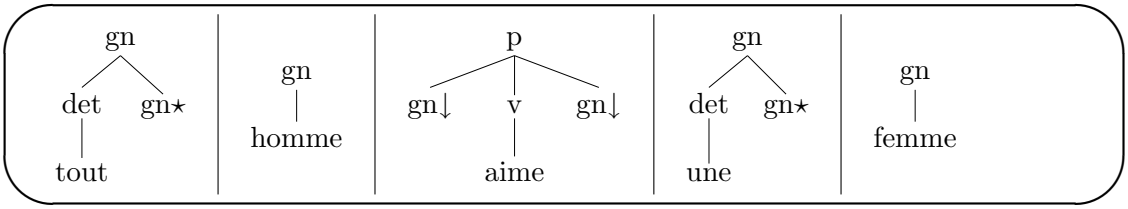


FIG. 2.25 – Grammaire TAG permettant d'analyse « tout homme aime une femme ».

Dans un premier temps, définissons la signature Σ_1 de l'ACG permettant d'encoder cette grammaire. Comme nous l'avons vu précédemment, cela revient à définir les types atomiques à partir desquels les types linéaires implicatifs seront construits, puis définir les constantes et la fonction de typage qui leur est associée. Dans le cas présent, les types atomiques considérés sont les suivants :

$$A_1 = \{p_A,\ v_A,\ gn_A,\ det_A,\ p_S,\ v_S,\ gn_S,\ det_S\}$$

Les constantes associées aux arbres élémentaires sont :

$$C_1 = \{c_{t\_tout},\ c_{t\_homme},\ c_{t\_aime},\ c_{t\_une},\ c_{t\_femme},\ I_{gn},\ I_{det},\ I_v,\ \ _p\}$$

et sont typées par la fonction τ_1 définie comme suit :

$$\begin{aligned}
\tau_1 :\ \mathbf{C_1} &\rightarrow \mathscr{T}(\mathbf{A_1}) \\
c_{t_tout} &\mapsto det_A \multimap gn_A \multimap gn_A \\
c_{t_homme} &\mapsto gn_A \multimap gn_S \\
c_{t_aime} &\mapsto p_A \multimap v_A \multimap gn_S \multimap gn_S \multimap p_S \\
c_{t_une} &\mapsto det_A \multimap gn_A \multimap gn_A \\
c_{t_femme} &\mapsto gn_A \multimap gn_S \\
I_{gn} &\mapsto gn_A \\
I_{det} &\mapsto det_A \\
I_v &\mapsto v_A \\
I_p &\mapsto p_A
\end{aligned}$$

Concernant la signature Σ_2, l'ensemble des types atomiques ne contient que le type τ : $A_2 = \{\tau\}$. Les constantes de C_2 sont les suivantes[39] :

$$C_2 = \{c_{tout},\ c_{homme},\ c_{aime},\ c_{une},\ c_{femme},\ p_1,\ p_2,\ p_3,\ gn_1,\ gn_2,\ v_1,\ det_1\}$$

Ces constantes sont typées par la fonction τ_2 définie comme suit :

$$\begin{array}{rrcl} \tau_{\mathbf{2}} : & \mathbf{C_2} & \rightarrow & \mathscr{T}(\mathbf{A_2}) \\ & c_{tout} & \mapsto & \tau \\ & c_{homme} & \mapsto & \tau \\ & c_{aime} & \mapsto & \tau \\ & c_{une} & \mapsto & \tau \\ & c_{femme} & \mapsto & \tau \\ & p_1 & \mapsto & \tau \multimap \tau \\ & p_2 & \mapsto & \tau \multimap \tau \multimap \tau \\ & p_3 & \mapsto & \tau \multimap \tau \multimap \tau \multimap \tau \\ & gn_1 & \mapsto & \tau \multimap \tau \\ & gn_2 & \mapsto & \tau \multimap \tau \multimap \tau \\ & v_1 & \mapsto & \tau \multimap \tau \\ & det_1 & \mapsto & \tau \multimap \tau \end{array}$$

Nous pouvons alors définir le lexique $\mathscr{L}$ permettant de réaliser la correspondance entre λ-termes linéaires de $\Lambda(\Sigma_1)$ et λ-termes linéaires de $\Lambda(\Sigma_2)$. Pour définir $\mathscr{L}$, il faut définir l'interprétation des types atomiques de A_1[40] :

$$\begin{array}{rcl} \mathscr{L}(X_A) & = & \tau \multimap \tau \\ \mathscr{L}(X_S) & = & \tau \end{array}$$

puis la réalisation des constantes de C_1 :

$$\begin{array}{rcl} \mathscr{L}(I_X) & = & \lambda x.x : \tau \multimap \tau \\ \mathscr{L}(c_{tout}) & = & \lambda F.\lambda G.\lambda x.F(gn_2\ G(det_1\ tout)\ x) : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \\ \mathscr{L}(c_{homme}) & = & \lambda F.F(gn_1\ homme) : \tau \multimap \tau \\ \mathscr{L}(c_{aime}) & = & \lambda F.\lambda G.\lambda x.\lambda y.F(p_3\ x\ G(v_1\ aime)\ y) : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau \\ \mathscr{L}(c_{une}) & = & \lambda F.\lambda G.\lambda x.F(gn_2\ G(det_1\ une)\ x) : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \\ \mathscr{L}(c_{femme}) & = & \lambda F.F(gn_1\ femme) : \tau \multimap \tau \end{array}$$

Enfin, le type distingué dans notre cas est p_S.

Si nous considérons alors l'arbre de dérivation de la phrase « tout homme aime une femme » donné figure 2.26, nous pouvons extraire de cet arbre de dérivation un λ-terme linéaire de $\Lambda(\Sigma_1)$ de la forme suivante[41] :

$$c_{t\_aime}\ (\ c_{t\_homme}\ c_{t\_tout}\ )\ (\ c_{t\_femme}\ c_{t\_une}\ )$$

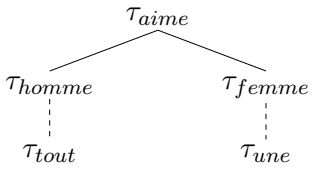


FIG. 2.26 – Arbre de dérivation de « tout homme aime une femme ».

En calculant, au moyen du lexique $\mathscr{L}$, la réalisation

$$\mathscr{L}(c_{t\_aime}\ (\ c_{t\_homme}\ c_{t\_tout}\ )\ (\ c_{t\_femme}\ c_{t\_une}\ ))$$

nous devons trouver un λ-terme linéaire de $\Lambda(\Sigma_2)$ représentant l'arbre dérivé correspondant. Cependant, comme évoqué précédemment, les nœuds internes des arbres élémentaires τ_{aime}, τ_{tout} et τ_{une} entraînent la présence de variables de second ordre dans les λ-termes linéaires correspondants de $\Lambda(\Sigma_2)$. Ces variables sont éliminées par l'utilisation des constantes abstraites I_X, via $\mathscr{L}(I_X) = \lambda x.x$. Finalement, nous calculons donc :

$$\mathscr{L}(c_{t\_aime}\ I_p\ I_v\ (\ c_{t\_homme}\ c_{t\_tout}\ I_{gn}\ I_{det}\ )\ (\ c_{t\_femme}\ c_{t\_une}\ I_{gn}\ I_{det}\ ))$$

or, nous avons les réductions suivantes :

$$\begin{aligned}
\mathscr{L}(c_{aime}\ I_p\ I_v) &= \lambda F.\lambda G.\lambda x.\lambda y.F(p_3\ x\ G(v_1\ aime)\ y)\ (\lambda p.p)\ (\lambda gn.gn)\\
\mathscr{L}(c_{aime}\ I_p\ I_v) &= \lambda x.\lambda y.(p_3\ x\ (v_1\ aime)\ y)\\
\mathscr{L}(c_{tout}\ I_{gn}\ I_{det}) &= \lambda F.\lambda G.\lambda x.F(gn_2\ G(det_1\ tout)\ x)\ (\lambda gn.gn)\ (\lambda det.det)\\
\mathscr{L}(c_{tout}\ I_{gn}\ I_{det}) &= \lambda x.(gn_2\ (det_1\ tout)\ x)\\
&\ etc.
\end{aligned}$$

ce qui permet directement de calculer l'expression réduite suivante :

$$(p_3\ (gn_2\ (det_1\ tout)\ (gn_1\ homme))\ (v_1\ aime)\ (gn_2\ (det_1\ une)\ (gn_1\ femme)))$$

que l'on peut représenter graphiquement comme illustré figure 2.27. Il s'agit de l'arbre dérivé.

C. Représentation du lien entre arbre de dérivation et sémantique à trous au moyen d'une ACG Dans ce contexte, (Pogodalla, 2004a) propose de réutiliser le même langage abstrait que précédemment pour encoder les arbres de dérivation de la grammaire TAG, et d'utiliser un langage objet représentant des formules logiques sous-spécifiées à la (Bos, 1995)[42] (voir figure 2.28).

[39] Rappelons que pour chaque symbole non-terminal α, nous introduisons les constantes $\alpha_1, \ldots, \alpha_i$ où i représente l'arité maximale d'un nœud étiqueté par α.

[40] X réfère ici à un symbole non-terminal de la grammaire G.

[41] L'extraction se fait en interprétant les relations de dominance comme des applications fonctionnelles.

[42] Nous ne revenons pas en détail sur la sémantique à trous de (Bos, 1995) qui a déjà été introduite en section 1.2.1.2 page 26 du présent document.

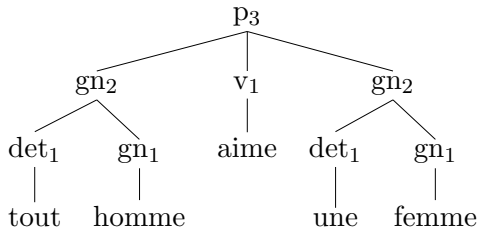


FIG. 2.27 – $\mathscr{L}(c_{t\_aime}\ I_p\ I_v\ (\ c_{t\_homme}\ c_{t\_tout}\ I_{gn}\ I_{det}\ )\ (\ c_{t\_femme}\ c_{t\_une}\ I_{gn}\ I_{det}\ ))$.

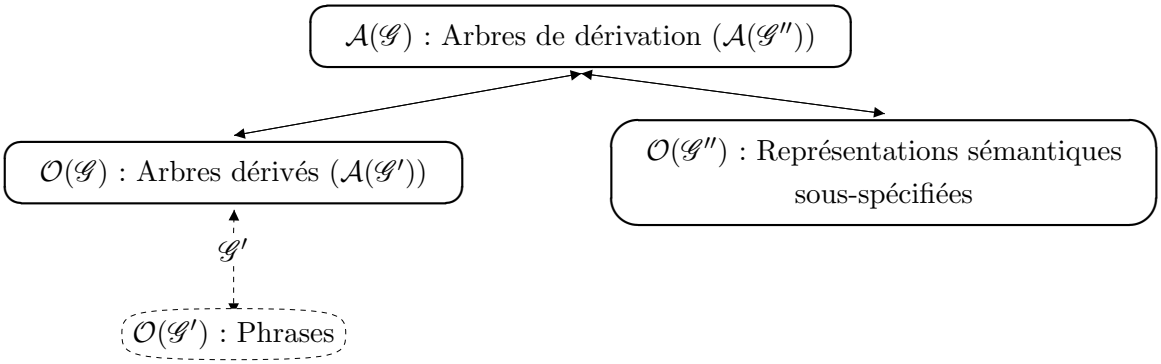


FIG. 2.28 – Interface syntaxe / sémantique pour TAG au moyen d'une ACG.

Nous cherchons donc à définir une ACG $\mathscr{G}''$ utilisant la même signature Σ_1 que celle du paragraphe précédent, et dont la signature Σ_2 permette de représenter des formules sémantiques sous-spécifiées (dans (Pogodalla, 2004a), ces formules sont représentées au moyen de λ-termes *non-linéaires*[43] simplement typés). Pour définir cette ACG $\mathscr{G}''$, il nous faut également définir un lexique $\mathscr{L}''$ permettant d'associer les λ-termes linéaires de $\Lambda(\Sigma_1)$ aux λ-termes (non-linéaires) de $\Lambda(\Sigma_2)$.

Concernant le premier point, à savoir la définition de $\Sigma_2 = \langle A_2, C_2, \tau_2\rangle$[44], nous procédons comme suit :

- les types atomiques sont ceux utilisés dans la sémantique Montagovienne, e pour les entités, t pour les valeurs de vérité, plus les types liés aux objets de la sémantique à trous, h pour les variables de trou, l pour les variables d'étiquettes, et p pour les prédicats :

$$A_2 = \{\, e,\ \ t,\ \ h,\ \ l,\ \ p\,\}$$

- on associe à chaque opérateur de la sémantique à trou, et à chaque constituant de la logique des prédicats (appliquée à la sémantique à trous, *cf* section 1.2.1.2) une

[43] Le fait que les λ-termes de $\Lambda(\Sigma_2)$ soient non-linéaires ne change pas les propriétés de décidabilité des ACG s, en particulier le formalisme est toujours réversible, cependant le langage objet ainsi défini ne peut plus être utilisé comme langage abstrait d'une autre ACG, pour plus de détails voir (Pogodalla, 2004b).

[44] Σ_2 est ici une signature d'ordre supérieur non-linéaire.

constante comme définie figure 2.29.

$\leq$	contrainte de portée entre variable de trou et variable d'étiquette
:	opérateur associant à un prédicat une variable d'étiquette
$\wedge$	opérateur de conjonction de formules
$\exists_l$	quantificateur existentiel pour les variables d'étiquette
$\exists_h$	quantificateur existentiel pour les variables de trou
$\exists_e$	quantificateur existentiel pour les variables d'entité
And	opérateur de conjonction de la logique des prédicats
Imp	opérateur d'implication de la logique des prédicats
Ex	quantificateur existentiel de la logique des prédicats
All	quantificateur universel de la logique des prédicats
$\mathbf{P_1}$	prédicat unaire de la logique des prédicats
$\mathbf{P_2}$	prédicat binaire de la logique des prédicats

FIG. 2.29 – Constantes de C_2.

– les constantes de C_2 sont alors typées par la fonction τ_2 définie comme suit[45] :

$$
\begin{array}{rrcl}
\tau_{\mathbf{2}} : & \mathbf{C_2} & \rightarrow & \mathscr{T}(\mathbf{A_2}) \\
& \leq & \mapsto & h \rightarrow l \rightarrow t \\
& : & \mapsto & l \rightarrow p \multimap t \\
& \wedge & \mapsto & t \multimap t \multimap t \\
& \exists_l & \mapsto & (l \rightarrow t) \multimap t \\
& \exists_h & \mapsto & (h \rightarrow t) \multimap t \\
& \exists_l & \mapsto & (e \rightarrow t) \multimap t \\
& \mathbf{And} & \mapsto & l \rightarrow h \rightarrow p \\
& \mathbf{Imp} & \mapsto & l \rightarrow h \rightarrow p \\
& \mathbf{Ex} & \mapsto & e \rightarrow l \rightarrow t \\
& \mathbf{All} & \mapsto & e \rightarrow l \rightarrow t \\
& \mathbf{P_1} & \mapsto & e \rightarrow p \\
& \mathbf{P_2} & \mapsto & e \rightarrow e \rightarrow p
\end{array}
$$

Enfin, concernant le second point, à savoir la définition du lexique $\mathscr{L}''$, cela revient à définir la transformation des types atomiques de A_1 en types de $\mathscr{T}(A_2)$, et des constantes de C_1 en λ-termes de $\Lambda(\Sigma_2)$.

[45] On note que $\mathscr{T}(A_2)$ (le codomaine de τ_2) réfère ici à l'ensemble des types implicatifs non-linéaires.

– Les types atomiques de la signature Σ_1 sont transformés comme indiqué ci-dessous :

$$\begin{aligned}
\mathscr{L}''(det_A) &= (e \to h \to l \to t) \multimap (e \to h \to l \to t) \\
\mathscr{L}''(gn_A) &= (e \to h \to l \to t) \multimap (e \to h \to l \to t) \multimap (h \to l \to t) \\
\mathscr{L}''(v_A) &= (h \to l \to t) \multimap (h \to l \to t) \\
\mathscr{L}''(p_A) &= (e \to h \to l \to t) \multimap (e \to h \to l \to t) \\
\mathscr{L}''(gn_S) &= (e \to h \to l \to t) \multimap (h \to l \to t) \\
\mathscr{L}''(p_S) &= h \to l \to t
\end{aligned}$$

Par exemple, le type de $\mathscr{L}''(v_A)$ nous indique qu'un nœud de type V_A est un modificateur de verbe, qu'il prend un objet de type $(h \to l \to t)$ et produit un objet de même type.

– Les constantes de C_1 se voient associées à un λ-terme non-linéaire de $\Lambda(\Sigma_2)$ reflétant la contribution sémantique de chaque nœud, par exemple pour c_{aimer} :

$$\mathscr{L}''(c_{aimer}) = \lambda baso.s(b(\lambda x.o(a(\lambda y h' l'.h' \leq l' \wedge\ l' : \mathbf{aimer}(x, y)))))$$

Les variables a et b réfèrent respectivement aux contributions des nœuds internes d'étiquette gv et p, le premier agissant sur l'ensemble de la relation, le second sur le prédicat[46].

Enfin, le type distingué de $\mathscr{G}''$ reste le même : p_S.

Nous pouvons calculer $\mathscr{L}''(c_{aime}\ I_p\ I_v\ (\ c_{homme}\ c_{tout}\ I_{gn}\ I_{det}\ )\ (\ c_{femme}\ c_{une}\ I_{gn}\ I_{det}\ ))$, c'est-à-dire le λ-terme de $\Lambda(\Sigma_2)$ associé à l'arbre de dérivation de la phrase « tout homme aime une femme », en considérant le lexique suivant[47] :

$$\begin{aligned}
\mathscr{L}''(I_X) &= \lambda x.x \\
\mathscr{L}''(c_{tout}) &= \lambda barp.ba(\lambda hl.\exists h_1 l_1 l_2 l_3 v_1 \\
&\quad (h \leq l_2 \wedge\ l_2 : \mathbf{All}(v_1, l_3) \wedge\ l_3 : \mathbf{Imp}(l_1, h_1) \\
&\quad \wedge\ h_1 \leq l \wedge\ r v_1 h l_1 \wedge\ p v_1 h l) \\
\mathscr{L}''(c_{homme}) &= \lambda q.q(\lambda x h l.h \leq l \wedge\ l : homme(x)) \\
\mathscr{L}''(c_{aime}) &= \lambda baso.s(b(\lambda x.o(a(\lambda y h' l'.h' \leq l' \wedge\ l' : \mathbf{aimer}(x, y))))) \\
\mathscr{L}''(c_{une}) &= \lambda barp.ba(\lambda hl.\exists h_1 l_1 l_2 l_3 v_1 \\
&\quad (h \leq l_2 \wedge\ l_2 : \mathbf{Ex}(v_1, l_3) \wedge\ l_3 : \mathbf{And}(l_1, h_1) \\
&\quad \wedge\ h_1 \leq l \wedge\ r v_1 h l_1 \wedge\ p v_1 h l) \\
\mathscr{L}''(c_{femme}) &= \lambda q.q(\lambda x h l.h \leq l \wedge\ l : femme(x))
\end{aligned}$$

La représentation sémantique sous-spécifiée (*i.e.*, le λ-terme de $\Lambda(\Sigma_2)$) que nous obtenons

[46] On remarque que si le nœud associé à l'objet était fils du nœud v dans l'arbre élémentaire, cette variable a porterait sur le prédicat et son second argument.

[47] On remarque la distinction entre restriction et portée des quantificateurs représentées respectivement par les variables r et p.

après réduction, est la suivante :

$$\lambda hl.\exists h_1 l_1 l_2 l_3 v_1 (h \leq l_2 \wedge\ l_2 : \mathbf{All}(v_1, l_3) \wedge\ l_3 : \mathbf{Imp}(l_1, h_1) \wedge\ h_1 \leq l \wedge\ l_1 : homme(v_1)$$
$$\wedge\ \exists h'_1 l'_1 l'_2 l'_3 v'_1 (h \leq l'_2 \wedge\ l'_2 : \mathbf{Ex}(v'_1, l'_3) \wedge\ l'_3 : \mathbf{And}(l'_1, h'_1) \wedge\ h'_1 \leq l \wedge\ l'_1 : femme(v'_1)$$
$$\wedge\ h \leq l \wedge\ l : \mathbf{aimer}(v_1, v'_1).$$

Cette formule sous-spécifiée peut être représentée graphiquement comme sur la figure 1.10 introduite précédemment (page 30).

Dans ce paragraphe, nous avons vu comment utiliser une grammaire catégorielle abstraite pour produire une représentation sémantique sous-spécifiée à partir de l'arbre de dérivation. Cette technique est particulièrement intéressante puisqu'elle fournit un cadre de travail adéquat non seulement à l'analyse sémantique mais également à la génération (via la réversibilité du formalisme). A ce jour, les phénomènes linguistiques supportés par cette approche regroupent la gestion des quantificateurs, les adverbes opaques, les verbes à montée, les dépendances à longue distance et les questions en *qu*. Cependant les verbes à contrôle restent problématiques.

Cette approche clôt la section sur les procédés de construction sémantique en TAG basés sur l'arbre de dérivation. A présent, nous allons étudier l'autre grande tendance, à savoir les procédés basés sur l'arbre dérivé.

2.2.4 Les méthodes basées sur l'arbre dérivé

2.2.4.1 L'approche de Frank et Van Genabith (2001) : utilisation de la logique linéaire

La première tentative de calcul sémantique à partir de l'arbre dérivé provient de (Frank et Van Genabith, 2001). Les auteurs partent du constat que l'opération d'adjonction est utilisée en TAG à la fois pour le traitement des modificateurs (optionnels) tels que les adverbes (*cf* exemple 3a), et pour le traitement des structures de complémentation[48] tels que les verbes à contrôle ou autres composants récursifs (*cf* exemple 3b).

(3) a. Jean pense *souvent* à Marie.

b. Jean *semble* penser que Marie adore les plats épicés.

Cependant, ces deux types de constituants syntaxiques diffèrent dans leur interprétation sémantique en terme de dépendances (*cf* problème du lien manquant, voir section 2.2.1 page 45).

Les auteurs en concluent l'existence d'un non-isomorphisme entre adjonction syntaxique et adjonction sémantique, rendant les approches de construction sémantique basées sur l'arbre de dérivation inadaptées.

Pour parer à ce problème, (Frank et Van Genabith, 2001) proposent d'utiliser l'arbre dérivé comme support à un procédé de composition sémantique utilisant la logique linéaire.

[48] Représentés au moyen d'arbres auxiliaires prédicatifs.

Plus précisément, ils proposent d'adapter le formalisme de la *sémantique à colle* (*Glue Semantics*) (Dalrymple et al., 1995), utilisé notamment pour le calcul sémantique avec les grammaires lexicales fonctionnelles (*Lexical Functional Grammars – LFG*), aux grammaires d'arbres adjoints.

Dans ce contexte, la construction sémantique se fait via le procédé suivant :

1. chaque arbre élémentaire de la grammaire est associé à un *constructeur de sens* (*meaning constructor*) de la forme
$$P : L$$
où P correspond à un λ-terme et L à une formule de logique linéaire (appelée *interface de collage* ou *glue part*), voir exemples figure 2.30 ci-dessous ;
2. les nœuds de l'arbre élémentaire sont étiquetés au moyen de traits sémantiques dont les valeurs sont des variables partagées avec l'interface de collage ;
3. lors de la réécriture d'arbres au moyen des opérations d'adjonction et de substitution, des équations de variables sont réalisées (ce qui permet de mettre à jour les interfaces de collage des arbres élémentaires combinés) ;
4. enfin, à l'issue d'une dérivation complète, on calcule une preuve par déduction à partir des constructeurs de sens des arbres élémentaires impliqués. La déduction de cette preuve est guidée par les interfaces de collage (*i.e.*, par une formule de logique linéaire), et produit la représentation sémantique[49] de la phrase.

Remarque Une motivation à l'utilisation de la logique linéaire pour la construction sémantique est que cette logique permet de représenter le fait qu'un constituant syntaxique apporte une information sémantique utilisée exactement une fois. En outre, la déduction comme calcul sémantique permet de s'assurer que chaque information sémantique nécessaire est présente, et qu'aucune information sémantique reste inutilisée. La logique linéaire permet donc de représenter une notion de besoin / ressource.

Principes d'étiquetage des arbres élémentaires Lors de l'écriture de la grammaire TAG, les arbres élémentaires sont annotés sémantiquement en respectant les principes suivants :

(i) tout nœud ancre[50] d'un arbre α reçoit, dans sa structure de traits *bottom*, un trait sémantique L_b de valeur $x_\perp$ (où x est une nouvelle variable) :
$$L_b(ancre(\alpha)) = x_\perp$$

(ii) tout arbre auxiliaire β associé à un modificateur est annoté de façon à ce que son nœud racine et son nœud pied reçoivent le trait sémantique x (x désigne une nouvelle

[49] *Les* représentations sémantiques en cas d'ambigüité.
[50] *I.e.*, un nœud étiqueté par un item lexical.

variable) comme suit :

$$L_b(racine(\beta)) \;=\; L_t(pied(\beta)) \;=\; x$$

(iii) les projections des nœuds ancres sur les nœuds internes d'un arbre α sont étiquetés sémantiquement au moyen des nouvelles variables $x_\perp, x_1, \ldots, x_n, x_\top$.

(iv) tout nœud référant à un argument du prédicat associé à un arbre α reçoit un trait sémantique, dans sa structure *top*, identique à celui de la structure *bot* de son nœud père, et étendu avec la fonction grammaticale FG de cet argument :

$$L_t(arg_N(\alpha)) \;=\; L_b(\text{père}(arg_N(\alpha))) : FG$$

Les principes (i) et (iii) permettent de marquer la projection de la tête lexicale, ce qui est nécessaire pour le traitement sémantique des constructions impliquant des dépendances non-locales[51] (*e.g.* modificateurs ou verbes à contrôle). En effet, cette projection permet de définir le type d'objet modifié en mettant à jour les interfaces de collage qui guideront le calcul sémantique par déduction (effet des variables i et $f_\top$ sur la figure 2.31 ci-dessous).

Le principe (ii) est utilisé de manière comparable aux structures de traits utilisées en TAG pour exprimer le lien entre racine et pied d'un modificateur.

On note enfin l'annotation des arguments du prédicat avec un trait sémantique marquant leur position par rapport à la projection de la tête lexicale et leur fonction grammaticale (via (iv)). La position permet de traiter les cas de dépendances à longue distance[52]. La fonction grammaticale permet de s'assurer que chaque fonction grammaticale est présente exactement une fois lors du calcul sémantique.

Quelques exemples d'arbres élémentaires annotés sémantiquement et de constructeurs de sens associés sont donnés figure 2.30[53].

Equations de variables réalisées lors de la réécriture d'arbres Lorsque deux arbres sont combinés au cours de la dérivation, les équations entre variables suivantes sont réalisées :

(i) (substitution) lors de la substitution de l'arbre γ au nœud n_α de l'arbre α, l'équation suivante est posée :

$$L_t(n_\alpha) \;=\; L_b(racine(\gamma))$$

(ii) (adjonction) lors de l'adjonction de l'arbre β sur le nœud n_α de l'arbre α, on distingue deux cas :

1. si β est un arbre auxiliaire associé à un modificateur, on effectue l'équation suivante :

$$L_t(n_\alpha) \;=\; L_b(racine(\beta))$$

[51] Par opposition aux dépendances locales représentées par des nœuds de substitution.

[52] Non-présenté ici, voir (Frank et Van Genabith, 2001)

[53] Les traits L_t sont notés en exposants et les traits L_b sont notés en indices.

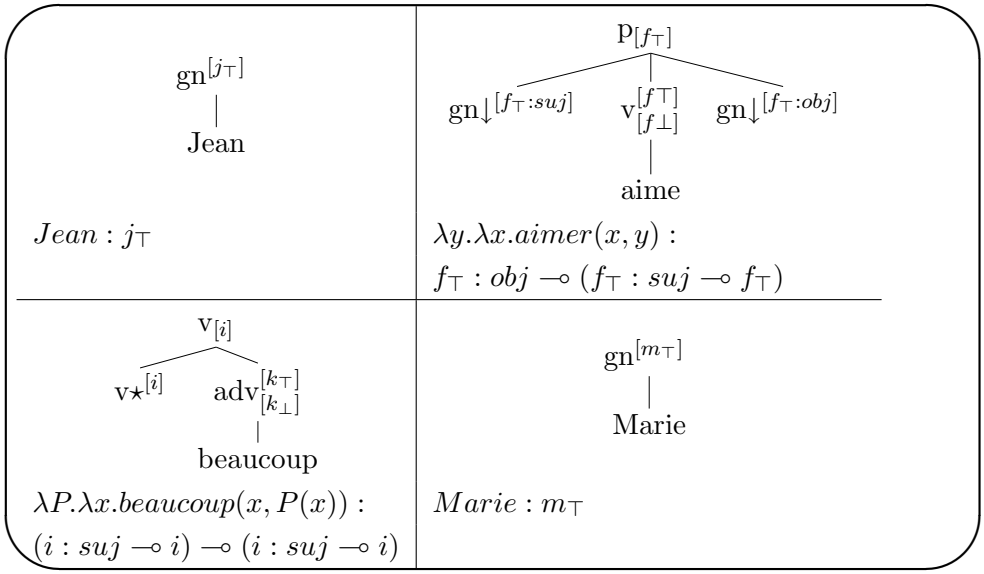


FIG. 2.30 – Arbres élémentaires munis de constructeurs de sens.

2. si β est un arbre auxiliaire prédicatif, on effectue l'équation suivante :

$$L_b(n_\alpha) \;=\; L_t(pied(\beta))$$

L'équation (i) permet de renseigner le type sémantique d'un argument du prédicat (les arguments correspondent à des nœuds de substitution). On note que, dans le cas de l'adjonction, les équations de variables diffèrent selon le type d'arbre auxiliaire considéré (modificateur ou composant de complémentation). Pour les composants de complémentation, le nœud pied de l'arbre auxiliaire dénote un argument de l'ancre lexicale. L'équation (ii-2) traduit le fait que, dans ce cas, le nœud pied de l'arbre auxiliaire se comporte comme un nœud de substitution. Pour les modificateurs, le nœud racine de l'arbre auxiliaire reçoit l'information sémantique du nœud auquel il s'adjoint, ainsi le modificateur est intégré à la phrase (équation (ii-1)).

Cette distinction entre arbre auxiliaire modificateur et arbre auxiliaire prédicatif est le pendant du problème de non-isomorphisme entre adjonction syntaxique et sémantique mentionné précédemment.

Enfin, lorsqu'un nœud n comporte deux traits sémantiques L_t et L_b ayant pour valeur respective les variables t et t', nous prenons la convention de réaliser la substitution suivante entre t et t' dans le constructeur de sens mcs associé à l'arbre :

$$(L_b(n) \;=\; t \;\wedge\; L_t(n) \;=\; t') \rightarrow mcs_{[t/t']}$$

en d'autres termes, nous substituons la variable du trait L_b par celle du trait L_t.

Ces substitutions de variables permettent de mettre à jour les interfaces de collage qui sont utilisées comme guide pour le calcul sémantique, en fonction des arbres élémentaires qui sont combinés.

Un exemple A présent que nous avons définit un procédé d'annotation sémantique des arbres élémentaires, ainsi que des règles d'équation de variables lors de la dérivation, voyons sur un exemple comment le calcul sémantique s'opère.

Considérons la phrase « Jean aime beaucoup Marie ». Cette phrase est analysée au moyen des arbres élémentaires de la figure 2.30, et l'arbre dérivé est illustré figure 2.31.

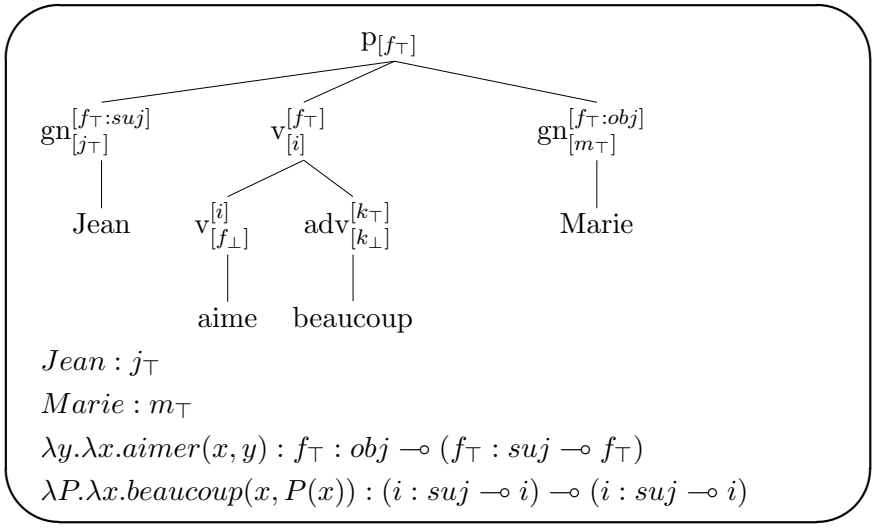


FIG. 2.31 – Arbre dérivé et constructeurs de sens pour « Jean aime beaucoup Marie ».

A partir de l'arbre dérivé et des équations entre variables introduites ci-dessus, les substitutions suivantes sont réalisées :

$$j_\top \rightarrow f_\top : suj \qquad m_\top \rightarrow f_\top : obj \qquad i \rightarrow f_\top$$

L'étape finale du calcul sémantique correspond alors à la déduction d'une preuve pour $f_\top$ (trait sémantique étiquetant la racine de l'arbre dérivé) :

$$\frac{\lambda y.\lambda x.aimer(x,y) : f_\top : obj \multimap (f_\top : suj \multimap f_\top) \qquad Marie : f_\top}{\lambda x.aimer(x,Marie) : (f_\top : suj \multimap f_\top)}$$

$$\frac{\lambda x.aimer(x,Marie) : (f_\top : suj \multimap f_\top) \qquad \lambda P.\lambda x.beaucoup(x,P(x)) : (f_\top : suj \multimap f_\top) \multimap (f_\top : suj \multimap f_\top)}{\lambda x.beaucoup(x,\lambda x.aimer(x,Marie)(x)) : (f_\top : suj \multimap f_\top)}$$

$$\frac{\lambda x.beaucoup(x,\lambda x.aimer(x,Marie)(x)) : (f_\top : suj \multimap f_\top) \qquad Jean : f_\top}{beaucoup(Jean,aimer(Jean,Marie) : f_\top}$$

Traitement des quantificateurs Ce procédé de construction sémantique à base de *glue semantics* permet également de traiter les ambiguïtés de portée. Nous illustrons cela au moyen de l'exemple (usuel) « tout homme aime une femme ».

Les arbres élémentaires associés aux quantificateurs sont donnés figure 2.32.

L'arbre dérivé correspondant est donné figure 2.33.

De cet arbre dérivé, nous déduisons les substitutions entre variables suivantes :

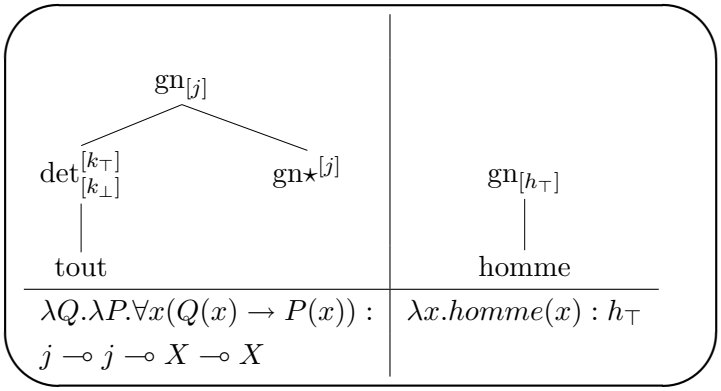


FIG. 2.32 – Arbres élémentaires et constructeurs de sens pour les quantificateurs.

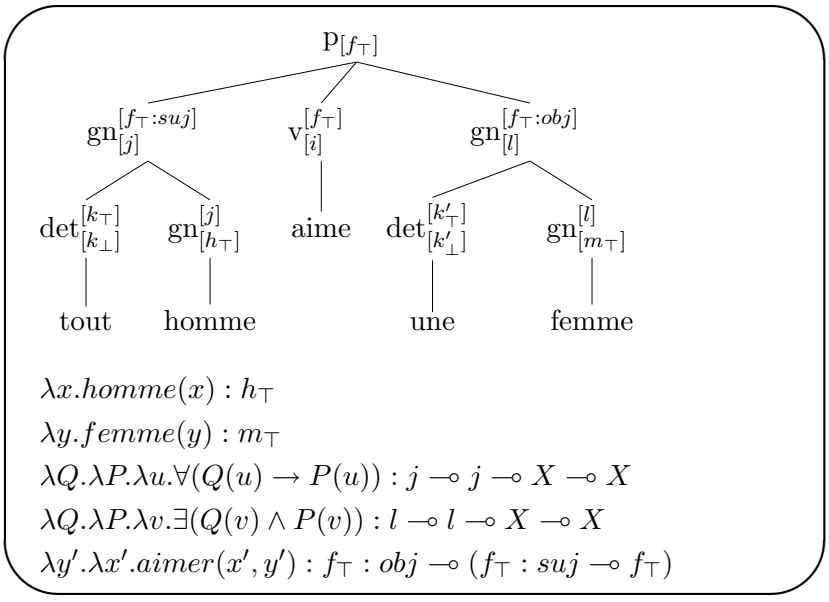


FIG. 2.33 – Arbre dérivé et constructeurs de sens pour « tout homme aime une femme ».

(a) $j \rightarrow f_\top : suj$ (b) $h_\top \rightarrow j$ (c) $m_\top \rightarrow l$ (d) $l \rightarrow f_\top : obj$

De (a) et (b) (respectivement (c) et (d)), nous pouvons déduire $h_\top \rightarrow f_\top : suj$ (respectivement $m_\top \rightarrow f_\top : obj$).

A présent, il reste à calculer une preuve pour $f_\top$ à partir des constructeurs de sens et des substitutions de variables :

$$\frac{\lambda Q.\lambda P.\lambda v.\exists(Q(v) \wedge P(v)) : f_\top : obj \multimap f_\top : obj \multimap X \multimap X \qquad \lambda y.femme(y) : f_\top : obj}{\lambda P.\lambda v.\exists(\lambda x.femme(x)(v) \wedge P(v)) : f_\top : obj \multimap X \multimap X}$$

$$\frac{\lambda P.\lambda v.\exists(femme(v) \wedge P(v)) : f_\top : obj \multimap X \multimap X \qquad \lambda y'.\lambda x'.aimer(x', y') : f_\top : obj \multimap (f_\top : suj \multimap f_\top)}{\lambda v.\exists(femme(v) \wedge \lambda y'.\lambda x'.aimer(x', y)(v)) : f_\top : suj \multimap f_\top}$$

$$\frac{\lambda Q.\lambda P.\lambda u.\forall(Q(u) \rightarrow P(u)) : f_\top : suj \multimap f_\top : suj \multimap X \multimap X \qquad \lambda x.homme(x) : f_\top : suj}{\lambda P.\lambda u.\forall(\lambda x.homme(x)(u) \rightarrow P(u)) : f_\top : suj \multimap X \multimap X}$$

$$\frac{\lambda P.\lambda u.\forall(homme(u) \rightarrow P(u)) : f_\top : suj \multimap X \multimap X \qquad \lambda v.\exists(femme(v) \wedge \lambda x'.aimer(x', v)) : f_\top : suj \multimap f_\top}{\lambda u.\forall(homme(u) \rightarrow \lambda v.\exists(femme(v) \wedge \lambda x'.aimer(x', v))(u)) : f_\top}$$

Finalement, nous obtenons la représentation sémantique suivante :

$$\lambda u.\forall(homme(u) \rightarrow \lambda v.\exists(femme(v) \wedge aimer(u, v))) : f_\top$$

ce qui correspond à l'interprétation sémantique de la phrase « tout homme aime une femme » lorsque le quantificateur universel a une portée englobant celle du quantificateur existentiel.

La seconde interprétation possible (avec une portée maximale pour le quantificateur existentiel) est calculée en considérant le constructeur de sens suivant pour le verbe *aimer* :

$$\lambda x'.\lambda y'.aimer(x', y') : f_\top : suj \multimap (f_\top : obj \multimap f_\top)$$

Ce constructeur est équivalent à celui considéré en figure 2.32, et permet de déduire la formule suivante :

$$\lambda v.\exists(femme(v) \wedge \lambda u.\forall(homme(u) \rightarrow aimer(u, v))) : f_\top$$

Dans cette section, nous avons présenté une approche de construction sémantique basée sur l'arbre dérivé. Cette approche utilise des arbres élémentaires associés à des formules sémantiques sous forme de λ-termes typés par une formule de logique linéaire. Cette formule, partageant des variables avec les nœuds des arbres, constitue l'interface entre syntaxe et sémantique.

Cette méthode de construction sémantique a été utilisée avec succès pour traiter sémantiquement les phénomènes syntaxiques suivants : verbes à contrôle, ambiguïtés de portée, interactions entre verbes à montée et adverbes et dépendances longue distance dans le cas des questions.

La principale limite à cette approche concerne le passage à grande échelle du procédé. En effet, la définition des formules de logique linéaire guidant le calcul sémantique peut s'avérer très délicat sur des grammaires de taille réelle. De plus, cette approche ne tire pas partie de l'opération d'unification présente dans le formalisme TAG. Ainsi, le système de types guidant la construction sémantique est défini au moyen d'équations de traits additionnelles.

2.2.4.2 L'approche de Gardent et Kallmeyer (2003) : utilisation d'une sémantique sous-spécifiée et de variables d'unifications partagées avec l'arbre dérivé

Dans la lignée de l'approche précédente, une seconde proposition pour la construction sémantique à partir de l'arbre dérivé a été proposée par (Gardent et Kallmeyer, 2003).

Cette proposition reprend certaines idées de (Frank et Van Genabith, 2001), à savoir :

- association d'une formule sémantique à chaque arbre élémentaire ;
- partage de variables entre nœuds de l'arbre syntaxique et formule sémantique ;
- annotation sémantique de la projection de la tête lexicale.

Dans ce contexte, la définition de l'interface syntaxe / sémantique se fait comme suit :

1. chaque arbre élémentaire est associé à une représentation en sémantique plate[54]. Cette représentation correspond à un ensemble de formules. Une formule est soit un prédicat associé à une variable d'étiquette, soit une contrainte de portée ($\geq$) ;
2. certaines variables de la représentation sémantique (arguments des prédicats ou variables d'étiquette) sont partagées avec les traits *idx* (index) et *lab* (label) des structures *top* et *bottom* dont disposent les nœud des arbres élémentaires en TAG ;
3. lors de la dérivation, les structures de traits associées aux nœuds sont unifiées via les opérations d'adjonction et de substitution (ce qui inclut l'unification des traits sémantiques *idx* et *label*). La conséquence de cela est la mise à jour des représentations sémantiques élémentaires ;
4. à l'issue de la dérivation, la représentation sémantique finale (sous-spécifiée) correspond à l'union des représentations sémantiques associées aux arbres élémentaires impliqués dans la dérivation, modulo les unifications des variables argumentales réalisées au cours de la dérivation.

Annotation sémantique des arbres élémentaires L'annotation sémantique des arbres élémentaires, en d'autres termes, le placement des variables sémantiques partagées dans les structures de traits *top* et *bottom* des nœuds se fait en respectant les règles suivantes :

a) Annotation des arbres initiaux

(arguments) les arguments du prédicat associé à l'arbre élémentaire correspondent à des variables partagées avec les traits *idx* des structures *top* des nœuds désignant ces arguments[55] ;

(étiquettes) les variables d'étiquette sont placées dans les traits *lab* des structures *top* des nœuds des arguments, et annotent également la projection de la tête lexicale.

b) Annotation des arbres auxiliaires

[54] *Cf* (Bos, 1995) et section 1.2.1.2 page 26.

[55] Via le principe de coocurrence prédicat-arguments, tout arbre TAG contient un nœud pour chaque argument du prédicat.

(arguments) dans le cas d'arbres auxiliaires prédicatifs, les arguments du prédicats sont partagés avec les traits *idx* de la structure *top* du nœud de l'argument (sujet des verbes à contrôle par exemple) et de la structure *bottom* du nœud pied (sujet du verbe contrôlé par exemple).
Dans le cas des quantificateurs, la variable quantifiée est partagée avec le trait *idx* de la structure *top* du nœud racine et de la structure *bottom* du nœud pied.

(étiquettes) dans le cas d'arbres auxiliaires modificateurs, la variable d'étiquette de la formule sémantique est partagée avec le trait *lab* de la structure *bottom* du nœud racine. La variable d'étiquette utilisée pour représenter la portée du modificateur est passée au trait *lab* de la structure *bottom* du nœud pied (voir figure 2.34 ci-dessous).
Dans le cas des quantificateurs, deux variables de trou sont utilisées pour représenter respectivement la portée et la restriction du quantificateur, et sont partagée respectivement avec le trait *lab* de la structure *top* du nœud racine, et de la structure *bottom* du nœud pied.

Des exemples d'arbres élémentaires avec représentations sémantiques sont donnés figure 2.34[56][57].

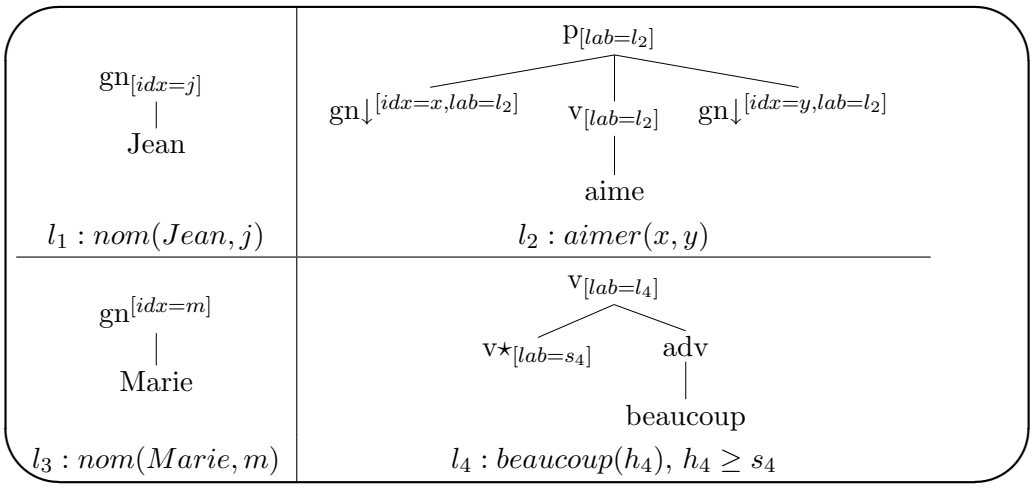


FIG. 2.34 – Arbres élémentaires et représentation sémantique.

Unification des traits sémantiques Les traits sémantiques annotant les nœuds des arbres élémentaires sont *unifiés* lors des opérations de substitution et d'adjonction ayant lieu pendant et à l'issue de la dérivation, en suivant les règles usuelles d'unification pour TAG, règles présentées en section 2.1.2 page 39.

[56] Les structures *top* sont notées en exposants, et les structures *bottom* en indices.

[57] Les arbres associés aux noms propres n'ont pas de trait *label* dans leur structure *bot*, à la différence des arbres associés aux noms communs. Cet absence a pour but de prévenir un échec d'unification du à l'absence de déterminant.

Un exemple Dans un premier temps, considérons la phrase « Jean aime beaucoup Marie », dont l'arbre dérivé est donné figure 2.35.

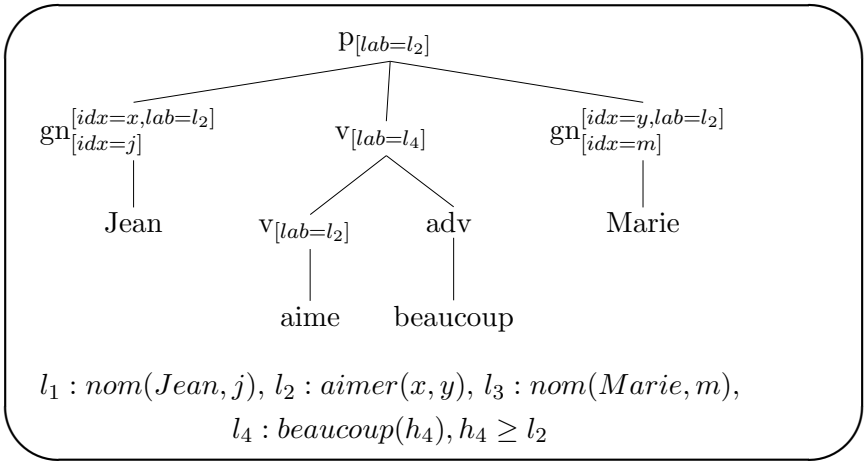


FIG. 2.35 – Arbre dérivé et représentation sémantique pour « Jean aime beaucoup Marie ».

Lors de la dérivation, l'adjonction de l'arbre $\tau_{beaucoup}$ sur le nœud v de l'arbre τ_{aimer}, la variable d'étiquette s_4 et la variable d'étiquette l_2 sont unifiées. De plus, à l'issue de la dérivation, les structures de traits *top* et *bottom* de chaque nœud de l'arbre dérivé sont unifiées, ce qui provoque l'unification des variables d'index j et x d'une part, et m et y d'autre part. On obtient alors la formule sémantique sous-spécifiée suivante :

$$l_1 : nom(Jean, j),\ l_2 : aimer(j, m),\ l_3 : nom(Marie, m),\ l_4 : beaucoup(h_4), h_4 \geq l_2$$

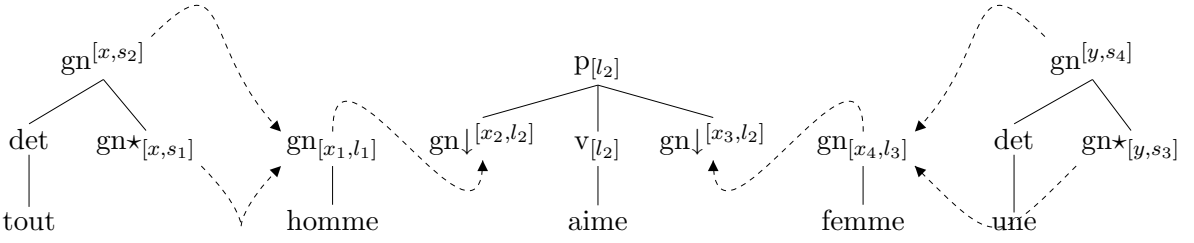
On note qu'il existe une seule assignation entre variables de trou et variables d'étiquette pour cette formule, à savoir $h_4 \leftrightarrow l_2$, ce qui produit la formule spécifiée finale

$$l_1 : nom(Jean, j),\ l_2 : aimer(j, m),\ l_3 : nom(Marie, m),\ l_4 : beaucoup(l_2)$$

Traitement des quantificateurs Considérons à présent un exemple introduisant une ambiguïté de portée, comme la phrase aux deux quantificateurs « tout homme aime une femme ». La dérivation de cette phrase est illustrée figure 2.36[58].

Lors de la dérivation, l'adjonction de τ_{tout} sur le nœud racine de l'arbre τ_{homme} provoque l'unification des traits d'index et de label, x et x_1 sont unifiées ainsi que s_1 et l_1 (unification des structures *bottom* du nœud pied et du nœud site de l'adjonction). De manière similaire, l'adjonction de τ_{une} sur τ_{femme} provoque l'unification de y et x_4 et s_3 et l_3. Par la suite, la substitution de l'arbre dérivé $\tau_{tout\ homme}$ sur τ_{aimer} provoque l'unification de x et x_2 et de s_2 et l_2 (unification des structures *top* des nœuds substitués). De même, la substitution de $\tau_{une\ femme}$ sur $\tau_{tout\ homme\ aime}$ déclenche l'unification de y et x_3 d'une part, et de s_4 et l_2 d'autre part.

[58] Par manque de place, les noms des traits sont omis.



$l_0 : \forall(x, h_1, h_2)$ $l_1 : homme(x_1)$ $l_2 : aimer(x_2, x_3)$ $l_3 : femme(x_4)$ $l_4 : \exists(y, h_3, h_4)$
$h_1 \geq s_1, h_2 \geq s_2$ $h_3 \geq s_3, h_4 \geq s_4$

FIG. 2.36 – Dérivation de « tout homme aime une femme ».

A l'issue de la dérivation, nous obtenons (modulo renommage lié aux unifications suscitées) l'ensemble de formules sémantiques suivant :

$$l_0 : \forall(x, h_1, h_2),\ l_1 : homme(x),\ l_2 : aimer(x, y),\ l_3 : femme(y),\ l_4 : \exists(y, h_3, h_4),$$

$$h_1 \geq l_1, \quad h_2 \geq l_2, \quad h_3 \geq l_3, \quad h_4 \geq l_2$$

Cet ensemble de formules peut se représenter graphiquement (en notant $\geq$ via une flèche), comme illustré figure 2.37[59].

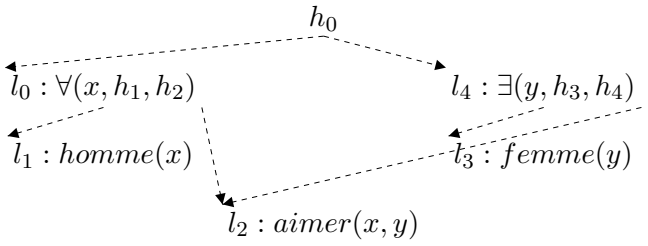


FIG. 2.37 – Représentation sémantique sous-spécifiée de « tout homme aime une femme ».

On remarque que cette figure est équivalente à la figure 1.10 introduite page 30, et illustrant le traitement sémantique de « tout homme aime une femme » dans la sémantique plate de (Bos, 1995).

Si nous souhaitons calculer l'ensemble des formules spécifiées, nous devons définir une fonction d'assignation des variables de trous h_i aux variables d'étiquettes l_j qui satisfasse l'ensemble des contraintes de portée. Dans le cas présent, nous pouvons définir deux assignations, correspondant aux deux interprétations possibles :

[59] La variable de trou h_0 symbolise la portée maximale.

$$\begin{cases} h_0 \rightarrow l_0 \\ h_1 \rightarrow l_1 \\ h_2 \rightarrow l_4 \\ h_3 \rightarrow l_3 \\ h_4 \rightarrow l_2 \end{cases} \qquad \begin{cases} h_0 \rightarrow l_4 \\ h_1 \rightarrow l_1 \\ h_2 \rightarrow l_2 \\ h_3 \rightarrow l_3 \\ h_4 \rightarrow l_0 \end{cases}$$

$$\forall x.homme(x) \rightarrow (\exists y.femme(y) \wedge aimer(x,y)) \qquad \exists y.femme(y) \wedge (\forall x.homme(x) \rightarrow aimer(x,y))$$

Nous venons de voir un procédé de construction sémantique basé sur l'arbre dérivé. Ce procédé permet de construire une représentation sous-spécifiée du sens d'énoncés connus comme problématiques pour les approches basées sur l'arbre de dérivation[60], ces énoncés incluent la quantification, les adjectifs intersectifs, la portée des modificateurs et les verbes à contrôle. De plus, le calcul sémantique[61] est réalisée par l'opération d'unification inhérente aux structures de traits étiquetant les nœuds des arbres élémentaires en TAG, il n'y a donc pas besoin de définir d'opérations annexes (*cf Glue semantics*). Comme le mentionne (Kallmeyer et Romero, 2004), l'inconvénient de cette approche réside dans l'utilisation de variables d'étiquettes et individuelles (variables d'arguments) en nombre potentiellement infini dans les structures de traits des nœuds, rendant le formalisme grammatical plus complexe que les TAG s classiques. Cependant, en pratique, le nombre de variables utilisées dans une grammaire, même de taille réelle, est fini.

2.2.5 Les méthodes hybrides

Après avoir étudié les approches basées exclusivement soit sur l'arbre de dérivation, soit sur l'arbre dérivé, nous présentons à présent des approches utilisant de l'information issues de chacune de ces deux structures.

2.2.5.1 L'approche de Kallmeyer (2002) : enrichissement de l'arbre de dérivation

L'approche de (Kallmeyer, 2002a; Kallmeyer, 2002b) est une extension de (Kallmeyer et Joshi, 1999), afin notamment de traiter les quantificateurs représentés par des arbres auxiliaires (cas des grammaires TAG de l'anglais (XTAG-Research-Group, 2001) et du français (Abeillé et al., 1999)).

Si nous représentons les quantificateurs par un arbre auxiliaire, l'arbre du quantificateur est adjoint à celui du nom, puis l'arbre du nom est substitué à celui du verbe (voir figure 2.38). En conséquence, l'arbre de dérivation ne contient aucun lien direct entre le

[60] A l'exception de (Kallmeyer et Romero, 2004) qui passe également par l'unification de traits sémantiques définis en fonction des arbres élémentaires et dérivés.

[61] Ici, nous considérons le calcul d'une formule sous-spécifiée uniquement, et non sa résolution en formules spécifiées.

quantificateur et le verbe. Ce lien manquant est problématique dans la mesure où la variable liée par le quantificateur est un argument du verbe, et la proposition introduite par le verbe est inclue dans la portée du quantificateur (relation $l_5 \leq h_2$ dans l'exemple du traitement de la phrase « tout homme aime une femme » par l'approche (Kallmeyer et Joshi, 1999), exemple introduit page 55).

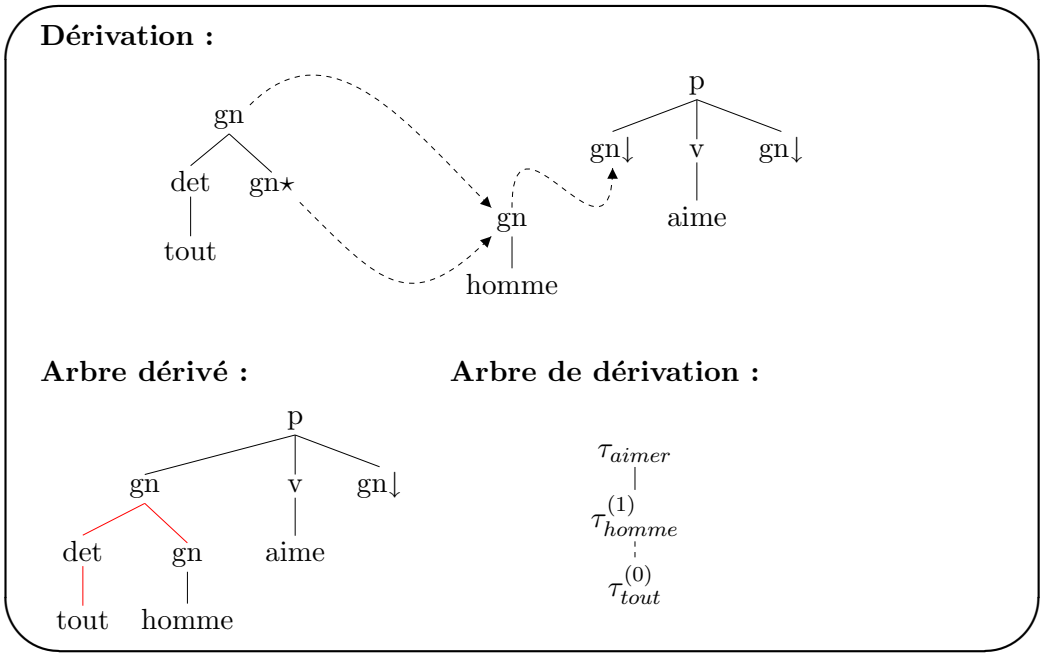


FIG. 2.38 – Analyse de la quantification par adjonction.

Avec une représentation du quantificateur sous forme d'arbre auxiliaire et une distinction entre relation prédicat-argument et relation de portée sous forme de MC-TAG, nous avons le problème suivant : le formalisme des MC-TAG n'est plus utilisé dans sa version locale, puisque l'arbre de la *portée* du quantificateur serait adjoint à l'arbre du verbe, alors que l'arbre de la *restriction* du quantificateur serait adjoint sur l'arbre du groupe nominal (*i.e.*, les deux arbres d'un ensemble seraient adjoints sur des arbres élémentaires distincts). Or les MC-TAG non-locales ont un pouvoir génératif supérieur aux TAG et leur complexité d'analyse est également supérieure. Il est donc préférable d'éviter cela.

Dans cette optique, (Kallmeyer, 2002a) propose d'enrichir l'arbre de dérivation pour pouvoir construire la représentation sémantique des quantificateurs adjoints.

L'intuition sur laquelle l'auteur se fonde est que les arbres élémentaires du quantificateur et du verbe apparaissent côte à côte dans l'arbre dérivé (sous-arbre coloré en rouge sur la figure 2.38). Cette relation de voisinage au niveau syntaxique justifie la présence d'un lien sémantique. De manière plus précise, l'auteur énonce qu'une opération d'adjonction sur un nœud racine d'un arbre élémentaire γ résulte non seulement en la création d'un lien sémantique entre l'arbre adjoint et γ mais également entre l'arbre adjoint et l'arbre avec lequel γ est réécrit dans la dérivation. La structure contenant ces liens sémantiques (ceux

de l'arbre de dérivation et les liens additionnels) est appelée structure de dérivation étendue ou *e-derivation*. L'e-derivation pour « tout homme aime une femme » est donné figure 2.39 (les liens additionnels et sous-arbres correspondants sur l'arbre dérivé sont colorés en rouge et bleu respectivement).

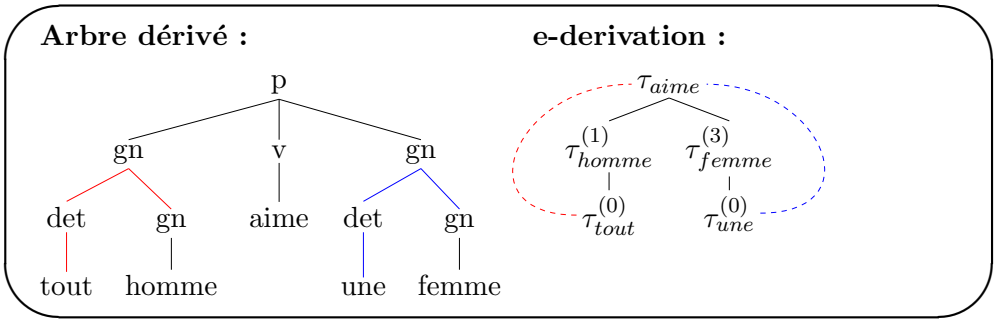


FIG. 2.39 – e-derivation pour « tout homme aime une femme »

Définition 39 (e-derivation)**.** *L'e-derivation est un graphe dont les nœuds correspondent aux arbres élémentaires utilisés dans la dérivation, et dont les arcs sont définis comme suit :*

- *tout arc de l'arbre de dérivation* TAG *est un arc de la structure e-derivation,*
- *deux nœuds γ et β de l'e-derivation sont liés par un arc s'il existe des nœuds γ', $\beta_1, \ldots, \beta_n$ appartenant à l'arbre de dérivation et tels que :*

$$\begin{cases} \gamma' \text{ est un nœud fils de } \gamma, \\ \beta_1 \text{ est un nœud fils de } \gamma', \text{ et l'arc est noté 0 (adjonction à la racine)}, \\ \beta_{i+1} \text{ est un nœud fils de } \beta_i, \text{ et l'arc est noté 0 (pour tout } i \text{ tel que } 1 \leq i < n), \\ \beta_n = \beta. \end{cases}$$

Dans ce contexte, les quantificateurs ne sont plus représentés au moyen d'un ensemble comportant deux arbres élémentaires et deux représentations sémantiques, mais au moyen d'un arbre auxiliaire associée à la représentation sémantique $\sigma_{\tau_{tout}}$ suivante :

$l_3 : tout(x, h_1, h_2), l_4 : p_1(x)$
$l_4 \leq h_1, s_2 \leq h_2$
arg : s_2, p_1

Dans le cas de la phrase *tout homme aime une femme*, cette représentation est appliquée à celles de *homme* et de *aimer* en suivant les liens de l'e-derivation représentée figure 2.39. Nous avons donc les applications suivantes :

- la représentation $\sigma_{\tau_{tout}}$ est appliquée à σ_{τ_homme} (adjonction au nœud racine), ce qui a pour conséquence d'assigner le prédicat *homme* à l'argument p_1,
- la représentation $\sigma_{\tau_{aimer}}$ est appliquée à la représentation $\sigma_{\tau_{tout}}$ (lien additionnel de

l'e-derivation), la variable d'étiquette l_5 est assignée à l'argument s_2, et[62] la variable individuelle x est assignée à l'argument x_1 (paramètre du prédicat *aimer*).

Nous obtenons à l'issue de ces applications, la représentation $\sigma_{\tau_{aimer}}$ suivante :

$l_3 : tout(x, h_1, h_2), l_4 : homme(x), l_5 : aimer(x, x_2)$ $l_4 \leq h_1, l_5 \leq h_2$
arg : x_2

Cette représentation est ensuite complétée de manière similaire pour l'expression quantifiée « une femme », on retrouve alors la représentation sémantique sous-spécifiée introduite page 55.

L'approche présentée dans ce paragraphe recourt à des représentations sémantiques sous-spécifiées qui sont combinées suivant les arcs d'une structure de graphe correspondant à une version enrichie de l'arbre de dérivation. L'enrichissement provient de la contigüité des arbres élémentaires dans l'arbre dérivé. Comme le montre (Kallmeyer, 2002a), cette technique permet de traiter des phénomènes reconnus comme problématiques tels que les dépendances illimités dans les phrases interrogatives enchassées, les modificateurs multiples et les verbes à montée. Il est important de noter que cette approche utilise non plus uniquement l'arbre de dérivation comme fondement pour la composition sémantique, mais également de l'information contenue dans l'arbre dérivé.

2.2.5.2 L'approche de Seddah (2004) : extraction d'un graphe de dépendances à partir de la forêt de dérivation

L'approche proposée par (Seddah, 2004)[63] montre comment extraire un graphe de dépendances à partir de la forêt de dérivation[64] calculée lors de l'analyse syntaxique, et d'informations issues de l'arbre dérivé (en l'occurrence, les arguments syntaxiques non-réalisés). Le procédé proposé est illustré sur le cas des verbes à contrôle. Dans un premier temps, nous introduisons le problème lié à la modélisation des verbes à contrôle en TAG, puis nous présentons le procédé d'extraction du graphe de dépendances.

Représentation des verbes à contrôle en TAG On parle de verbe à contrôle (du sujet[65]) pour les verbes dont le sujet contrôle le verbe d'un argument phrastique, comme par exemple *espérer* dans les exemples suivants :

[62] Bien que non-précisé dans (Kallmeyer, 2002a), il semblerait que les arcs additionnels de l'e-derivation engendrent une application symétrique des représentations sémantiques, permettant l'assignation de variables de chacune des représentation.

[63] Voir aussi (Seddah et Gaiffe, 2005b; Seddah et Gaiffe, 2005a).

[64] Une forêt de dérivation est une structure compacte représentant l'ensemble des dérivations d'une phrase, voir section 7.3.2 page 197 du présent document et (Vijay-Shanker et Weir, 1993).

[65] Il existe également des cas de contrôle de l'objet comme « La pomme semble avoir été mangée ».

(4) a. Jean espère dormir.

b. Jean espère parler à Marie.

Un verbe à contrôle est représenté en TAG au moyen d'un arbre auxiliaire venant s'adjoindre à la racine de l'arbre associé au prédicat (Abeillé, 1998) (voir figure 2.40[66]).

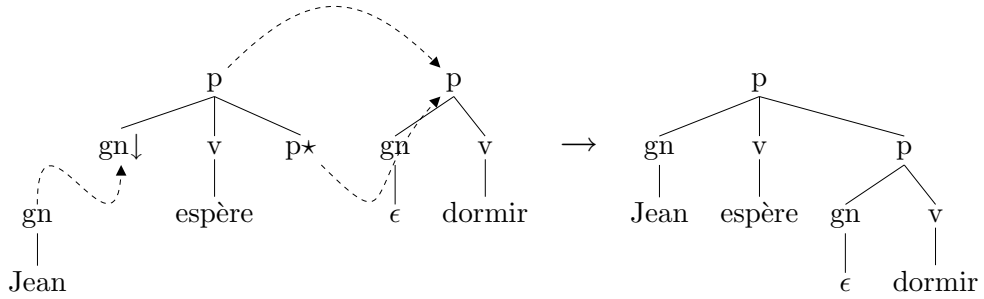


FIG. 2.40 – Représentation des verbes à contrôle en TAG.

On remarque que dans la dérivation de la figure 2.40, l'arbre élémentaire lié au nom *Jean* n'est pas combiné avec l'arbre lié au prédicat *dormir*. De ce fait, l'arbre de dérivation ne comporte aucun lien entre *Jean* et *dormir*. C'est ce lien que l'on souhaite rétablir dans le graphe de dépendance extrait.

Extraction du graphe de dépendances (Seddah, 2004) se base sur le constat que l'arbre d'un verbe à contrôle s'adjoint sur un arbre initial ayant au moins un nœud vide marquant un argument non-réalisé (Abeillé, 1998). Le verbe à contrôle *transfère* donc l'un de ses arguments au prédicat du verbe élémentaire sur lequel a lieu l'adjonction. L'auteur propose une modélisation de ce processus de transfert d'argument sous forme de graphe de dépendances.

(Seddah, 2004) se fonde sur un parcours de la forêt de dérivation issue de l'analyse syntaxique. Comme l'ont montré (Vijay-Shanker et Weir, 1993; Lang, 1994), cette forêt peut être représentée sous la forme d'une grammaire hors-contexte[67]. La forêt considérée par (Seddah, 2004) a la forme d'une grammaire hors-contexte[68], avec des règles étendues par l'ajout d'une pile bornée. Cette pile permet de garder une trace, lors du calcul de la dérivation, des adjonctions en cours. Chaque terme d'une règle de la forêt de dérivation est un item de dérivation à la *Cocke-Kasami-Younger* (Kasami, 1965) de la forme suivante :

$$\langle Pos, N, i, j, p, q, Pile \rangle$$

où Pos indique si une adjonction est encore possible ($Pos ::= \top$) ou non ($Pos ::= \bot$), N

[66] ϵ représente le mot vide, utilisé pour marquer un argument non-réalisé.
[67] Voir aussi section 7.3.2 page 197.
[68] Où les règles symbolisent des arcs de dérivation.

désigne un nœud de l'arbre élémentaire où l'opération a lieu[69], i et j sont des indices entiers marquant la portion de phrase couverte par le nœud N (p et q sont des indices entiers utilisés dans le cas d'une adjonction pour marquer la portion de phrase couverte par l'arbre adjoint), et $Pile$ sert à empiler les nœuds d'adjonction.

L'algorithme de (Seddah, 2004) d'extraction du graphe de dépendance à partir d'une telle forêt, procède par inférence pour déduire des arcs de dépendance. Les arcs de dépendances considérés sont de trois formes :

$$\begin{array}{ll} \langle N, \alpha, \gamma, subst \rangle & (a) \\ \langle N, \beta, \gamma, adj \rangle & (b) \\ \langle N, \alpha, -, - \rangle & (c) \end{array}$$

(a) désigne une dépendance issue de la substitution d'un arbre α au nœud N de l'arbre γ. (b) marque une dépendance provenant de l'adjonction de l'arbre β au nœud N de l'arbre γ. (c) désigne la reconnaissance d'un arbre α de racine N et couvrant toute la phrase (calcul d'une racine du graphe de dépendance).

Pour calculer ces arcs, l'algorithme de (Seddah, 2004) utilise les règles d'inférence de la forme :

$$\frac{X \rightarrow Y}{Z}$$

où $X \rightarrow Y$ désigne une règle hors-contexte de la forêt de dérivation, et Z un arc de dépendance. Les règles utilisées ici pour déduire les arcs de dépendances sont les suivantes :

$$\frac{p \rightarrow \langle \top, N_\alpha, 0, n, -, -, [\emptyset] \rangle}{\langle N, \alpha, -, - \rangle} \tag{2.1}$$

$$\frac{\langle \bot, N_\gamma, i, j, -, -, Pile \rangle \rightarrow \langle \top, N_\alpha, i, j, -, -, Pile \rangle}{\langle N, \alpha, \gamma, subst \rangle} \tag{2.2}$$

$$\frac{\langle \bot, N_\gamma, i, j, p, q, Pile \rangle \rightarrow \langle \top, N_\beta, i, j, p, q, [N_\gamma, Pile] \rangle}{\langle N, \beta, \gamma, adj \rangle} \tag{2.3}$$

$$\frac{\langle \bot, N_\gamma, i, j, -, -, Pile \rangle \rightarrow Vrai}{\langle N, \alpha, X, subst \rangle} \tag{2.4}$$

La règle (2.1) est validée lorsque la racine d'une dérivation est parcourue. La règle (2.2) est validée lorsque la trace d'une substitution de l'arbre α au nœud N de l'arbre γ est visitée. (2.3) est déclenchée au passage d'une règle de dérivation marquant l'adjonction de l'arbre β au nœud N de l'arbre γ. La règle (2.4) permet de tenir compte des arguments non-réalisés en définissant un arc de dépendance du nœud N de l'arbre α vers une variable d'arbre X.

[69] N est donc un identifiant de nœud, cela peut-être une adresse de Gorn, ou un nom de nœud défini par le designer de la grammaire.

Tous les arcs produits lors du parcours de la forêt de dérivation ne décrivent pas encore un graphe de dépendance. En effet, les arguments non-réalisés sont représentés par des arcs de dépendance pointant *dans le vide* (variable d'arbre X ci-dessus). Il reste à identifier les variables d'arbre avec des arbres élémentaires présents dans la dérivation. Pour réaliser cela (Seddah, 2004) utilise une information issue du lexique. Plus précisément, l'auteur propose de marquer tout nœud de l'arbre associé à un verbe à contrôle, marquant un transfert d'argument au moyen d'un *canevas de contrôle* de la forme $N_{i\rightarrow j}$, désignant le fait que le i^e argument du verbe à contrôle est transféré au j^e argument du prédicat. Par exemple, dans la phrase « Jean espère dormir », le nœud (de l'arbre élémentaire associé au verbe à contrôle) désignant le 1er argument du verbe est marqué comme suit : $gn_{1\rightarrow 1}$ (le 1er argument du verbe à contrôle devient 1er argument du prédicat *dormir*).

A partir des arcs de dépendances extraits de la forêt de dérivation, et des canevas de contrôle présents dans les nœuds des arbres élémentaires, l'auteur définit la dernière étape du procédé, étape nommée *fusion argumentale*. Cette étape utilise une règle d'inférence permettant de remplacer un arc de dépendance comportant une variable d'arbre (arc (D_3) dans la règle ci-dessous) par un arc de dépendance complètement défini (*i.e.* où la variable d'arbre a été remplacée par une constante), par exemple :

$$\frac{(D_1)\langle N, \beta, \gamma, adj\rangle\ (D_2)\langle N_{i\rightarrow j}, \alpha, \beta, subst\rangle\ (D_3)\langle N_j, X, \gamma, subst\rangle}{\langle N, \beta, \gamma, adj\rangle\ \langle N_{i\rightarrow j}, \alpha, \beta, subst\rangle\ \langle N_j, \alpha, \gamma, subst\rangle} \tag{2.5}$$

Cette règle traduit le fait que si nous avons un arc marquant un lien (d'adjonction) entre les arbres β et γ d'une part, et que nous avons également un lien (de substitution) entre les arbres α et β à un nœud N étiqueté $i \rightarrow j$ d'autre part, et enfin que nous avons un arc *non-lié* partant du nœud étiqueté N_j de l'arbre γ, alors cet arc est remplacé par un arc marquant un lien (de substitution) entre les arbres α et γ au nœud N_j.

Si nous revenons sur l'exemple (4a) illustré figure 2.40, l'algorithme proposé ci-dessus extrait de la forêt de dérivation les quatres arcs de dépendance suivant[70] :

$$\langle e1_{1\rightarrow 1}, \tau_{Jean}, \tau_{espere}, subst\rangle \tag{2.6}$$

$$\langle d0, \tau_{espere}, \tau_{dormir}, adj\rangle \tag{2.7}$$

$$\langle d1, X, \tau_{dormir}, subst\rangle \tag{2.8}$$

$$\langle d0, \tau_{dormir}, -, -\rangle \tag{2.9}$$

La fusion argumentale permet de remplacer l'arc (2.8) par :

$$\langle d1, \tau_{Jean}, \tau_{dormir}, subst\rangle \tag{2.10}$$

Ce qui nous permet de construire le graphe de dépendance représenté figure 2.41.

Le procédé de construction d'un graphe de dépendance proposé par (Seddah, 2004) permet donc de traiter le cas des verbes à contrôle par extraction d'arcs de dépendance à

[70] Ici, nous identifions les nœuds par la première lettre du mot ancrant l'arbre concaténé avec l'adresse de Gorn du nœud.

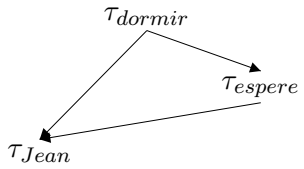


FIG. 2.41 – Graphe de dépendance de la phrase « Jean espère dormir ».

partir du parcours de la forêt de dérivation. Cette extraction utilise également l'information lexicale présente sur les nœuds des arbres élémentaires. Nous pouvons donc dire que cette approche utilise en fait à la fois l'arbre de dérivation et l'arbre dérivé, ces deux structures étant accessibles via la forêt de dérivation. Ce procédé a été étendu au cas des coordinations elliptiques (Seddah et Sagot, 2006).

Les inconvénients de cette approche résident dans (a) le fait que l'algorithme d'extraction du graphe de dépendance est fortement lié à l'algorithme d'analyse syntaxique produisant la forêt de dérivation, et (b) la nécessité d'introduire des canevas de contrôle. Pour annoter correctement les nœuds marquant un transfert argumental au sein des arbres élémentaires, il convient de mettre en place un système de nommage de nœud permettant de désigner le nœud recevant le transfert argumental. Enfin, il est difficile de prévoir dans quelle mesure ce procédé d'extraction d'un graphe de dépendance permet de traiter les cas d'ambiguïté de portée (comment produire plusieurs graphes à partir d'une analyse non-ambigue ?).

2.2.5.3 L'approche de Kallmeyer et Romero (2004, 2005) : construction sémantique à base d'unification sémantique

L'approche de (Kallmeyer et Romero, 2004; Romero et Kallmeyer, 2005) définit un cadre formel pour la construction sémantique, proche de celui de (Gardent et Kallmeyer, 2003) introduit précédemment[71]. Ce cadre associe à chaque arbre élémentaire une représentation sémantique composée de deux structures :

1. un ensemble de formules prédicatives étiquetées et de contraintes de portée de la forme $x \geq y$ où x, y réfèrent à des étiquettes ou à des variables correspondant aux *trous* de la sémantique plate de (Bos, 1995) (cette structure correspond à la représentation sémantique de (Kallmeyer et Joshi, 2003) sans le champ *arguments*) ;
2. une structure de traits sémantiques typée. Cette structure contient les valeurs de traits sémantiques associés à chaque nœud de l'arbre élémentaire[72]. Le lien avec un nœud de l'arbre se fait au moyen de son adresse de Gorn. Ainsi, cette structure de trait est définie comme suit :
   – la structure de traits sémantiques est de type *sem* ;

[71] Page 75.
[72] D'où l'analogie avec (Gardent et Kallmeyer, 2003).

– cette structure de traits contient un trait pour chaque nœud de l'arbre élémentaire, ce trait a la forme d'une adresse de Gorn[73]. La valeur de ce trait est une structure de traits de type *tb* (pour *top-bottom*) ;
– chaque structure de traits de type *tb* contient deux traits T et B de valeurs une structure de traits de type *bindings* ;
– une structure de traits de type *bindings* contient deux traits I et P de valeurs respectives une variable individuelle et une étiquette propositionnelle.

Un exemple de paires *(arbre élémentaire, représentation sémantique)* est donné figure 2.42.

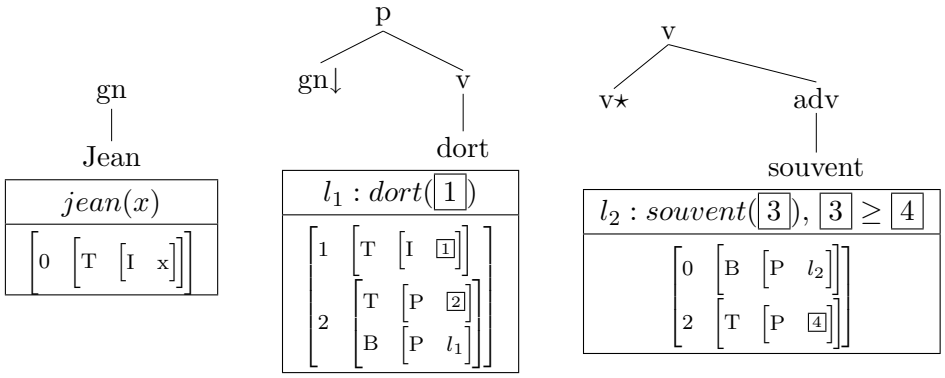


FIG. 2.42 – Arbres élémentaires et représentations sémantiques associées.

La construction sémantique est alors réalisée à partir d'une lecture de l'arbre de dérivation, lecture déclenchant l'unification des traits sémantiques associés à des nœuds dont les traits sont unifiés lors de la dérivation. Ainsi, pour chaque arc de l'arbre de dérivation liant le nœud associé à l'arbre élémentaire γ_1 à celui associé à γ_2 par combinaison (substitution ou adjonction) au nœud d'adresse p, les unifications sémantiques suivantes sont réalisées :

– les structures de traits sémantiques $\gamma_1.p.T$ et $\gamma_2.0.T$ sont unifiées (cette unification est le pendant des unifications induites par l'opération de substitution et d'une partie des unifications induites par l'adjonction, *cf* section 2.1.2) ;
– si γ_2 est un arbre auxiliaire, alors les structures de traits sémantiques $\gamma_1.p.B$ et $\gamma_2.f.B$ (f représente l'adresse de Gorn du nœud pied) sont unifiées. Ce qui correspond à l'unification des structures *bottom* lors de l'opération d'adjonction.

En plus de ces unifications, les unifications équivalentes à celles des structures *top* et *bottom* de chaque nœud, réalisées à l'issue de la dérivation, sont déclenchées comme suit :
pour chaque nœud γ de l'arbre de dérivation, pour chaque adresse de nœud p tel que p n'étiquette aucun arc de l'arbre de dérivation, $\gamma.p.T$ et $\gamma.p.B$ sont unifiées.

A l'issue de ces unifications, certaines variables sémantiques se voient assignées une valeur, d'autres restent indéterminées. Ces variables *libres* sont alors interprétées comme des variables de trous dans la sémantique plate de (Bos, 1995).

[73]Le « . » est omis, ainsi le nœud $i.j$ est noté ij.

Enfin, la représentation sémantique finale de la phrase représentée par l'arbre de dérivation correspond à l'*union* des formules sémantiques sous-spécifiées associées à chacun des nœuds de l'arbre de dérivation. Cette représentation sémantique doit être résolue pour déterminer l'ensemble des représentations sémantiques en cas d'ambiguïté (*cf* section 1.2.1.2 page 26).

Un exemple Dans un premier temps, intéressons nous à l'exemple « Jean dort souvent ». Les arbres élémentaires permettant d'analyser cette phrase sont donnés figure 2.42 présentée précédemment. L'arbre de dérivation pour cette phrase est représenté figure 2.43.

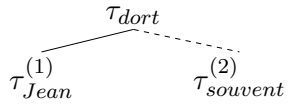


FIG. 2.43 – Arbre de dérivation de « Jean dort souvent ».

Les unifications réalisées dans le cadre de cette dérivation sont les suivantes[74] :

$$(\tau_{\mathbf{Jean}}.\mathbf{0}.\mathbf{T}) \sqcup (\tau_{\mathbf{dort}}.\mathbf{1}.\mathbf{T}) \text{ (substitution)}$$

$$(\tau_{\mathbf{souvent}}.\mathbf{0}.\mathbf{T}) \sqcup (\tau_{\mathbf{dort}}.\mathbf{2}.\mathbf{T}) \text{ (adjonction – 1)}$$

$$(\tau_{\mathbf{souvent}}.\mathbf{2}.\mathbf{B}) \sqcup (\tau_{\mathbf{dort}}.\mathbf{2}.\mathbf{B}) \text{ (adjonction – 2)}$$

$$(\tau_{\mathbf{Jean}}.\mathbf{0}.\mathbf{T}) \sqcup (\tau_{\mathbf{Jean}}.\mathbf{0}.\mathbf{B}) \text{ (unification top-bottom)}$$

$$(\tau_{\mathbf{souvent}}.\mathbf{0}.\mathbf{T}) \sqcup (\tau_{\mathbf{souvent}}.\mathbf{0}.\mathbf{B}) \text{ (unification top-bottom)}$$

$$(\tau_{\mathbf{souvent}}.\mathbf{2}.\mathbf{T}) \sqcup (\tau_{\mathbf{souvent}}.\mathbf{2}.\mathbf{B}) \text{ (unification top-bottom)}$$

Les trois premières unifications sont issues directement de la lecture des arcs de l'arbre de dérivation (unifications induites par les opérations de substitution et d'adjonction). Les trois dernières unifications correspondent à l'unification des traits T et B de chacun des nœuds non-utilisés pour marquer une dérivation. Nous pouvons représenter les unifications ainsi réalisées sur l'arbre de dérivation comme indiqué figure 2.44[75].

A l'issue de ces unifications, les variables des représentations sémantiques se voient attribuées les valeurs suivantes :

$$x \leftrightarrow \boxed{1} \quad l_2 \leftrightarrow \boxed{2} \quad l_1 \leftrightarrow \boxed{4}$$

La variable $\boxed{3}$ reste libre. La représentation sémantique (sous-spécifiée) ainsi calculée correspond à l'union des formules élémentaires :

$$l_1 : dort(x),\ jean(x),\ l_2 : souvent(\boxed{3}),\ \boxed{3} \geq l_1$$

[74] Le symbole $\sqcup$ représente l'unification.

[75] Les lettres encadrées marquent les structures de traits unifiées entre elles.

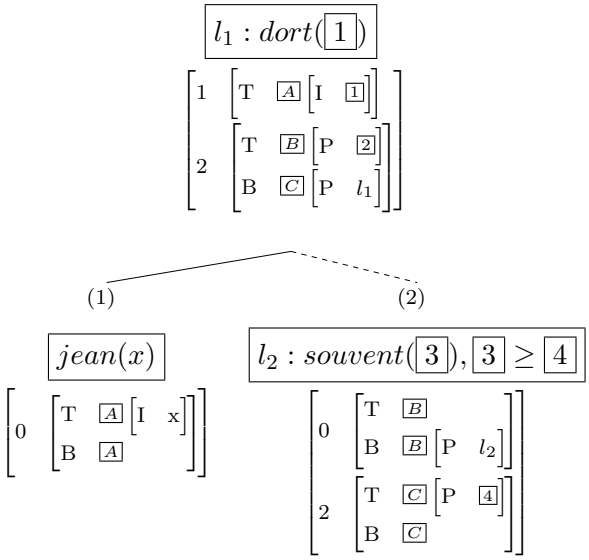


FIG. 2.44 – Arbre de dérivation avec traits sémantiques pour la phrase « Jean dort souvent ».

Pour obtenir les représentations sémantiques décrites par cette formule, il faut définir une assignation des variables (de trous) vers les étiquettes respectant les contraintes de portée. Dans le cas présent, la seule assignation est $l_1 \leftrightarrow \boxed{3}$, ce qui produit la représentation sémantique (attendue) :

$$l_1 : dort(x),\ jean(x),\ l_2 : souvent(l_1)$$

En conclusion, la proposition de (Kallmeyer et Romero, 2004) que nous venons de présenter, définit un procédé de construction sémantique compositionnel à partir de l'arbre de dérivation et d'informations issues de l'arbre dérivé (en l'occurrence les unifications entre structures de traits étiquetant les nœuds des arbres élémentaires impliqués). Cette proposition est très proche de (Gardent et Kallmeyer, 2003) à la différence près que les traits sémantiques sont extraits des arbres élémentaires et stockés dans une structure de traits dédiée. L'avantage de cette approche est de ne pas étendre le formalisme TAG, en particulier de ne pas étiqueter les nœuds des arbres élémentaires par des traits dont la valeur appartient à un ensemble potentiellement infini (cas des variables d'étiquette).

Notons que cette approche permet de traiter l'ensemble des phénomènes supportés par (Gardent et Kallmeyer, 2003), en particulier les quantificateurs, les adverbes, les verbes à montée, les questions (voir (Romero et al., 2004)), et les adjectifs enchâssés.

2.3 Conclusion

Dans ce chapitre, nous avons présenté les grammaires d'arbres adjoints et les principales méthodes de construction sémantique pour celles-ci. Les différentes approches de cal-

cul sémantique se distinguent de plusieurs points de vue. Tout d'abord elles diffèrent par la structure syntaxique sur laquelle elles s'appuient (arbre de dérivation ou arbre dérivé). Elles diffèrent ensuite par la forme de la représentation sémantique calculée (formule de logique du 1er ordre, formule de sémantique plate sous-spécifiée, graphes de dépendances). En outre, certaines de ces approches se placent dans un cadre formel plus large (encodage de la grammaire dans le formalisme des ACG), ou demandent un enrichissement de la grammaire TAG (ajout d'un canevas de contrôle, d'annotations sémantiques, ou de traits sémantiques sur les nœuds) ou du résultat de l'analyse syntaxique (ajout d'arcs ou de structures de traits dans les arbres de dérivation). Enfin, ces approches diffèrent par les phénomènes linguistiques pour lesquels elles permettent le calcul d'une représentation sémantique correcte.

Cette présentation des diverses approches de calcul sémantique en TAG clôt ce chapitre et cette partie d'état de l'art. Dans le cadre de cette thèse, dont le sujet vise à proposer une architecture pour la construction sémantique en TAG pour des grammaires de taille réelle, nous nous sommes basés sur l'approche de (Gardent et Kallmeyer, 2003) pour les raisons suivantes :

- cette approche propose un cadre formel uniforme et bien défini pour la construction sémantique[76] en utilisant les structures de traits étiquetant les nœuds des arbres TAG pour définir une interface syntaxe / sémantique, et l'opération d'unification pour le calcul sémantique à proprement parler ;
- cette approche ne demande pas d'extension significative de la tâche d'analyse syntaxique, puisque le calcul sémantique ne fait intervenir aucun procédé additionnel (tel que l'ajout de liens ou de traits dans l'arbre de dérivation par exemples) autre que l'unification des structures de traits des nœuds des arbres élémentaires ;
- cette approche est relativement souple par rapport au type d'arbre manipulé dans la grammaire, du fait de l'utilisation des structures de traits définir pour l'interface syntaxe / sémantique.

Dans ce contexte, notre proposition d'architecture pour le calcul sémantique en TAG repose sur (a) l'annotation sémantique des arbres de la grammaire (ajout de traits et de formules sémantiques), tâche qui peut s'avérer très complexe sur une grammaire de taille réelle, et (b) le développement d'un procédé d'analyse syntaxique permettant non seulement de produire le (ou les) arbre(s) dérivé(s) correspondant à un énoncé donné, mais également de manipuler et mettre à jour les formules sémantiques élémentaires associés aux arbres en fonction des unifications qui ont lieu, pour produire une représentation sémantique sous-spécifiée à la (Bos, 1995).

Le point (a) est traité dans la seconde partie de ce document, où nous présentons un procédé de production semi-automatique de grammaires, permettant l'annotation sémantique des arbres élémentaires. Le point (b) correspond à la troisième partie de cette thèse, où nous introduisons un procédé de calcul sémantique pour l'approche de (Gardent et Kall-

[76] Ce qui est également le cas de (Kallmeyer et Romero, 2004) et (Pogodalla, 2004a).

meyer, 2003) basé sur deux types d'analyseurs syntaxiques, un analyseur produisant des arbres dérivés, et un analyseur produisant une forêt de dérivation.

Deuxième partie

Compilation de Grammaires d'Arbres Adjoints à portée sémantique

La grammaire, avec son mélange de règle logique et d'usage arbitraire, propose au jeune esprit un avant-goût de ce que lui offriront plus tard les sciences de la conduite humaine, le droit ou la morale, tous les systèmes où l'homme a codifié son expérience instinctive.

Marguerite Yourcenar, Mémoires d'Hadrien

Chapitre 3

La production semi-automatique de grammaires TAG

Sommaire

Dans le cadre de nos travaux sur la construction sémantique en TAG, la première ressource dont nous avons besoin est une grammaire annotée sémantiquement. L'écriture d'une grammaire de taille réaliste est une tâche longue et complexe. En partant de ce constat, plusieurs théories visant la production semi-automatique de grammaires (et notamment de grammaires d'arbres) ont été élaborées. Dans ce chapitre, nous revenons sur les problèmes inhérents au développement de grammaires, introduisons le concept de *métagrammaire*, puis nous donnons un bref aperçu des systèmes existants, avant de conclure en donnant les

analogies et différences de ces systèmes métagrammaticaux[77].

3.1 Motivations

3.1.1 Problèmes liés au développement de grammaires

Les difficultés liées à l'écriture et de la maintenance de grammaires n'est pas un problème nouveau, et ont déjà suscité bon nombre d'attentions (Erbach et Uszkoreit, 1990).

Parmi les limitations les plus importantes au développement de grammaires, nous pouvons citer :

- le temps de développement important ;
- la difficulté de conserver la cohérence entre règles lorsque la grammaire atteint une certaine taille ;
- la difficulté de modulariser le développement ;
- la difficulté de représenter des concepts linguistiques généraux et traitant des exceptions ;
- les problèmes liés à l'évaluation de la grammaire.

Ces difficultés mettent en évidence la nécessité de disposer d'outils adéquats, tels que des éditeurs de grammaires, des analyseurs syntaxiques et autres suites de tests.

3.1.2 Vers une production semi-automatique de grammaires

Il existe plusieurs possibilités pour faciliter le développement de grammaires, parmi celles-ci nous pouvons citer l'extraction automatique de grammaires à partir de corpus (éventuellement arborés), ou encore l'utilisation d'un niveau d'abstraction (Kinyon et Prolo, 2002).

Ici, nous nous intéressons tout particulièrement à cette deuxième approche. L'abstraction en question correspond à une description des règles de la grammaire, appelée *méta-grammaire*, et à partir de laquelle il doit être possible d'automatiser la génération de la grammaire.

Les intérêts principaux des métagrammaires sont les suivants[78] :

(i) éviter la redondance et les incohérences entre règles au moyen d'un partage important de l'information,

(ii) permettre une généralisation linguistiquement motivée de l'information contenue dans les règles.

[77] Pour une présentation détaillé des systèmes de production semi-automatique de grammaires d'arbres, le lecteur pourra consulter (Crabbé, 2005b).

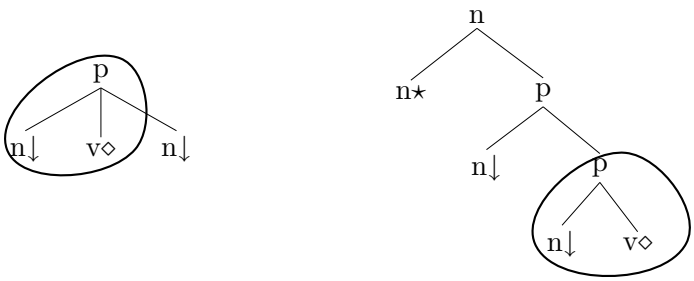
[78] Comme le mentionnent (Kinyon et Prolo, 2002), il serait envisageable d'acquérir automatiquement cette description abstraite, mais peu d'études sur cette possibilité ont eu lieu à ce jour.

3.1.3 Le cas des grammaires d'arbres lexicalisées

Considérons un cas particulier de grammaires, à savoir les grammaires d'arbres, dans leur version lexicalisée. Comme nous l'avons vu au chapitre 2, de telles grammaires se distinguent par le fait que chaque règle (*i.e.* chaque arbre) se voit associée à (au moins) un mot du lexique. Dans ce cas, la grammaire peut être vue comme une fonction associant à un mot du lexique un ensemble d'arbres représentant le contexte d'utilisation de ce mot dans une phrase.

L'une des conséquences de la lexicalisation de la grammaire est qu'un même arbre peut être employé un grand nombre de fois, créant une grande redondance dans la grammaire.

De plus, dans une grammaire d'arbres de taille réelle, il y a un nombre important de règles dont certaines partagent une sous-structure commune (voir figure 3.1 représentant l'utilisation de verbe transitif *manger* dans un contexte de sujet sous forme canonique et extraite).



Jean *mange une pomme*

La pomme que **Jean** *mange*

FIG. 3.1 – Redondance dans les règles de la grammaire.

Ces deux phénomènes combinés font que, lorsque l'on souhaite modifier l'information contenue dans les sous-structures communes (par exemple pour changer la représentation de l'accord sujet-verbe), cela implique la modification de nombreux arbres. Ce qui, en plus d'être fastidieux, nous ramène au problème de la cohérence des règles.

Dans la section qui suit, nous présentons un premier procédé de factorisation de grammaire TAG permettant d'éviter cette redondance.

3.2 Représentation factorisée d'une grammaire TAG

3.2.1 Fragments d'arbres et héritage

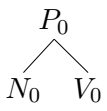
Une première tentative de réponse à ce problème de redondance a été donnée par (Vijay-Shanker et Schabes, 1992) où les auteurs proposent de représenter une grammaire TAG au moyen d'une hiérarchie d'héritage entre fragments d'arbres.

En bas de cette hiérarchie se trouvent les mots qui héritent chacun des fragments nécessaires à la construction des arbres élémentaires qui leur sont associés.

Au sein de la hiérarchie, un fragment est encapsulé dans une classe contenant les informations suivantes :

- **Super-classes :** liste des classes situées *immédiatement* en amont dans la hiérarchie.
- **Noeuds :** liste des nœuds associés à la classe.
- **Description :** description du fragment d'arbre correspondant à la classe. Cette description fait appel à une logique de description d'arbres à base de relations de dominance. Cette logique n'impose pas d'ordre entre nœuds frères.
- **Equations sur les nœuds :** équations représentant des contraintes d'unification sur les traits portés par chaque nœud.
- **Nœud argument :** définition du nœud pour lequel la classe apporte un argument.
- **Précédence linéaire :** définition de l'ordre des nœuds frères dans le fragment d'arbre.
- **Ancre :** Nœud père du nœud où va être inséré le mot ancrant la structure syntaxique.

Un exemple de classes est donné figure 3.2 (les descriptions utilisent des variables de nœuds homonymes aux catégories syntaxiques étiquetant les nœuds).

VERBE-INTRANS
Noeuds : P_0, V_0, N_0
Description : 
Equations : N_0.<cat> = n,
N_0.<accord> = V_0.<accord> ...
Noeud argument : N_0
Précédence : $N_0 < V_0$
ancre = V_0

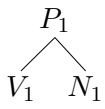
VERBE-TRANS
Super-classe : VERBE-INTRANS
Noeuds : P_1, V_1, N_1
Description : 
Equations : N_1.<cat> = n ...
Noeud argument : N_1
Précédence : $V_1 < N_1$
ancre = V_1

FIG. 3.2 – Informations de la classe des verbes transitifs.

La classe de droite, *VERBE-TRANS*, hérite l'information de celle de gauche, *VERBE-INTRANS*, qu'elle complète en ajoutant le nœud pour l'objet direct. De plus, la classe *VERBE-INTRANS* gère l'accord entre le sujet et le verbe au moyen d'équations de traits.

Dans ce contexte, pour construire l'arbre associé à un mot, on procède comme suit :

1. on sélectionne la classe finale associée au mot,
2. on récupère les descriptions d'arbres des classes héritées directement ou indirectement (héritage de l'information structurelle)[79],

[79] Cela peut nécessiter un renommage des variables de noeuds dans le cas de descriptions partielles employant les mêmes noms de variables de nœuds.

3. on identifie les noeuds de type *ancre* de chacune de ces descriptions, ce qui entraîne l'identification du père du nœud ancre, et par suite, nous pouvons déterminer l'arbre correspondant à la conjonction des descriptions des classes héritées.

Notons que les descriptions d'arbre utilisées ici peuvent inclure une relation de dominance immédiate entre nœuds, de dominance stricte, ou de dominance large[80]. Si cette dernière est utilisée, on parle alors de *quasi-arbre* (Rogers et Vijay-Shanker, 1992). Ce quasi-arbre représente un ensemble d'arbres, qui diffèrent par la distance entre deux nœuds en relation de dominance. Dans notre cas, parmi tous les arbres de cet ensemble, ceux qui nous intéressent sont les *référents minimaux*, c'est-à-dire ceux où les relations de dominance dites larges ont été résolues soit sous forme d'identifications de nœuds (lorsqu'il est possible d'unifier les traits associés à chacun des nœuds) soit sous forme de dominances immédiates. De manière intuitive, les référents minimaux correspondent donc aux arbres dont le nombre de dominances immédiates est minimal (*cf* figure 3.3[81]).

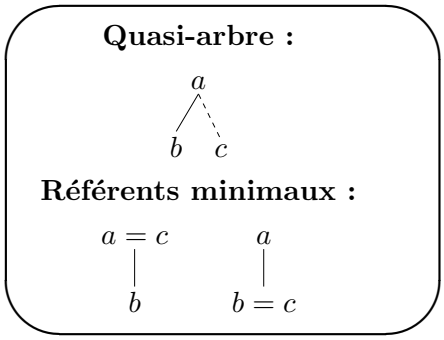


FIG. 3.3 – Référents minimaux d'un quasi-arbre.

Si nous revenons à notre exemple des verbes transitifs (*cf* figure 3.2), l'extraction de l'arbre associé au verbe *aimer* se fait comme suit :

1. récupération de la classe *VERBE-TRANS*,
2. récupération de la classe *VERBE-INTRANS* par héritage,
3. identification des nœuds ancre de chacune de ces classes, ainsi la variable V_0 est identifiée avec la variable V_1, ce qui entraîne l'identification de leurs nœuds père respectifs.

A l'issue de l'extraction, la structure produite est celle de la figure 3.4.

Remarque la description factorisée de (Vijay-Shanker et Schabes, 1992) utilise également un système de règles lexicales permettant d'associer à la description d'une classe, une nouvelle description. Cela est utilisé par exemple pour représenter les transformations liées à l'alternance actif / passif.

[80] On appelle dominance immédiate la relation de parenté, la dominance stricte correspond à la clôture transitive de la dominance et la dominance large à la clôture transitive et réflexive de la dominance.

[81] La relation de dominance large est notée en pointillées.

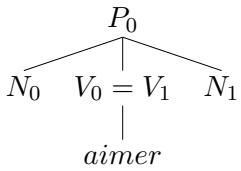


FIG. 3.4 – Arbre associé au verbe *aimer*.

3.2.2 Langages de représentation

Nous venons de voir un procédé de factorisation d'une grammaire en considérant des fragments d'arbres placés au sein d'une hiérarchie d'héritage. Pour pouvoir développer une telle grammaire factorisée, nous avons besoin d'un langage informatique permettant de représenter cette factorisation. Différents langages de représentation ont été développés pour les grammaires d'unification. Parmi les plus connus, nous pouvons citer PATR II (Shieber, 1984) et DATR (Evans et Gazdar, 1996).

Concernant le problème plus particulier de la représentation d'une grammaire TAG au moyen de ces langages, nous pouvons citer (Evans et al., 1995). Dans leur approche, les auteurs encodent une grammaire TAG dans un lexique au format DATR, en utilisant les idées de (Vijay-Shanker et Schabes, 1992).

L'encodage est ascendant, en d'autres termes, chaque arbre élémentaire de la grammaire est décrit en partant de son ancre et en utilisant les relations binaires *parent*, *gauche* et *droite* (précédence immédiate). Prenons l'exemple de l'arbre associé au verbe *aimer* avec arguments réalisés sous forme canonique (voir figure 3.5[82]). L'entrée lexicale associée à cet arbre est la suivante :

```
Aimer:
  <cat> == v
  <parent cat> == p
  <left cat> == gn
  <right cat> == gn.
```

Elle traduit le fait que le nœud ancre de l'arbre est de catégorie v, sont nœud père de catégorie p, et ses frères gauche et droit de catégorie gn.

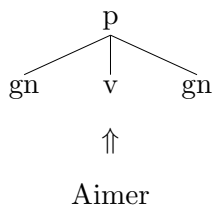


FIG. 3.5 – Arbre élémentaire associé au verbe *aimer* avec arguments sous forme canonique.

[82]Le nœud étiqueté par un losange correspond au nœud ancre.

En outre, il est possible de définir une entrée lexicale comme héritant d'une autre (champ `<> ==` sur l'exemple ci-dessous). Ainsi, nous pouvons abstraire la structure des verbes transitifs avec arguments sous forme canonique en définissant un ensemble d'entrées lexicales en relation d'héritage comme illustré figure 3.6 ci-dessous[83].

```
NODE:
  <> == undef
  <type> == internal.

ROOTNODE:
  <> == NODE
  <cat> == P.

VERB:
  <> == NODE
  <cat> == v
  <type> == anchor
  <parent> == ROOTNODE:<>.

GN:
  <> == NODE
  <cat> == gn
  <type> == subst.

VERB-INTRANS:
  <> == VERB
  <left> == GN:<>.

VERB-TRANS:
  <> == VERB-INTRANS
  <right> == GN:<>.

Aimer:
  <> == VERB-TRANS
  <root> == aimer.
```

FIG. 3.6 – Entrées lexicales au format DATR

Dans ce lexique, l'entrée `NODE` définit l'information associée à un nœud interne. Par héritage, cette entrée est utilisée pour définir l'entrée `ROOTNODE` désignant un nœud racine d'un arbre. Cette entrée `ROOTNODE` est elle-même utilisée par l'entrée `V` pour spécifier la place du nœud par rapport à la racine, etc.

Pour construire l'entrée associée au lemme *aimer*, il faut hériter de l'information associée à l'entrée `VERB-TRANS`, qui elle-même hérite de `VERB-INTRANS`. Ces relations d'héritage entraînent les unifications des équations de chemin encodées dans les différentes entrées. Ainsi, l'héritage entre `VERB` et `VERB-INTRANS` produit l'entrée suivante :

```
<cat> == v
<type> == anchor
<left cat> == gn
<left type> == subst
```

En d'autres termes, l'héritage ajoute un nœud à gauche de l'ancre, nœud correspondant à l'ancre de l'entrée lexicale `GN`. La hiérarchie d'héritage considérée ici est représentée figure 3.7.

Il convient de noter deux points :

[83] Précisons que le champ `<root>` est utilisé pour définir la valeur de l'ancre lexicale.

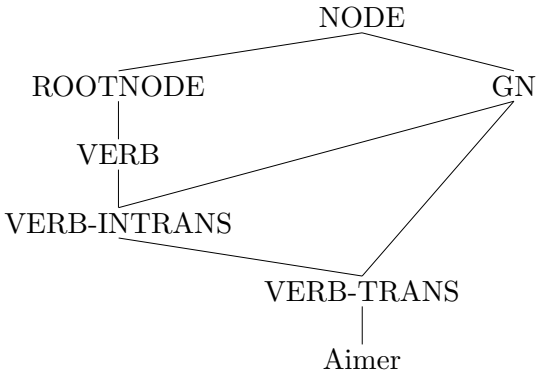


FIG. 3.7 – Hiérarchie d'héritage encodant un lexique TAG jouet.

1. la relation d'héritage est *non-monotone*. Certains champs d'une entrée lexicale se voient attribués une valeur par défaut, lorsque cette entrée est impliquée dans une relation d'héritage. En conséquence, l'ordre dans lequel les relations d'héritage sont résolues a un impact sur la forme de l'entrée lexicale décrite.
2. les descriptions d'arbres sont encodées au moyen des relations de parenté, frère gauche et frère droit (précédence immédiate), c'est-à-dire qu'il n'y a pas de sous-spécification. La factorisation est donc moindre que dans l'approche de (Vijay-Shanker et Schabes, 1992).

Notons enfin que la méthodologie d'encodage d'une grammaire TAG au moyen du langage DATR proposée par (Evans et al., 1995) utilise également un système de règles lexicales afin de dériver de nouvelles entrées lexicales par transformation d'entrées lexicales existantes.

L'inconvénient de cette approche réside dans la non-monotonie de l'héritage qui rend difficile la définition de grammaires de taille réelle.

3.3 Les systèmes métagrammaticaux existants

Jusqu'ici, nous avons introduit les idées à la base de la production semi-automatique de grammaires TAG par abstraction, à savoir *factorisation* du lexique et manipulation de descriptions d'arbres. Ces idées ont été utilisées dans le développement de langages de représentation spécifiques aux grammaires TAG. Dans cette section, nous présentons ces langages et leur implantation.

3.3.1 Le compilateur de métagrammaire de Paris 7 (Marie-Hélène Candito)

La première implantation d'un système de production semi-automatique de grammaire TAG est à mettre à l'actif de l'Université Paris 7, dans le cadre de (Candito, 1999). Ces

travaux sont à l'origine du terme *métagrammaire*. L'auteur reprend l'idée de factorisation du lexique émise par (Vijay-Shanker et Schabes, 1992), et émet les desideratas suivants :

– la description d'une grammaire TAG doit être modulaire et hiérarchisée ;
– la hiérarchie utilisée doit rendre compte de propriétés linguistiques.

Dans ce contexte, M.H. Candito définit une description abstraite de la grammaire exprimée au moyen d'une hiérarchie tri-dimensionnelle de classes (Candito, 1996) :

- les classes de la dimension 1 définissent *les cadres de sous-catégorisation initiaux*. Ces cadres définissent le nombre et le type des arguments d'un verbe (*e.g.*, *n0Vn1* pour les verbes transitifs avec arguments sous forme canonique) ;
- la dimension 2 contient les définitions des *redistributions des fonctions syntaxiques*. Ces redistributions associent de nouvelles fonctions syntaxiques aux arguments du verbe tels que définis en dimension 1. Cela permet par exemple de représenter l'alternance actif / passif ;
- la dimension 3 contient les différentes réalisations syntaxiques de chacune des fonctions grammaticales de la dimension 2 (par exemple sujet canonique, sujet extraposé, etc).

Formellement, les classes des différentes dimensions peuvent contenir deux types d'informations : des *équations de traits* et des *descriptions d'arbres*. Ces descriptions prennent la forme d'une formule de logique de description d'arbres, en l'occurrence la logique utilisée est celle de (Rogers et Vijay-Shanker, 1994). Cette logique utilise un langage de description à base de dominance immédiate, de dominance stricte, de dominance large, de précédence immédiate, et d'identité entre variables de nœuds. On remarque en outre que les formules des différentes classes utilisent des variables de nœud dont la portée est *globale*[84].

A partir de cette description, qui constitue la métagrammaire, M.H. Candito définit un algorithme de compilation permettant de construire les arbres[85] de la grammaire. Cet algorithme réalise le *croisement*[86] de classes suivant :

1. une classe C^1 de la dimension 1 est sélectionnée,
2. elle est croisée avec une classe C^2 de la dimension 2, ce qui produit un cadre de sous-catégorisation dans lequel chaque argument du verbe se voit attribué une fonction grammaticale,
3. $C^1 \times C^2$ est alors croisé avec les classes C^3_i ($i \in [0..n]$, n est le nombre d'arguments du verbe) de la dimension 3, représentant la réalisation de chacun des arguments.

Ce croisement est effectué pour chaque classe de la dimension 1, et produit une conjonction de descriptions d'arbres. Rappelons que les descriptions utilisent des variables de nœud à portée globale, ce qui implique que deux occurrences d'une même variable de nœud désignent

[84] Nous revenons sur ce point dans la suite de nos propos.
[85] Pour être plus précis, l'algorithme produit des arbres non-ancrés appelés *schèmes*, voir aussi chapitre 6.
[86] Par croisement de classe, nous entendons prendre la conjonction des descriptions d'arbre, et réaliser les équations de traits associées aux classes.

le même nœud. Ceci a un impact non-négligeable dans la définition du contenu des classes de la métagrammaire.

Enfin, une fois les croisements effectués, on calcule les référents minimaux de chacune des descriptions d'arbres produites par croisement, ces référents correspondent aux arbres de la grammaire.

La figure 3.8 illustre ce croisement et montre un arbre ainsi généré.

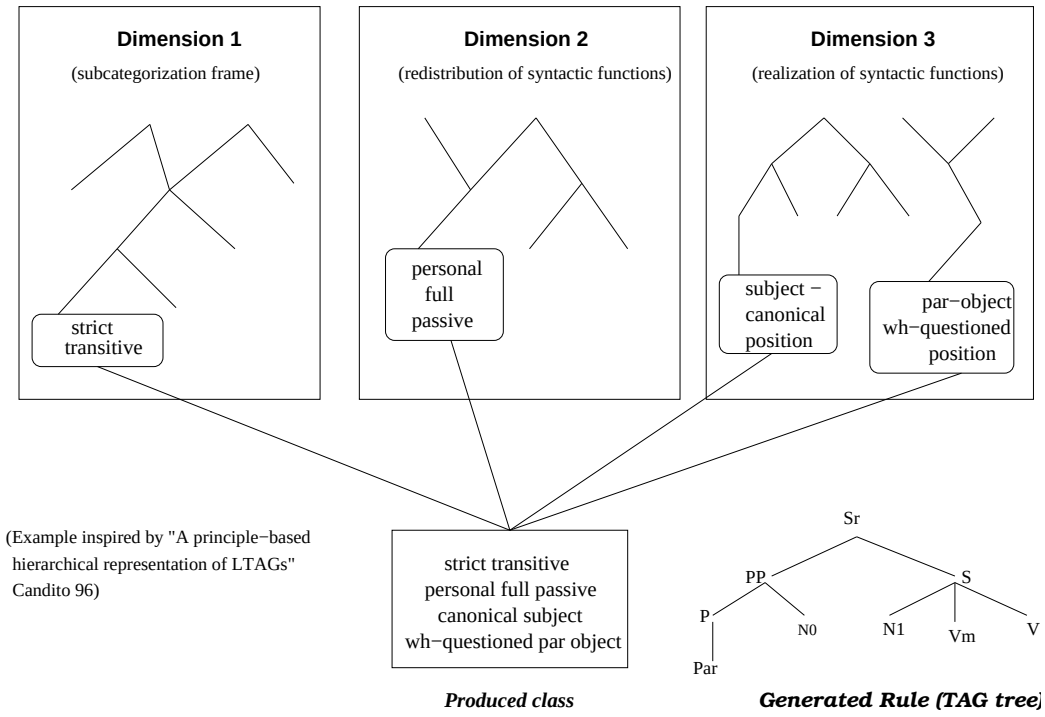


FIG. 3.8 – Hiérarchie de classes dans la métagrammaire de M.H. Candito.

Les inconvénients de cette métagrammaire sont (a) que la forme de la description abstraite de la grammaire et l'algorithme de compilation de celle-ci sont fortement liés, ce qui limite le type de généralisation linguistique que l'on peut émettre, (b) l'algorithme de croisement est non-monotone dans le sens où le croisement avec une classe donnée peut provoquer l'effacement d'une autre classe (ce qui est utilisé pour représenter le passif sans agent), ce qui peut rendre la définition de la description plus complexe, (c) les descriptions d'arbre utilisent des variables globales, ce qui implique une gestion des noms de variables qui peut s'avérer délicate sur des métagrammaires de taille importante, et (d) la forte combinatoire liée aux croisements pose des problèmes en termes d'efficacité et de prédiction des arbres produits (manque de contrôle).

3.3.2 Le système de UPenn (Fei Xia)

Un second système de production semi-automatique de grammaires TAG à partir des idées de (Vijay-Shanker et Schabes, 1992) a été proposé par (Xia, 2001)[87]. Comme nous allons le voir, Ce système partage certaines des caractéristiques de (Candito, 1999).

Dans cette approche, la description métagrammaticale est composée de trois constituants :

1. un ensemble de *cadres de sous-catégorisation canoniques*. Ces cadres encodent la catégorie de l'ancre principale, le nombre, la position, et la catégorie des arguments de l'ancre, ainsi que des équations de traits.
2. un ensemble de *blocs élémentaires*. Ces blocs peuvent être de deux types : les blocs de sous-catégorisation, et les blocs de transformations. Chacun de ces blocs contient une description d'arbres exprimée dans la logique de (Rogers et Vijay-Shanker, 1994), mais où les variables de nœuds ont une portée *locale*. Les blocs de transformations se distinguent des blocs de sous-catégorisation dans la mesure où ils contiennent l'information liée à des structures syntaxiques particulières (mouvements à longue distance de la conjonction interrogative par exemple).
3. un ensemble de *règles lexicales de redistribution* (RLR), ces règles permettent de dériver de nouveaux cadres de sous-catégorisation à partir des cadres canoniques.

A partir de cette métagrammaire, les arbres de la grammaire sont produits par l'algorithme suivant :

1. sélection d'un cadre de sous-catégorisation canonique ;
2. application des RLR sur ce cadre pour produire de nouveaux cadres de sous-catégorisation ;
3. combinaisons[88] des blocs (de sous-catégorisation puis de transformation) correspondant aux cadres générés.

Ce procédé est réitéré pour chaque cadre de sous-catégorisation canonique.

Il est important de noter que (i) l'ordre d'application des RLR a un impact sur les structures produites, et (ii) certaines RLR vont supprimer l'information contenue dans certains cadres canoniques (cas du passif sans agent), on retrouve le problème de non-monotonie de la description de (Candito, 1999).

Chaque cadre canonique définit une *famille d'arbres*, pour laquelle l'algorithme ci-dessus produit un conjonction de descriptions d'arbres. On remarque que, les variables de nœuds utilisées ayant une portée locale, chaque élément d'une conjonction de description est indépendante des autres, deux variables homonymes apparaissant dans deux descriptions différentes ne sont pas interprétées comme désignant le même nœud. La conséquence de cela est, comme le notent (Crabbé et Duchier, 2004), la possibilité de sur-générer lors de

[87] Voir aussi (Xia et al., 1998; Xia et al., 1999).

[88] Par combinaison, nous entendons la conjonction des descriptions et la résolution des équations de traits éventuelles.

la recherche des référents minimaux correspondants (les seuls contraintes dont on dispose sont celles de bonne formation de l'arbre et les annotations des nœuds – telles que traits, marque d'ancre, etc – par opposition à (Candito, 1999) qui utilise en plus les noms des variables de nœud).

L'approche de (Xia, 2001), tout comme celle de (Candito, 1999), propose un cadre métagrammatical (langage et algorithme de compilation) permettant de produire semi-automatiquement une grammaire TAG. Ces deux approches se sont intéressées principalement aux verbes et utilisent toutes les deux le langage logique de descriptions d'arbres de (Rogers et Vijay-Shanker, 1994). La distinction majeure entre ces deux approches concerne le niveau de contrôle dans la combinaison des descriptions d'arbres élémentaires. (Xia, 2001) propose un contrôle au niveau des règles lexicales de redistribution, alors que (Candito, 1999) utilise les noms des variables de nœuds. Cette dernière combine des classes pour lesquelles aucun référent minimal ne pourra être calculé, ce que tente d'éviter (Xia, 2001) en contrôlant plus finement quels descriptions vont être combinés.

3.3.3 Le compilateur de métagrammaire du LORIA (Bertrand Gaiffe)

En partant du constat que l'approche de (Candito, 1999) (1) manquait de contrôle dans la définition des croisements de classe, et (2) que la forme de la description métagrammaticale basée sur trois dimensions était trop rigide, (Gaiffe et al., 2002) ont proposé un nouveau système de compilation de métagrammaire.

Les innovations de cette proposition sont nombreuses. Tout d'abord, ce système permet de définir un nombre quelconque de dimensions (*i.e.*, d'hiérarchies d'héritage de classe), il introduit un système de contrôle des croisements de classes basé sur une notion de *besoins / ressources*, et enfin il permet de manipuler des *hypertags* (Kinyon, 2000), c'est-à-dire d'associer à chaque arbre produit une structure de traits globale.

Dans leur approche, la métagrammaire représente donc une (ou plusieurs) hiérarchie(s) d'héritage de classes, où chaque classe contient :

- un nom ;
- un ensemble de super-classes ;
- une structure de traits (servant à construire l'hypertag) ;
- une description d'arbre dans la logique de (Rogers et Vijay-Shanker, 1994) (dans cette description les variables de nœud ont une portée globale à l'ensemble de la métagrammaire) ;
- un ensemble de besoins (symboles atomiques) que la classe nécessite pour permettre la description d'un arbre TAG ;
- un ensemble de ressources (symboles atomiques) que la classe fournit.

Un besoin (respectivement ressource) qu'une classe demande (respectivement fournit) peut par exemple correspondre à la réalisation syntaxique d'une fonction grammaticale donnée.

La relation d'héritage est un héritage *multiple* dont la sémantique est telle que, lorsqu'une classe c_2 hérite d'une classe c_1, une classe c_{12} est créée où :

- la liste des super-classes correspond à l'union des super-classes de c_1 et c_2,
- les structures de traits de c_1 et c_2 sont unifiées,
- la description d'arbre correspond à la conjonction des descriptions d'arbres de c_1 et c_2,
- l'ensemble des besoins est la différence de l'union des besoins de c_1 et c_2, et de l'union des ressources de c_1 et c_2,
- l'ensemble des ressources est la différence de l'union des ressources de c_1 et c_2, et de l'union des besoins de c_1 et c_2.

Pour produire la grammaire, ces hiérarchies de classes sont compilées au moyen de l'algorithme suivant :

1. toutes les classes finales (*i.e.*, les classes qui ne sont super-classe d'aucune autre) sont combinées[89] entre elles ;
2. toutes les combinaisons de classes dont les besoins et ressources sont des ensembles vides sont conservées ;
3. les référents minimaux des descriptions contenues par ces classes *équilibrées* sont calculés et correspondent aux arbres de la grammaire.

Le système métagrammatical proposé par (Gaiffe et al., 2002) étend celui de (Candito, 1999) en introduisant une plus grande flexibilité dans la description (via un nombre quelconque de hiérarchies de classes) et une combinaison des classes plus contrôlée (via un concept de besoins / ressources). Cependant, il souffre de deux limitations principales :

- les descriptions d'arbres manipulent des variables de nœud à portée globale, ce qui implique (i) une gestion des noms de variables qui peut vite devenir problématique quand la métagrammaire atteint une certaine taille, et (ii) l'impossibilité de combiner deux instances de la même classe[90] ;
- le contrôle des combinaisons de classes est encore trop large, dans la mesure où la combinatoire liée à la recherche de l'annulation des besoins / ressources reste très importante.

Ce deuxième inconvénient, combiné à la complexité de la résolution des descriptions d'arbres, impacte fortement l'efficacité du compilateur en termes de temps de calcul. Pour donner une idée, une métagrammaire décrivant un grammaire TAG d'environ 3000 arbres est compilée en près d'une heure avec ce système[91].

[89] La sémantique de cette combinaison est la même que celle de l'héritage présentée ci-dessus.

[90] Ce qui est utile pour représenter les cas de double groupe prépositionnel par exemple.

[91] Sur une architecture comparable, le compilateur présenté au chapitre 5 prend dix minutes pour produire ces arbres.

3.3.4 Le MGCOMP (Eric de la Clergerie)

La dernière proposition de compilateur de métagrammaire en date est celle de (Thomasset et Villemonte de la Clergerie, 2005; Villemonte de la Clergerie, 2005b). Les auteurs reprennent certaines idées des approches antérieures (manipulation de descriptions d'arbres, hiérarchie de classes à base d'héritage multiple, combinaison de classes contrôlée par besoins / ressources), mais se distinguent par le type de structures qu'ils produisent. A la différence des approches antérieures qui génèrent des arbres en calculant les référents minimaux d'une description d'arbre, (Thomasset et Villemonte de la Clergerie, 2005) proposent de produire des arbres *factorisés*.

Un exemple d'arbre factorisé est donné figure 3.9[92]. Sur cet arbre, on note la présence d'un nœud étiqueté par le symbole *##*, indiquant qu'il s'agit d'un entrelacement. Cela signifie que l'ordre des fils de ce nœud est libre. Ainsi cet arbre factorisé permet d'analyser à la fois *Jean donne un livre à Marie* et *Jean donne à Marie un livre*.

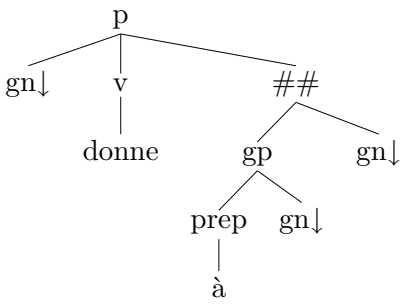


FIG. 3.9 – Exemple d'arbre factorisé.

Pour parvenir à produire de telles structures, le langage utilisé pour la description métagrammatical a été étendu par rapport aux approches antérieures. Cette extension concerne en premier lieu le langage logique de description d'arbres. En plus des relations usuelles de dominance[93], la précédence stricte, l'égalité entre nœuds, il est possible d'annoter les nœuds de la description au moyen d'opérateurs spécifiques représentant l'*optionnalité*, l'*alternative*, la *séquence*, ou encore la *répétition* (étoile de Kleene). Les nœuds peuvent en outre, comme c'est le cas des approches précédentes, être décorés au moyens de structures de traits. De plus, il est possible de spécifier la parenté d'un nœud et également son rang dans la fratrie (premier ou dernier). Enfin, il est possible de spécifier l'existence (ou la non-existence) d'un nœud au moyen d'un système de *gardes* (conditions sur des équations de chemin).

Dans ce contexte, la métagrammaire correspond à une hiérarchie de classes contenant des descriptions d'arbres *étendues*, des arguments éventuels, et un ensemble de besoins et de ressources.

[92]Cet arbre illustre de manière simplifiée le concept de factorisation, les arbres produits par une métagrammaire réaliste contiennent une bien plus grande factorisation.

[93]Dans sa version stricte ou sa clôture réflexive et transitive.

Une innovation forte par rapport à (Gaiffe et al., 2002) concerne la possibilité de placer ces besoins / ressources dans des espaces de noms spécifiques. L'intérêt de ces espaces de noms est de permettre la double utilisation d'une même ressource (cas du double groupe prépositionnel par exemple). De plus, ces espaces de noms peuvent également être utilisés pour gérer la portée des variables de nœuds inclues dans les descriptions. Cela permet d'éviter les conflits des variables *globales* de (Candito, 1999) ou *locales* de (Xia, 2001).

La compilation de la métagrammaire correspond alors à la combinaison[94] des classes terminales par point fixe, afin de dériver un ensemble de classes équilibrées (annulation des besoins et ressources).

Chacune de ces classes équilibrées est alors résolue, c'est-à-dire que l'on calcule les référents minimaux correspondants. On note que ces référents sont légèrement différents de ceux de (Rogers et Vijay-Shanker, 1994) puisque certaines sous-spécifications ne sont pas résolues (comme l'alternative par exemple), de plus le compilateur détecte les sous-spécifications dans l'ordonnancement d'une fratrie et introduit un nœud d'entrelacement (comme celui de la figure 3.9). En d'autres termes, les arbres factorisés ainsi calculés ne sont pas expansés.

Nous venons ainsi de voir une nouvelle génération de métagrammaires, qui se distingue par son expressivité (extension du langage de description d'arbres, et contrôle fin de la combinaison des classes, tant au niveau des besoins / ressources qu'au niveau des nœuds). Le principal inconvénient[95] de cette approche est qu'elle produit non pas des arbres mais des arbres factorisés rendant la tâche de vérification de la grammaire plus délicate. Ainsi, la validation de la métagrammaire passe par l'analyse syntaxique (ce qui présuppose que l'analyseur utilise des arbres factorisés, ce que seul le système DyALog (Villemonte de la Clergerie, 2005a) permet actuellement).

3.4 Conclusion

3.4.1 Caractéristiques d'un système métagrammatical

Un système métagrammatical repose sur la définition (et la compilation) d'une description abstraite de la grammaire. Cette description a pour but de factoriser l'information contenue dans la grammaire, en l'occurrence dans le cas des TAG, la factorisation concerne des structures arborescentes. Ces structures sont découpées en fragments élémentaires. Elles sont ensuite décrites en termes de combinaisons de ces fragments. Le langage logique de description d'arbres utilisé pour décrire chacun de ces fragments est celui de (Rogers et Vijay-Shanker, 1994). Il utilise les relations de dominance et de précédence entre nœuds ainsi que leurs clôtures réflexives transitives.

[94] L'opération de combinaison correspond à la conjonction des descriptions d'arbres étendues modulo la satisfiabilité des contraintes logiques qu'elles contiennent.

[95] Qui représente également un avantage selon le point de vue auquel on se place.

La grande différence parmi les systèmes métagrammaticaux existants concerne le contrôle de la combinaison des fragments. Ce contrôle peut soit être défini par la forme de la métagrammaire (cas des trois dimensions de (Candito, 1999)), soit par des règles lexicales (cas de (Xia, 2001)), soit encore par une notion de besoins / ressources (cas de (Gaiffe et al., 2002; Thomasset et Villemonte de la Clergerie, 2005)).

Dans le chapitre qui suit, nous introduisons un nouveau formalisme métagrammatical dans lequel les combinaisons entre classes sont contrôlées explicitement par les opérateurs logiques de disjonction et de conjonction.

3.4.2 La gestion des espaces de noms : un problème récurrent

La gestion de la portée des noms associés aux variables de nœuds utilisées dans les descriptions représente un réel problème pour la factorisation des structures arborescentes.

Une première approche consiste à utiliser un système de nommage global. Celle-ci a été adoptée dans les métagrammaires de (Candito, 1999; Gaiffe et al., 2002) notamment. L'inconvénient est que le développeur de la métagrammaire doit gérer manuellement une table de noms afin de savoir exactement dans quel classe un nom de nœud a été déclaré et à quoi il réfère, et ce afin d'éviter tout conflit de nom. Bien entendu, cette tâche devient vite délicate au fur et à mesure que la métagrammaire grandit.

Une seconde approche consiste à utiliser un système de nommage local. C'est le choix fait par (Xia, 2001). Dans ce cas, la combinaison des fragments peut facilement sur-générer s'il n'est pas contraint par un ensemble d'équations de nœuds.

Pour éviter ces problèmes, le formalisme que nous présentons au chapitre suivant intègre un mécanisme de gestion des espaces de nom flexible basé sur une notion d'import/export.

Chapitre 4

XMG : un formalisme métagrammatical extensible

Sommaire

Dans ce chapitre, nous présentons en détail le formalisme métagrammatical que nous avons proposé pour produire semi-automatiquement des grammaires d'arbres à portée sémantique : *eXtensible MetaGrammar* (XMG)[96].

Ce chapitre est organisé comme suit : dans un premier temps, nous définissons la syntaxe abstraite[97] noyau du formalisme XMG. Nous portons une attention particulière à la factorisation d'arbres et la gestion des espaces de noms. Rappelons que ce dernier point était l'un des problèmes majeurs des approches antérieures. Dans un second temps, nous abordons deux extensions du formalisme. Une première extension vise à permettre la manipulation

[96] Ce travail a été mené en étroite collaboration avec Benoît Crabbé, Denys Duchier et Joseph Le Roux.
[97] La syntaxe concrète du formalisme sera introduite dans le chapitre 6 et présentée en détail en annexe C.

de plusieurs niveaux de description linguistique (syntaxe et sémantique en l'occurrence). Une seconde extension du formalisme permet d'énoncer des principes de bonne formation des structures produites par rapport au formalisme grammatical cible.

4.1 Syntaxe abstraite du formalisme XMG

4.1.1 Définition de blocs élémentaires

Comme nous l'avons vu au chapitre 3, une idée communément admise dans les approches métagrammaticales est de considérer comme unités de base des fragments d'arbres. Ces fragments sont réutilisables dans plusieurs contextes[98], ce qui permet de représenter de manière factorisée la grammaire.

Pour pouvoir réutiliser un fragment, il convient de pouvoir le nommer. Cela se traduit, dans le formalisme XMG, par le concept d'*abstraction* permettant d'associer un nom à un contenu (le fragment d'arbre en question). Cette association se fait au sein d'une structure nommée *Classe* :

$$\mathit{Classe} \quad ::= \quad \mathit{Nom} \rightarrow \{\, \mathit{Contenu} \,\} \tag{4.1}$$

Dans notre cas, le contenu des classes correspond à des descriptions d'arbres définies au moyen d'une logique d'arbre à base de relations de dominance et précédence (immédiates, strictes ou larges) :

$$\mathit{Contenu} \quad ::= \quad \mathit{Description} \tag{4.2}$$

La logique d'arbre utilisée dans le formalisme XMG inclut les opérateurs suivants[99] :

$$\begin{aligned} \mathit{Description} \quad ::= \quad & X \rightarrow Y \mid X \rightarrow^{+} Y \mid X \rightarrow^{*} Y \mid \\ & X \prec Y \mid X \prec^{+} Y \mid X \prec^{*} Y \mid \\ & X = Y \mid X[f{:}E] \mid X(p{:}E) \mid \\ & \mathit{Description} \wedge \mathit{Description} \mid \mathit{Description} \vee \mathit{Description} \end{aligned} \tag{4.3}$$

où X, Y représentent des variables de nœuds, $\rightarrow$ la dominance immédiate, $\rightarrow^{+}$ la dominance stricte[100], $\rightarrow^{*}$ la dominance large[101], $\prec$ la précédence immédiate, $\prec^{+}$ la précédence stricte, $\prec^{*}$ la précédence large, $=$ l'identification de nœuds, $x[f{:}E]$ l'association du trait f de valeur l'expression E au nœud représenté par la variable X, et enfin $X(p{:}E)$ l'association de la

[98] Pour une plus grande factorisation, ces fragments sont généralement définis au sein d'une hiérarchie d'héritage, ce que permet également notre formalisme comme nous le verrons plus loin

[99] Nous prenons les notations issues de la Programmation Logique, à savoir les symboles commençant par une majuscule désignent des variables, ceux commençant par une minuscule des constantes.

[100] Nous rappelons que par dominance stricte nous désignons la clôture transitive de la relation de dominance.

[101] Le terme "large" réfère ici à la clôture transitive et réflexive de la relation.

propriété[102] p de valeur l'expression E à ce même nœud dénoté par X. Par expression, nous entendons une variable, une constante, ou encore une disjonction de valeurs atomiques.

Exemple 1 (Descriptions d'arbres)**.** *Pour fixer les idées, voici (i) un exemple de fragment d'arbre et (ii) la description équivalente définie au moyen du langage de description utilisé dans* XMG *:*

(i)

(ii) $(X[cat:p] \rightarrow Y[cat:v]) \quad \wedge \quad (X \rightarrow Z[cat:n]) \quad \wedge \quad (Y \prec Z)$

Notons que, par défaut, chacune des variables de nœuds introduites dans nos descriptions d'arbres a une portée locale[103], ce qui évite au développeur de la métagrammaire de devoir gérer des noms de variables globaux et lui permet ainsi d'éviter des conflits de noms.

4.1.2 Combinaison des blocs élémentaires

A partir des blocs élémentaires (contenant des fragments d'arbres), nous utilisons des opérateurs de combinaison permettant d'accumuler des fragments en vue de produire des arbres.

4.1.2.1 Héritage

Une première combinaison de fragments correspond à l'**héritage** de classe. L'idée sous-jacente (présentée originellement dans (Vijay-Shanker et Schabes, 1992) et reprise dans les approches de (Candito, 1999; Gaiffe et al., 2002) notamment) est de permettre de réutiliser le contenu d'un fragment dans un autre fragment. Ce dernier *spécialise* alors le fragment duquel il hérite.

Dans notre formalisme, nous noterons l'héritage comme suit :

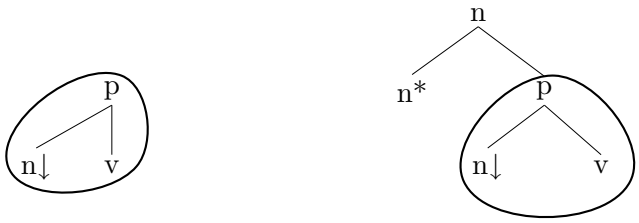
$$Classe \quad ::= \quad NomB \quad \angle \quad NomA \rightarrow \{\, Contenu \,\} \tag{4.4}$$

Dans ce cas, la définition d'une classe correspond donc à l'association à un nom $NomB$ d'un contenu, tout en héritant le contenu associé à l'entité désignée par $NomA$. Notons que nous utilisons le terme d'**import** de classe pour désigner l'héritage. Ainsi, dire qu'une classe B hérite d'une classe A revient à dire que B *importe* A.

Exemple 2 (Héritage)**.** *Pour illustrer l'usage de l'héritage, considérons les deux fragments d'arbres de la figure 4.1, représentant respectivement la fonction syntaxique sujet (nominal) réalisée sous forme canonique et sous forme relativisée.*

[102] Une propriété de nœud correspond à une caractéristique utilisée lors de la résolution des descriptions pour éliminer certains modèles, *cf* paragraphe 4.3.

[103] Comme nous le verrons ultérieurement, XMG intègre un mécanisme d'extension de la portée des noms de variable.



Sujet_Canonique *Sujet_Relativisé*

FIG. 4.1 – Héritage d'un fragment d'arbre.

On remarque que le fragment d'arbre représentant le sujet relativisé peut être vu comme une spécialisation du fragment associé au sujet canonique (cf structure encerclée sur la figure 4.1).

Ainsi nous avons dans notre syntaxe abstraite :

$$Sujet\_Relativis\mathbf{é} \angle Sujet\_Canonique$$

Plusieurs questions se posent dans ce contexte d'héritage, à savoir :

1. Comment accéder aux variables de nœuds de la classe mère à partir de la classe fille dans le cas où les variables de nœuds ont une portée locale (*cf supra*[104]) ?
2. L'héritage est-il *multiple* ? Et si oui, pouvons nous hériter plusieurs fois de la même classe ?

4.1.2.2 Extension de la portée des variables

Pour permettre de réutiliser un fragment par héritage, le formalisme XMG intègre un mécanisme d'**export** de variable. Ainsi il est possible de déclarer les noms des variables[105] qui seront accessibles en dehors de la classe. En d'autres termes, il est possible pour toute variable X introduite localement, d'étendre la portée de X aux classes filles.

Plus précisément, à chaque classe est associée une structure de traits contenant les noms des variables exportées. Ainsi la définition d'abstraction (4.1) devient :

$$Classe \quad ::= \quad \langle V_1, \ldots, V_n \rangle \Leftarrow Nom \quad \rightarrow \quad \{\, Contenu \,\} \tag{4.5}$$

Cette définition traduit le fait qu'une classe, en plus d'associer un nom avec un contenu, lui associe également un enregistrement contenant la liste des noms des variables exportées, *i.e.*, accessibles en dehors de la classe elle-même (ici les noms V_1 à V_n). Cet export de noms explicite permet (1) de référer directement à une variable déclarée dans une super classe et (2) d'éviter les conflits de noms puisqu'il est possible de définir précisément la portée d'un

[104] Cette portée locale est analogue aux variables dites *privées* en programmation orientée objet.

[105] Que ce soient les noms de variables de nœuds ou encore les noms de variables de traits.

nom, soit localement (par défaut, le nom peut donc être réutilisé), soit globalement (par extension de portée, le nom réfère alors à une variable introduite auparavant).

4.1.2.3 Restriction de la portée des variables à l'Import

L'export des noms de variables tel que présenté ci-dessus n'est pas sans problème, puisqu'il a pour conséquence d'introduire des noms dans l'espace de noms de la classe fille. Ce comportement d'extension de la portée des identifiants peut mener à deux cas critiques :

1. Imaginons que nous définissons une classe C_1 contenant une variable X de portée étendue, puis deux classes C_2 et C_3 héritant chacune de C_1. Imaginons que C_2 ait besoin d'accéder à la variable X de C_1, par exemple si C_2 ajoute de l'information sur le nœud identifié par X. Imaginons également que C_3 n'ait pas besoin d'accéder à cette variable X. Dans ce cas le nom X est réservé dans chacune des classes C_2 et C_3 (ainsi que dans toutes leurs classes filles). En particulier, l'ensemble des noms libres dans C_3 s'en trouve réduit.
2. Que se passe-t-il si nous souhaitons, pour une raison ou une autre, à la fois (i) référer à une variable introduite précédemment et (ii) réutiliser le nom qui lui était associé auparavant pour désigner une variable locale ?

Pour éviter ces problèmes, XMG intègre un système flexible de restriction de la portée des noms de variables lors de l'héritage d'une classe.

Cas 1. Tout d'abord, il est possible d'assigner à un nom une portée semi-globale en utilisant un **import paramétré**. Le long d'une branche d'héritage, il est possible de « bloquer » la portée d'un nom exporté précédemment en réalisant un import paramétré (ou import sélectif). Ainsi, la définition de l'import (4.4) est étendue de la façon suivante[106] :

$$\begin{aligned} \mathit{Classe} \quad ::= \quad & \langle V_1, \ldots, V_n \rangle \Leftarrow \mathit{NomB} \quad \angle \quad \mathit{NomA}[I_1, \ldots, I_m] \\ & \rightarrow \quad \{\, \mathit{Contenu} \,\} \end{aligned} \tag{4.6}$$

Ici, nous définissons une classe comme l'association d'un nom *NomB* à un contenu, tout en récupérant par héritage le contenu de la classe référencée par *NomA*, mais en limitant les noms de variable issus de *NomA* et réservés dans *NomB* à $I_1, \ldots, I_m$[107].

Exemple 3 (Import paramétré)**.** *Imaginons qu'une classe A exporte les nom* `X` *et* `Y`*, et qu'une classe B hérite de A tout en introduisant localement un nom* `X`*. Nous souhaitons alors bloquer la portée du premier* `X` *à la classe B (i.e. B n'importera que la variable* `Y`*). Cela peut se faire comme suit :*

$$\langle X \rangle \Leftarrow B \quad \angle \quad A[Y] \quad \rightarrow \quad \{\, \mathit{Contenu} \,\}$$

Ici, la classe B hérite de la classe A, et ne peut accéder, parmi tous les noms exportés (directement ou transitivement) par la classe A, qu'au nom `Y`*.*

[106] $I_1, \ldots, I_m$ représentent les variables importées de $NomB$.
[107] $\{I_1, \ldots, I_m\}$ forme un sous-ensemble des noms exportés par la classe $NomA$.

Cas 2. L'autre configuration problématique dans le contexte de l'héritage correspond à la définition locale d'une variable homonyme d'une variable importée. Afin de pouvoir conserver une liberté quasi-totale dans le choix des noms de variables pour une classe donnée (et ainsi faciliter la tâche du développeur de la métagrammaire), XMG intègre un système de **renommage à l'import** :

$$\begin{array}{lll} \mathit{Classe} & ::= & \langle V_1, \ldots, V_n \rangle \Leftarrow \mathit{NomB} \quad \angle \quad \mathit{NomA}[I_1 = J_1, \ldots, I_m = J_m] \\ & & \quad \rightarrow \quad \{\, \mathit{Contenu} \,\} \end{array} \tag{4.7}$$

Ici, nous voyons qu'une classe (nommée *NomB*), en plus d'hériter le contenu d'une classe (*NomA*), peut restreindre l'import de celle-ci aux variables I_1 à I_m en les renommant respectivement J_1 à J_m. Ainsi les noms $I_1 \ldots I_m$ restent libres (*i.e.* disponibles) dans la classe *NomB*.

Exemple 4 (Renommage à l'import)**.** *Définissons une classe A exportant les nom* `X` *et* `Y`*, et une classe B héritant de A tout en introduisant localement un nom* `X`*. Pour éviter le conflit de nom lié au double emploi de l'identifiant* `X`*, nous allons renommer le premier* `X` *issu de la classe A en* `XA` *:*

$$\langle X \rangle \Leftarrow B \quad \angle \quad A[X = XA\,,\, Y] \quad \rightarrow \quad \{\, \mathit{Contenu} \,\}$$

Ici, la classe B hérite de la classe A, et peut réutiliser les noms `XA`*, pour référer à la variable initialement nommée* `X` *dans A, et* `Y`*. Ainsi, le nom* `X` *visible dans la classe B correspond à la variable introduite localement.*

4.1.2.4 Héritage multiple

L'héritage intégré au formalisme XMG est un **héritage multiple**, une classe peut hériter de plusieurs autres, la définition de l'héritage (4.4) peut donc être étendue comme suit[108] :

$$\begin{array}{lll} \mathit{Classe} & ::= & \mathit{Nom} \quad \angle \quad C_1 \wedge C_2 \wedge \ldots C_n \\ & & \rightarrow \{\, \mathit{Contenu} \,\} \end{array} \tag{4.8}$$

L'intérêt de l'héritage multiple est de permettre un plus haut degré de factorisation des blocs élémentaires puisque l'on peut réutiliser plusieurs blocs au sein d'une même classe.

Dans le cas de l'héritage multiple, il faut bien noter que les contenus des classes $C_1 \ldots C_n$ sont rendus accessibles dans la classe fille, et cela de manière *conjonctive*. Plus précisément, les descriptions définies dans les classes importées sont inclues dans la description de la classe courante. Ainsi, il est possible par héritage de spécialiser une classe ou plusieurs classes en ajoutant de l'information (nouveaux nœuds ou mise à jour des matrices de traits associées à des nœuds existants).

[108] Pour des raisons de lisibilité, nous omettons ici les exports de noms et restrictions d'héritage introduits précédemment.

Conflits de noms liés à l'héritage multiple. Une conséquence de cet héritage multiple est qu'il est possible d'avoir des conflits de noms, tel qu'illustré dans l'exemple suivant.

Exemple 5 (Portée des noms lors d'un héritage multiple)**.** *Imaginons que nous définissons des classes C_1 et C_2 contenant chacune une description manipulant une variable de nœud X. Ensuite, nous définissons la classe C qui hérite des classes C_1* et *C_2. A quoi réfère alors le nom X dans la classe C ?*

Deux réponses à cette question sont envisageables :

(i) *nous pouvons considérer que les deux variables ayant le nom X dans chacune des classes mères réfèrent à la* même information *(nœud, traits, etc). Cela revient à dire que nous ajoutons une unification* implicite *entre ces deux variables dans la description de la classe fille.*

(ii) *nous pouvons aussi considérer que les deux variables nommées X dans les classes mères désignent des informations différentes. Comment savoir alors à laquelle de ces variables nous accédons dans la classe fille ?*

Nous allons voir qu'il existe deux façons d'éviter cette ambiguïté de la portée des noms lors de l'héritage multiple, XMG permettant d'identifier les deux variables homonymes (les variables désignées par X dans la classe C dans l'exemple précédent), ou au contraire de les différencier. Notons au passage que, dans l'implantation actuelle du système XMG, si une telle situation se produit (héritage multiple impliquant des variables homonymes), l'interprétation par défaut est celle de (ii). Le nom X désigne alors, dans l'espace de noms de la classe fille, une et une seule des variables des classes mères (définie selon l'ordre d'import des classes mères).

Désambiguïsation – Solution 1. La première possibilité, pour gérer une telle ambiguïté de nom, est d'utiliser les mécanismes d'*import paramétré* et de *renommage à l'import* introduits précédemment pour (a) ne pas introduire ces deux noms dans le même espace ou (b) utiliser deux noms différents. En effet, (a) revient à ne pas hériter le nom X de chacune des classes mères (utile si nous ne souhaitons pas spécialiser chacune des deux informations référée par X), ou (b) renommer l'un (ou les deux) occurrence(s) de X lors de l'import des classes mères.

Désambiguïsation – Solution 2. Une autre façon d'éviter l'ambiguïté liée à cette configuration d'héritage est d'utiliser une **conjonction** de classes associée à l'opérateur *dot*. Cet opérateur est présenté en détail dans le paragraphe 4.1.2.5 page 120.

Héritage multiple d'une même classe. Un cas particulier de l'héritage multiple consiste en l'héritage multiple de la même classe. Quel est le sens d'un tel héritage ? Est-ce souhaitable ? Cela conduit-il à des problèmes (de représentation ou d'implantation) ?

Tout d'abord, voici un exemple présentant l'intérêt d'un double import d'un même fragment.

Exemple 6 (Double groupe prépositionnel et héritage)**.** *Considérons les classes C_1, C_2 et C de la figure 4.6. C_1 contient le fragment d'arbre de la morphologie verbale à l'actif et C_2 celui du groupe prépositionnel. Si nous souhaitons représenter le fragment d'arbre du double groupe prépositionnel tel que dans la classe C avec un maximum de factorisation, il nous faut pouvoir déclarer que C importe deux fois C_2.*

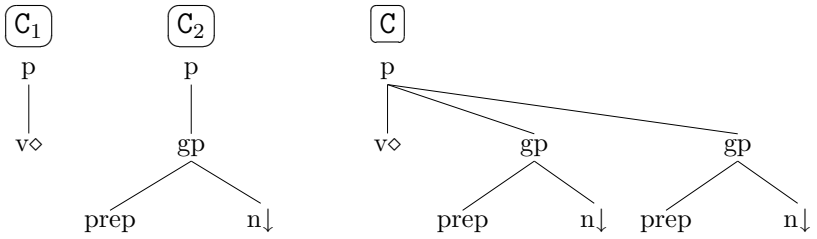


FIG. 4.2 – Cas du double groupe prépositionnel et héritage.

Ce double import de C_2 n'est pas sans problème. En effet, prenons une variable nommée X déclarée dans C_2. A quoi réfère X dans C ? Notons qu'ici nous ne pouvons pas avoir que X réfère aux deux variables issues de chacune des instances de C_2, sinon la description contenue dans C ne contiendrait qu'un seul groupe prépositionnel (en outre, elle serait redondante puisque ce même fragment serait décrit deux fois).

Le cas de l'héritage multiple est problématique puisqu'il introduit un conflit de noms comme illustré dans l'exemple précédent. En conséquence, le choix qui a été fait au sein du formalisme XMG est de ne pas permettre l'héritage multiple d'une même classe. Pour pouvoir traiter les cas de réutilisation multiple d'un même fragment, XMG intègre un autre mécanisme de combinaison de classe : la *conjonction*, que nous introduisons ci-dessous.

4.1.2.5 Conjonction et Disjonction

Comme nous l'avons vu, nous avons besoin, dans notre formalisme métagrammatical, d'un opérateur de **conjonction** de classes. De plus, la volonté de représenter des alternatives d'une même structure syntaxique (telle que les différentes réalisations d'une même fonction grammaticale) motive l'introduction d'un opérateur de choix : la **disjonction**.

Ainsi, XMG permet la combinaison conjonctive et disjonctive de classes. La spécification de notre langage, définition (4.2), se trouve étendue par l'ajout de la définition suivante :

$$\begin{array}{lll} \mathit{Contenu} & ::= & \mathit{Description} \mid \mathit{Nom} \mid \\ & & \mathit{Contenu} \vee \mathit{Contenu} \mid \mathit{Contenu} \wedge \mathit{Contenu} \end{array} \tag{4.9}$$

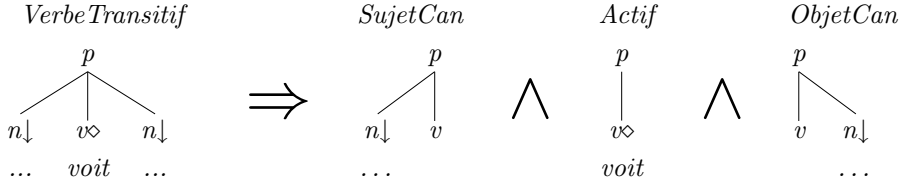
Cette définition traduit le fait que le contenu d'une classe peut être soit une description

(généralement partielle) d'arbre, l'instanciation d'une autre classe[109], la conjonction de deux classes ou la disjonction de deux classes (instanciation non-déterministe de classe). Nous introduisons ainsi, en plus d'un langage de description de fragments, et d'un concept d'héritage, un langage de contrôle de la combinaison de classes.

Exemple 7 (Utilisation de la conjonction)**.** *Si l'on fait l'approximation qu'un verbe transitif s'obtient par la* **conjonction** *d'un fragment d'arbre représentant la morphologie verbale à la voix active, la réalisation syntaxique d'un sujet en position canonique, et celle d'un objet également en position canonique,* XMG *permet de représenter un verbe transitif comme suit :*

$$VerbeTransitif \quad \rightarrow \quad SujetCan \quad \wedge \quad Actif \quad \wedge \quad ObjetCan$$

Ce qui correspond à la description factorisée de l'arbre $VerbeTransitif$ *de la manière suivante :*



Exemple 8 (Utilisation de la disjonction)**.** *Comment faire pour étendre l'exemple précédent afin de générer les différentes réalisations syntaxique du sujet et de l'objet ? Nous pouvons utiliser la* **disjonction** *pour énumérer les différentes réalisations possibles, comme illustré ci-dessous :*

$$\begin{aligned} Sujet \quad &\rightarrow \quad SujetCan \quad \vee \quad SujetExt \quad (\vee \quad \ldots) \\ Objet \quad &\rightarrow \quad ObjetCan \quad \vee \quad ObjetQu \quad (\vee \quad \ldots) \end{aligned}$$

Ainsi, nous définissons une abstraction sur la fonction syntaxique sujet qui regrouperait l'ensemble de ses réalisations possibles (forme canonique, extraite, etc), de la même manière pour la fonction syntaxique objet (forme canonique, questionnée, etc). En reprenant notre règle de description des verbes transitifs introduite précédemment, en la modifiant comme suit :

$$VerbeTransitif \quad \rightarrow \quad Sujet \wedge Actif \wedge Objet$$

et si l'on restreint les réalisations syntaxiques du sujet et de l'objet uniquement aux formes canoniques et extraites, notre description permet d'engendrer un plus grand nombre d'arbres, à savoir ceux de la figure 4.3[110]*.*

[109] Ce qui revient à définir un nouveau nom pour une classe donnée.

[110] On remarquera la présence de 3 arbres alors que 2 réalisations syntaxiques sont définies pour le sujet, 2 également pour l'objet, ce qui laisserait présager 2×2 solutions. Cependant la double extraction étant interdite, il n'y a que 3 solutions. Le procédé d'élimination de la double extraction est présenté en section 4.3.

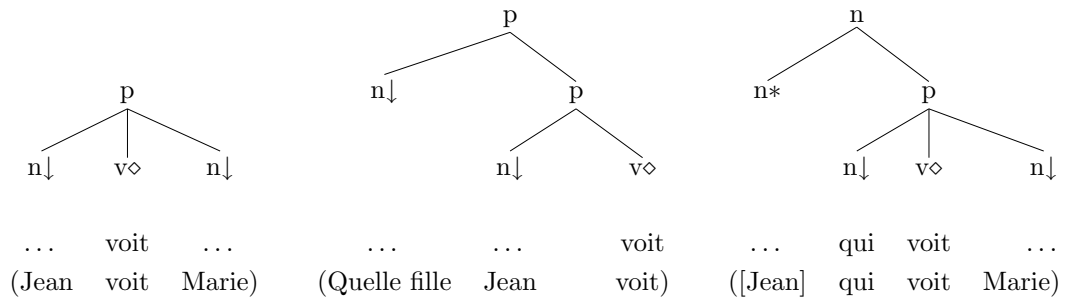


FIG. 4.3 – Arbres générés par la classe $VerbeTransitif$.

Au delà de l'intuition présentée ci-dessus, comment s'opère l'identification (ou la différenciation) de variables apparaissant dans chacune des classes combinées ? En d'autres termes, comment accéder aux noms définis dans chacune des classes combinées afin d'unifier les variables désignant les mêmes informations ? Ou encore, comment s'assurer dans notre exemple précédent que les variables désignant les nœuds racines dans les classes *Sujet*, *Active* et *Objet* sont identifiées ?

4.1.2.6 Opérateur *dot*

Pour pouvoir identifier deux variables présentes dans deux classes distinctes combinées par instanciation (et non par héritage), il nous faut (a) pouvoir accéder à chacune de ces variables (leur nom doit être visible) dans la classe courante, et (b) pouvoir unifier ces variables (au moyen d'un opérateur d'unification).

Le point (a) est rendu possible, dans XMG, d'une part au moyen de l'export de noms présenté précédemment. Cet export de nom permet d'étendre la visibilité d'un nom de variable en dehors de la classe où ce nom est introduit. D'autre part, une fois exporté, ce nom peut être accédé dans une autre classe au moyen de l'opérateur « . » (*dot*). Enfin l'unification entre variables originaires de classes combinées conjointement, point (b), s'opère au moyen d'une équation d'unification (opérateur =).

Exemple 9 (Utilisation de l'opérateur *dot*)**.** *Considérons les trois classes SujetCan, Actif et ObjetCan avec leurs enregistrements d'export tel que représentées ci-dessous (on notera qu'ici les noms locaux des variables de nœud correspondent à la catégorie syntaxique et sont donc les mêmes dans chacune des classes) :*

$$\begin{array}{lllll} \langle S, V\rangle \Leftarrow SujetCan & \rightarrow & \{\, S\,[cat:p] \;\rightarrow\; N\,[cat:n]\ldots\} & \wedge \\ \langle S, V\rangle \Leftarrow Actif & \rightarrow & \{\, S\,[cat:p] \;\rightarrow\; V\,[cat:v]\,\} & \wedge \\ \langle S, V\rangle \Leftarrow ObjetCan & \rightarrow & \{\, S\,[cat:p] \;\rightarrow\; N\,[cat:n]\ldots\} & \end{array}$$

Pour pouvoir accéder aux variables exportées, nous stockons dans des variables locales les enregistrements d'export de chacune des classes combinées :

$$VerbeTransitif \quad \rightarrow \quad SU = SujetCan \quad \wedge \quad AC = Actif \quad \wedge \quad OB = ObjectCan \quad \wedge \quad \ldots$$

Enfin, dans la classe $VerbeTransitif$, nous pouvons utiliser l'opérateur d'unification pour définir les unifications de nœuds nécessaires :

$$\ldots \quad SU.S = AC.S \quad \wedge \quad SU.V = AC.V \quad \wedge \quad AC.S = OB.S \quad \wedge \quad AC.V = OB.V$$

Ainsi, nous pouvons unifier *explicitement* des variables introduites dans des classes différentes (*i.e.*, qui ne sont pas en relation d'héritage l'une avec l'autre).

Cependant cela demande de définir un certain nombre d'équations d'unification, ce qui peut s'avérer un travail fastidieux sur des métagrammaires de taille réaliste. Comme nous le verrons en section 4.3, XMG intègre un système de contraintes additionnelles, dont un système de polarisation des nœud, qui a pour effet de déclencher l'unification *implicite* de variables de nœud, ce qui réduit considérablement la taille de la métagrammaire et par là, facilite la gestion des noms.

4.1.2.7 Interfaces entre classes

Nous avons vu différents procédés de gestion de la portée des noms de variable. En plus de l'héritage et de l'opérateur *dot*, XMG intègre un mécanisme de globalisation de la portée d'un nom. Ce mécanisme, appelé **interface** de classe, correspond à la définition, pour chaque classe, d'une matrice de traits. Cette matrice permet d'associer un nom global, le trait, à une variable, la valeur du trait. Ce mécanisme est compatible avec l'héritage et les conjonctions et disjonctions de classes. Ainsi, lorsqu'une classe B hérite d'une classe A, les interfaces de A et de B sont unifiées. En conséquence, si deux variables se sont vues donner le même nom global, elles sont unifiées. De la même manière, lorsque deux classes A et B sont conjointes, leurs interfaces sont unifiées et là encore, si deux variables de chacune des deux classes ont été associées au même nom global, elles sont unifiées.

Exemple 10 (Unification par interface)**.** *Considérons les classes $SujetCan$, $Actif$ et $ObjetCan$ de l'exemple 7. Si nous appelons X, Y et Z les variables dénotant respectivement le nœud de catégorie S dans les classes $SujetCan$, $Actif$ et $ObjetCan$, nous pouvons forcer l'unification de ces nœuds sans les exporter, de la façon suivante (l'opérateur $*=$ symbolise la définition d'interface) :*

$$\begin{array}{lll} VerbeTransitif \quad \rightarrow \quad SujetCan & *= [sNode = X] & \wedge \\ \qquad\qquad\qquad\quad Actif & *= [sNode = Y] & \wedge \\ \qquad\qquad\qquad\quad ObjetCan & *= [sNode = Z] & \end{array}$$

4.1.2.8 Descriptions totales

A ce stade, nous disposons d'un langage abstrait intégrant des procédés d'**abstraction**, d'**héritage** et enfin de combinaison **conjonctive** et **disjonctive** de classes. Il nous manque cependant un moyen de distinguer, parmi les classes de notre métagrammaire, celles qui correspondent à des descriptions totales. En recherchant les modèles minimaux pour les descriptions auxquelles réfèrent ces classes, nous génèreront l'ensemble des arbres de notre grammaire.

Cette distinction se fait au moyen d'une **évaluation** de classe :

$$Evaluation \quad ::= \quad evalue \quad Nom \tag{4.10}$$

Exemple 11 (Evaluation des verbes transitifs)**.** *Nous avons vu précédemment que les verbes transitifs étaient décrits comme étant construit à partir de la réalisation d'un sujet, d'une morphologie verbale*[111] *et de la réalisation d'un objet :*

$$\begin{aligned} VerbeTransitif \rightarrow\ & Sujet \quad \wedge \quad Actif \quad \wedge \quad Object \\ Sujet \rightarrow\ & SujetCan \vee SujetExt \\ Objet \rightarrow\ & ObjetCan \vee ObjetQu \end{aligned}$$

La classe nommée VerbeTransitif contient donc (après développement des règles de combinaison de classes) une disjonction de descriptions totales. Chacune des ces descriptions décrit un ensemble de structures syntaxiques pour des verbes intransitifs. Pour spécifier que cette classe contient les descriptions pour lesquelles nous souhaitons calculer les modèles d'arbres minimaux, nous déclarons que la classe VerbeTransitif doit être évaluée :

$$evalue \quad VerbeTransitif$$

La conséquence de cette déclaration est que le compilateur va calculer l'ensemble des modèles d'arbres minimaux pour *chacune* des descriptions contenues dans la disjonction en question. Pour cet exemple, le résultat correspond aux arbres présentés figure 4.3 page 122.

4.2 Extension 1 : différents niveaux de description linguistique, intégration d'une dimension sémantique

A présent, nous allons voir comment étendre notre langage de description afin de pouvoir prendre en compte différents niveaux de description linguistique, tel que la sémantique par exemple.

[111] Pour simplifier l'exemple, nous ne considérons que la morphologie active ici.

4.2.1 Dimensions dans le formalisme XMG

Dans la section précédente, nous avons vu comment décrire et manipuler des fragments d'arbres avec le formalisme XMG. Ces fragments d'arbres représentent des structures syntaxiques reflétant le lien entre les constituants de la phrase. Lors du calcul des arbres de la grammaire, ces fragments d'arbres sont *accumulés*. En d'autres termes, les héritages de fragments et autres combinaisons conjonctives et disjonctives sont développés, en partant des classes évaluées. Par exemple : $A \wedge (B \vee C)$ devient $(A \wedge B) \vee (A \wedge C)$, *i.e.* nous obtenons une énumération d'accumulations[112]. Si nous souhaitons étendre le langage de description du formalisme XMG pour décrire d'autres types d'informations qu'uniquement des descriptions de structures syntaxiques arborescentes, il convient de gérer non plus une simple opération d'accumulation, mais un ensemble d'*accumulations* distinguées. Ces accumulations manipulent différents types de description, appelés aussi **dimensions**. Cette accumulation *multi-dimensionnelle* correspond à l'extension suivante de notre langage :

$$\begin{array}{lll} \mathit{Contenu} & ::= & \mathit{Dimension} \mathrel{+}= \mathit{Description} \mid \mathit{Nom} \mid \\ & & \mathit{Nom} \vee \mathit{Nom} \mid \mathit{Nom} \wedge \mathit{Nom} \end{array} \tag{4.11}$$

Cette définition traduit le fait que la description contenue dans la classe courante est accumulée (opérateur $+=$) au sein de la dimension adéquate. Cette notion de dimension permet de distinguer le type de l'information décrite. Un avantage de cette distinction est qu'il sera possible d'effectuer un traitement additionnel de certaines dimensions. Ainsi s'il s'agit de descriptions d'arbres (dimension syntaxique), nous procéderons à la recherche de leurs modèles d'arbres minimaux.

4.2.2 Un langage de description pour une sémantique plate

Un autre exemple de dimension est la description d'informations sémantiques. Moyennant le fait que nous disposons d'un langage de description de formules sémantiques, nous pouvons, en plus d'accumuler des descriptions d'arbres, accumuler des formules sémantiques. XMG intègre un langage de description pour les formules d'un langage de sémantique plate tel que celui de (Bos, 1995). Ce langage est défini comme suit :

$$\mathit{Description} \ ::= \ \ell{:}p(E_1, ..., E_n) \mid \ \neg\ell{:}p(E_1, ..., E_n) \mid \ E_i \ll E_j \tag{4.12}$$

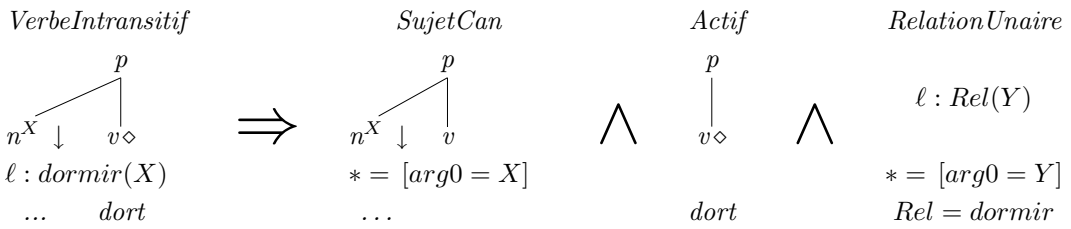
Ici, $\ell{:}p(E_1, ..., E_n)$ représente le prédicat p avec les arguments $E_1, .., E_n$, et étiqueté par ℓ, $\neg$ l'opérateur de négation, et $E_i \ll E_j$ la portée entre les variables sémantiques E_i et E_j.

Notons qu'à la différence des descriptions d'arbres, cette accumulation de formules sémantiques n'est pas résolue. Le traitement additionnel appliqué à cette dimension sémantique correspond donc à l'identité.

Grâce à cette extension, nous pouvons définir dans notre métagrammaire des classes purement syntaxiques, purement sémantiques ou encore incluant à la fois des informations syntaxiques et sémantiques. Cela est illustré dans l'exemple ci-dessous.

[112] Chacune des ces accumulations contient une description d'arbre totale.

Exemple 12 (Utilisation d'accumulateurs)**.** *Nous pouvons définir l'arbre associé au verbe intransitif* **dormir** *avec sujet en position canonique en y incluant une information sémantique de la façon suivante :*



Ici, nous avons défini trois classes élémentaires, représentant respectivement le sujet nominal en position canonique, la morphologie verbale active et le prédicat sémantique unaire dormir. Cette dernière classe contient donc des informations appartenant uniquement à la dimension sémantique. La conjonction de ces trois classes nous permet de construire l'arbre de gauche, dans lequel le nœud sujet et l'argument du prédicat sémantique ont été unifiés au moyen d'un trait d'interface nommé arg0*.*

Remarque Les variables introduites dans une classe peuvent être utilisées dans chacune des dimensions que cette classe manipule. Ainsi, nous pourrions très bien définir une classe dont certaines variables (de trait, voire de nœud ou de prédicat) seraient partagées entre la dimension syntaxique et la dimension sémantique, comme dans l'exemple ci-dessous :

$$\begin{aligned} CanonSuj \rightarrow\ & syn+ = X\,[cat : p] \wedge Y\,[cat : n, ind : I] \wedge Z\,[cat = v] \\ & \wedge\ X \rightarrow\ Y \wedge X \rightarrow\ Z \wedge Y \prec\ Z \\ & \wedge\ sem+ = \ell : Rel(I) \end{aligned}$$

Dans ce paragraphe, nous avons vu une première extension du formalisme XMG qui permet de prendre en charge différents niveaux de description linguistique, en l'occurrence la syntaxe et la sémantique. Cette extension est rendue aisée car notre formalisme distingue clairement un langage de contrôle permettant de conjoindre ou disjoindre des blocs élémentaires et un langage de description de ces blocs. Nous avons vu deux exemples de *type* de blocs (dimension) : une *dimension syntaxique* utilisant un langage de descriptions de fragments d'arbres et une *dimension sémantique* utilisant un langage de description de formules logiques.

4.3 Extension 2 : vers une librairie de contraintes de bonne formation grammaticale

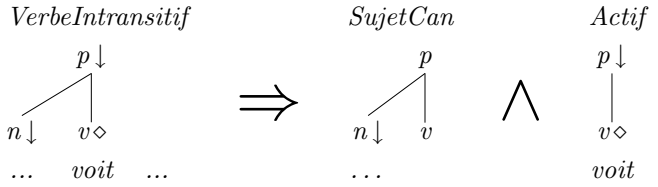
Dans cette section, nous allons voir une seconde extension du formalisme permettant d'imposer des contraintes de bonne formation grammaticale sur les structures décrites dans la métagrammaire.

4.3.1 Motivations

Lors de la définition d'une métagrammaire de taille importante, il est très facile de faire des erreurs d'étiquetage de nœud, ou encore de sur-générer en produisant des structures invalides du fait d'une description d'arbre sous-spécifiée. Ces deux cas de figures sont illustrés dans l'exemple 13.

Exemple 13 (Erreurs métagrammaticales)**.** *Les erreurs métagrammaticales présentées dans cet exemple ont comme cause commune une description incomplète ou erronée des fragments d'arbres. La conséquence de cela est la production d'arbres non-valides au sens* TAG*, ce qui empêche leur utilisation par un analyseur* TAG*.*

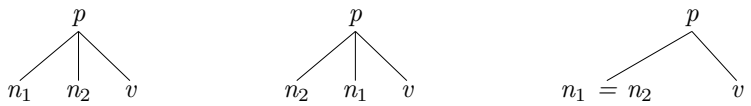
(i) Etiquetage de nœud incohérent *Imaginons que nous décrivons les classes suivantes :*



Clairement, en étiquetant le nœud racine de la classe Actif comme nœud de substitution (symbolisé par la flèche), nous introduisons une erreur dans la grammaire produite.

(ii) Sous-spécification de fragment d'arbre *Prenons les classes définies ci-dessous*[113] *:*

Supposons en outre que nous avons unifié les variables désignant les nœuds de même catégorie. Il reste un couple de nœud dont la relation est sous-spécifiée, il s'agit de n_1 et n_2. Ainsi la résolution de cette conjonction de fragments va produire les trois arbres ci-dessous dont potentiellement un seul correspond à la solution désirée :



Pour éviter que de telles situations se produisent, le formalisme XMG intègre un système de contraintes (dont certaines sont optionnelles) qui ont pour but d'éliminer ou de corriger automatiquement les arbres produits à partir de la description métagrammaticale.

4.3.2 Classification des contraintes

Les contraintes que le formalisme XMG offre au développeur de la métagrammaire peuvent être classées en quatre catégories, les contraintes **formelles**, les contraintes **opérationnelles**, les contraintes **dépendantes du langage** et les contraintes **théoriques**.

[113] Sur cet exemple, nous omettons les étiquetage de nœuds pour simplifier la représentation.

Dans ce qui suit, nous allons introduire chacune de ces catégories de contraintes, puis nous donnerons les contraintes implantées actuellement dans le système XMG, et conclurons.

4.3.2.1 Contraintes formelles

La première catégorie de contraintes correspond aux contraintes de bonne formation par rapport à un formalisme grammatical cible. Dans notre cas, nous décrivons des grammaires d'arbres adjoints. A ce titre, les structures arborescentes que nous produisons doivent respecter certains critères, tels que le fait que, dans un arbre, tout nœud frontière est étiqueté soit *nœud lexical*[114], *nœud ancre*, soit *nœud de substitution*, soit *nœud pied*, ou encore qu'un nœud interne ne peut pas être étiqueté *nœud pied* ou *nœud de substitution*. L'ensemble des contraintes formelles, pour les grammaires d'arbres adjoints est donné dans la figure 4.4.

Tout nœud dispose d'une catégorie syntaxique.
Tout nœud frontière doit être étiqueté *nœud lexical*, *nœud ancre*, *nœud de substitution* ou *nœud pied*.
Tout nœud interne est étiqueté soit *nœud de non-adjonction* ou *nœud standard*.
Tout arbre dispose d'au moins un nœud frontière étiqueté *nœud ancre*.
Tout arbre dispose d'au plus un nœud étiqueté *nœud pied*.
La catégorie syntaxique du nœud étiqueté *nœud pied* est la même que celle du nœud racine.

FIG. 4.4 – Contraintes formelles pour TAG

Notons que la définition d'autres ensembles de contraintes formelles permet de modifier les structures produites par le système XMG suivant d'autres critères liés au formalisme grammatical cible. Ainsi, XMG permet de décrire, en plus des grammaires d'arbres adjoints, des grammaires d'interaction (Perrier, 2003).

4.3.2.2 Contraintes opérationnelles

La seconde catégorie de contraintes correspond aux contraintes opérationnelles. Ces contraintes étendent l'expressivité de la logique de description d'arbres utilisée en associant aux variables de nœuds des *propriétés*[115] contraignant les modèles valides. De manière intuitive, elles ont pour effet de réduire le nombre de modèles pour une description d'arbres en introduisant un mécanisme de contrôle semi-automatique de l'identification entre nœuds de la description.

Dans notre cas, ce mécanisme de contrôle prend la forme d'un système de polarisation des nœuds. Ces polarités vont entraîner l'identification de certains nœuds (et l'unification

[114] Un nœud lexical est un nœud étiqueté par un symbole terminal, en l'occurrence, un mot du lexique.

[115] Rappelons que le formalisme XMG utilise un langage logique de description permettant d'associer à une variable de nœud non seulement une matrice de traits mais également un ensemble de propriétés, *cf* définition (4.3) page 114.

des structures de traits qui leur sont associées).

Dans un premier temps, nous allons présenter le système de polarisation utilisé dans le formalisme XMG, puis nous donnerons un exemple de son utilisation, avant de faire un résumé des avantages et inconvénients d'un tel système.

Polarisation de nœuds Le formalisme XMG intègre un système de polarisation de nœuds sous forme d'un langage de couleurs. Ainsi, il est possible d'annoter les nœuds des descriptions d'une couleur parmi *Rouge*, *Noir* et *Blanc*. Le *Rouge* représente la saturation d'un nœud (aucune information additionnelle n'est attendue pour ce nœud). Le *Noir* correspond à une *ressource*, *i.e.* un nœud potentiellement saturé (des informations peuvent cependant être ajoutées à ce nœud par fusion de nœuds). Enfin, le *Blanc* représente la non-saturation du nœud. Celui-ci correspond alors à un *besoin*, il va donc devoir se combiner avec un nœud ressource qu'il complétera. Les combinaisons de nœuds autorisées sont présentées en figure 4.5.

	$\bullet_N$	$\bullet_R$	$\circ_B$	$\bot$
$\bullet_N$	$\bot$	$\bot$	$\bullet_N$	$\bot$
$\bullet_R$	$\bot$	$\bot$	$\bot$	$\bot$
$\circ_B$	$\bullet_N$	$\bot$	$\circ_B$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

FIG. 4.5 – Règles de combinaison des couleurs.

Dans ce contexte, la résolution d'une description correspond à l'ensemble des modèles minimaux *saturés*, c'est-à-dire pour lesquels tous les nœuds sont colorés en *Noir* ou en *Rouge*. Ce qui traduit le fait que tous les besoins ont été absorbés, *i.e.* tous les nœuds blancs ont été identifiés avec un nœud noir.

Exemple 14 (Description du double groupe prépositionnel)**.** *Dans notre approche métagrammaticale de description de fragments d'arbres, une façon de représenter le cas du double groupe prépositionnel*[116] *est d'utiliser un fragment pour l'épine verbale et deux fois le fragment du groupe prépositionnel (cf figure 4.6).*

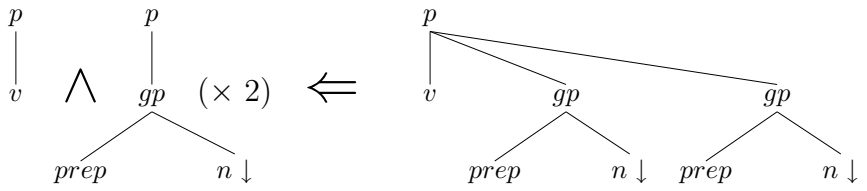


FIG. 4.6 – Cas du double groupe prépositionnel.

[116] *E.g.* Jean conduit de Pau à Toulouse.

Dans une telle situation, l'utilisation de nœuds colorés permet une description concise dans la mesure ou il suffit d'annoter la racine du groupe prépositionnel en blanc, tous les autres nœuds étant noirs (cf figure 4.7). Ainsi, la double instanciation du groupe prépositionnel associé à l'instanciation de l'épine verbale entraînera le calcul de l'unique fragment attendu. On note qu'on évite, par l'emploi des couleurs (1) les conflits de noms de nœuds issus de la présence de deux instances d'une même classe (illustré précédemment, cf section 4.1.2), et (2) d'avoir à définir plusieurs équations de nœuds pour s'assurer que les racines des différents fragments coïncident (cf (Gardent et Parmentier, 2006)).

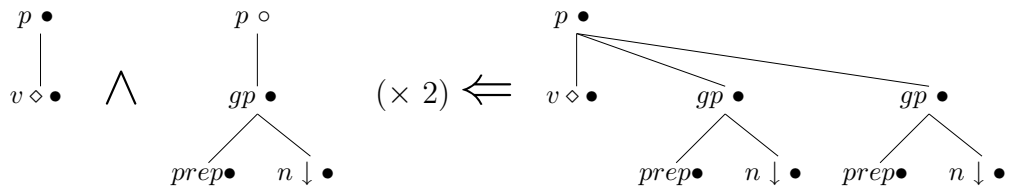


FIG. 4.7 – Cas du double groupe prépositionnel avec descriptions colorées.

Résumé Dans ce paragraphe, nous avons vu une seconde catégorie de contraintes, appelées contraintes opérationnelles. Celles-ci permettent, au moyen d'un langage de couleurs, de contrôler semi-automatiquement l'identification de nœuds. L'intérêt de telles contraintes est d'obtenir un langage de description métagrammatical purement déclaratif, où l'utilisateur économise la tâche (fastidieuse pour des grammaires de taille importante) de définir des équations de nœuds, afin de s'assurer de la correcte combinaison des fragments d'arbres.

Le danger principal de ce langage de couleurs est qu'il est à manipuler avec une certaine méthodologie, dans le cas contraire le concepteur de la métagrammaire peut facilement sous-estimer le nombre de modèles pour une description colorée donnée (voir section 4.4.3[117]).

4.3.2.3 Contraintes dépendantes du langage

Une troisième catégorie de contraintes regroupe les contraintes dépendantes du langage décrit. Ces contraintes ont pour but d'assurer que les arbres produits respectent certains critères dépendant du langage cible. En ce qui concerne le français, un exemple de telles contraintes correspond à l'unicité et l'ordonnancement des clitiques tel qu'illustré ci-dessous.

Exemple 15 (Unicité et ordonnancement des clitiques)**.** *En français, les clitiques sont des particules ayant les deux propriétés décrites dans (Perlmutter, 1970), à savoir :*

- *(i) les clitiques précèdent le verbe dans un ordre fixé par leur rang,*
- *(ii) deux clitiques précédant un verbe ne peuvent pas avoir le même rang.*

[117] Voir aussi (Cohen-Sygal et Wintner, 2006b), où les auteurs abordent ce type de problème pour le développement de grammaires syntagmatiques guidées par les têtes.

Ainsi, sachant que les clitiques le, la ont le rang 3 et lui le rang 4, les phrases suivantes sont telles que (1a) respectent (i) mais pas (1b), et (1c) viole (ii) :

(1) a. Jean le$_3$ lui$_4$ donne

*b. *Jean lui$_4$ le$_3$ donne*

*c. *Jean le$_3$ la$_3$ donne*

Pour gérer un tel cas de figure, nous allons annoter les nœuds de catégorie clitique avec un entier désignant leur rang. Ainsi, il sera possible d'émettre les contraintes (a) qu'un arbre contenant deux clitiques de même rang et (b) qu'un arbre dont l'ordre des rangs des clitiques ne respecte pas l'ordre sur $\mathcal{N}$ ne sont pas valides.

Ainsi, dans la métagrammaire de (Crabbé, 2005b), le cas des clitiques est traité comme exemplifié en figure 4.8.

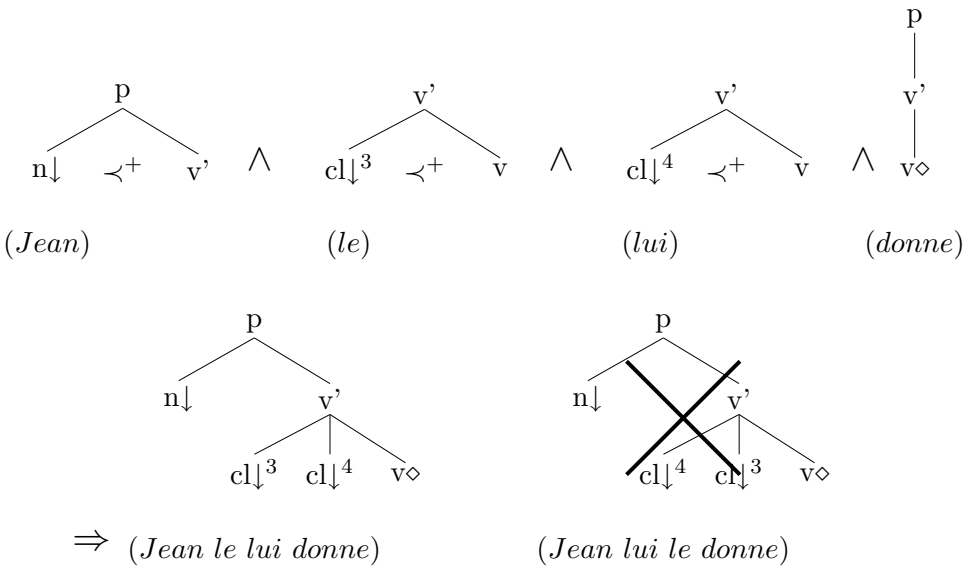


FIG. 4.8 – Traitement des clitiques.

Dans la métagrammaire, on définit les quatre fragments pour le sujet, le clitique de rang 3 (le), le clitique de rang 4 (lui) et l'épine verbale respectivement. On décrit alors l'arbre du verbe avec arguments clitiques de rang 3 et 4 comme la conjonction de ces fragments. On note qu'on n'a définit aucune relation entre les deux nœuds de catégorie clitique, la description est ainsi laissée volontairement sous-spécifiée. Deux arbres devraient donc être calculés, cependant le principe d'ordonnancement des clitiques permet d'éviter la construction de l'arbre erroné.

4.3.2.4 Contraintes théoriques

Une quatrième catégorie de contraintes est ce que nous appelons les contraintes théoriques. Ce terme désigne les contraintes indépendantes du langage cible mais inhérentes à l'utilisation du formalisme grammatical considéré. Dans le cas des grammaires d'arbres adjoints, une telle contrainte correspond au Principe de Coocurrence Prédicat-Argument (PCPA). Comme nous l'avons vu dans l'introduction aux grammaires TAG [118], cette contrainte (spécifiant qu'un arbre élémentaire associé à un prédicat doit contenir un nœud pour chaque argument de ce prédicat) est utilisée pour définir des arbres élémentaires fondés linguistiquement. D'autres contraintes théoriques sont données dans (Frank, 2002). Notons que cette catégorie de contraintes n'est pas implantée actuellement au sein du système XMG.

A ce jour, l'implantation de XMG inclut les contraintes suivantes :

1. Contraintes de bonne formation des arbres par rapport au formalisme TAG.
2. Contraintes de polarisation des nœuds, aussi appelées *principe des couleurs*.
3. Contraintes d'unicité d'une propriété de nœud[119], d'ordonnancement des clitiques, et d'arité des nœuds[120].

Dans le chapitre 5, nous verrons comment ces contraintes sont implantées dans le système XMG au moyen du paradigme de la Programmation par Contraintes, ce qui permet un traitement efficace de celles-ci. En effet, dans ce contexte, ces contraintes permettent activement de ne construire que les arbres valides (*i.e.* il ne s'agit pas d'une élimination *a posteriori*).

Notons enfin que ces différentes contraintes sont optionnelles[121], le but étant de laisser l'utilisateur libre de les appliquer ou non. Ainsi il devient possible de définir une librairie de contraintes dans laquelle le développeur de la métagrammaire choisirait en fonction du formalisme grammatical cible et de la langue décrite celles qu'il conviendrait d'appliquer.

4.4 Conclusion

Dans ce chapitre, nous avons introduit le formalisme métagrammatical nous permettant de produire semi-automatiquement des grammaires d'arbres adjoints de taille réaliste et dont les arbres sont associés à des formules de sémantique plate. Ce formalisme, dont la forme initiale a été introduite par (Duchier, 2003), permet d'atteindre un haut degré de factorisation. En effet, (Crabbé, 2005b) l'a utilisé pour décrire une grammaire TAG du français de taille importante, qui contient près de 6000 arbres produits à partir d'une description contenant un peu moins de 300 classes.

[118] Voir section 2.1.4 page 41.

[119] Cette propriété de nœud est un paramètre de la contrainte d'unicité.

[120] *I.e.*, on peut spécifier le nombre maximum de fils d'un nœud donné.

[121] A l'exception des contraintes de validité TAG qui sont appelées implicitement dans l'implantation actuelle lorsqu'une grammaire TAG est produite.

4.4.1 Sur l'expressivité du formalisme

L'un des points forts du formalisme XMG réside dans son expressivité. Comme nous l'avons vu, il intègre un traitement efficace des noms de variables au moyen d'un concept d'import / export inspiré par la Programmation Orientée Objets. Ce traitement des noms de variables peut être complété par un système de polarisation permettant d'atteindre un plus haut niveau de factorisation, cela sans rompre le caractère intuitif du langage. (Gardent et Parmentier, 2006) montrent en particulier que ce langage est particulièrement adapté à la description de différents phénomènes linguistiques.

Enfin, (Crabbé, 2005a) a montré comment ce langage peut être utilisé pour décrire relativement rapidement une grammaire d'arbres adjoints de couverture significative.

4.4.2 Sur l'extensibilité du formalisme

Une autre caractéristique du formalisme XMG est son extensibilité, tel qu'introduit dans (Duchier et al., 2005; Parmentier et Le Roux, 2005). En effet, ce formalisme distingue un langage de définition de blocs élémentaires et un langage de contrôle des combinaisons de ces blocs. En outre, le formalisme XMG intègre un concept de *dimension*, qui peut être vu comme un typage de ces blocs élémentaires. Ainsi, moyennant le fait que nous pouvons définir un langage pour différentes dimensions, il devient possible d'étendre la description métagrammaticale à d'autres niveaux. Actuellement, XMG intègre deux dimensions, une dimension syntaxique permettant de décrire des structures d'arbres (utilisé pour décrire des grammaires d'arbres adjoints et des grammaires d'interaction (Perrier, 2003)) et une dimension sémantique pour un sémantique plate à la (Gardent et Kallmeyer, 2003) (voir chapitre 6). La définition de nouvelles dimensions pourrait permettre par exemple de tester différents types de représentation sémantique (λ-calcul, etc).

L'extensibilité de ce formalisme s'exprime également dans son procédé de contraintes optionnelles visant à la bonne formation des structures produites. En définissant de nouvelles contraintes appropriées à un formalisme grammatical cible ou à un langage cible, il devient possible d'étendre le contrôle des structures produites.

4.4.3 Sur les limites du formalisme

Bien qu'utilisé avec succès pour décrire une grammaire TAG du français de couverture importante (Crabbé, 2005b), et pour y inclure une portée sémantique (Gardent, 2006), le formalisme comporte des limites. Les deux principales limites à ce jour concernent (i) le traitement des héritages « en losange », et (ii) l'utilisation des couleurs.

Concernant (i), comme nous l'avons vu en section 4.1.2, le cas de l'héritage multiple d'une même classe (bas du losange) est problématique. En effet, lorsque l'on hérite en double d'une classe, nous créons deux instances de cette classe. Ainsi toutes les variables sont créées en deux exemplaires. Pour indiquer qu'il s'agit du même fragment, il faut définir

à la main un ensemble d'équations de nœuds assurant l'unification des deux instances de chacun des nœuds.

Concernant (ii), l'utilisation des couleurs est dangereuse dans le sens où son interprétation par le concepteur de la métagrammaire est sujette à une sous-estimation du nombre de modèles admissibles.

Tout d'abord, récapitulons le rôle des couleurs. Les couleurs sont intégrées au langage de description d'arbre. Celles-ci permettent d'annoter les nœuds de la description afin d'indiquer lesquels dénotent un apport d'information (notion de ressource) ou un besoin d'information. Ainsi, les couleurs guident la combinaison des fragments d'arbres en déclenchant des identifications de nœuds lors de la recherche des modèles minimaux d'une description.

Notons que lors de la combinaison des fragments d'arbres, les descriptions les représentant sont accumulées puis tous les modèles minimaux correspondant à la description totale sont calculés. Or si le concepteur de la métagrammaire utilise une séquence de combinaisons pour prédire les arbres qu'il décrit, il peut être surpris par la production d'arbres non-envisagés liée au fait que le langage de couleurs est non-associatif (Cohen-Sygal et Wintner, 2007). Nous illustrons cela dans l'exemple qui suit[122].

Exemple 16 (Influence des couleurs sur la description métagrammaticale)**.** *Considérons les trois fragments d'arbres colorés de la figure 4.9.*

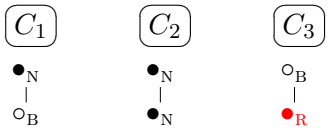


FIG. 4.9 – Influence des couleurs sur la description métagrammaticale.

Quelles solutions trouvons-nous si nous calculons les modèles de $(C_2 \wedge C_3)$ *? La réponse correspond aux arbres de la figure 4.10.*

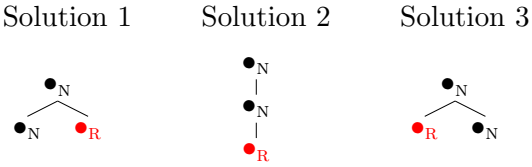


FIG. 4.10 – Résultat de $(C_2 \wedge C_3)$.

A présent, nous considérons également C_1*, en plus des trois arbres décrits dans la figure 4.10. Cette nouvelle description a trois solutions, celles de la figure 4.11.*

A présent, on s'intéresse aux solutions de $(C_1 \wedge C_2) \wedge C_3$*. Cette combinaison de descriptions a pour modèles les trois solutions données ci-dessus, plus deux nouvelles solutions, données figure 4.12.*

[122]Cet exemple est issu de (Cohen-Sygal et Wintner, 2006a).

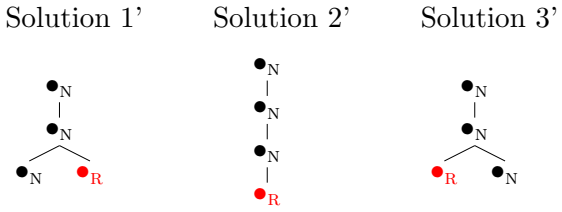


FIG. 4.11 – Solutions de $(C_2 \wedge\ C_3) \wedge\ C_1$.

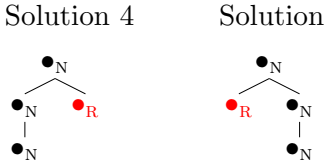


FIG. 4.12 – Nouvelles solutions de $(C_1 \wedge\ C_2) \wedge\ C_3$.

En conclusion, $(C_1 \wedge\ C_2) \wedge\ C_3 \neq C_1 \wedge\ (C_2 \wedge\ C_3)$*. Or, le compilateur de métagrammaire va rechercher l'ensemble des solutions pour la description* $C_1 \wedge\ C_2 \wedge\ C_3$ *(indépendemment de l'ordre de combinaison des classes). Ainsi, si le développeur de la métagrammaire ne prévoit pas toutes les solutions de toutes les combinaisons de classes possibles, il peut être surpris par le nombre élevé d'arbres. L'utilisation de notre formalisme doit donc s'accompagner d'une méthodologie rigoureuse.*

Chapitre 5

XMG : un compilateur de métagrammaires multi-paradigme

Sommaire

Dans ce chapitre, nous allons présenter l'implantation du formalisme XMG. Nous montrerons notamment comment l'emploi de différents paradigmes de programmation nous permet d'avoir un outil extensible et efficace.

5.1 La métagrammaire vue comme une grammaire de clauses définies

Nous allons voir que le langage métagrammatical défini précédemment se ramène à une *grammaire de clauses définies* (*Definite Clause Grammar* –DCG), ce qui a une incidence directe sur son traitement automatique.

5.1.1 Comparaison entre métagrammaire et programme logique

Comme nous l'avons vu au chapitre 4, le formalisme XMG intègre un concept d'abstraction (classe), un langage de description de fragments (contenu de classe), et un langage de contrôle des combinaisons de ces fragments (conjonction et disjonction de classe).

Notons que l'héritage dans XMG a pour but de rendre accessible directement le contenu d'autres classes (extension de la portée de certaines variables). A ce titre, l'héritage peut être considéré comme une conjonction de classes associée à une fusion des espaces de noms. En conséquence, nous omettons volontairement ce concept dans cette comparaison.

Ainsi, si nous considérons les descriptions d'arbres comme des symboles terminaux, le langage du formalisme XMG est comparable à un programme logique, *cf* figure 5.1.

Programme Logique	**Métagrammaire XMG**
Clause	Classe
Tête de clause	Nom
Corps de clause	Contenu de classe
Symbole terminal	Description
Conjonction	Conjonction de contenus
Disjonction	Disjonction de contenus
Requête	Evaluation

FIG. 5.1 – Comparaison entre un programme logique et une métagrammaire XMG.

Plus formellement, nous pouvons définir notre langage métagrammatical comme suit :

$$\textit{Clause} \quad ::= \quad \textit{Nom} \rightarrow \textit{But} \tag{5.1}$$

$$\textit{But} \quad ::= \quad \textit{Description} \mid \textit{Nom} \mid \textit{But} \vee \textit{But} \mid \textit{But} \wedge \textit{But} \tag{5.2}$$

$$\textit{Requête} \quad ::= \quad \textit{Nom} \tag{5.3}$$

Les définitions (4.1), (4.2) et (4.10) correspondent respectivement à (5.1), (5.2) et (5.3) introduites dans le chapitre 4. En d'autres termes, notre métagrammaire n'est autre qu'une DCG.

Remarque Dans une DCG, les variables introduites dans une clause ont une portée locale à la clause. Ainsi, dans notre comparaison, nous omettons les exports de noms de variables (nous considérons uniquement des variables dont la portée est locale à la classe).

Cette comparaison nous montre, qu'à traitement des noms de variables près, une métagrammaire dans le formalisme XMG correspond à une DCG dans laquelle les symboles terminaux sont remplacés par des descriptions d'arbres.

En conclusion, l'accumulation des descriptions partielles contenues dans les classes de notre métagrammaire revient à rechercher le langage généré par la DCG sous-jacente. Notons que dans le cas d'une DCG, nous avons un symbole distingué (l'axiome) servant de point de départ aux dérivations. Dans notre cas, il est possible de spécifier un nombre quelconque d'axiomes (les évaluations de notre métagrammaire).

5.1.2 Traitement des différents niveaux de description

Il nous manque le support des différents niveaux de description (appelés *dimension* précédemment). Comment, dans le contexte des DCG, pouvons nous distinguer différents types de *symboles terminaux*?

Tout d'abord revenons sur ce qu'est une DCG. Conceptuellement, une DCG est un formalisme permettant de décrire des grammaires hors-contextes sous forme d'un programme logique.

Techniquement, une DCG est un enrobage syntaxique sur les différences de listes (voir (Blackburn et al., 2006)). Ces différences de listes permettent de réaliser l'analyse syntaxique de suites de symboles terminaux (en l'occurrence des mots du vocabulaire). Plus précisément, ces listes *accumulent* les symboles terminaux.

Dans notre approche multi-dimensionnelle, nous souhaitons accumuler différents types de symboles terminaux. Pour réaliser cela, nous utilisons une extension des DCG : les grammaires de clauses définies étendues (EDCG) présentées dans (Van Roy, 1990). Ce formalisme permet la définition d'accumulateurs distincts pour stocker les différents types d'information, et également d'associer à ces accumulateurs un traitement spécifique (opération logique ou algébrique). Pour information, dans leur implantation, les EDCG sont compilées en DCG s contenant une liste d'accumulateurs, au moyen d'un traitement pré-processeur.

5.2 1ère étape : compilation de la métagrammaire en code intermédiaire

Nous venons de voir qu'une métagrammaire n'est autre qu'une EDCG (à traitement des espaces de noms près). En conséquence, le traitement automatique de cette métagrammaire en vue de produire une grammaire se fait au moyen d'un compilateur de programmes logiques. Comme dans tout compilateur, le traitement du programme d'entrée passe par une phase de traduction du code en code abstrait, code qui sera exécuté, dans notre cas,

par une machine virtuelle, qui constitue le coeur du compilateur. De plus, nous aurons une troisième phase de résolution des descriptions d'arbres (*cf* section 5.4).

5.2.1 Analyse de la syntaxe concrète de la métagrammaire et vérifications statiques

Le premier traitement que subit la métagrammaire correspond à l'analyse syntaxique. Cette analyse de la syntaxe concrète de la métagrammaire va permettre de construire l'arbre de syntaxe abstraite, structure sur laquelle seront effectués les traitements suivants.

Dans notre cas, nous utilisons un segmenteur et un analyseur syntaxique générés automatiquement à partir d'une spécification de la syntaxe concrète de notre langage sous forme d'une grammaire hors-contexte. Le générateur d'analyseur que nous avons utilisé est GUMP (sur-couche des outils libres `flex` et `bison`), il fait partie de l'environnement de développement Mozart (Oz-Mozart, 2005).

Une fois l'arbre de syntaxe abstraite construit, celui-ci sert de base à une phase de vérification statique (*i.e.* avant exécution du code abstrait). Cette vérification a pour but de détecter les erreurs dans la métagrammaire et d'émettre un maximum d'avertissements sur des erreurs éventuelles (par exemple, omission d'une information telle que la couleur d'un nœud[123]).

Les vérifications qui sont effectuées concernent principalement l'existence des classes importées (vérification de la cohérence du graphe d'héritage), l'appel de classe avec le bon nombre de paramètres et l'utilisation de variables ayant été préalablement déclarées.

Remarque Comme nous le verrons dans le chapitre 6, le formalisme XMG intègre un *typage* des données manipulées (structures de traits, propriétés de nœuds). Cependant, à l'heure actuelle, ce typage n'est pas utilisé dynamiquement, et seulement légèrement statiquement. Plus précisément, le typage en place actuellement est utilisé principalement dans le but de déterminer les constantes[124].

5.2.2 Traitement des espaces de noms

L'étape suivant la vérification de l'arbre de syntaxe abstraite est la gestion des espaces de noms. Le but de ce traitement est d'associer à chaque variable de la métagrammaire une adresse mémoire permettant d'y référer.

Cette résolution des noms de variables se fait en examinant les déclarations d'Import / Export de chaque classe, en partant des classes *finales* de la métagrammaire (les

[123] On notera que l'absence pour un nœud donné d'une couleur, même pour une métagrammaire utilisant cette contrainte opérationnelle des couleurs, ne peut être considéré comme une erreur puisque le nœud en question peut être unifié explicitement avec un nœud coloré ailleurs dans la métagrammaire.

[124] La syntaxe concrète du formalisme XMG n'utilise aucune convention de notation pour différencier les constantes des variables.

classes évaluées). Pour chaque classe parcourue, nous calculons (a) les variables accessibles localement et (b) les variables exportées[125].

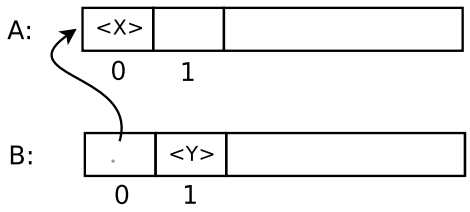
Plus précisément, nous associons à chaque variable une structure de données indiquant la provenance de la variable. Cette structure se présente sous forme d'un enregistrement[126] et sera utilisée par la machine virtuelle lors de l'unification de variables. En particulier, si une variable est accessible par héritage, son enregistrement dans la classe fille sera complexe. Cela est illustré dans l'exemple suivant :

considérons les deux classes A et B suivantes :

$$\begin{aligned} \langle X \rangle \Leftarrow A \quad &\rightarrow \quad \{ \dots X \dots \} \\ \langle X, Y \rangle \Leftarrow B \quad \angle \quad A \quad &\rightarrow \quad \{ \dots X \dots Y \dots \} \end{aligned}$$

Nous avons une première classe A introduisant une variable X et une seconde classe B héritant de A et introduisant une variable Y. Lors de notre traitement des espaces de noms pour associer à chacune de variables accessibles pour une classe donnée, nous allons :

1. créer des espaces (tableaux) pour chaque classe, en réservant des cases pour les classes importées, ici, la première case du tableau de la classe B est réservé pour contenir l'adresse du tableau de la classe A,
2. associer à chaque variable un enregistrement représentant l'adresse de la variable dans cet espace de noms. Ici, X dans la classe A est associé avec l'enregistrement *index(0)*, et X dans la classe B avec *index(0 index(0))* traduisant le fait que X est la variable occupant la première case de la structure dont l'adresse est contenue dans la première case de l'espace de nom de la classe courante (B) :



5.2.3 Production de code intermédiaire

A l'issue de la phase de traitement des espaces de noms, nous pouvons effectuer la production du code abstrait. Ce code contient des instructions pour la machine virtuelle.

Le code abstrait produit par la compilation résulte d'un dépliage de la description métagrammacale. Par exemple, la classe suivante :

$$\langle X, Y \rangle \Leftarrow A \quad \rightarrow \quad X\,[cat : p] \rightarrow Y\,[cat : v]$$

[125] NB : (b) est un sous-ensemble de (a).
[126] Structure de données plus connue dans la littérature en tant que *Record*.

est compilée en[127] :

```
conj(make_node(X)
  conj(synfeatget(X X1)
    conj(make_avm(X1 [ (cat,p) ])
      conj(make_node(Y)
        conj(synfeatget(Y Y1)
          conj(make_avm(Y1 [ (cat,v) ])
            conj(syndom(X Y strict)
              noop)))))))
```

En d'autres termes, cette classe contient la conjonction d'une instruction de création d'un nœud d'adresse `X`, et de structure de trait associée d'adresse `X1`, et de contenu une liste de couple (attribut,valeur), ici (cat,p), et d'une conjonction d'une instruction de création d'un nœud `Y` et d'une structure de trait d'adresse associée d'adresse `Y1` et de contenu (cat,v).

On remarque que notre code abstrait a la forme d'un enregistrement, dont le contenu est lui même un enregistrement, etc. Chacun de ces enregistrements correspond à une instruction reconnue par la machine virtuelle.

5.3 2e étape : exécution du code par une machine virtuelle de type WAM

Le code produit à l'issue de la compilation est exécuté par une machine virtuelle spécifique. Cette machine virtuelle est inspirée de la *Warren's Abstract Machine* (WAM) (Ait-Kaci, 1991; Van Roy, 1993)[128].

5.3.1 Modèle d'exécution de notre machine virtuelle

Notre machine virtuelle est composée des structures de données suivantes :

(a) la **pile d'exécution** (*STACK*), contenant les instructions du code intermédiaire associées à l'environnement dans lequel celles-ci doivent être interprétées,

(b) la **table des symboles** où sont stockés les constantes à portée globale,

(c) la **trace d'exécution** (*TRAIL*) où sont enregistrés les liens (*bindings*) entre variables unifiées au cours de l'évaluation d'une alternative d'une disjonction,

(d) le **point de choix** (*Choice Point*), compteur permettant de numéroter les alternatives d'une disjonction, de manière à pouvoir revenir en arrière si une unification échoue. L'incrémentation du point de choix provoque la création d'un nouvel environnement (voir ci-dessous),

[127] Pour des raisons de clarté, nous utilisons ici le nom de la variable de nœud et non pas son enregistrement index.

[128] Le lecteur intéressé pourra également se référer à (Büttcher, 2002) présentant de manière informelle l'implantation d'une WAM en Java.

(e) l'**environnement** courant, tableau contenant les liens entre variables utilisées dans la classe en cours d'évaluation,

(f) la **pile des environnements** où sont stockés les environnements créés à chaque évaluation d'une contenu d'une classe,

(g) les **listes d'accumulation**, structures recevant les descriptions accumulées par la machine virtuelle. Une liste correspond à une dimension de description (syntaxe, sémantique), et contient les instructions exécutées par la machine virtuelle, suivant la dimension à laquelle l'instruction se rapporte (par exemple une instruction de type `synnode` est enregistrée dans la liste d'accumulation de la dimension syntaxique).

En outre, notre machine virtuelle utilise des structures spécifiques au traitement des interfaces (pile, trace et environnement d'interface). Toutes ces structures constituent les registres de notre machine virtuelle.

Le cœur de la machine virtuelle est la **boucle d'exécution**, qui dépile puis exécute les couples (instruction, environnement) contenus dans la pile d'exécution, jusqu'à ce que celle-ci soit vide. Lors de l'exécution d'une disjonction, nous enregistrons l'état de la machine virtuelle (le contenu de ses registres), nous augmentons le compteur *Choice Point*, et nous exécutons le code contenu dans l'alternative en question. Si une unification (entre variables de nœuds ou variables de traits) échoue, ou s'il reste des alternatives à explorer (point de choix supérieur à un), nous faisons un retour-arrière (*backtracking*). L'effet du retour-arrière est de dépiler la trace d'exécution, ce qui correspond à défaire les liens entre variables effectués lors de l'évaluation de l'alternative précédente. Lorsque la pile est vide, si aucune unification n'a échouée, des descriptions accumulées sont stockées dans les registres dédiés à cet effet.

Dans ce contexte, la pile d'exécution contient au départ le code correspondant aux classes dont l'évaluation est demandée (les requêtes de notre langage métagrammatical). Chaque appel de classe provoque la création d'un nouvel environnement pour exécuter le code de cette classe. Cet environnement est placé dans l'environnement courant, et est associé à une adresse de retour, afin de pouvoir réaliser le retour-arrière.

5.3.2 Jeu d'instructions de notre machine virtuelle

Notre machine virtuelle exécute les instructions appartenant à la liste suivante :

`exec_loop` démarre la boucle d'exécution.

`backtrack` vérifie s'il y a des alternatives à explorer avant d'effectuer l'annulation des liens entre variables (via `trail_unwind`).

`reset` remet à zéro les registres de la machine.

`trail_unwind` dépile la trace d'exécution et exécute l'instruction d'annulation des liens (instruction préfixée par `trail_`).

`noop` ne fait rien.

`conj(E1 E2)` exécute l'instruction `E1` et place `E2` en sommet de la pile d'exécution.

`disj(E1 E2)` augmente le *Choice Point*, enregistre l'état des registres et la prochaine instruction à exécuter (`E2`) dans la trace d'exécution, et exécute `E1`.

`trail_exec(E Env Stack Cp Iface Ifenv Ifstack Envstack L)` repositionne les registres dans l'état défini par les paramètres de la fonction, exécute l'instruction `E` (qui représente l'alternative d'une disjonction), et relance la boucle d'exécution.

`ifunify(E)` unifie l'interface `E` avec l'interface actuelle

`call(A L Slot)` crée un nouvel environnement pour interpréter le code correspondant à la classe `A`, d'arguments `L`. `Slot` représente la valeur de retour (adresse de l'environnement du call dans l'environnement courant, permet d'accéder aux variables héritées).

`trail_slot(Env Slot)` remet à zéro l'adresse `Slot` de l'environnement Env.

`unify(X EnvX Y EnvY)` unifie les termes `X` et `Y` d'environnements respectifs `EnvX` et `EnvY`.

`trail_avm(B)` annule le lien `B` entre deux structures de traits.

`unify_avm(X EnvX Y EnvY)` unifie les structures de traits `X` et `Y` d'environnement respectifs `EnvX` et `EnvY`.

`unify_list(L1 E1 L2 E2)` unifie les listes `L1` et `L2` d'environnement respectif `E1` et `E2`.

`make_avm(V L)` crée une nouvelle structure de traits contenant les attributs/valeurs spécifiés par `L` et l'unifie avec la variable `V`.

`bind(Env I Binding)` crée le lien `Binding` à l'adresse `I` de l'environnement `Env`. Si cette adresse contient une suspension, l'exécute (variable nécessaire à une instruction suspendue).

`trail_binding(Env I V)` défait le lien situé à l'adresse `I` de l'environnement `Env`, et le remplace par l'ancien lien `V`.

`unify_node(NodeX EnvX NodeY EnvY)` unifie les nœuds `NodeX` et `NodeY` d'environnement respectif `EnvX` et `EnvY`.

`trail_node(B)` défait le lien `B` entre deux nœuds.

`dot(V E1 E2)` unifie la variable `V` avec le résultat de `E1.E2` (qui est soit une variable de l'enregistrement des exports, ou une variable d'une structure de traits).

`suspend(Env I Code)` crée une suspension (accès au contenu d'une variable non-encore unifiée). En pratique stocke l'instruction `Code` à l'adresse `I` de l'environnement `Env` pour indiquer une variable dont la valeur est attendue par `Code`.

`synnode(E)` crée une structure de type nœud et l'unifie avec la variable `E`. Stocke de plus cette instruction et l'environnement courant dans la liste d'accumulation de la dimension syntaxique.

`syndom(E1 OP E2)` demande une suspension si `E1` ou `E2` est une variable libre. Stocke cette instruction et l'environnement courant dans la liste d'accumulation syntaxique.

`eq(E1 E2)` unifie `E1` et `E2` dans l'environnement courant.

`synpropget(E1 E2)` unifie `E2` avec la structure de traits des propriétés du nœud `E1`. Crée une suspension si `E1` est une variable libre.

`synfeatget(E1 E2)` unifie `E2` avec la structure de traits du nœud `E1`. Crée une suspension si `E1` est une variable libre.

`semlit(L P A)` stocke un littéral d'étiquette `L`, de prédicat `P` et d'arguments `A` dans la liste d'accumulation de la dimension sémantique.

`semvar(V)` stocke une variable `V` dans la liste d'accumulation sémantique.

`semdom(S1 O S2)` stocke un opérateur de dominance de portée entre les variables `S1` et `S2` dans la liste d'accumulation sémantique.

`export_result` vérifie que l'environnement courant ne contient plus aucune suspension et retourne l'état des listes d'accumulation.

`trail_env(E CP)` associe le point de choix `CP` à l'environnement `E` (utilisé lors du retour-arrière).

`deref(X EnvX XX EnvXX)` recherche le lien d'une variable `X` dans l'environnement `EnvX` et retourne la valeur `XX` et son environnement d'interprétation (`EnvXX` peut être `EnvX` lui-même si deux variables locales sont unifiées).

5.3.3 Comparaison entre notre machine virtuelle et la WAM

Notre machine virtuelle a un mode de fonctionnement très proche de celui de la WAM (utilisation d'une pile d'exécution, d'une trace d'exécution et d'un point de choix). Cependant, à la différence de la WAM, notre machine virtuelle n'utilise pas la technique de copie de structure (*structure copying*), mais celle du partage de structure (*structure sharing*). Cette technique consiste à ne pas dupliquer les structures lors de l'unification de termes, mais plutôt à représenter un terme au moyen d'un couple (squelette, environnement d'interprétation). La conséquence de l'utilisation du partage de structure est (a) qu'il faut effectuer un déréférencement de pointeurs lors de l'unification, ce qui peut prendre du temps, et (b) que l'on économise de la mémoire.

Pourquoi ne pas utiliser la WAM directement ? Comme nous le voyons, la machine virtuelle de XMG a un fonctionnement similaire à un compilateur de programmes logiques tel que la WAM. On pourrait alors se demander pourquoi ne pas utiliser une machine virtuelle existante (c'est-à-dire n'utiliser qu'un pré-processeur compilant la métagrammaire en une DCG classique) ?

La réponse est que la définition d'une machine virtuelle spécifique nous permet (a) d'avoir une machine virtuelle extensible pour pouvoir supporter l'unification de structures de données non-standards (à l'heure actuelle les structures unifiées sont les nœuds et les structures de traits), (b) de pouvoir étendre l'opération d'unification pour supporter par

exemple l'unification de traits polarisés (*cf* formalisme des grammaires d'interaction (Perrier, 2003)), et (c) nous avons besoin d'effectuer un traitement additionnel sur la sortie de la machine virtuelle (qui sont, dans notre cas, des descriptions d'arbres).

D'un point de vue technique, notre machine virtuelle est implantée au moyen du paradigme de la programmation orientée objet, plus précisément, chaque méthode correspond à une instruction du code intermédiaire. Cela rend l'extension du jeu d'instructions, et en conséquence de la machine virtuelle, plus aisée.

5.4 3e étape : résolution de descriptions

Chaque contenu des accumulateurs retourné par la machine virtuelle, ($Desc_1$, ..., $Desc_n$), est envoyé à un module de résolution au sein duquel chaque dimension i est traitée au moyen d'un résolveur spécifique S_i. Ce résolveur spécifique a pour but de calculer les modèles de chacune des descriptions : $S_i(Desc_i)$.

Ainsi, les éléments de notre grammaire, pour un contenu des accumulateurs donné, correspondent aux modèles calculés par le module de résolution : $\{(M_1, \ldots, M_n) \mid M_i \in S_i(Desc_i)\ avec\ 1 \leq i \leq n\}$.

Notons que dans l'implantation actuelle de XMG, seules trois dimensions (*i.e.* trois accumulateurs) sont traitées, une dimension *syntaxique* dont le contenu est une description d'arbres, une dimension *sémantique* dont le contenu est une liste de prédicats, et une dimension dite *dynamique* dont le contenu est une structure de traits utilisée pour l'ancrage de la grammaire avec le lexique (voir chapitre 6).

Le traitement effectué par le résolveur de la dimension sémantique est trivial, puisqu'il retourne les prédicats accumulés par la machine virtuelle (*i.e.*, la description sémantique elle-même[129]). Il en est de même pour la résolution de la dimension dynamique. Par contre, concernant la dimension syntaxique, le résolveur de descriptions doit calculer l'ensemble des modèles d'arbres minimaux correspondant à la description.

5.4.1 Résolution de descriptions d'arbres

Pour résoudre les descriptions d'arbres de notre dimension syntaxique, nous utilisons un résolveur pour logiques de descriptions d'arbres. Ce résolveur est implanté au moyen du paradigme de la Programmation par Contraintes (*Constraint Programming*).

L'application de la Programmation par Contraintes au problème de la résolution de descriptions à base de dominance[130] n'est pas nouveau, une formalisation de la définition du problème de la recherche des modèles minimaux d'une description en tant que problème de satisfaction de contraintes (*Constraints Satisfaction Problem* – CSP) est donnée dans (Duchier et Gardent, 1999; Duchier, 1999; Duchier et Niehren, 2000; Duchier, 2000).

[129] En d'autres termes, $S_sem ::= Id$ (l'Identité).

[130] L'utilisation, dans le cadre des métagrammaires, d'une logique de dominance plutôt qu'une logique de parenté moins expressive, remonte à (Rogers et Vijay-Shanker, 1992).

La Programmation par Contraintes offre un cadre de travail dans lequel il est possible de définir une spécification élégante et déclarative du problème de recherche des modèles, cette spécification constituant le programme de recherche des solutions lui-même.

En outre, les contraintes utilisées ici portent sur des ensembles d'entiers naturels pour lesquels il existe un environnement de programmation efficace au sein du système Oz-Mozart (Smolka, 1995; Oz-Mozart, 2005).

Nous avons donc réutilisé cette technique dans notre approche en l'étendant pour pouvoir supporter les contraintes spécifiques données en section 4.3. Nous allons tout d'abord présenter la résolution de descriptions d'arbres « classique », puis son extension aux contraintes spécifiques.

5.4.1.1 Mise en place d'une représentation d'arbre basée sur les ensembles d'entiers

L'idée générale de la résolution d'une description d'arbres ϕ au moyen de contraintes est de convertir les relations entre variables de nœuds de la description en contraintes sur des ensembles d'entiers.

Cette résolution se fait en deux temps : (a) mise en place d'une représentation d'arbre utilisant des ensembles et (b) conversion des relations entre variables de nœuds de la description en contraintes sur les ensembles contenus dans nos représentations d'arbre.

Concernant (a), nous associons à chaque variable de nœud X de ϕ un entier i. Ensuite nous représentons le nœud N_X^i correspondant à la réalisation de X dans un arbre d'un modèle par un n-uplet définissant la position de ce nœud dans l'arbre :

$$\begin{aligned} N_X^i \quad ::= \quad & (Eq, Up, Down, Left, Right, \\ & EqDown, EqUp, Side, Children, Parent, \\ & IsRoot, Label, Mark) \end{aligned} \tag{5.4}$$

où i est un entier unique[131], Eq (respectivement Up, $Down$, $Left$, $Right$) est l'ensemble des nœuds[132] égaux à (respectivement ancêtres de, descendants de, à gauche de, à droite de) N_X^i dans l'arbre (*cf* figure 5.2). Notons que ces cinq champs forment une partition de l'ensemble $\mathcal{V}_\phi$ des nœuds de la description, en d'autres termes :

$$\mathcal{V}_\phi \;=\; N_{X.Eq}^i \uplus N_{X.Up}^i \uplus N_{X.Down}^i \uplus N_{X.Left}^i \uplus N_{X.Right}^i \tag{5.5}$$

et étant donné que i est l'entier référant à X, nous avons :

$$i \in N_{X.Eq}^i \tag{5.6}$$

[131] Cet entier réfère de manière non-ambigüe à une variable de nœud de la description.
[132] Il faut lire « des entiers désignant les nœuds ».

Le champ $EqUp$ représente l'union disjointe des ensembles Eq et Up, il en est de même pour $EqDow$ et $Side$, ainsi :

$$N^i_{X.EqUp} = N^i_{X.Eq} \uplus N^i_{X.Up} \tag{5.7}$$
$$N^i_{X.EqDown} = N^i_{X.Eq} \uplus N^i_{X.Down} \tag{5.8}$$
$$N^i_{X.Side} = N^i_{X.Left} \uplus N^i_{X.Right} \tag{5.9}$$

Ces trois champs sont utilisés dans un but d'optimisation, en effet ils permettent d'améliorer la propagation des contraintes en constituant des résultats intermédiaires lors de la recherche des solutions.

Le champ $Children$ (respectivement $Parent$) réfère à l'ensemble des nœuds fils de N^i_X (respectivement à l'ensemble des nœuds pères[133]), ainsi nous avons les contraintes suivantes :

$$N^i_{X.Children} \subseteq N^i_{X.Down} \tag{5.10}$$
$$N^i_{X.Parent} \subseteq N^i_{X.Up} \tag{5.11}$$

$IsRoot$ est une variable booléenne indiquant si N^i_X est un nœud racine de l'arbre. $IsRoot$ est définie par :

$$IsRoot ::= cardinal(N^i_{X.Up}) = 0 \tag{5.12}$$

$Label$ réfère à une structure de traits ouverte associée au nœud et enfin $Mark$ est une variable d'unification contenant la valeur de la marque du nœud (ancre, pied, substitution, non-adjonction ou standard).

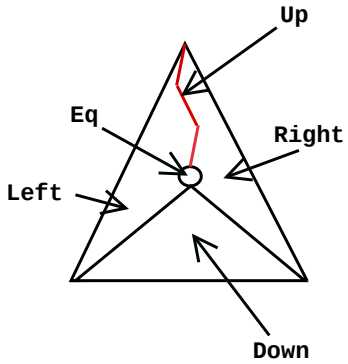


FIG. 5.2 – Représentation d'un nœud dans un modèle d'arbre.

Dans ce contexte, la résolution de la description consiste en le calcul d'une fonction d'interprétation associant à chaque variable de nœud de la description un nœud de l'arbre. Cette fonction et cet arbre forment le modèle recherché.

[133] Contrairement à ce que l'on pourrait penser, le cardinal de cet ensemble n'est pas 1 puisque le père de X peut correspondre à plusieurs nœuds de la description qui ont été identifiés.

5.4.1.2 Traduction des relations entre nœuds contenues dans les descriptions en contraintes sur des ensembles d'entiers

Concernant (b), la conversion des relations de notre description d'entrée en contraintes sur les ensembles contenus dans les représentations d'arbres se fait de manière explicite comme nous allons le voir.

Rappelons tout d'abord les relations de notre langage de description donnée en section 4.1 :

$$\begin{array}{lll} \textit{Description} & ::= & X \rightarrow Y \mid X \rightarrow^{+} Y \mid X \rightarrow^{*} Y \mid \\ & & X \prec Y \mid X \prec^{+} Y \mid X \prec^{*} Y \mid \\ & & X = Y \mid X[f{:}E] \mid X(p{:}E) \mid \\ & & \textit{Description} \wedge \textit{Description} \mid \textit{Description} \vee \textit{Description} \end{array} \tag{5.13}$$

En d'autres termes, les relations R pour lesquelles nous cherchons un équivalent sous forme de contraintes sur des ensembles sont celles ci-dessus, que nous étendons, dans un but de simplification (*cf supra*), par la possibilité de définir deux nœuds comme étant distincts[134] (représenté par $\neq$) :

$$R ::= \rightarrow \mid \rightarrow^{+} \mid \rightarrow^{*} \mid \prec \mid \prec^{+} \mid \prec^{*} \mid = \mid \neq \tag{5.14}$$

En outre, nous remarquons que certaines relations peuvent être définies en fonction d'autres :

$$\rightarrow^{+} ::= \rightarrow^{*} \wedge \neq \tag{5.15}$$

$$\prec^{+} ::= \prec^{*} \wedge \neq \tag{5.16}$$

Ainsi, nous cherchons un équivalent en termes de contraintes pour les relations de base suivantes :

$$R' ::= \rightarrow \mid \rightarrow^{*} \mid \prec \mid \prec^{*} \mid = \mid \neq \tag{5.17}$$

Dominance ($\rightarrow$) La première relation à considérer est la dominance directe entre nœuds. Dire que deux nœuds X et Y (associés respectivement aux entiers i et j) sont tels que $X \rightarrow Y$ revient à émettre les contraintes suivantes sur les champs des n-uplets représentant X et Y dans notre modèle :

$$\begin{array}{lllllllllll} X \rightarrow Y \equiv & & N^{i}_{X.Eq} & = & N^{j}_{Y.Parent} & \text{(i)} & \wedge & N^{i}_{X.Children} & \supseteq & N^{j}_{Y.Eq} & \text{(ii)} \\ & \wedge & N^{i}_{X.EqUp} & = & N^{j}_{Y.Up} & \text{(iii)} & \wedge & N^{i}_{X.Down} & \supseteq & N^{j}_{Y.EqDown} & \text{(iv)} \\ & \wedge & N^{i}_{X.Left} & \subseteq & N^{j}_{Y.Left} & \text{(v)} & \wedge & N^{i}_{X.Right} & \subseteq & N^{j}_{Y.Right} & \text{(vi)} \end{array} \tag{5.18}$$

[134] Le langage du formalisme est légèrement plus expressif que présenté précédemment, la relation de distinction n'apparaissant pas dans la syntaxe concrète, cette relation a été volontairement omise.

où (i) traduit le fait que l'ensemble des nœuds égaux (*i.e.*, identifiables) à X est inclus dans les nœuds parents de Y. (ii) signifie que l'ensemble des nœuds enfants de X contient l'ensemble des nœuds égaux à Y. (iii) représente le fait que l'ensemble des nœuds ancêtres ou égaux à X est égal à l'ensemble des nœuds strictement ancêtres de Y. (iv) illustre le fait que l'ensemble des nœuds descendants de X contient l'ensemble des nœuds égaux ou descendants de Y (*cf* figure 5.3). Enfin (v) et (vi) traduisent le fait que l'ensemble des nœuds à gauche (respectivement droite) de X sont contenus dans l'ensemble des nœuds à gauche (respectivement droite) de Y.

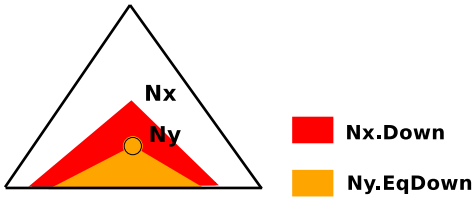


FIG. 5.3 – Contrainte sur ensembles liée à la dominance (iv).

Dominance large ($\rightarrow^*$) La seconde relation pour laquelle nous souhaitons définir une traduction en terme de contraintes sur des ensembles est la dominance large[135]. Cette conversion est la suivante :

$$\begin{array}{lllllllll} X \rightarrow^* Y \equiv & N^i_{X.EqDown} & \supseteq & N^j_{Y.EqDown} & \text{(i)} & \wedge & N^i_{X.EqUp} & \subseteq & N^j_{Y.EqUp} \quad \text{(ii)} \\ \wedge & N^i_{X.Left} & \subseteq & N^j_{Y.Left} & \text{(iii)} & & & & \\ \wedge & N^i_{X.Right} & \subseteq & N^j_{Y.Right} & \text{(iv)} & & & & \end{array} \tag{5.19}$$

où (i) représente le fait que si X domine largement Y alors le champ l'ensemble des nœuds égaux ou descendants de X contient l'ensemble des nœuds égaux ou descendants de Y. (ii) est le dual de (i), à savoir l'ensemble des nœuds ancêtres ou égaux à X est contenu dans l'ensemble des nœuds égaux ou ancêtres à Y. Enfin (iii) et (iv) signifient que l'ensemble des nœuds sur la gauche (respectivement droite) de X est contenu dans l'ensemble des nœuds sur la gauche (respectivement droite) de Y.

Précédence ($\prec$) La précédence entre nœud se traduit comme suit :

$$\begin{array}{lllllll} X \prec Y \equiv & N^i_{X.Right} & = & (N^j_{Y.EqDown} & \uplus & N^j_{Y.Right}) & \text{(i)} \\ \wedge & (N^i_{X.Left} & \uplus & N^i_{X.EqDown}) & = & N^j_{Y.Left} & \text{(ii)} \end{array} \tag{5.20}$$

Ainsi, si X précède Y, (i) l'ensemble des nœuds égaux ou descendants de Y et celui des nœuds à droite de Y forment une partition de l'ensemble des nœuds à droite de X (*cf*

[135]Par dominance large, nous entendons la clôture réflexive transitive de la relation de dominance.

figure 5.4). De manière symétrique, (ii) l'ensemble des nœuds à gauche de X et celui des nœuds égaux ou descendants de X forment une partition de l'ensemble des nœuds à gauche de Y.

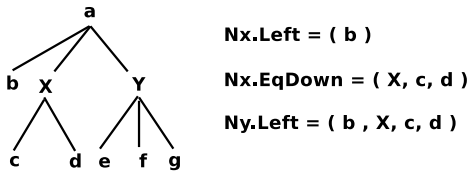


FIG. 5.4 – Exemple de précédence stricte en terme d'ensembles (i).

Précédence large ($\prec^*$) La précédence large[136] entre nœuds donne lieu à la définition des contraintes suivantes :

$$\begin{aligned} X \prec^* Y \equiv \quad & N^i_{X.EqDown} \subseteq N^j_{Y.Left} \quad \text{(i)} \quad \wedge \quad N^i_{X.Right} \supseteq N^j_{Y.EqDown} \quad \text{(ii)} \\ \wedge \quad & N^i_{X.Right} \supset N^j_{Y.Right} \quad \text{(iii)} \quad \wedge \quad N^i_{X.Left} \subset N^j_{Y.Left} \quad \text{(iv)} \end{aligned} \tag{5.21}$$

(i) traduit le fait que si X précède largement Y alors l'ensemble des nœuds égaux ou descendants de X est contenu dans l'ensemble des nœuds à gauche de Y (*cf* figure 5.5), le dual étant vrai (ii) : l'ensemble des nœuds à droite de X contient l'ensemble des nœuds égaux ou descendants de Y. Enfin (iii) signifie que l'ensemble des nœuds à droite de X contient l'ensemble des nœuds à droite de Y. De même, l'ensemble des nœuds à gauche de X est contenu dans l'ensemble des nœuds à gauche de Y (iv).

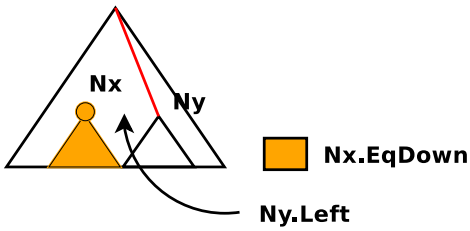


FIG. 5.5 – Contrainte sur ensembles liée à la précédence large (i).

Egalité (=) La relation d'égalité entre nœuds se traduit explicitement comme suit :

$$X = Y \equiv \quad N^i_X \quad = \quad N^j_Y \quad \text{(i)} \tag{5.22}$$

où l'égalité entre n-uplets (membre de droite de la définition) réfère à l'unification[137].

[136] Par précédence large, nous désignons la clôture réflexive transitive de la précédence.

[137] On notera que deux nœuds de la description sensé référé au même nœud dans un modèle doivent être tels que leurs matrices de traits soient unifiables.

Distinction ($\neq$) La distinction entre nœuds utilise la notion de disjonction d'ensembles (représentée par $\|$) :

$$\begin{array}{rcccc} X \neq Y \equiv & N^i_{X.Eq} & \| & N^j_{Y.Eq} & \text{(i)} \\ \wedge & N^i_{X.Children} & \| & N^j_{Y.Children} & \text{(ii)} \end{array} \tag{5.23}$$

(i) traduit que les ensembles de nœuds égaux à X et ceux égaux à Y sont disjoints. Nous utilisons dans un but d'optimisation la contrainte additionnelle que les ensembles de nœuds fils de X et fils de Y sont également disjoints (ii).

5.4.1.3 Recherche des solutions

Lorsque la description d'entrée ϕ a été convertie en contraintes, nous pouvons lancer une recherche des solutions. Dans notre cas, une solution est une assignation de valeurs à chacun des ensembles d'entiers contenus dans nos représentations de nœuds.

Cette recherche utilise le fait que, pour toute paire de nœuds (X^i, Y^j), l'une des relations mutuellement exclusives suivantes est valide :

$$X^i = Y^j \mid X^i \rightarrow^+ Y^j \mid Y^j \rightarrow^+ X^i \mid X^i \prec^+ Y^j \mid Y^j \prec^+ X^i \tag{5.24}$$

Nous exprimons cela au moyen d'une variable de choix C^{ij} et de clauses disjonctives traduisant la relation entre les nœuds X^i et Y^j :

$$X^i = Y^j \wedge C^{ij} = 1 \vee C^{ij} \neq 1 \wedge X^i \neq Y^j \tag{5.25}$$
$$X^i \rightarrow^+ Y^j \wedge C^{ij} = 2 \vee C^{ij} \neq 2 \wedge X^i \not\rightarrow^+ Y^j \tag{5.26}$$
$$Y^j \rightarrow^+ X^i \wedge C^{ij} = 3 \vee C^{ij} \neq 3 \wedge Y^j \not\rightarrow^+ X^i \tag{5.27}$$
$$X^i \prec^+ Y^j \wedge C^{ij} = 4 \vee C^{ij} \neq 4 \wedge X^i \not\prec^+ Y^j \tag{5.28}$$
$$Y^j \prec^+ X^i \wedge C^{ij} = 5 \vee C^{ij} \neq 5 \wedge Y^j \not\prec^+ X^i \tag{5.29}$$

De plus, la variable C^{ij} est contrainte dans l'intervalle [1..5].

Ce cadre étant posé, nous utilisons une stratégie de recherche pour explorer les assignations consistantes pour ces variables de choix (et pour les ensembles de nœuds) par rapport à la description ϕ.

Stratégie de recherche Le but de ce paragraphe est de donner une brève introduction à la résolution d'un problème au moyen d'un programme par contraintes, afin de pouvoir définir la stratégie de recherche utilisée dans notre recherche d'arbres[138].

La recherche de solutions à un problème de satisfaction de contraintes utilise deux règles d'inférence dans le but de trouver les assignations valides pour les variables du problème : la *propagation* de contraintes et la *distribution* de contraintes. Le but de la propagation de contraintes est de restreindre les valeurs admissibles des variables, celle de la distribution

[138] Pour les lecteurs intéressés par le domaine, un très bon document est (Schulte, 2002)

est de s'assurer de la complétude de la résolution. Etant donné une contrainte C et un problème P, distribuer la contrainte C revient à calculer les solutions de $P \cup C$ et $P \cup \neg C$. Lors de la résolution d'un problèmes, ces deux règles de propagation et de distribution sont appliquées successivement et de manière cyclique, jusqu'à obtention d'un état stable (*i.e.*, il n'y a plus de contraintes à appliquer) ou d'échec.

Dans notre contexte, la stratégie de recherche mentionnée précédemment est une stratégie de distribution, qui a pour but, lorsqu'une phase de distribution est nécessaire, de choisir parmi les variables indéterminées du problème, celle pour laquelle la distribution va être appliquée (*i.e.*, une contrainte ajoutée).

Pour notre problème de recherche d'arbres, la stratégie de recherche utilisée est appelée *first-fail*. Cette stratégie sélectionne la variable indéterminée la plus à gauche[139] et pour laquelle l'espace de recherche est le plus restreint. On notera que cette stratégie fournit de bons résultats (*cf* section 5.5), et qu'il est possible de définir ses propres stratégies de recherche.

En revenant à nos variables de choix C^{ij}, lors de la recherche de la relation entre deux nœuds dans notre arbre, cette distribution va permettre d'éliminer plusieurs possibilités infructueuses en associant une des alternatives à cette variable.

5.4.1.4 Optimisation

Dans un but d'optimisation, nous avons utilisé une seconde variable de choix pour la relation de parenté entre nœuds. Ainsi, partant du constat que deux nœuds (X^i, Y^j) sont tels que soit X^i est père de Y^j, soit Y^j est père de X^i, soit X^i et Y^j ne sont pas en relation de parenté :

$$X^i \rightarrow Y^j \mid Y^j \rightarrow X^i \mid X^i \parallel Y^j \tag{5.30}$$

nous définissons une variable de choix C_1^{ij}) contrainte dans l'intervalle [1..3], et telle que :

$$X^i \rightarrow Y^j \quad \wedge \quad C_1^{ij} = 1 \quad \vee \quad C_1^{ij} \neq 1 \quad \wedge \quad X^i \not\rightarrow Y^j \tag{5.31}$$

$$Y^j \rightarrow X^i \quad \wedge \quad C_1^{ij} = 2 \quad \vee \quad C_1^{ij} \neq 2 \quad \wedge \quad Y^j \not\rightarrow X^i \tag{5.32}$$

$$X^i \parallel Y^j \quad \wedge \quad C_1^{ij} = 3 \quad \vee \quad C_1^{ij} \neq 3 \quad \wedge \quad X^i \not\parallel Y^j \tag{5.33}$$

Là aussi, nous utilisons une stratégie de type *first-fail* pour déterminer les assignations solutions.

5.4.2 Résolution de contraintes spécifiques

Jusqu'à présent, nous avons vu comment nous résolvons des descriptions d'arbres. Maintenant, nous allons voir comment étendre ce procédé pour prendre en charge les contraintes spécifiques (dites de bonne formation grammaticale) introduites en section 4.3.

[139] Dans le vecteur des variables.

5.4.2.1 Les contraintes formelles

La première catégorie de contraintes concerne la validité des structures produites par rapport au formalisme TAG. Nous allons ajouter à la spécification du problème des contraintes permettant de restreindre les assignations de variables par rapport à des critères de marquage principalement.

Les contraintes que nous considérerons sont les suivantes[140] :
- marquage des nœuds (suivant qu'ils sont feuilles ou non),
- existence et unicité du nœud marqué ancre,
- unicité du nœud marqué pied.

Marquage des nœuds Nous souhaitons exprimer le fait que dans tout arbre valide au sens TAG, (a) tout nœud frontière a pour marque *ancre (anchor)*, *subst* ou *pied (foot)*, et (b) tout nœud interne a pour marque *standard (std)* ou *non-adjonction (nadj)*.

Les contraintes sur les nœuds feuilles (a) sont exprimées via :

$$N^i_X(mark : anchor) \equiv N^i_{X.Down} = \emptyset \tag{5.34}$$

$$N^i_X(mark : subst) \equiv N^i_{X.Down} = \emptyset \tag{5.35}$$

$$N^i_X(mark : foot) \equiv N^i_{X.Down} = \emptyset \tag{5.36}$$

et celles sur les nœuds internes (b) via :

$$N^i_X(mark : std) \equiv N^i_{X.Down} \neq \emptyset \tag{5.37}$$

$$N^i_X(mark : nadj) \equiv N^i_{X.Down} \neq \emptyset \tag{5.38}$$

Existence et unicité du nœud marqué ancre Nous voulons en outre exprimer le fait que tout arbre valide contient un et exactement un nœud marqué ancre. Pour réaliser cette contrainte, nous allons étendre la représentation des nœuds de nos modèles afin que celle-ci (*n.b.*, un n-uplet) intègre une variable booléenne $IsAnchor$ indiquant si le nœud en question est marqué ancre ou non. Ainsi, en notant $\mathcal{V}^\phi_{IsAnchor}$ l'ensemble des entiers référant à des nœuds étant marqués ancre dans la description ϕ, nous avons :

$$N^i_{X.IsAnchor} ::= (N^i_{X.Eq} \cap \mathcal{V}^\phi_{IsAnchor}) \neq \emptyset \tag{5.39}$$

Ensuite, si l'on associe la valeur de vérité $Vrai$ avec l'entier 1 et $Faux$ avec 0[141], nous pouvons calculer la somme des $N^i_{X.IsAnchor}$ pour chaque nœud de l'arbre et contraindre cette somme a valoir 1 :

$$\sum_{X \in \mathcal{V}^\phi} N^i_{X.IsAnchor} = 1 \tag{5.40}$$

[140] Nous ne parlons pas ici de la contrainte de présence du trait cat (catégorie syntaxique sur chaque nœud) et de la contrainte de similarité entre catégorie syntaxique du nœud pied et du nœud racine qui sont réalisées par élimination (*i.e., a posteriori*).

[141] On dit que l'on réifie la valeur booléenne.

Unicité du nœud marqué pied La contrainte pour l'unicité du nœud marqué pied est très proche de la contrainte vue précédemment pour le nœud ancre. A nouveau, nous allons utiliser une variable booléenne, nommée $IsFoot$, pour indiquer si le nœud est marqué ancre ou non. Cette variable sera définies comme suit ($\mathcal{V}^{\phi}_{IsFoot}$ réfère à l'ensemble des entiers dénotant des nœuds marqués pieds dans ϕ) :

$$N^i_{X.IsFoot} ::= (N^i_{X.Eq} \cap \mathcal{V}^{\phi}_{IsFoot}) \neq \emptyset \tag{5.41}$$

Après réification de cette variable booléenne, la contrainte d'unicité du nœud pied force la somme des $N^i_{X.IsRoot}$ à être strictement inférieure à deux :

$$\sum_{X \in \mathcal{V}^{\phi}} N^i_{X.IsFoot} < 2 \tag{5.42}$$

5.4.2.2 Les contraintes opérationnelles

Une seconde catégorie de contraintes que le formalisme XMG permet d'exprimer sont les contraintes opérationnelles, que nous avons appelées précédemment contraintes de coloration des nœuds. Pour mémoire, nous souhaitons contrôler semi-automatiquement la combinaison des fragments d'arbres. Pour cela, nous utilisons un langage de couleurs associé à une règle de saturation (*cf* section 4.3.2.2). L'idée est que chaque nœud de la description est coloré en rouge, noir ou blanc. Lors de la résolution des descriptions, des contraintes additionnelles permette de forcer certains nœuds à fusionner[142]. Les règles de fusion de nœuds sont récapitulées sur la figure 5.6.

$$\begin{array}{lcccl}
\circ_{\mathrm{B}} & + & \circ_{\mathrm{B}} & = & \circ_{\mathrm{B}} \\
\bullet_{\mathrm{N}} & + & \circ_{\mathrm{B}} & = & \bullet_{\mathrm{N}} \\
\bullet_{\mathrm{N}} & + & \bullet_{\mathrm{N}} & = & \bot \\
\bullet_{\mathrm{R}} & + & \{\ \circ_{\mathrm{B}}\ ;\ \bullet_{\mathrm{N}}\ ;\ \bullet_{\mathrm{R}}\} & = & \bot
\end{array}$$

FIG. 5.6 – Règles de fusion des nœuds colorés.

La règle de saturation impose que les modèles considérés comme valides soient ceux où l'arbre ne contient plus que des nœuds colorés en noir ou rouge (tous les nœuds blancs ont été consommés).

Pour appliquer ce type de contraintes dans notre résolveur de descriptions d'arbre, nous procédons comme suit : (a) nous étendons la représentation des nœuds dans nos modèles, et (b) nous convertissons les contraintes entre couleurs en contraintes sur les ensembles contenus dans ces représentations de nœuds.

[142]Par fusionner, nous entendons que les deux nœuds dans la description sont interprété comme un seul et unique nœud de l'arbre du modèle, et ce nœud de l'arbre se voit associé comme structure de traits l'unification des structures de traits des nœuds fusionnés.

L'extension (a) correspond à l'ajout au n-uplet représentant un nœud de deux champs : $TheRB$ et $ChildrenRB$. Le premier est un entier[143] référant au nœud de couleur noire ou rouge avec lequel le nœud de la description a potentiellement été fusionné. Si ce nœud n'était pas blanc, nous avons :

$$X \in (\mathcal{V}_R \cup \mathcal{V}_N) \Rightarrow N^i_{X.TheRB} = i \tag{5.43}$$

où $\mathcal{V}_R$ (respectivement $\mathcal{V}_N$) représente l'ensemble des nœuds colorés en rouge (respectivement noir). Pour les nœuds blancs, nous avons une contrainte moins stricte :

$$X \in \mathcal{V}_B \Rightarrow N^i_{X.TheRB} \in \mathcal{V}^\phi_N \tag{5.44}$$

où $\mathcal{V}_B$ représente l'ensemble des nœuds colorés en blanc, et $\mathcal{V}^\phi_N$ l'ensemble des nœuds colorés en noir dans notre description ($\mathcal{V}^\phi_N = \mathcal{V}^\phi \cap \mathcal{V}_N$). De plus, deux nœud de champs $TheRB$ différents ne peuvent dénoter le même nœud de la description, en conséquence nous étendons la définition de la distinction entre nœuds (5.23) :

$$\begin{array}{rcccc} N^i_X \neq N^j_Y \equiv & N^i_{X.Eq} & \| & N^j_{Y.Eq} & \text{(i)} \\ \wedge & N^i_{X.Children} & \| & N^j_{Y.Children} & \text{(ii)} \\ \wedge & N^i_{X.TheRB} & \neq & N^j_{Y.TheRB} & \text{(iii)} \end{array} \tag{5.45}$$

Le champ $ChildrenRB$ désigne l'ensemble des nœuds fils de couleur rouge ou noire. Clairement, $ChildrenRB$ est le sous-ensemble des nœuds rouge ou noir de l'ensemble $Children$:

$$N^i_{X.ChildrenRB} = N^i_{X.Children} \cap (\mathcal{V}^\phi_R \cup \mathcal{V}^\phi_N) \tag{5.46}$$

Le point (b) correspond à la définition de contraintes sur ces représentations étendues, contraintes qui traduisent les règles de combinaison des nœuds colorés. La première contrainte traduit le fait qu'un nœud rouge ne peut être fusionné avec aucun autre nœud :

$$X \in \mathcal{V}_R \Rightarrow N^i_{X.Eq} = \{i\} \tag{5.47}$$

De plus, l'union disjointe des ensembles Eq des nœuds noirs forme une partition de l'union des ensembles Eq des nœuds noirs et blancs[144], d'où :

$$\biguplus_{X \in \mathcal{V}_N} N^i_{X.Eq} = \bigcup_{Y \in (\mathcal{V}_B \cup \mathcal{V}_N)} N^j_{Y.Eq} \tag{5.48}$$

Cette dernière est une conséquence du fait que les nœuds blancs sont obligatoirement fusionnés avec un nœud noir.

[143] Et non un ensemble d'entiers à la différence des champs Eq, Up, etc.

[144] Pour rappel, l'ensemble Eq contient les entiers dénotant les nœuds qui ont été fusionnés avec le nœud courant.

Les contraintes suivantes illustrent l'utilisation du champ $TheRB$ pour calculer les valeurs des champs Eq, $Parent$, $Down$ et Up des nœuds blancs[145] :

$$\forall X \in \mathcal{V}_B \qquad N^i_{X.Eq} = \langle \bigoplus_{Y \in \mathcal{V}_N} N^j_{Y.Eq} \rangle \, [N^i_{X.TheRB}] \tag{5.49}$$

$$\forall X \in \mathcal{V}_B \qquad N^i_{X.Parent} = \langle \bigoplus_{Y \in \mathcal{V}_N} N^j_{Y.Parent} \rangle \, [N^i_{X.TheRB}] \tag{5.50}$$

$$\forall X \in \mathcal{V}_B \qquad N^i_{X.Down} = \langle \bigoplus_{Y \in \mathcal{V}_N} N^j_{Y.Down} \rangle \, [N^i_{X.TheRB}] \tag{5.51}$$

$$\forall X \in \mathcal{V}_B \qquad N^i_{X.Up} = \langle \bigoplus_{Y \in \mathcal{V}_N} N^j_{Y.Up} \rangle \, [N^i_{X.TheRB}] \tag{5.52}$$

(5.49) traduit le fait que l'ensemble des nœuds fusionnés avec un nœud blanc est déterminé en sélectionnant parmi les ensembles Eq des nœuds noirs ceux qui sont consistents avec la valeur du champ $TheRB$ du nœud blanc considéré. De manière similaire, (5.50) exprime la contrainte selon laquelle les parents d'un nœud blanc sont déterminés en sélectionnant parmi les ensembles $Parent$ des nœuds noirs ceux qui sont consistents avec la valeur $TheRB$ du nœud blanc considéré. Une contrainte de sélection comparable est utilisée pour déterminer les valeurs des champs $Down$ (5.51) et Up (5.52).

Enfin, nous pouvons définir les deux contraintes suivantes[146] :

$$\forall X \in \mathcal{V} \qquad N^i_{X.Children} = \cup\langle \bigoplus_{Y \in (\mathcal{V}_N \cup \mathcal{V}_R)} N^j_{Y.Eq} \rangle \, [N^i_{X.ChildrenRB}] \tag{5.53}$$

$$\forall X \in \mathcal{V} \qquad N^i_{X.Down} = \cup\langle \bigoplus_{Y \in (\mathcal{V}_N \cup \mathcal{V}_R)} N^j_{Y.EqDown} \rangle \, [N^i_{X.ChildrenRB}] \tag{5.54}$$

(5.53) traduit le fait que l'ensemble des nœuds fils d'un nœud X (de couleur quelconque) est déterminé en prenant l'union de tous les ensembles Eq des nœuds fils de X qui sont de couleur noire ou rouge. De manière similaire (5.54) exprime le fait que l'ensemble des nœuds descendants d'un nœud X quelconque est déterminé en prenant l'union des tous les ensembles $EqDown$ des nœuds fils de X qui sont colorés en noir ou en rouge.

Les contraintes (5.49) à (5.54) découlent toutes du fait qu'un nœud blanc est obligatoirement fusionné avec un nœud noir.

Interaction entre contrainte des couleurs et contraintes formelles Les contraintes de coloration introduites ici peuvent interagir avec les contraintes formelles introduites précédemment, en particulier les contraintes d'unicité.

[145] $\langle ... \rangle [...]$ désigne l'opérateur de contrainte de sélection. Plus précisément $X = \langle V_1 \dots V_n \rangle [I]$ exprime le fait que les variables I et X sont liées de telle façon que I soit un ensemble contenant les indices j des variables V_j dont la valeur n'est pas inconsistente avec celle de X. $\bigoplus$ représente l'opérateur de construction de liste.

[146] Ces contraintes utilisent l'opérateur d'union de sélections contrainte $\cup\langle ... \rangle [...]$. Par exemple $S = \cup\langle S_1 \dots S_n \rangle [SI]$ contraint les valeurs des variables S et SI de telle manière que S soit égal à l'union des ensembles S_i pour tous les indices i de SI.

Reprenons l'exemple de l'unicité du nœud marqué pied. A présent que les nœuds sont colorés, certains nœuds (les nœuds blancs) vont être fusionnés avec d'autres (les nœuds noirs). De ce fait, l'unicité de la marque pied doit-elle concerner l'ensemble des nœuds de la description ou bien uniquement ceux qui sont colorés en noir ou en rouge ? En effet, il est possible que dans une description, deux nœuds soient marqués pied, mais qu'ils dénotent un seul et même nœud de l'arbre du modèle (ces nœuds ont été fusionnés).

Ainsi il nous faut considérer, lors du calcul de la somme des nœuds marqué pied ($IsRoot ::= Vrai$), uniquement les nœuds de l'arbre du modèle appartenant à l'ensemble $\mathcal{V}_{RN}$ défini par :

$$\mathcal{V}^{\phi}_{RN} = \mathcal{V}^{\phi} \cap (\mathcal{V}_R \cup \mathcal{V}_N)$$

Enfin la contrainte forçant l'existence est l'unicité du nœud marqué ancre est dans ce cas :

$$\sum_{X \in \mathcal{V}^{\phi}_{RN}} N^i_{X.IsFoot} < 2 \tag{5.55}$$

5.4.2.3 Les contraintes dépendantes du langage cible

Une troisième catégorie de contraintes spécifiques sont celles dépendantes du langage cible, présentées en section 4.3.2.3. Dans cette catégories, les contraintes considérées sont (a) l'unicité et (b) l'ordonnancement des clitiques.

Le point (a) peut être vu comme une généralisation du cas de l'unicité de la marque pied. Nous allons donc généraliser le procédé vu plus haut. Plutôt que d'ajouter à notre représentation de nœud un champ $Clitique$, nous allons paramétriser la contrainte d'unicité. Ainsi, notre contrainte exprime le fait que dans un modèle valide, il n'y a qu'un seul et unique nœud ayant la propriété p (dans notre cas, la catégorie syntaxique clitique). Nous étendons notre représentation d'un nœud x en y incluant un champ has_p référant à une variable booléenne dont la valeur est $Vrai$ si le nœud a la propriété p et $Faux$ sinon. Ainsi, nous avons :

$$N^i_{X.has\_p} ::= (N^i_{X.Eq} \cap \mathcal{V}^{\phi}_p) \neq \emptyset \tag{5.56}$$

Cette définition traduit le fait qu'un nœud X a la propriété p si l'un des nœuds qui lui sont égaux dans le modèle dénote un nœud ayant la propriété p dans la description ϕ.

Ensuite, nous procédons comme auparavant, nous réifions cette variable booléenne et faisons la somme qui doit être inférieure à deux[147] :

$$\sum_{X \in \mathcal{V}^{\phi}} N^i_{X.has\_p} < 2 \tag{5.57}$$

[147] La remarque sur l'interaction entre couleurs et unicité reste valable ici.

Enfin, comme nous l'avons vu, les clitiques de même rang[148] doivent être uniques. Ainsi, nous appliquons la contrainte d'unicité avec le paramètre $p \in \{rang = 1, rang = 2, rang = 3, rang = 4, rang = 5, rang = 6, rang = 7\}$.

Pour contraindre l'ordonnancement des clitiques (point b), nous utilisons la variable de choix C^{ij} introduite précédemment. Ainsi nous allons contraindre la forme que les arbres prennent en contraignant les relations admissibles entre deux nœuds N_X^i et N_Y^j de rangs respectifs R_X et R_Y. Nous allons distinguer quatre cas de figure :

$$(R_X = 0 \mid\ R_Y = 0) \ \mid\ \ R_X = R_Y \ \mid\ \ R_X < R_Y \ \mid\ \ R_X > R_Y$$

$(R_X = 0 \mid\ R_Y = 0)$ Dans ce cas, l'un des nœuds N_X^i ou N_Y^j n'a pas de rang (il n'est donc pas de catégorie clitique). Aucune contrainte n'est imposée.

$R_X = R_Y$ Si les deux nœuds sont de même rang, on impose que leurs parents ne soient pas les mêmes :

$$N^i_{X.Parent} \neq N^j_{Y.Parent} \tag{5.58}$$

$R_X < R_Y$ Si N_X^i est de rang inférieur à N_Y^j alors nous avons :

$$N^i_{X.Parent} = N^j_{Y.Parent} \Rightarrow C^{ij} = 4 \tag{5.59}$$

ce qui signifie que si N_X^i et N_Y^j ont les mêmes parents, la variable C^{ij} est mise à quatre (ce qui impose que N_X^i précède strictement N_Y^j).

$R_X > R_Y$ De manière symétrique, si N_X^i est de rang supérieur à N_Y^j :

$$N^i_{X.Parent} = N^j_{Y.Parent} \Rightarrow C^{ij} = 5 \tag{5.60}$$

En d'autres termes, si N_X^i et N_Y^j ont les mêmes parents, C^{ij} vaut cinq, ce qui impose que N_Y^j précède strictement N_X^i.

5.5 Conclusion

L'architecture du compilateur de métagrammaire présenté ici se distingue par son caractère multi-paradigme. En particulier celle-ci utilise des techniques de la programmation logique (WAM, programmation par contraintes) qui permettent un traitement efficace de la métagrammaire (voir chiffres ci-dessous).

Les paradigmes de programmation qui ont été utilisés sont les suivants :

- **Programmation orientée objets** syntaxe concrète à base de classes, d'imports / exports, et code source de la machine virtuelle,

[148] Pou rappel, le rang est une propriété de nœud, comme il est possible d'en définir avec le formalisme XMG.

- **Programmation Logique** noyau de la machine virtuelle de type WAM pour la compilation d'un langage de contrôle similaire à une DCG,
- **Programmation par contraintes** module de résolution de descriptions utilisant les librairies pour la programmation par contraintes du langage Oz (Oz-Mozart, 2005).

Le choix de ces différents paradigmes a été guidé principalement par des soucis d'efficacité (cas de la compilation de la DCG sous-jacente à la métagrammaire et de la résolution de descriptions par contraintes) ou d'extensibilité (cas de la machine virtuelle orientée objets).

Sur la machine virtuelle de type WAM La machine virtuelle du compilateur XMG est inspirée par la WAM, et partage son schéma de résolution des requêtes (*valuations* dans notre cas).

En outre, notre machine virtuelle est développé sous forme d'une classe dont les méthodes correspondent au jeu d'instruction du code intermédiaire. La conséquence de cela est que l'extension de la machine virtuelle pour supporter de nouvelles instructions est facilitée.

Enfin, le fait de disposer de notre propre machine virtuelle nous permet d'effectuer des unifications entre structures de données non-standards (telles que les structures de traits polarisées utilisées en grammaires d'interaction (Perrier, 2003)).

Sur le module de résolution de descriptions Le problème de la résolution de contraintes de dominance est NP-complet (Koller et al., 1998). Compte tenu de cette complexité élevée, l'utilisation de techniques de programmation par contraintes (et plus particulièrement de programmation par contraintes concurrente (Schulte, 2000)) permet de mettre un oeuvre un algorithme efficace d'élimination des modèles non valides.

Quelques chiffres Enfin, voici quelques chiffres permettant d'avoir une meilleure idée de la disponibilité, l'utilisabilité et l'efficacité du compilateur XMG :

- le compilateur XMG est écrit dans le langage Oz dont l'environnement de développement est disponible gratuitement sous licence GPL pour les platesformes les plus courantes (Oz-Mozart, 2005). XMG lui-même est distribué gratuitement sous licence CeCILL[149] à l'adresse `http://sourcesup.cru.fr/xmg`.
- la métagrammaire pour le français de (Crabbé, 2005b), utilisant les contraintes spécifiques (couleurs, unicité, ordonnancement) et comptant près de 300 classes, est compilée en moins de 15 minutes sur une machine dont le processeur est cadencé à 2,6 Ghz, et comprenant 1 Go de mémoire vive. La grammaire ainsi produite contient plus de 6000 arbres.
- il existe des outils de conversion des grammaires produites par XMG pour pouvoir les interfacer avec des analyseurs TAG (Kow et al., 2006).

[149] Qui est une adaptation de la GPL au droit français, plus d'information sur `http://www.cecill.info`.

Chapitre 6

Cas pratique : description d'une grammaire à portée sémantique

Sommaire

Dans ce chapitre, nous allons voir comment, concrètement, définir une métagrammaire à portée sémantique. Nous introduirons notamment la syntaxe concrète du formalisme XMG. Notons que nous ne souhaitons pas détailler ici la définition d'une grammaire de taille réelle pour le français[150], nous donnerons plutôt une description générale de la méthodologie de développement d'une grammaire noyau à portée sémantique en nous basant notamment sur

[150]Pour cela, nous renvoyons le lecteur vers (Candito, 1999; Abeillé, 2002; Crabbé, 2005b).

(Crabbé, 2005a; Gardent, 2006), et en nous focalisant sur certains phénomènes illustrant la puissance du formalisme XMG.

La définition d'une métagrammaire se fait en trois étapes. (1) Tout d'abord, le concepteur déclare les contraintes à appliquer aux arbres, les types des données (catégorie syntaxique, etc) ainsi que leur domaine de définition. Ces informations permettent de typer les propriétés et traits associés aux nœuds. (2) Ensuite, il définit un ensemble de blocs de base (fragments d'arbres) qu'il combine via disjonction et conjonction. (3) Enfin, il évalue ces combinaisons de fragments. Nous allons à présent étudier plus en détail chacune de ces étapes.

6.1 Déclarations

Au préalable à la définition des fragments d'arbres composant le coeur de notre métagrammaire, il convient de déclarer certaines informations utiles au traitement automatique de la métagrammaire.

6.1.1 Déclaration des contraintes utilisées

La première des déclarations concerne ce que nous avons appelé précédemment les *contraintes spécifiques*. Comme nous avons vu en 4.3, XMG permet au développeur de la métagrammaire de choisir, parmi une librairie de contraintes, celles qu'il souhaite appliquer à la grammaire qu'il décrit, afin d'éviter la production de structures non valides. La directive du compilateur permettant d'appeler une contrainte est la suivante :

```
use X with (P1 , ... , Pn) dims (D1 ... Dn)
```

où `X` est le nom de la contrainte instanciée, `P1, ... , Pn` une liste de paramètres (éventuellement vide) et `D1 ... Dn` une liste de dimensions sur les descriptions desquelles la contrainte s'applique.

Les contraintes disponibles dans la version actuelle du formalisme XMG sont données en figure 6.1[151].

Notons que si le concepteur de la métagrammaire instancie une contrainte indisponible dans le formalisme, un message d'erreur est renvoyé.

[151] XMG dispose également d'une contrainte d'arité permettant de restreindre le nombre de fils d'un nœud donné. Cette contrainte n'est pas présentée ici car n'entrant pas dans notre classification.
De plus, les contraintes de bonne formation TAG sont activées par défaut, il n'existe donc actuellement aucune instruction pour celles-ci.
L'unicité est utilisée par exemple pour garantir que les arbres produits ne contiennent pas deux nœuds de catégorie clitique de rang identique.

Contrainte	Paramètres	Instruction en syntaxe concrète
Ordonnancement	/	`rank`
Unicité	propriété de nœud	`unicity`
Couleurs	/	`color`

FIG. 6.1 – Contraintes disponibles dans le système XMG.

6.1.2 Déclarations de types

Une fois les contraintes déclarées, l'utilisateur défini un ensemble de types énumérés pour les informations qu'il va manipuler (catégories syntaxiques, etc). La primitive de déclaration de type est :

```
type X = {val1 , ... , valN}
```

où `X` représente le nom du type et `val1` à `valN` la liste des valeurs que le type comporte (son domaine de définition).

Exemple 17 (La catégorie syntaxique)**.** *Nous pouvons définir l'ensemble des valeurs que le trait catégorie syntaxique d'un nœud peut prendre :*

```
type CAT = {p, n, v, gv, gp, p, d}
```

6.1.3 Déclarations de propriétés

Lorsque les types de données sont déclarés, nous pouvons les utiliser pour typer les propriétés de nœuds (*cf* section 4.1). La déclaration d'une propriété de nœud utilise la syntaxe suivante :

```
property X : T
```

où `X` est le nom de la propriété et `T` son type.

Exemple 18 (Rang d'un nœud)**.** *Le type de la propriété de nœud nommée rang se définit via :*

```
property rang : RANG
```

où `RANG` *a été défini comme l'ensemble* $\{0, 1, 2, 3, 4, 5, 6, 7\}$.

Rappelons qu'une propriété de nœud est différente d'un trait dans la mesure où elle sert à définir des contraintes sur les arbres produits mais ne fait pas partie du formalisme TAG.

6.1.4 Déclarations de traits

Enfin, de manière analogue à la définition de propriétés de nœud typées, il est possible de définir des traits typés :

```
feature X : T
```

où `X` est le nom du trait et `T` son type.

Exemple 19 (Personne)**.** *Le trait personne (*`pers`*) est définit comme suit :*

```
feature pers : PERS
```

où `PERS` *a été défini comme l'ensemble* {*sing,plur*}*.*

6.1.5 Remarque : Utilisation du typage

Le typage mis en place dans le système XMG, dans sa version actuelle, peut être qualifié de faible, dans la mesure il n'est employé que pour pouvoir construire la table des symboles (données à portée globale, *cf* section 5.3).

Dans une version future de XMG, il serait intéressant d'utiliser ces déclarations de typage pour réaliser un typage statique de la métagrammaire, ce qui permettrait de détecter un certain nombre d'erreurs avant même la compilation. Une autre utilisation intéressante de ces déclarations serait de mettre en place des règles de complétion de la métagrammaire à partir des types de traits utilisés, ce qui passerait par exemple par la mise en place d'une hiérarchie de type comme dans le formalisme des grammaires syntagmatiques guidées par les têtes (HPSG).

6.2 Définition des blocs

A présent, nous allons voir (a) comment définir les blocs élémentaires de notre (méta)grammaire, puis (b) comment les combiner entre eux pour produire les arbres attendus. Cette définition de blocs élémentaires et de « blocs de combinaison » constitue le coeur de la métagrammaire. Les blocs utilisés ici proviennent de (Crabbé, 2005a).

6.2.1 Blocs élémentaires

Si l'on considère une métagrammaire du verbe (ce qui constitue la majeure partie d'une métagrammaire) pour le français, un bloc élémentaire (fragment d'arbre) représente soit la réalisation d'un prédicat, soit celle d'un de ses arguments.

Prenons l'exemple de l'argument sujet nominal en position canonique (figure 6.2). Ce fragment d'arbre se définit comme suit :

$$\begin{aligned} CanonSuj \ \rightarrow \ syn+ = & X\,[cat : p] \wedge Y\,[cat : n] \wedge Z\,[cat = v] \\ & \wedge\ X \rightarrow\ Y \wedge X \rightarrow\ Z \wedge Y \prec\ Z \end{aligned}$$

ce qui donne, dans notre syntaxe concrète :

```
class CanonSuj
export ?X ?Y ?Z
declare ?X ?Y ?Z
{ <syn> {
          node ?X [cat=p] {
                            node ?Y [cat=n]
                            node ?Z [cat=v]
                          }
        }
}
```

où `node ?X { node ?Y}` représente la dominance ($X \rightarrow Y$), et `node ?X node ?Y` la précédence ($X \prec Y$). Notons que les dominance et précédence larges ($\rightarrow^*$ et $\prec^*$) sont représentées par `node ?X { ...node ?Y}` et `node ?X ,,,node ?Y` respectivement[152]. Enfin, la section `declare` introduit des variables (de nœuds ou de traits) et la section `export` permet de sélectionner parmi les variables déclarées celles qui sont visibles en dehors de la classe.

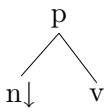


Fig. 6.2 – Fragment sujet nominal en position canonique.

Dans la grammaire réduite de (Crabbé, 2005a), l'auteur définit les dix blocs élémentaires présentés figure 6.3.

On distingue parmi ces fragments, deux réalisations pour la fonction grammaticale *sujet* (à savoir sujet sous forme canonique et sujet sous forme relativisée), deux réalisations pour l'*objet* (forme canonique et forme questionnée), deux réalisations pour l'*objet indirect* (formes canonique et questionnée), deux réalisations pour le *par-objet* (formes canoniques et questionnées) et enfin deux morphologies verbales (actif et passif).

En outre, ces blocs sont définis au sein d'une hiérarchie d'héritage afin d'avoir une meilleure factorisation et de faciliter la gestion de variables de nœud (*cf* section 4.1.2).

Si l'on excepte les deux blocs de la morphologie verbale, cette hiérarchie correspond à celle de la figure 6.4.

Cette figure nous montre comment l'information structurelle est partagée au sein des arguments verbaux. Par exemple, la structure commune aux blocs *objet indirect sous forme canonique* (classe $CanonIndObj$) et *par-objet sous forme canonique* (classe $CanonParObj$) est définie dans le bloc *groupe prépositionnel sous forme canonique* (classe $CanGP$) représenté figure 6.5.

[152]La syntaxe concrète de XMG est détaillée dans le manuel disponible en ligne à l'adresse `http://wiki.loria.fr/wiki/XMG/Documentation`.

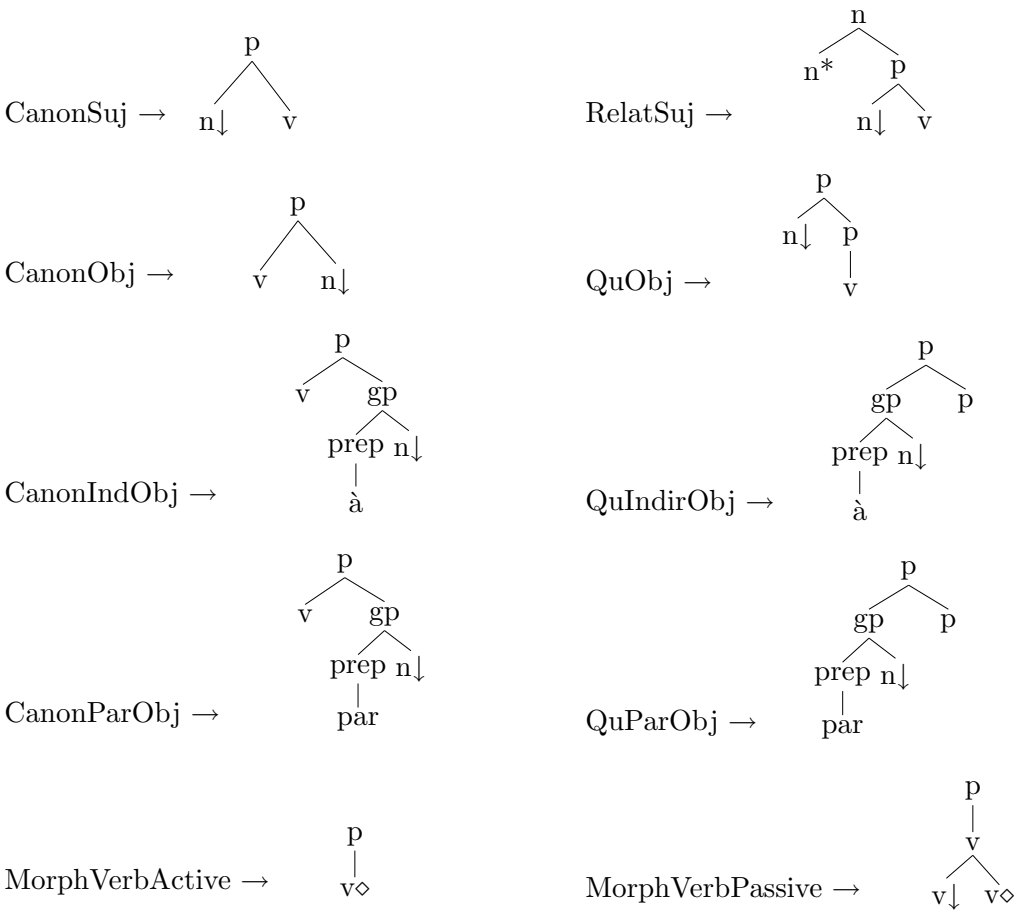


FIG. 6.3 – Blocs élémentaires pour une grammaire réduite du français.

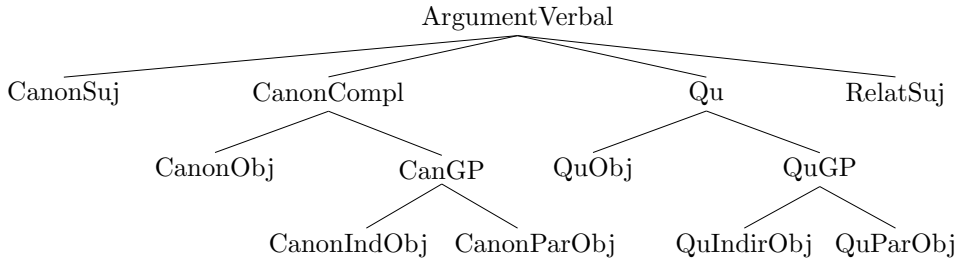


FIG. 6.4 – Hiérarchie d'héritage entre blocs élémentaires.

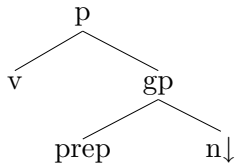


FIG. 6.5 – Contenu du fragment groupe prépositionnel sous forme canonique.

Ainsi, la définition de la classe *CanonIndObj* se fait par spécialisation du fragment *CanGP*. Dans notre syntaxe concrète, cela s'opère comme suit[153] :

```
class CanGP
import CanonComp[]
export ?C ?D ?E
declare ?C ?D ?E
{ <syn> {
          node ?C [cat=gp] {
                              node ?D [cat=prep]
                              node ?E (mark=subst)[cat=n]
                            }
        }
}

class CanonIndObj
import CanGP[]
declare ?X
{ <syn> {
         node ?D {
                    node ?X (mark=lex) [cat=à]
                  }
        }
}
```

On remarque que l'héritage permet à la classe *CanonIndObj* de réutiliser directement le nœud de la classe *CanGP* représenté par la variable `?D`, pour définir la position de la préposition *à* dans le fragment d'arbre de l'objet indirect sous forme canonique.

6.2.2 Combinaisons de blocs

D'après la définition originelle du terme métagrammaire donnée dans (Candito, 1999), une métagrammaire doit « reposer sur des principes linguistiques de hiérarchisation des structures syntaxiques ». Appliquée à notre approche, cette définition impose que les combinaisons de blocs élémentaires expriment des généralisations linguistiques. Suivant ce principe, (Crabbé, 2005a) exprime deux types de généralisations au moyen du langage de contrôle du formalisme XMG.

Le premier type de généralisation que (Crabbé, 2005a) exprime dans sa métagrammaire concerne les fonctions syntaxiques. En effet, nous avons vu précédemment des blocs élémentaires représentant différentes réalisations d'une même fonction syntaxique (sujet sous forme canonique et sujet sous forme relativisée par exemple). Ainsi, il définit des *abstractions* sur les différentes réalisations possibles pour une fonction syntaxique donnée. Dans

[153] Le fragment `CanGP` lui-même hérite de l'information du fragment `CanonComp` représentant le complément sous forme canonique, d'où la présence dans `CanGP` uniquement du groupe prépositionnel.

notre cas, nous avons les quatre abstractions suivantes :

Sujet	$Sujet \rightarrow CanonSuj \vee RelatSuj$	(a)
Objet	$Objet \rightarrow CanonObj \vee QuObj$	(b)
Objet indirect	$ObjetInd \rightarrow CanonIndObj \vee QuIndirObj$	(c)
Par-objet	$ParObjet \rightarrow CanonParObj \vee QuParObj$	(d)

Une telle abstraction s'écrit, dans la syntaxe concrète du formalisme XMG, comme suit (cas a) :

```
class Sujet { CanonSuj[] | RelatSuj[] }
```

Le second type de généralisation exprimée par (Crabbé, 2005a) concerne les alternances de diathèse. Les alternances de diathèse représentent les différentes identifications possibles entre arguments du verbe et fonctions syntaxiques. Nous considérons ici les alternances liées à la forme du verbe. Ainsi, nous souhaitons généraliser le fait que pour un verbe transitif à la forme active, le premier argument correspond à la fonction grammaticale sujet et le second à la fonction objet, alors que pour un verbe à la forme passive, le premier argument est réalisé par la fonction grammaticale par-objet et le second argument par la fonction sujet. Ce type de généralisation s'exprime, dans le formalisme XMG, par l'utilisation des opérateurs de conjonction et disjonction comme suit :

$$\begin{aligned} AlternativeTransitive \rightarrow & (\, Sujet \wedge MorphVerbActive \wedge Objet \,) \\ & \vee (\, Sujet \wedge MorphVerbPassive \wedge ParObjet \,) \end{aligned}$$

Ce qui, dans notre syntaxe concrète donne :

```
class AlternativeTransitive
{  { Sujet[] ; MorphVerbActive[] ; Objet[] }
 | { Sujet[] ; MorphVerbPassive[] ; ParObjet[] } }
```

Remarque 1 Ici, il convient de préciser que chacune des conjonctions de classes, `Sujet`, `MorphVerbActive` et `Objet` d'une part, et `Sujet`, `MorphVerbPassive` et `ParObjet` d'autre part, ne suffisent à décrire les arbres des verbes transitifs à la forme active et passive respectivement, que dans le cas où les fragments d'arbres utilisés sont colorés. En effet, si ce n'est pas le cas, il faut ajouter aux conjonctions énoncés dans notre exemple des opérations d'identification de nœuds, afin de s'assurer par exemple que les nœuds racines du sujet et de l'objet vont être fusionnés (*cf* section 4.1.2.5).

Remarque 2 L'utilisation du langage de contrôle fournit par le formalisme XMG permet de traiter de manière élégante le cas du passif sans agent, puisqu'à la différence de (Candito, 1999), il n'est pas utile d'effacer de l'information (*i.e.*, de supprimer une partie de la

description). Dans notre cas, le traitement du passif sans agent se fait par l'ajout d'une alternative à la disjonction représentant les alternances d'un verbe transitif :

$$\begin{aligned} AlternativeTransitive \rightarrow & (Sujet \wedge MorphVerbActive \wedge Objet) \\ & \vee (Sujet \wedge MorphVerbPassive \wedge ParObjet) \\ & \vee (Sujet \wedge MorphVerbPassive) \end{aligned}$$

6.2.3 Définition de familles

Il existe un autre de type de généralisation exprimable par le formalisme XMG, il s'agit de la notion de famille d'arbres utilisée par (XTAG-Research-Group, 2001). Cette notion permet de regrouper les arbres partageant certaines caractéristiques, en l'occurrence, ayant la même valence (nombre d'arguments), et des arguments de même catégorie syntaxique (*i.e.*, les arbres de même *cadre de sous-catégorisation*). L'un des intérêts de ce regroupement réside, comme nous allons le voir en section 6.4, de pouvoir « décomposer » notre grammaire TAG lexicalisée en *schèmes*[154] associés à un *lexique syntaxique*, ce qui permettra à l'analyseur syntaxique de ne manipuler qu'une sous-grammaire.

La définition d'une famille dans le formalisme XMG se fait au moyen du langage de contrôle basé sur les opérateurs de conjonction et disjonction. Un premier exemple de famille concerne les verbes ayant un seul argument (verbes intransitifs) de sujet nominal, $n0V$[155] :

$$n0V \rightarrow (Sujet \wedge MorphVerbActive) \tag{6.1}$$

De la même manière, la famille des arbres transitifs à arguments nominaux n'est autre que la classe *AlternanceTransitive* introduite précédemment :

$$\begin{aligned} n0Vn1 \rightarrow & (Sujet \wedge MorphVerbActive \wedge Objet) \\ & \vee (Sujet \wedge MorphVerbPassive \wedge ParObjet) \\ & \vee (Sujet \wedge MorphVerbPassive) \end{aligned} \tag{6.2}$$

Enfin, la famille des verbes ditransitifs à arguments nominaux se définit alors en réutilisant les classes impliquées dans la définition des verbes transitifs[156] :

$$\begin{aligned} n0Vn1pn2 \rightarrow & (Sujet \wedge MorphVerbActive \wedge Objet \wedge ObjetInd) \\ & \vee (Sujet \wedge MorphVerbPassive \wedge ParObjet \wedge ObjetInd) \\ & \vee (Sujet \wedge MorphVerbPassive \wedge ObjetInd) \end{aligned} \tag{6.3}$$

[154] Un schème est un arbre dans lequel un nœud feuille est marqué pour recevoir un item lexical, on parle de nœud ancre.

[155] NB : ces verbes ne se passivent pas.

[156] Notons que nous pourrions réutiliser la famille $n0Vn1$ dans la définition de la famille $n0Vn1pn2$ comme suit : $n0Vn1pn2 \rightarrow n0Vn1 \wedge ObjetInd$. Cependant, en vue de l'intégration d'information sémantiques, nous définissons n0Vn1pn2 en réutilisant les alternances pour les verbes transitifs.

6.2.4 Ajout de l'information sémantique

A présent, nous allons voir comment étendre la métagrammaire réduite du français introduite ci-dessus afin de produire une grammaire TAG à portée sémantique, et qui soit utilisable dans un contexte de construction sémantique. L'interface syntaxe / sémantique que nous utilisons est celle présentée en section 2.2.4.2. Ainsi, nous souhaitons étendre notre description métagrammaticale pour produire des arbres syntaxiques associés chacun à une représentation sémantique avec laquelle ils partagent des variables d'unification (les indices sémantiques). L'intégration d'une dimension sémantique présentée ici s'inspire de (Gardent, 2006).

6.2.4.1 Représentations sémantiques

Pour rappel, les représentations sémantiques que nous souhaitons associer à nos arbres TAG sont des formules prédicatives pour une logique plate. Pour la grammaire réduite introduite précédemment, les représentations utilisées sont les suivantes :

a. $\ell : p(e)\ ;\ \ell : theta_1(e,x)$ (verbes intransitifs)
b. $\ell : p(e)\ ;\ \ell : theta_1(e,x)\ ;\ \ell_k : theta_2(e,y)$ (verbes transitifs)
c. $\ell : p(e)\ ;\ \ell : theta_1(e,x)\ ;\ \ell : theta_2(e,y)\ ;\ \ell : theta_3(e,z)$ (verbes ditransitifs)

où $p, theta_1, theta_2, theta_3$ représentent des variables de prédicat dont la valeur sera donnée par le lexique (voir 6.4 ci-dessous). e représente la variable événementielle et x, y, z les arguments des prédicats. Enfin, les ℓ représentent les étiquettes du formalisme sémantique (utilisées pour le traitement des opérateurs de portée tels que les quantificateurs).

Prenons l'exemple d'un verbe intransitif tel que *dormir*. La représentation sémantique qui lui sera associée sera la suivante :

$$\ell : dormir(e)\ ;\ \ell : agent(e,x)$$

6.2.4.2 Association arbres - représentations sémantiques

Afin de bénéficier d'une meilleure factorisation, l'association entre arbres et représentations sémantiques va se faire comme suit :

1. Nous définissons des classes ayant uniquement une dimension sémantique, contenant les représentations sémantiques introduites ci-dessus. En outre, ces classes sont placées dans une hiérarchie d'héritage afin de pouvoir réutiliser l'information.
2. Nous utilisons le regroupement des arbres par famille pour pouvoir associer une représentation sémantique donnée à l'ensemble des arbres correspondant. En effet, les familles regroupent les arbres ancrés par le même prédicat, et dans chacun de ces arbres il existe un nœud pour chaque argument du prédicat (via le principe de Co-ocurrence Prédicat-Arguments auquel nous nous sommes conformés en écrivant la métagrammaire).

Le premier point est réalisé par la définition des classes sémantiques $RelUnaire$, $RelBinaire$ et $RelTernaire$ suivantes[157] :

$$RelUnaire \rightarrow (L : P(E) \wedge L : Theta_1(E, X)) \quad (6.4)$$

$$RelBinaire \angle RelUnaire \rightarrow (L : Theta_2(E, Y)) \quad (6.5)$$

$$RelTernaire \angle RelBinaire \rightarrow (L : Theta_3(E, Z)) \quad (6.6)$$

Dans la syntaxe concrète du formalisme XMG, la définition de telles classes sémantiques se fait comme dans l'exemple de la relation binaire ci-dessous :

```
class RelBinaire
import RelUnaire[]
export ?L ?Theta2 ?Y
declare ?L ?Theta2 ?Y
{ <sem> {
          ?L : ?Theta2(?E,?Y)
        }
}
```

Le second point, l'association information sémantique – famille d'arbres, se fait en modifiant la définition des familles comme suit. (6.1), (6.2) et (6.3) deviennent respectivement :

$$n0V \rightarrow (Sujet \wedge MorphVerbActive \wedge RelUnaire) \quad (6.7)$$

$$\begin{aligned} n0Vn1 \rightarrow (&(Sujet \wedge MorphVerbActive \wedge Objet) \\ &\vee (Sujet \wedge MorphVerbPassive \wedge ParObjet) \\ &\vee (Sujet \wedge MorphVerbPassive)) \wedge RelBinaire \end{aligned} \quad (6.8)$$

$$\begin{aligned} n0Vn1pn2 \rightarrow (&(Sujet \wedge MorphVerbActive \wedge Objet \wedge ObjetInd) \\ &\vee (Sujet \wedge MorphVerbPassive \wedge ParObjet \wedge ObjetInd) \\ &\vee (Sujet \wedge MorphVerbPassive \wedge ObjetInd)) \wedge RelTernaire \end{aligned} \quad (6.9)$$

Ainsi, nous définissons la famille des arbres intransitifs, classe $n0V$, comme l'ensemble des arbres ayant un sujet et une morphologie verbale active associés à une relation sémantique unaire. De même, la famille $n0Vn1$ regroupe les arbres pour les verbes transitifs associés à une relation sémantique binaire et enfin la famille $n0Vn1pn2$ regroupe les arbres pour les verbes ayant un sujet, un objet et un objet indirect associés à une relation sémantique ternaire. Cependant, ces définitions ne suffisent pas à produire les associations (arbre syntaxique – représentation sémantique) attendues puisque nous n'avons pas encore traité la gestion des variables partagées entre ces deux structures.

[157] Pour simplifier la représentation, nous ne représentons pas les variables exportées, toutes les variables commençant par une majuscule.

6.2.4.3 Gestion des variables partagées

En section 2.2.4.2, nous avons présenté une interface syntaxe / sémantique dans laquelle les arbres TAG partageaient certaines variables de trait avec des variables issues de prédicats sémantiques, tel qu'illustré en figure 6.6[158].

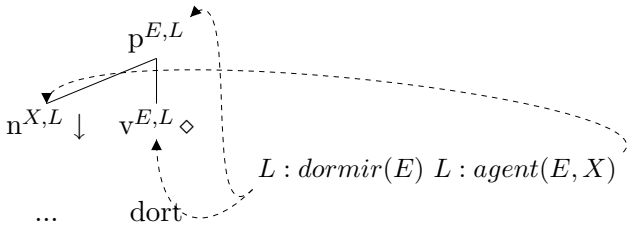


FIG. 6.6 – Partage de variables entre structures syntaxique et sémantique.

Sur cette figure, l'argument sémantique du prédicat *dormir*, variable X, est lié à une variable de trait du nœud réalisant la fonction grammaticale sujet, et les variables événementielle et étiquette du prédicat, variables E et L respectivement, sont liées à des variables de trait des nœuds de la projection du nœud ancre.

Ce partage de variables d'unification entre dimensions syntaxique et sémantique va permettre, lors de l'analyse syntaxique, de construire une représentation sémantique dont les arguments des prédicats seront liés aux constituants syntaxiques correspondants.

Pour mettre en place ce partage de variables, nous allons distinguer deux types de variables d'unification : les variables de la dimension sémantique, que nous appellerons **indices sémantiques**, et les variables de la dimension syntaxique (variables de trait), appelées **indices d'arbres**[159].

Définition des indices sémantiques Les indices sémantiques sont des variables locales à la classe sémantique où elles sont introduites (et aux sous-classes de celle-ci). Pour pouvoir référer à ces variables en dehors des classes sémantiques, nous allons utiliser l'interface de classe pour leur donner un nom global (*cf* section 4.1.2.7). Nous appellerons le premier argument sémantique arg_1, le second arg_2 et ainsi de suite. Nous aurons également une variable d'étiquette *label* et une variable événementielle *evt*.

Ainsi, à partir des classes sémantiques définies en (6.4), (6.5) et (6.6), nous allons définir

[158] Dans un souci de simplification, les noms des traits de nœud ne sont pas indiqués.
[159] Nous nous conformons ainsi à la terminologie de (Gardent, 2006)

les interfaces suivantes :

$$RelUnaire \rightarrow (\, L : P(E) \wedge \; L : Theta_1(E,X)\,)* = [\, label : L, \; evt : E, \; arg_1 : X\,] \tag{6.10}$$

$$RelBinaire \angle RelUnaire \rightarrow (\, L : Theta_2(E,Y)\,)* = [\, arg_2 : Y\,] \tag{6.11}$$

$$RelTernaire \angle RelBinaire \rightarrow (\, L : Theta_3(E,Z)\,)* = [\, arg_3 : Z\,] \tag{6.12}$$

Dans notre syntaxe concrète, cette définition d'interface se fait comme dans l'exemple de la classe `RelBinaire` ci-dessous :

```
class RelBinaire
import RelUnaire[]
export ?L ?Theta2 ?Y
declare ?L ?Theta2 ?Y
{ <sem> {
          ?L : ?Theta2(?E,?Y)
        }
}*=[arg2=?Y]
```

On note que, via héritage, les interfaces d'une sous-classe et de sa classe mère sont unifiées, ainsi l'interface de la classe `RelBinaire` contient les noms globaux *evt*, *label*, arg_1 et arg_2. De même, l'interface de la classe `RelTernaire` contient les noms globaux *evt*, *label*, arg_1, arg_2 et arg_3.

Remarque : Nous utilisons les interfaces pour rendre accessibles les indices sémantiques à partager avec certaines variables de trait. A partir du moment où ces indices sémantiques sont exportés par les classes sémantiques (ce qui est le cas ici), nous pourrions très bien utiliser l'opérateur *dot* pour référer à ces variables en dehors des classes sémantiques (et ainsi unifier explicitement les variables partagées). Nous préférons l'interface car elle permet une réduction non-négligeable de la description.

Définition des indices d'arbres Les indices d'arbres que nous souhaitons définir sont liés soit (a) à un argument sémantique, soit (b) à un prédicat (indice lié à la variable événementielle).

Les nœuds recevant des indices de type (a) sont appelés **nœuds besoin** et ceux recevant des indices de type (b) **nœuds ressource**. En outre, tous ces nœuds reçoivent un trait contenant une variable d'étiquette de prédicat (utilisé pour la gestion des quantificateurs).

Les arbres étant définis suivant le principe de coocurrence prédicat-argument, nous savons que les nœuds besoin correspondent à des nœuds feuilles marqués pour substitution (relation prédicat-argument) ou marqués pied (relation modificateur / modifié). Nous associons donc, dans la métagrammaire, à chacun de ces nœuds besoin deux variables associées aux traits *ind* et *label* représentant respectivement l'indice sémantique (variable liée à l'argument sémantique) et l'étiquette du prédicat. Un exemple de la définition d'un tel nœud dans la syntaxe concrète de XMG est :

```
... node ?X [cat=n, top=[ind=?I,label=?L]] ...
```

Les nœuds ressource sont situés sur la projection du nœud ancre, *cf* (Frank et Van Genabith, 2001) (sur la figure 6.6, les nœuds ressource sont la racine et l'ancre). En effet, cette projection contient les nœuds sujets à adjonction d'un modificateur du verbe. Sur chacun de ces nœuds, nous allons donc placer des variables associées aux traits *ind* et *label* représentant respectivement l'indice sémantique (dans ce cas, variable liée à l'événement du prédicat) et l'étiquette du prédicat. Pour ces nœuds, nous pouvons éviter une annotation manuelle des indices d'arbres en utilisant une classe $Projection\_Sem$ contenant la coindexation des traits sémantiques (indices d'arbres) entre deux nœuds voisins de l'épine dorsale du verbe. Cette classe $Projection\_Sem$ est définie comme suit :

$$Projection\_Sem \rightarrow \begin{array}{c} \mathrm{X}^{[ind=I,label=L]} \\ | \\ \mathrm{Y}^{[ind=I,label=L]} \end{array}$$

On remarque que les nœuds de ce fragment d'arbre ne contiennent pas de catégorie syntaxique. En effet, cette classe va être spécialisée par héritage pour ainsi définir les différentes projections possibles (*e.g.* entre nœuds de catégories respectives v et p). Ce seront ces classes spécialisées qui seront combinées avec celles définissant la morphologie verbale pour avoir des arbres dont les nœuds sont annotés par des indices d'arbres.

Enfin, nous allons, comme pour les indices sémantiques, donner des noms globaux aux indices d'arbres afin de faciliter leur accès depuis des classes différentes de celles où ils sont déclarés. Cela permettra de lier les indices sémantiques aux indices d'arbres (voir ci-dessous).

Pour réaliser ce nommage global, nous utilisons à nouveau les interfaces. Plus précisément, **(a)** nous associons à la classe définissant un nœud besoin (classe $CanonSuj$ par exemple) des noms globaux significatifs pour les indices d'arbres. Ainsi la classe $CanonSuj$ est étendue par l'ajout de l'interface suivante :

$$\begin{aligned} CanonSuj \rightarrow\ & (\, syn+ = X\,[cat : p] \wedge \\ & \quad Y\,[cat : n, top : [ind : I, label : L]] \wedge \\ & \quad Z\,[cat = v] \wedge X \rightarrow Y \wedge X \rightarrow Z \wedge Y \prec Z\,) \\ & \quad * = [\, sujIdx : I\, label : L] \end{aligned}$$

(b), les interfaces des classes où sont définis les nœuds ressource, classe $Projection\_Sem$ par exemple, sont étendues de la même manière :

$$\begin{aligned} Projection\_Sem \rightarrow\ & (\, syn+ = X\,[top : [ind : I_1, label : L_1]] \wedge \\ & \quad Y\,[top : [ind : I_1, label : L_1]] \wedge \\ & \quad X \rightarrow Y\,) * = [\, projIdx : I_1\, label : L_1] \end{aligned}$$

Ainsi, si nous appelons $Projection\_Sem\_VP$ la classe contenant les indices d'arbres pour la projection $v - -p$, nous pouvons étendre la définition de la morphologie verbale

active pour qu'elle intègre les indices d'arbres fournis par la classe $Projection\_Sem\_VP$:

$$MorphVerbActiveSem \rightarrow (\, MorphVerbActive \wedge \; Projection\_Sem\_VP \,) \tag{6.13}$$

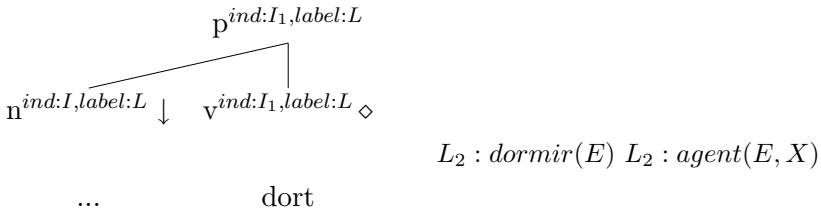
Il convient de remarquer ici à nouveau que la conjonction des classes $MorphVerbActive$ et $Projection\_Sem\_VP$ ne suffit pas à décrire les fragments attendus. Il faut encore s'assurer que les nœuds de $Projection\_Sem\_VP$ soient correctement identifiés avec ceux de $MorphVerbActive$. Pour cela, nous pouvons utiliser l'opérateur *dot* et des équations de nœuds, ou utiliser le système de couleurs pour éviter la définition de ces équations, c'est cette dernière approche que choisit (Gardent, 2006).

Enfin, nous redéfinissons la famille $n0V$ en étendant la définition (6.7) comme suit[160] :

$$n0V \rightarrow (\, Sujet \wedge \; MorphVerbActiveSem \wedge \; RelUnaire \,) \tag{6.14}$$

La classe $Sujet$ instanciant la classe $CanonSuj$, son interface contient les traits $sujIdx$ et $label$. De même la classe $MorphVerbActiveSem$ instanciant $Projection\_Sem\_VP$ contient les traits $projIdx$ et $label$. De plus, la classe $n0V$ instanciant les classes $Sujet$ et $MorphVerbActiveSem$, leurs interfaces sont unifiées. Ainsi, nous obtenons, pour la classe $n0V$, l'interface suivante[161] : $[\, sujIdx : I, label : L\, projIdx : L]$. On note que les traits $label$ des interfaces des classes $Sujet$ et $Projection\_Sem\_VP$ ont été unifiés[162] (ici les variables L et L_1).

Si l'on revient à notre exemple du verbe *dormir*, nous avons, avec la définition ci-dessus de la famille $n0V$, l'arbre pour le sujet sous forme canonique suivant :



C'est-à-dire que nous avons les indices sémantiques et les indices d'arbres définis, de plus la projection verbale est correctement coindexée (partage des indices d'arbres entre nœuds de catégorie p et v). De plus, les variables d'étiquette sont partagée entre les nœuds ressource et les nœuds besoin. Il nous manque cependant encore le partage entre les indices sémantiques et les indices d'arbres (lien entre I et X, entre I_1 et E et entre L et L_2).

Partage par interfaces de classes Pour réaliser les coindexations entre indices d'arbres et indices sémantiques, nous allons utiliser les interfaces de classe. Lors de la définition des familles, nous avons accès à tous les noms globaux, puisque les classes sémantiques et

[160] Les définitions des familles $n0Vn1$ et $n0Vn1pn2$ peuvent être étendues de manière similaire moyennant la définition de la classe $MorphVerbPassiveSem$.

[161] Nous ne considérons ici que les noms globaux ayant trait aux indices sémantiques et aux indices d'arbres.

[162] Et donc les indices *label* des nœuds ressource et des nœuds besoin également.

syntaxiques y sont instanciées. Nous allons donc associer, dans les interfaces des classes combinées lors de la définition des familles, les traits devant être unifiés avec la même variable. Les définitions des familles $n0V$, $n0Vn1$ et $n0Vn1pn2$ de notre métagrammaire réduite deviennent donc :

$$\begin{aligned} n0V \rightarrow (\, & Sujet* = [sujIdx : X, label : L] \wedge \\ & MorphVerbActiveSem* = [projIdx : E, label : L] \wedge \\ & RelUnaire* = [label : L, evt : E, arg1 : X]\,) \end{aligned} \tag{6.15}$$

$$\begin{aligned} n0Vn1 \rightarrow (\, & (\, Sujet* = [sujIdx : X, label : L] \wedge \\ & MorphVerbActiveSem* = [projIdx : E, label : L] \wedge \\ & Objet* = [objIdx : Y, label : L]\,) \\ & \vee\ (\, Sujet* = [sujIdx : Y, label : L] \wedge \\ & MorphVerbPassiveSem* = [projIdx : E, label : L] \wedge \\ & ParObjet* = [pobjIdx : X, label : L]\,) \\ & \vee\ (\, Sujet* = [sujIdx : Y, label : L] \wedge \\ & MorphVerbPassiveSem* = [projIdx : E, label : L]\,)\,) \\ \wedge\ & RelBinaire* = [label : L, evt : E, arg1 : X, arg2 : Y] \end{aligned} \tag{6.16}$$

$$\begin{aligned} n0Vn1pn2 \rightarrow (\, & (\, Sujet* = [sujIdx : X, label : L] \wedge \\ & MorphVerbActiveSem* = [projIdx : E, label : L] \wedge \\ & Objet* = [objIdx : Y, label : L] \wedge \\ & ObjetInd* = [objIndIdx : Z, label : L]\,) \\ & \vee\ (\, Sujet* = [sujIdx : Y, label : L] \wedge \\ & MorphVerbPassiveSem* = [projIdx : E, label : L] \wedge \\ & ParObjet* = [pobjIdx : X, label : L] \wedge \\ & ObjetInd* = [objIndIdx : Z, label : L]\,) \\ & \vee\ (\, Sujet* = [sujIdx : Y, label : L] \wedge \\ & MorphVerbPassiveSem* = [projIdx : E, label : L] \wedge \\ & ObjetInd* = [objIndIdx : Z, label : L]\,)\,) \\ \wedge\ & RelTernaire* = [label : L, evt : E, arg1 : X, arg2 : Y, arg3 : Z] \end{aligned} \tag{6.17}$$

Dans (6.15) nous contraignons les variables $sujIdx$ représentant le constituant syntaxique sujet et $arg1$ représentant l'argument du prédicat à partager la même valeur. De même, en coindexant $projIdx$ et evt, nous nous assurons que la projection du nœud ancre va bien recevoir l'indice du fonction sémantique.

Il est intéressant de noter comment le cas du passif est traité dans (6.16). Pour l'alternance active, les traits d'interface $sujIdx$ et $arg1$ d'une part, et $objIdx$ et $arg2$ d'autre part sont partagés, alors que pour l'alternance passive, le partage concerne $objIdx$ et $arg1$ d'une part et $sujIdx$ et $arg2$ d'autre part.

Remarque Nous aurions très bien pu utiliser des paramètres de classe pour partager les variables entre les dimensions sémantique et syntaxique. Cependant, cela aurait nécessité de transmettre la valeur des paramètres via chacune des classes instanciées (par héritage ou conjonction).

6.3 Evaluation

Après avoir défini nos blocs élémentaires, les avoir combiner, les avoir regroupés en famille et y avoir ajouté l'information sémantique, nous pouvons demander au compilateur de calculer la grammaire ainsi décrite.

Comme nous l'avons vu au chapitre 5, la métagrammaire correspond à une DCG, dans laquelle les symboles terminaux sont des descriptions d'arbres. Dans ce contexte, la grammaire décrite par notre métagrammaire correspond au langage couvert par cette DCG. Pour calculer la grammaire, il faut donc dériver les phrases du langage couvert par la DCG. Pour cela, nous partons habituellement d'un axiome. Dans notre cas, il y en a plusieurs, ce sont les familles décrites précédemment. En effet celles-ci contiennent des descriptions totales des arbres. Pour que le compilateur sache qu'il s'agit des axiomes, il convient de déclarer ces familles comme des classes évaluables, cela se fait au moyen de la syntaxe suivante :

```
value n0V
```

Exemple 20 (Dérivation de la classe $n0Vn1$)**.** *Si nous prenons l'exemple de la famille* $n0Vn1$*, dont nous rappelons la définition ci-dessous*[163] *:*

$$\begin{aligned} n0Vn1 \rightarrow (\ &(\ Sujet \wedge\ MorphVerbActiveSem \wedge\ Objet\ ) \\ &\vee\ (\ Sujet \wedge\ MorphVerbPassiveSem \wedge\ ParObjet\ ) \\ &\vee\ (\ Sujet \wedge\ MorphVerbPassiveSem\ )\ ) \wedge\ RelBinaire \end{aligned}$$

le compilateur va développer l'expression logique, pour aboutir à une énumération de conjonctions de descriptions, dans notre cas :

$$\begin{aligned} n0Vn1 \rightarrow (\ &(\ Sujet \wedge\ MorphVerbActiveSem \wedge\ Objet \wedge\ RelBinaire\ ) \\ &\vee\ (\ Sujet \wedge\ MorphVerbPassiveSem \wedge\ ParObjet \wedge\ RelBinaire\ ) \\ &\vee\ (\ Sujet \wedge\ MorphVerbPassiveSem \wedge\ RelBinaire\ )\ ) \end{aligned}$$

[163] Nous omettons volontairement les matrices interfaces dans un souci de clarté.

Prenons la première alternative de cette définition (forme active) :

$$(\ Sujet \wedge\ MorphVerbActiveSem \wedge\ Objet \wedge\ RelBinaire\ )$$

Sachant que les fonctions grammaticales sujet et objet sont définies via :

$$Sujet \rightarrow (\ CanonSuj \vee\ RelatSuj\ )$$
$$Objet \rightarrow (\ CanonObj \vee\ QuObj\ )$$

Les accumulations de descriptions calculées par le compilateur pour la forme active de la famille $n0Vn1$ correspondent aux quatre conjonctions de descriptions suivantes :

$$\begin{array}{lccccccccl}
(\ & CanonSuj & \wedge & MorphVerbActiveSem & \wedge & CanonObj & \wedge & RelBinaire &) & a \\
(\ & RelatSuj & \wedge & MorphVerbActiveSem & \wedge & CanonObj & \wedge & RelBinaire &) & b \\
(\ & CanonSuj & \wedge & MorphVerbActiveSem & \wedge & QuObj & \wedge & RelBinaire &) & c \\
(\ & RelatSuj & \wedge & MorphVerbActiveSem & \wedge & QuObj & \wedge & RelBinaire &) & d
\end{array}$$

Une fois les descriptions accumulées, celles-ci sont résolues afin de construire les modèles d'arbres minimaux constituant la grammaire. Par exemple, pour la description accumulée du verbe transitif à l'actif avec sujet et objet sous forme canonique (notée a ci-dessus), nous produisons l'arbre donné en figure 6.7.

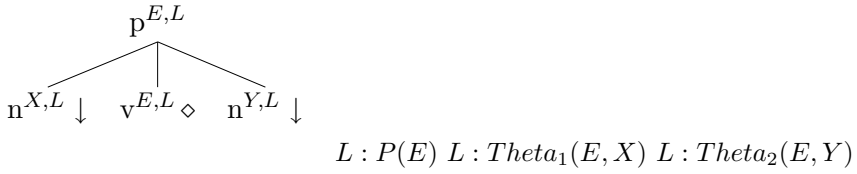


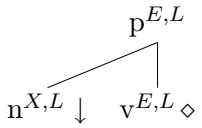
FIG. 6.7 – Arbre de la famille $n0Vn1$.

6.4 Ancrage de la grammaire avec le lexique

Comme mentionné en section 6.2.3, la métagrammaire décrite ici ne produit pas des arbres TAG à proprement parler, mais plutôt des *schèmes*. Un schème est un arbre TAG dans lequel le nœud feuille accueillant un item lexical est supprimé, et le père de ce nœud est étiqueté par un symbole spécifique : $\diamond$ (on désigne alors ce nœud par le terme de nœud *ancre*). Plus précisément, notre métagrammaire produit des entrées associant un schème à une représentation sémantique plate tel qu'illustré en figure 6.8 (arbre de la famille $n0V$).

L'intérêt de ce type de structure est qu'elle évite à l'analyseur syntaxique d'avoir à gérer un nombre très important d'arbres, qui potentiellement partagent une structure similaire (à item lexical près). L'analyseur ne chargera donc que les schèmes, et à partir d'un lexique, reconstruira le nœud de l'item lexical (on parle de l'opération d'*ancrage*).

Dans notre cas, l'ancrage a plusieurs effets, (a) construction du nœud de l'item lexical, (b) résolution des équations d'ancrage (équations de chemin) et (c) instanciation du prédicat



$L : P(E)\ L : Theta_1(E, X)$

FIG. 6.8 – Exemple d'entrée produite par le compilateur de métagrammaire.

et des arguments sémantiques. Dans ce contexte, le lexique que nous utilisons avec notre métagrammaire doit contenir des informations syntaxiques et sémantiques.

En outre, le lexique associé à une grammaire TAG comporte deux niveaux :

1. une association schème – lemme(s).
2. une association lemme – forme(s) fléchie(s).

En d'autres termes, à un schème est associé un ou plusieurs lemmes, et à un lemme est associé une ou plusieurs formes fléchies. Ce double lexique permet une meilleure factorisation de l'information syntaxique en isolant l'information morphologique.

Dans ce qui suit, nous allons voir ce que contient ce double lexique, tout d'abord nous allons nous intéresser au lexique des lemmes, puis à celui des formes fléchies.

6.4.1 Lexique de lemmes

Les informations contenues dans le lexique de lemmes sont énumérées dans le tableau de la figure 6.9.

Champ	**Exemple**
Lemme	dormir
Catégorie syntaxique	v
Représentation sémantique	$RelUnaire, P = dormir, Theta_1 = agent$
Famille	$n0V$
Restrictions d'ancrage	$\emptyset$
Equations d'ancrage	`anc -> aux = avoir`
Coancres	$\emptyset$

FIG. 6.9 – Contenu d'une entrée du lexique de lemmes.

Ainsi, une entrée du lexique de lemmes contient les informations syntaxe et sémantiques partagées par un certain nombre de formes fléchies, à savoir le nom du lemme, la catégorie syntaxique du lemme, la représentation sémantique qui lui est associée, la famille d'arbres, des restrictions d'ancrage éventuelles (exprimées sous forme d'une structure de traits qui doit être unifiable avec l'interface de l'arbre), des équations d'ancrage permettant de définir certains traits syntaxiques des arbres (tel que l'auxiliaire associée au verbe *dormir* dans

notre exemple), et enfin les éventuelles coancres (*e.g.* préposition pour les verbes intransitifs).

6.4.2 Lexique morphologique

Le lexique morphologique contient les informations liées aux formes fléchies, à savoir la forme elle-même, le lemme correspondant, la catégorie syntaxique du noeud ancre dans l'arbre, et un ensemble de traits morpho-syntaxiques (qui seront unifiées avec la structure *bot* du nœud ancre).

Un exemple d'entrée du lexique morphologique est celle de la forme fléchie *dort* :

```
dort    dormir    [pos = v ; mode = ind ; pers = 3 ; num = sg]
```

Dans cette entrée, on indique que la forme fléchie *dort* réfère au lemme *dormir*, que sa catégorie syntaxique (*part of speech, pos*) est v, son mode l'indicatif, troisième personne du singulier.

6.4.3 Opération d'ancrage

Dans ce contexte, l'opération d'ancrage se fait comme suit.

1. A partir du résultat de la segmentation de la phrase à analyser et des entrées du lexique morphologique, l'analyseur syntaxique récupère les lemmes correspondants.
2. Pour chaque couple (forme fléchie, lemme), l'analyseur va récupérer les schèmes des familles d'arbres correspondantes, et utiliser le lexique de lemmes pour réaliser les opérations suivantes :
   - instanciation des informations sémantiques (prédicat et rôles thématiques),
   - ajout d'un certain nombre de couples (attribut, valeur) à la structure de traits top du nœud noté ancre,
   - résolution des équations d'ancrage et ainsi mettre à jour les traits associés aux nœuds.

   Ainsi pour l'exemple du lemme *dormir* introduite précédemment, tous les arbres de la famille $n0V$ vont voir leur prédicat prendre la valeur *dormir* et le rôle thématique $Theta_1$ la valeur *agent*. De plus, la structure de traits top du nœud ancre recevra un trait *aux* associé à la valeur *avoir*.
3. Enfin, l'analyseur va ajouter un nœud sous le nœud marqué ancre, ce nœud contiendra le mot du lexique morphologique (lexicalisation du schème), et l'ensemble des traits morphologiques seront placés dans la structure de traits top du nœud père du nœud anciennement marqué ancre. Dans l'exemple du verbe *dort*, les traits d'accord (*mode*, *pers*, et *num*) seront ajouté au nœud ancre. Cette instanciation de traits pourra éventuellement être partagée à d'autres traits dans l'arbre par coindexation. Le résultat de cette opération d'ancrage produit l'arbre de la figure 6.10.

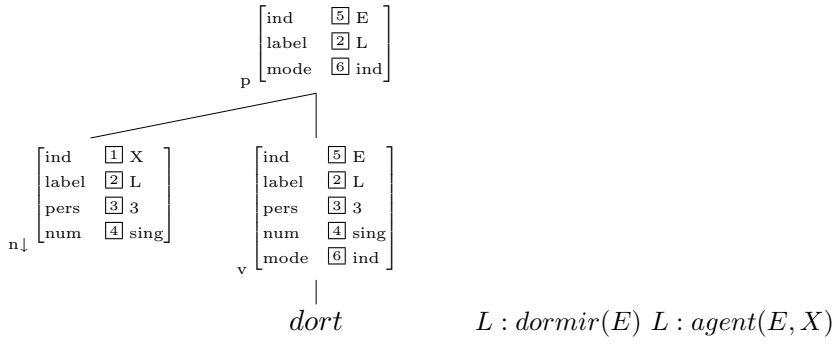


FIG. 6.10 – Exemple d'arbre ancré.

6.5 Conclusion

Nous venons de voir comment définir une métagrammaire réduite du français avec le formalisme XMG, et comment l'étendre pour y inclure l'information sémantique nécessaire à la tâche de construction sémantique. Il convient de préciser ici qu'il ne s'agit que d'un extrait de la grammaire noyau du français décrite par (Crabbé, 2005b) et étendue sémantiquement par (Gardent, 2006). C'est cette dernière grammaire que nous avons utilisée pour notre évaluation (voir chapitre 8).

Pour définir notre métagrammaire réduite du français, nous avons tiré parti des fonctionnalités offertes par le formalisme XMG, à savoir :

- factorisation permise pas l'héritage entre fragments (concerne aussi bien les classes syntaxiques que sémantiques),
- abstraction de haut niveau permise par l'opérateur de disjonction (regroupement des différentes réalisation syntaxiques d'une fonction grammaticale donnée, définition des familles TAG),
- référencement global des variables au moyen des interfaces de classes (partage des indices sémantiques et des indices d'arbres),
- automatisation des identifications de noeuds permise par l'utilisation de nœuds colorés (gestion de la combinaison des blocs élémentaires, gestion des projections sémantiques).

Dans le chapitre 7, nous allons voir comment utiliser les ressources présentées ici (métagrammaire à portée sémantique, lexiques de lemmes et morphologique) pour construire la représentation sémantique d'énoncés à partir du résultat de l'analyse syntaxique.

Troisième partie

Construction sémantique pour Grammaires d'Arbres Adjoints : implantation et évaluation

C'est par l'expérience que la science et l'art font leur progrès chez les hommes.

Aristote

Chapitre 7

L'analyse syntaxique comme base de la construction sémantique

Sommaire

Au chapitre 2, nous avons présenté les procédés de construction sémantique existants pour TAG. En particulier, nous avons distingué différentes tendances, à savoir les approches basées sur l'arbre de dérivation, celles basées sur l'arbre dérivé et enfin les approches hybrides. Comme nous l'avons annoncé alors, l'approche que nous avons choisie pour notre plate-forme est celle de (Gardent et Kallmeyer, 2003)[164] utilisant l'arbre dérivé (ou plus précisément les unifications effectuées pendant la dérivation). Les raisons à ce choix proviennent du fait que cette proposition :

[164] Introduite section 2.2.4.2 page 75.

- définit un cadre formel homogène pour la construction sémantique de différent phénomènes linguistiques,
- utilise l'opération d'unification du formalisme TAG pour la composition sémantique,
- est intégrable dans un système de production semi-automatique de grammaires.

Au chapitre 6, nous avons vu comment produire semi-automatiquement une grammaire TAG incluant une interface syntaxe / sémantique pour cette approche. A présent, nous allons montrer comment utiliser une telle grammaire dans le cadre de la construction sémantique, conformément aux idées de (Gardent et Kallmeyer, 2003). Plus précisément, nous allons présenter deux procédés de construction sémantique pour ce type de grammaire, un premier procédé réalisant le calcul sémantique pendant la dérivation, et un second procédé le réalisant *a posteriori*, à partir du résultat de l'analyse syntaxique (en l'occurrence une forêt de dérivation). Précisons enfin que par construction sémantique, nous entendons le calcul d'une formule sémantique sous-spécifiée. Nous ne cherchons pas à calculer les fonctions d'assignation permettant de produire toutes les formules sémantiques totalement spécifiées correspondantes (*cf* section 1.2.1 page 23).

7.1 Rappel : le calcul sémantique proposé par (Gardent et Kallmeyer, 2003)

Dans un premier temps, nous revenons brièvement sur le calcul sémantique pour TAG proposé par (Gardent et Kallmeyer, 2003).

Les grandes lignes de ce calcul sont les suivantes :

- chaque arbre élémentaire est associé à une formule de sémantique plate,
- les structures de traits décorant les nœuds des arbres élémentaires partagent des variables d'unification avec les formules sémantiques (arguments des prédicats notamment),
- les arguments sémantiques sont déterminés via l'unification des structures de traits inhérentes au formalisme TAG,
- à l'issue d'une dérivation complète, la représentation sémantique de la phrase décrite correspond à l'union des formules sémantiques associées aux arbres élémentaires impliqués (modulo l'unification des variables sémantiques).

Par exemple, la phrase « Jean aime Marie » est dérivée tel qu'illustré sur la figure 7.1[165] et produit la formule sémantique :

$$l_0 : \ aimer(e, j, m), \quad l_1 : \ jean(j), \quad l_2 : \ marie(m)$$

[165] La structure de traits *top* (respectivement *bot*) est noté en exposant (respectivement indice).

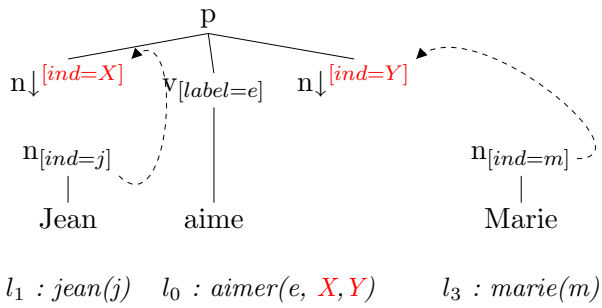


FIG. 7.1 – Calcul sémantique pour la phrase « Jean aime Marie ».

7.2 Un procédé de construction sémantique lors de la dérivation TAG

Une première façon (la plus directe) de construire une représentation sémantique en suivant (Gardent et Kallmeyer, 2003) est d'utiliser la dérivation TAG pour réaliser la composition sémantique et de récupérer les formules sémantiques élémentaires à l'issue de la dérivation. Cela présuppose que nous disposons d'un analyseur syntaxique permettant de manipuler des paires *(arbre élémentaire, formule sémantique)*, ce qui n'est pas le cas. Un moyen de contourner cette limitation est d'intégrer les formules sémantiques dans les arbres élémentaires. Nous pourrons ainsi partager les variables d'unification entre traits décorant les nœuds des arbres et formules sémantiques.

7.2.1 Intégration des formules sémantiques dans les arbres

L'intégration des formules sémantiques dans les arbres passe par l'utilisation d'un nœud spécifique, décoré avec une structure de traits contenant les informations sémantiques (étiquette(s), prédicat(s) et argument(s)). Pour pouvoir intégrer les formules dans les arbres, nous devons respecter les contraintes suivantes :

1. Les formules sémantiques associées à un arbre élémentaire peuvent contenir plusieurs prédicats, nous allons donc avoir besoin de structures de traits récursives pour stocker les formules sémantiques[166]. Un exemple de formule sémantique représentée sous forme de structure de traits est donné figure 7.2[167].

2. la composition sémantique correspond à l'*union* des formules sémantiques élémentaires, en d'autres termes, il ne doit pas y avoir d'unifications directement entre formules sémantiques élémentaires. Ce qui implique que le nœud désigné pour accueillir la formule sémantique ne doit pas s'unifier avec un nœud du même type lors de la

[166]Le nombre de prédicats associés à un arbre élémentaire étant fini, il serait possible de passer par une structure de traits non-récursive, mais qui rendrait l'écriture des formules plus difficiles

[167]Nous associons au trait *semf* la formule sémantique.

dérivation. Pour cette raison, nous choisissons le nœud *ancre* comme site de la formule sémantique. En effet, deux nœuds *ancre* ne peuvent être unifiées lors d'une dérivation.

NB : Théoriquement, l'utilisation de structures de traits récursives nous fait sortir du cadre formel des TAG (en termes de complexité d'analyse), cependant en pratique le problème ne se pose pas ici, puisque cette structure récursive n'est unifiée qu'à l'ancrage et non en cours d'analyse.

$$\ell : dormir(e) \quad ; \quad \ell : agent(e, x) \quad \Leftrightarrow \quad \begin{bmatrix} \text{semf} & \begin{bmatrix} \text{label} & \boxed{1}\ \ell \\ \text{pred} & \text{dormir} \\ \text{arg0} & \boxed{2}\ e \\ \text{semf} & \begin{bmatrix} \text{label} & \boxed{1} \\ \text{pred} & \text{agent} \\ \text{arg0} & \boxed{2} \\ \text{arg1} & \boxed{3}\ x \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

FIG. 7.2 – Représentation sémantique sous forme de structure de traits.

Un autre avantage de l'utilisation du nœud *ancre* pour recevoir la formule sémantique est qu'il est possible d'utiliser l'opération d'ancrage pour instancier les informations sémantiques liées au lemme (*e.g.*, prédicat et arguments sémantiques). Cela est illustré figure 7.3.

Dans ce contexte, la représentation sémantique finale est récupérée via l'extraction de l'union des traits *semf* de l'arbre dérivé. Ces traits auront été mis à jour en cours de dérivation via l'unification des variables partagées lors des unifications inhérentes aux opérations d'adjonction et de substitution (*cf* section 2.1.2 page 39).

Pour illustrer cela, prenons la grammaire de la figure 7.4[168]. Cette grammaire permet d'analyser la phrase « *Jean aime beaucoup Marie* ». Lors de l'analyse, les arbres associés respectivement à *Jean* et à *Marie* vont être substitués sur l'arbre associé à *aime*. La conséquence de ces substitutions est que les indices sémantiques des arguments du prédicat *aimer*, représentés ici par les variables X et Y, vont être unifiés avec les indices j et m représentant respectivement les constituants *Jean* et *Marie*.

De même, l'arbre associé à *beaucoup* va être adjoint sur le noeud ancre de l'arbre associé à *aime*, l'unification des structures de traits mises en jeu vont alors lier les variables évé-

[168] Afin de simplifier le schéma, (1) nous représentons les formules sémantiques sous la forme d'un simple trait, et non d'une structure de trait, et (2) nous omettons les indices d'étiquette de prédicat.

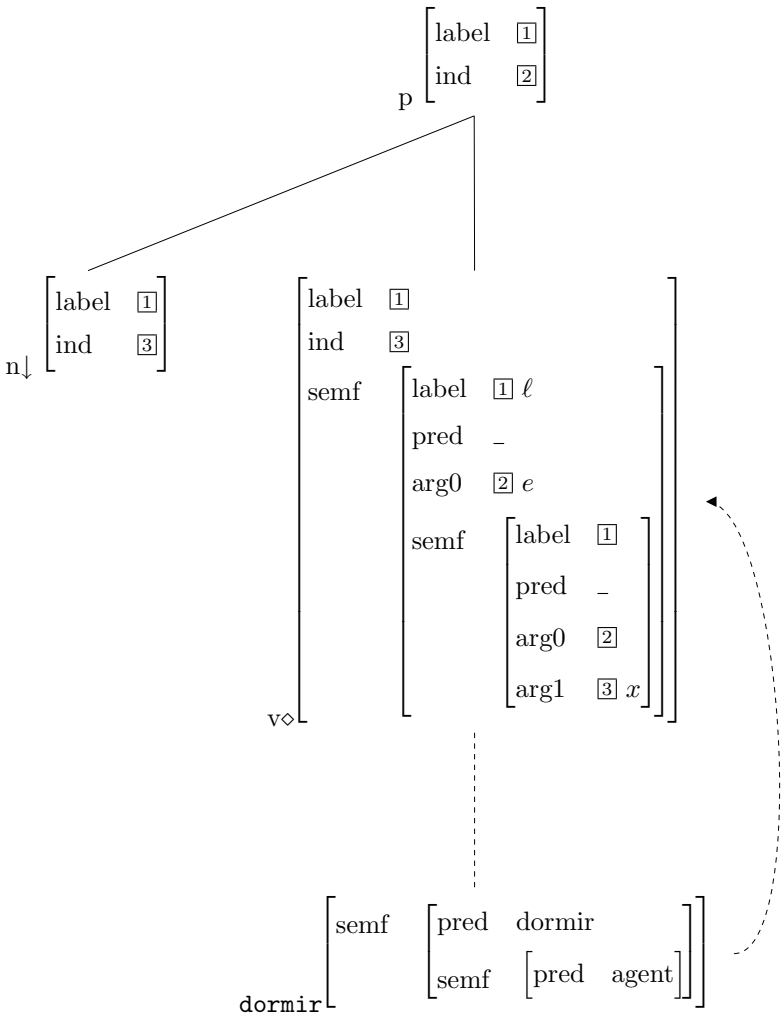


FIG. 7.3 – Ancrage avec instanciation des informations sémantiques.

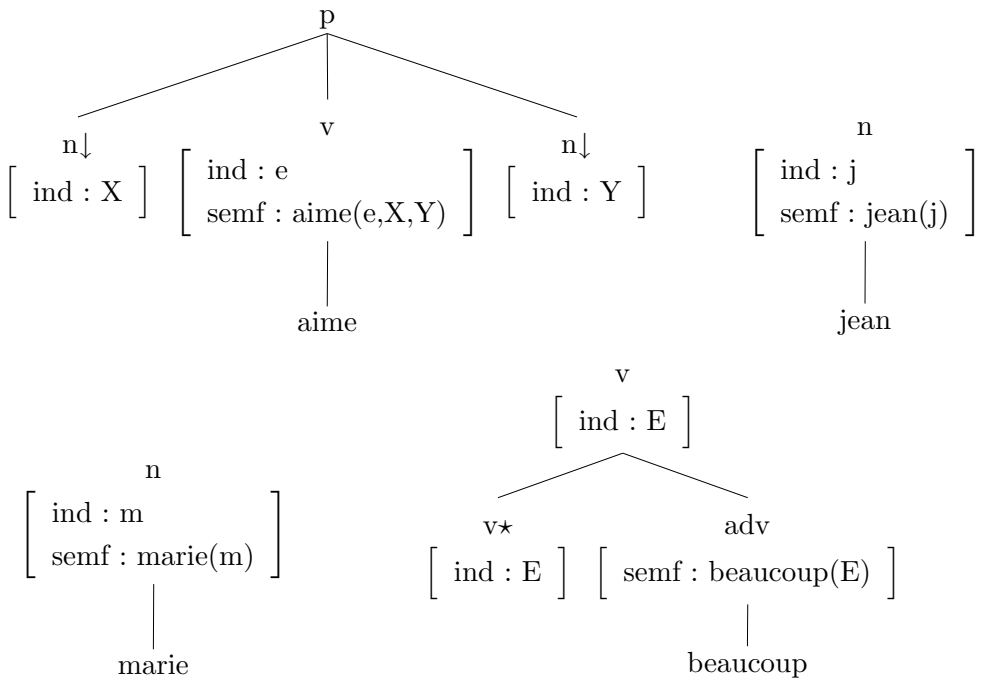


FIG. 7.4 – Arbres élémentaires TAG avec nœud sémantique.

nementielles du verbe et de l'adverbe. Ces unifications sont visibles sur l'arbre dérivé pour cette phrase, représenté figure 7.5.

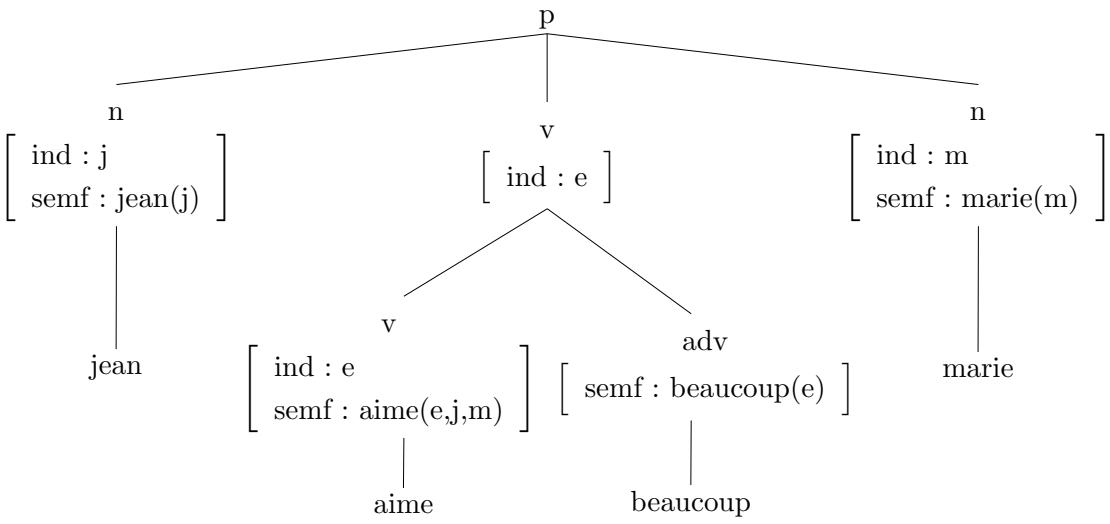


FIG. 7.5 – Arbre dérivé avec information sémantique.

Ainsi, il est possible d'extraire la représentation sémantique de la phrase en parcourant l'arbre dérivé, et en collectant l'ensemble des traits *semf*. L'union de ces traits forme la représentation sémantique. Dans notre exemple, cette représentation est la suivante :

$$\{\, jean(j)\,,\, aime(e,j,m)\,,\, beaucoup(e)\,,\, marie(m)\,\}$$

7.2.2 Implantation au moyen de l'analyseur LLP2

Concernant l'implantation d'un tel procédé de construction sémantique, nous ne nécessitons qu'un analyseur syntaxique classique pour grammaires TAG. En effet, la composition sémantique est réalisée par les opérations de substitution et d'adjonction. Aucune modification de l'analyseur n'est alors nécessaire.

Pour réaliser notre prototype, nous avons utilisé l'analyseur *Loria LTAG Parser 2* (LLP2)[169]. Cet analyseur utilise un algorithme d'analyse par connexité (algorithme de type ascendant) décrit dans (Lopez, 1999), et est fourni avec un ensemble d'outils permettant une utilisation interactive. Parmi ces outils, on compte un segmenteur, un visualisateur d'arbres et un validateur XML pour grammaires TAG au format TagML (Bonhomme et Lopez, 2000).

Notons que cet analyseur ne supporte pas les adjonctions englobantes, et formellement, ne permet que l'analyse des Grammaires d'Insertion d'Arbres (*Tree Insertion Grammars, TIG*). En pratique cependant, les grammaires utilisées n'intègre que rarement des arbres auxiliaires à adjonction englobante (avec un nœud pied précédé et précédant d'autres nœuds).

Pour pouvoir intégrer chaque formule sémantique élémentaire (initialement associée à l'arbre) sous forme d'une structure de traits inclue dans l'arbre, nous devons réaliser un pré-traitement de la grammaire[170].

Une fois la grammaire et les lexiques chargés par l'analyseur syntaxique (après conversion au format TagML), nous pouvons procéder à l'analyse de phrases en mode interactif (*cf* figure 7.6[171]) ou de corpus en mode *batch*. Notons enfin que cet analyseur produit les résultats d'analyse dans un format XML, à partir duquel nous pouvons extraire les formules sémantiques.

7.3 Un procédé de construction sémantique post-dérivation TAG

En plus du procédé de construction sémantique introduit précédemment, nous avons exploré une seconde voie consistant à réaliser le calcul sémantique après la dérivation. Comme nous allons le voir, l'intérêt de cette seconde voie est de bénéficier de l'efficacité des algorithmes d'analyse tabulaires, dans lesquels les calculs intermédiaires sont stockés afin d'être réutilisés. Ces algorithmes produisent un résultat d'analyse sous une forme compacte : la *forêt de dérivation* (voir section 7.3.2 ci-dessous).

[169] Disponible à l'adresse `http://www.loria.fr/~azim/LLP2/`.

[170] La grammaire étant produite automatiquement par XMG dans un format XML, nous utilisons un programme de conversion XSLT pour intégrer les traits sémantiques dans les arbres élémentaires.

[171] On remarque que sur cet exemple de construction sémantique pour une grammaire *jouet*, nous avons placé la formule sémantique dans le nœud racine de l'arbre et non dans le nœud ancre (grammaire jouet ne contenant aucun arbre auxiliaire s'adjoignant sur la racine).

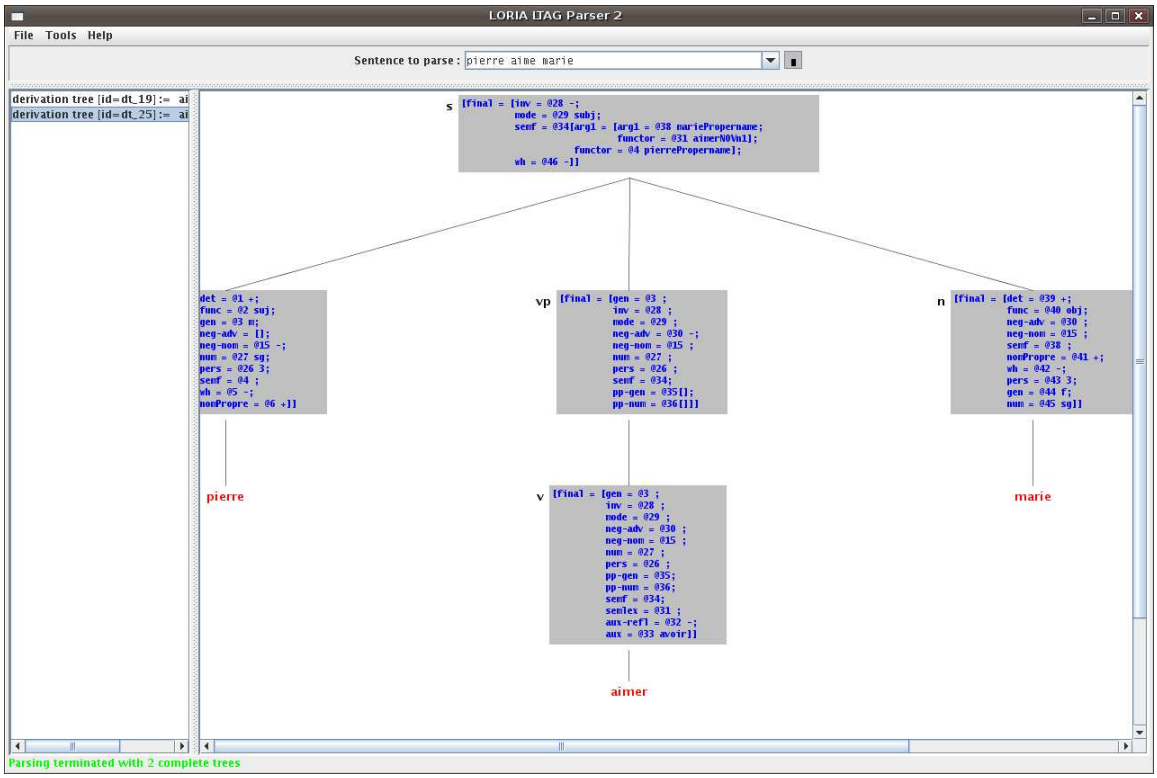


FIG. 7.6 – Construction sémantique avec l'analyseur LLP2.

Ce procédé de construction sémantique se fait en trois temps :

1. Tout d'abord, nous extrayons de chacun des arbres de la grammaire G, l'ensemble des informations sémantiques (adresses de Gorn et traits sémantiques des nœuds de l'arbre, et formule sémantique). Ce qui nous permet de construire, ce que nous appellerons un *lexique sémantique*, noté G'. Nous remarquons que la grammaire après cette phrase d'extraction, correspond à une TAG « classique », notée G''.
2. Ensuite, nous réalisons l'analyse syntaxique de la phrase à partir de la grammaire G''. Le résultat de l'analyse correspond à une forêt de dérivation.
3. Enfin, nous utilisons cette forêt de dérivation et le lexique sémantique G' pour recalculer les unifications ayant eu lieu sur des nœuds ayant un rôle sémantique. Ces unifications produisent la représentation sémantique recherchée.

7.3.1 Construction d'un lexique sémantique

La première étape de notre procédé consiste à extraire les informations sémantiques de notre grammaire TAG produite automatiquement pour disposer (1) d'un lexique sémantique G' que nous utiliserons, avec le résultat de l'analyse syntaxique, pour la construction sémantique (*cf infra*), et (2) d'une grammaire TAG G''.

Nous procédons comme suit :

1. Nous nommons les nœuds des arbres au moyen de leur adresse de Gorn[172], ce qui produit la grammaire TAG G_{noms}.
2. Pour chaque arbre de G_{noms}, nous créons (a) un arbre initial TAG (purement syntaxique) et (b) une entrée dans le lexique sémantique, entrée décrivant une structure d'arbre identique à (a), mais dont les nœuds sont annotés uniquement avec des traits sémantiques.

Cette extraction du lexique sémantique est illustrée en figure 7.7, où l'arbre associé au mot *dort* dans la grammaire à portée sémantique G[173] est traité pour construire un arbre syntaxique appartenant à la grammaire G'' et une entrée dans le lexique sémantique G'.

Le lexique sémantique que nous obtenons ainsi contient des squelettes d'arbres dont les nœuds ne contiennent que des traits sémantiques. Notons que, comme nous l'avons vu au chapitre 6, les arbres de notre grammaire correspondent à des *schèmes*[174], nous utilisons donc le lexique de lemmes pour ajouter à ce lexique sémantique les lemmes pouvant ancrer l'arbre. Plus précisément, chaque entrée dans le lexique sémantique va contenir les informations suivantes :

– nom de l'arbre initial TAG associé,

[172]Nous rappelons que l'adresse de Gorn est définie comme suit : le nœud racine a pour adresse 0, ses fils $1, \ldots, n$ et le k^e fils d'un nœud d'adresse j a pour adresse $j.k$.

[173]L'arbre initial TAG de la figure ne comporte pas l'ensemble des traits syntaxiques habituellement utilisés dans le formalisme, mais uniquement ceux marquant l'accord.

[174]Nous rappelons qu'un schème est un arbre non-ancré.

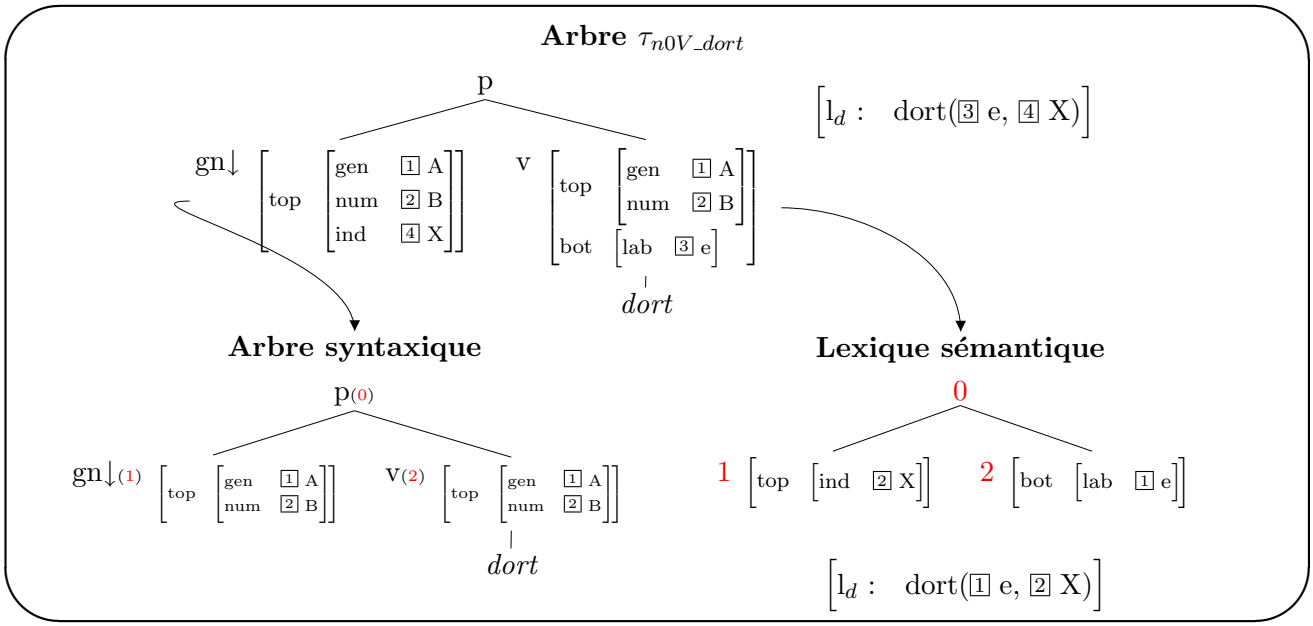


FIG. 7.7 – Création du lexique sémantique.

- nom du lemme correspondant,
- représentation sémantique plate,
- quatre ensembles d'équations de chemins définissant la position des traits sémantiques dans l'arbre initial TAG : (i) l'ensemble des équations référant à des nœuds de substitution, (ii) l'ensemble des équations référant à des nœuds d'adjonction autorisée, (iii) l'ensemble des équations pour les traits sémantiques du nœud racine et enfin (iv) l'ensemble des équations pour les traits sémantique du nœud pied s'il existe (ensemble vide sinon).

Enfin, notons que, pour un arbre donné, on produit autant d'entrées dans le lexique sémantique qu'il y a de lemmes ancrant l'arbre. Cela nous permet de disposer ainsi d'un lexique sémantique indexé par des clés contenant (a) le lemme et (b) le nom d'arbre.

Si nous prenons l'exemple du lemme *dormir* avec sujet réalisé sous forme canonique (figure 7.7 ci-dessus), l'entrée de notre lexique sémantique a la forme suivante :

```
Arbre : T1-n0V
Lemme : dormir
SemR  : L:dormir(e,X)
AdjN  : { 2.bot.[label = e] }
SubN  : { 1.top.[ind = X] }
Root  : {}
Foot  : {}
```

Le lexique sémantique ainsi extrait va nous servir, à l'issue de l'analyse syntaxique, à reconstituer la représentation sémantique plate associée à l'énoncé analysé.

7.3.2 Analyse syntaxique et extraction des informations de la forêt de dérivation

La seconde étape de notre procédé de construction sémantique consiste à réaliser l'analyse syntaxique de l'énoncé pour produire une forêt de dérivation de laquelle sont extraites les informations nécessaires à la construction sémantique.

Analyse syntaxique avec tabulation La définition d'un algorithme d'analyse repose sur la définition de deux composants importants (voir (Villemonte de la Clergerie, 2006)) :

1. **La stratégie d'analyse.** Celle-ci décrit la façon dont les étapes de calcul sont réalisées. En d'autres termes, elle décrit le parcours des sous-arbres lors de l'analyse. Les stratégies les plus courantes peuvent être *ascendantes* (*bottom-up*), dans ce cas, on part des symboles terminaux de la grammaire et on remonte dans l'arbre dérivé par reconnaissance de sous-arbres, *descendantes* (*top-down*), on part de l'axiome de la grammaire et on parcourt l'arbre dérivé en prédisant les sous-arbres menant aux symboles terminaux, ou encore *hybrides*, auquel cas on mélange reconnaissances et prédictions de sous-arbres.
2. **La stratégie de contrôle.** Celle-ci décrit la façon dont est traité l'indéterminisme de la grammaire (ce qui inclut le traitement des boucles). En d'autres termes, cette stratégie décrit l'application des règles de déduction de la stratégie d'analyse. C'est la stratégie de contrôle qui va utiliser la tabulation pour assurer le partage des résultats intermédiaires. Un exemple de stratégie de contrôle est le retour-arrière de Prolog.

Enfin, on note que ces deux stratégies sont indépendantes.

Concernant les stratégies d'analyse pour TAG, les implantations les plus connues correspondent à des extensions des algorithmes **Cocke-Kasami-Younger** (Kasami, 1965) et **Earley** (Earley, 1970). Le premier est décrit dans (Vijay-Shanker et Joshi, 1985) et utilise une stratégie ascendante combinée à une technique de tabulation des sous-arbres dérivés reconnus. Le second, décrit dans (Schabes et Joshi, 1988), utilise une stratégie hybride, plus précisément, il étend l'algorithme CKY avec des règles de prédiction permettant de réduire l'espace de recherche des sous-arbres dérivés.

Tous ces algorithmes d'analyses sont basés sur des invariants d'arbre et présentent des limitations (1) en terme d'efficacité issues de la production de sous-arbres inutiles pour l'analyse de la phrase dans son ensemble (algorithme CKY) ou (2) en terme d'extensibilité, car la correction de telles extensions est délicate à prouver (algorithme Earley).

Un autre type d'algorithme, défini à l'origine pour traiter de manière déterministe une sous-classe des grammaires hors contextes, utilisent des automates à états finis (Tomita, 1987). L'idée est de compiler la grammaire hors-ligne en un automate à états finis qui permet de guider l'analyse (il encode la stratégie d'analyse). Cette technique a été étendue aux grammaires d'unification sous la forme des automates logiques à pile (*Logical Push-Down Automata – LPDA*) (Lang, 1988). Ces automates utilisent une pile dont les éléments

sont des termes et dont les transitions sont appliquées modulo unification.

Dans le prolongement de ces travaux, des algorithmes d'analyse à base d'automates ont été défini pour les grammaires d'arbres adjoints. Ici, une pile ne suffit plus pour gérer l'opération d'adjonction, il a donc fallu étendre le modèle. Une première extension constitue les automates à piles de piles (*Embedded Push-Down Automata– EPDA*) (Joshi et Schabes, 1997). Un autre type d'automates pour l'analyse des TAG correspond aux automates à deux piles (*2-stack automata*) (Becker, 1994), et en particulier à une forme spécifique de ces automates : les automates à deux piles fortement dirigés (*Strongly-Driven 2 Stacks Automata – SD-2SA*). C'est ce dernier type d'algorithme qui est utilisé dans le système d'analyse que nous employons.

L'intérêt de ces algorithmes à base d'automates est de pouvoir encoder différentes stratégies d'analyse. La complexité moyenne d'une stratégie à la Earley, implantée par un automate de type SD-2SA, est $\mathcal{O}(n^6)$.

Dans ce contexte, on utilise les automates pour décrire les stratégies d'analyse et les techniques de tabulation pour l'évaluation de ces automates (*cf* figure 7.8[175]).

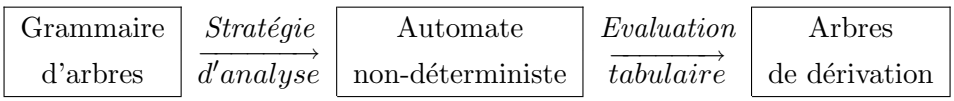


FIG. 7.8 – Analyse syntaxique et tabulation.

A l'issue de cette évaluation d'automates, il est possible d'extraire des objets tabulés non seulement les arbres dérivés, mais également une forêt de dérivation, comme nous allons le voir dans le paragraphe suivant.

Forêt partagée et forêt de dérivation La *forêt partagée* correspond à une représentation compacte de l'ensemble des arbres dérivés correspondant à l'analyse d'une phrase donnée (Billot et Lang, 1989). Plus précisément, pour une grammaire hors-contexte G, la forêt partagée est équivalente à une grammaire hors-contexte G_f qui n'est autre qu'une instanciation de G en annotant, dans les règles, les symboles non-terminaux par des indices représentant la portion de la phrase couverte (Lang, 1991). Un corollaire de cela est que cette forêt n'est autre que l'intersection du langage généré par la grammaire avec un langage régulier généré par la chaîne d'entrée. Il est possible de représenter ce langage régulier par un automate à états finis.

Il existe un résultat similaire pour les grammaires TAG (Lang, 1994; Alonso et al., 2004). La forêt partagée issue de l'analyse pour une grammaire TAG T correspond à l'intersection de T avec un langage régulier. En outre, il est possible de représenter tous les arbres de dérivation pour une phrase donnée sous la forme d'une grammaire hors-contexte. Cette

[175] Figure issue de (Villemonte de la Clergerie, 2006).

représentation est ce que nous appelons la *forêt de dérivation*[176]. Elle contient toutes les dérivations possibles pour une phrase donnée. Pour les TAG, chacune de ces forêts est extraite directement à partir des objets tabulés (Villemonte de la Clergerie, 2005a). Un exemple de forêt de dérivation pour l'analyse de la chaîne *aabbeccdd* au moyen de la grammaire TAG introduite figure 2.7 page 41 (rappelée ici) est donné figure 7.9[177][178].

Dans la forêt de dérivation de l'exemple, exprimée sous forme de grammaire hors-contexte, les règles ont l'un des formats suivants :

$\alpha(g,d)$: dérivation de l'arbre initial α couvrant les indices g à d de la chaîne analysée.

$\beta(g_1,d_1)(g_2,d_2)$: dérivation de l'arbre auxiliaire β pour une adjonction couvrant les indices g_1 à d_1 de la chaîne, avec un nœud pied couvrant les indices g_2 à d_2.

$adj(x)(g_1,d_1)(g_2,d_2)$: dérivation d'une adjonction sur le nœud x couvrant les indices g_1 à d_1 de la chaîne, avec éventuellement dans le résultat un nœud pied (dominé par x) couvrant les indices g_2 à d_2.

$pied(x)(g_1,d_1)(g_2,d_2)$: dérivation du sous-arbre de racine x couvrant les indices g_1 à d_1 de la chaîne, avec éventuellement un nœud pied (dominé par x) couvrant les indices g_2 à d_2.

Construction sémantique par lecture de la forêt de dérivation Pour effectuer la construction sémantique, nous allons utiliser cette représentation factorisée que constitue la forêt de dérivation. Plus précisément, nous allons parcourir cette forêt (sous forme d'une grammaire hors-contexte) et la reformater pour produire des règles ne contenant plus que les informations pertinentes. Le format de nos règles est le suivant :

$$\begin{aligned} DTNodeId_0 :: ElTreeId \leftarrow\ & (DTNodeId_1\ /\ Op_1.ElTreeNodeId_1)+ \\ & (\ |\ (DTNodeId_i\ /\ Op_i.ElTreeNodeId_i)+\ )\star . \\ ElTreeId :: Lemma.TreeName. & \end{aligned} \tag{7.1}$$

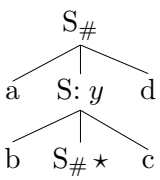
où $DTNodeId_j$ représente un identifiant de nœud dans la forêt de dérivation. $ElTreeId$ réfère à un identifiant d'arbre élémentaire. Op_j représente le type de l'opération j ($Op_j ::= a$ pour une adjonction et $Op_j ::= s$ pour une substitution). Enfin, $ElTreeNodeId_j$ représente le nœud de l'arbre élémentaire sur lequel l'opération j est réalisée. « | » représente la disjonction (utilisée en cas d'ambiguïté d'analyse). Enfin $Lemma$ et $TreeName$ représentent respectivement le lemme et le nom d'arbre définissant l'arbre élémentaire impliqué dans une dérivation.

(7.1) se lit comme suit : l'arbre de dérivation $DTNodeId_0$ est construit en réalisant l'opération Op_1 de l'arbre dérivé désigné par $DTNodeId_1$ sur le nœud $ElTreeNodeId_1$ de l'arbre élémentaire $ElTreeId$.

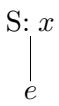
[176] On note que pour les grammaires hors-contextes, forêt partagée et forêt de dérivation sont isomorphes.
[177] Cet exemple est inspiré de (Villemonte de La Clergerie, 2000).
[178] Les structures de traits étiquetant les nœuds sont omis dans la forêt de dérivation pour simplifier la représentation.

Grammaire :

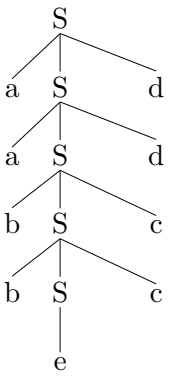
(β)



(α)



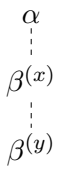
Arbre dérivé :

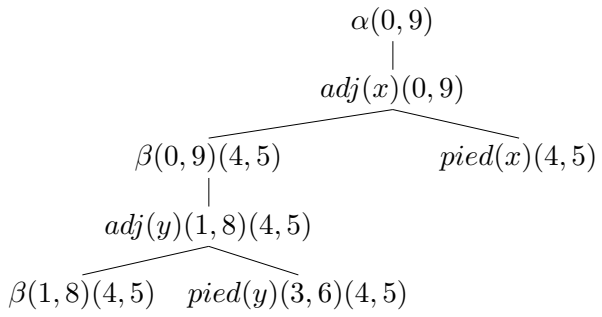


Chaîne :

$$0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 4 \xrightarrow{e} 5 \xrightarrow{c} 6 \xrightarrow{c} 7 \xrightarrow{d} 8 \xrightarrow{d} 9$$

Forêts de dérivation :





Forêt de dérivation sous forme de grammaire hors-contexte :

$\alpha(0,9)$	`1 <- 2`
$adj(x)(0,9)$	`2 <- 3 4`
$\beta(0,9)(4,5)$	`3 <- 5`
$pied(x)(4,5)$	`4 <-`
$adj(y)(1,8)(4,5)$	`5 <- 6 7`
$\beta(1,3)(4,5)$	`6 <-`
$pied(y)(3,6)(4,5)$	`7 <-`

FIG. 7.9 – Forêt de dérivation pour la chaîne $a^2b^2ec^2d^2$.

Exemple 21 (Traitement de la forêt de dérivation)**.** *Pour illustrer ce traitement de la forêt de dérivation, prenons tout d'abord l'exemple du verbe dormir introduit précédemment. L'analyse de la phrase « Jean dort » fournit la forêt de dérivation de la figure 7.10*

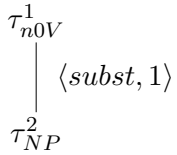


FIG. 7.10 – Forêt de dérivation de la phrase « Jean dort ».

Dans notre système de représentation, cette forêt de dérivation est définie comme suit :

$$r1 :: (t1 \leftarrow (t2/s.1)).$$
$$t1 :: dormir.\tau^1_{n0V}.$$
$$t2 :: jean.\tau^2_{NP}.$$

Cette forêt traduit le fait que l'arbre ancré par « Jean » est substitué au nœud d'adresse 1 *de l'arbre ancré par « dort ».*

Prenons à présent un exemple un peu plus significatif, à savoir « Jean regarde Anne avec un télescope ». Cette phrase est ambigüe, puisque le groupe prépositionnel « avec un télescope » peut être considéré comme un modificateur du nom « Anne » ou du verbe « regarde ». Ce qui donne la forêt de dérivation de la figure 7.11[179]*.*

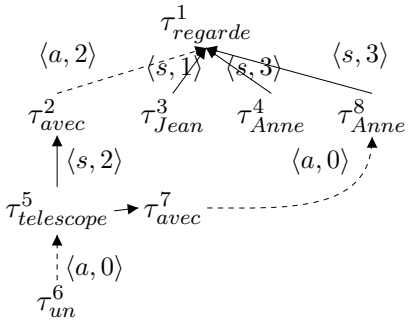


FIG. 7.11 – Forêt de dérivation de la phrase « Jean regarde Anne avec un télescope ».

[179]Les arcs représentant des substitutions sont marqués au moyen d'un trait plein et ceux représentant des adjonctions au moyen d'un trait en pointillés.

Cette forêt est convertie de manière à produire les règles de dérivation suivantes :

$$1 :: (t3 \leftarrow ([5/a.2, 4/s.3, 2/s.1] \mid \ [7/s.3, 2/s.1])). \tag{7.2}$$
$$12 :: (t7 \leftarrow [15/a.0]). \tag{7.3}$$
$$14 :: (t8 \leftarrow [12/s.2]). \tag{7.4}$$
$$5 :: (t1 \leftarrow [12/s.2]). \tag{7.5}$$
$$7 :: (t5 \leftarrow [14/a.0]). \tag{7.6}$$
$$t8 :: avec.\tau^{7}_{avec}. \tag{7.7}$$
$$t7 :: telescope.\tau^{5}_{telescope}. \tag{7.8}$$
$$15 :: un.\tau^{6}_{un}. \tag{7.9}$$
$$t5 :: Anne.\tau^{8}_{Anne}. \tag{7.10}$$
$$2 :: Jean.\tau^{3}_{Jean}. \tag{7.11}$$
$$t3 :: regarde.\tau^{1}_{regarde}. \tag{7.12}$$
$$4 :: Anne.\tau^{4}_{Anne}. \tag{7.13}$$
$$t1 :: avec.\tau^{2}_{avec}. \tag{7.14}$$

Si l'on prend par exemple la règle 7.2, celle-ci nous indique que l'arbre dérivé (représenté par un nœud arbitrairement nommé 1 *dans la forêt de dérivation) est produit par deux analyses (d'où la disjonction) :*

1. *soit par adjonction au nœud d'identifiant* 2 *de l'arbre dérivé par la règle de dérivation* 5 *(avec un télescope), substitution au nœud d'identifiant* 3 *de l'arbre dérivé par* 4 *(Anne) et par substitution au nœud* 1 *de l'arbre dérivé par* 2 *(Jean) sur l'arbre dérivé par* $t3$ *(regarde),*
2. *soit par substitution au nœud* 3 *de l'arbre dérivé par* 7 *(Anne avec un télescope) et par substitution au nœud* 1 *de l'arbre dérivé par* 2 *(Jean) sur l'arbre dérivé par* $t3$ *(regarde).*

Les autres règles s'interprètent de manière similaire.

7.3.3 Calcul de la représentation sémantique

A présent que nous avons (a) extrait de notre grammaire un lexique sémantique, (b) analysé syntaxiquement l'énoncé et récupéré sa forêt de dérivation sous forme de règles hors-contextes, nous pouvons réaliser la troisième étape de notre procédé de construction sémantique : calculer la représentation sémantique correspondant à l'énoncé.

Cette construction sémantique est réalisée au moyen d'un parcours descendant de la forêt de dérivation (reformatée comme mentionné ci-dessus). Lors de ce parcours, pour chaque nœud de la forêt (*i.e.*, pour chaque arbre élémentaire impliqué), nous extrayons l'entrée correspondante dans le lexique sémantique. Pour chaque arc de dérivation, nous calculons les unifications entre traits sémantiques associés aux nœuds utilisés par l'opération de dérivation (adjonction ou substitution). A l'issue de ce calcul, nous récupérons les

formules sémantiques de chacune des entrées du lexique sémantique utilisées (ces entrées contiennent des variables unifiées).

Ainsi, imaginons que nous souhaitons construire la représentation sémantique Sem d'une dérivation de la forme suivante :

$$DTNodeId :: ElemTreeId \leftarrow Daughters \tag{7.15}$$

où $DTNodeId$ représente l'identifiant du nœud racine dans l'arbre de dérivation, $ElemTreeId$ l'identifiant de l'arbre élémentaire correspondant à ce nœud de l'arbre de dérivation, et $Daughters$ une combinaison conjonctive et / ou disjonctive (en cas d'ambiguïté d'analyse) d'opérations de dérivation.

Pour réaliser la construction sémantique, nous procédons comme suit :

1. Nous récupérons le lemme et le nom d'arbre dénotés par $ElemTreeId$ via une fonction nommée $terminal$:

$$Lemma.TreeName \leftarrow terminal(ElemTreeId) \tag{7.16}$$

2. Nous récupérons du lexique sémantique (précédemment extrait de la grammaire) les informations sémantiques associées au lemme $Lemma$ et à l'arbre $TreeName$. Ces informations sont la position des traits sémantiques dans l'arbre et la représentation sémantique plate (avec partage de variables d'unification). Nous appelons $lexSem$ la fonction d'extraction des informations du lexique sémantique :

$$HeadSem \leftarrow lexSem(Lemma.TreeName) \tag{7.17}$$

3. Nous poursuivons le parcours de la forêt en recherchant les informations sémantiques associées aux autres dérivations. Cela se fait au moyen de la fonction $dtrsSem$ introduite plus loin :

$$DtrsSem \leftarrow dtrsSem(HeadSem, Daughters) \tag{7.18}$$

   On note que cette fonction prend comme arguments les informations du lexique sémantique pour le nœud de dérivation courant ($HeadSem$), et les opérations de dérivations filles ($Daughters$).

4. Finalement, la représentation sémantique Sem correspond à l'accumulation modulo unification (notée + ci-dessous) des informations issues du lexique sémantique et de celles issues des dérivations $Daughters$:

$$Sem \leftarrow HeadSem + DtrsSem \tag{7.19}$$

   Lorsque l'on accumule la sémantique des dérivations filles ($DtrsSem$) avec celle de la dérivation courante ($HeadSem$), nous pouvons unifier les structures top et bot des nœuds du lexique sémantique pour l'arbre dérivé décrit par la règle courante. En effet, ces nœuds ne subiront plus d'adjonction.

La construction de la représentation sémantique des dérivations filles, lors du parcours descendant de la forêt, se fait de la manière suivante. Les dérivations filles ont la forme suivante :

$$(DTNodeId \ / \ Op.ElTreeNodeId) \ | \ \ ODtrs \tag{7.20}$$

Nous devons récupérer les informations du lexique sémantique concernant l'arbre élémentaire dénoté par le nœud $DTNodeId$ dans l'arbre de dérivation (7.22), puis unifier ces informations avec celles provenant du lexique pour l'arbre élémentaire dénoté par le nœud père dans l'arbre de dérivation ($HeadSem$) en respectant les contraintes liées à l'opération Op et l'identifiant du nœud où cette opération a lieu, $ElTreeNodeId$ (7.23). Enfin, on continue le parcours de la forêt de dérivation en construisant la sémantique des autres dérivations $ODtrs$ (7.24) et en l'accumulant (modulo unification) avec la sémantique de la dérivation précédemment calculée (7.25) :

$$Lemma.TreeName' \leftarrow terminal(DTNodeId) \tag{7.21}$$

$$HeadSemD1 \leftarrow lexSem(Lemma.TreeName') \tag{7.22}$$

$$tagUnify(Op, EltreeNodeId, HeadSem, HeadSemD1) \tag{7.23}$$

$$ODtrsSem \leftarrow dtrsSem(HeadSem, ODtrs) \tag{7.24}$$

$$DtrsSem \leftarrow HeadSemD1 + semODtrs \tag{7.25}$$

Exemple 22 (Construction sémantique)**.** *Reprenons l'exemple de la phrase « Jean dort » et voyons comment s'effectue la construction sémantique dans ce cas. Nous rappelons ci-dessous la forêt de dérivation reformatée :*

$$r1 :: (t1 \leftarrow (t2/s.1). \tag{7.26}$$

$$t1 :: dormir.\tau^1_{n0V}. \tag{7.27}$$

$$t2 :: jean.\tau^2_{NP}. \tag{7.28}$$

Aussi, les entrées du lexique sémantique pour les arbres τ^1_{n0V}–dormir et τ^2_{NP}–jean sont rappelées ci-dessous :

```
Arbre : T1-n0V
Lemme : dormir
SemR  : L:dormir(e,X)
AdjN  : { 2.bot.[label = e] }
SubN  : { 1.top.[ind = X] }
Root  : {}
Foot  : {}
```

```
Arbre : T2-NP
Lemme : jean
SemR  : Lj:jean(j)
AdjN  : {}
SubN  : {}
Root  : { 0.bot.[ind = j] }
Foot  : {}
```

Si nous prenons la règle de dérivation de départ (7.26), celle-ci traduit le fait que l'arbre élémentaire ancré par Jean (τ^2_{NP}) se substitue à celui ancré par dort (τ^1_{n0V}) sur le nœud d'adresse 1. Le résultat de cette substitution est que les traits top du nœud racine (adresse

0) de τ^2_{NP} et du nœud d'adresse 1 de τ^1_{n0V} sont unifiés (les règles (7.27) et (7.28) servent à extraire du lexique les entrées correspondantes aux arbres élémentaires impliqués dans la dérivation) :

$$(0.top)\,[]\ \sqcup\ [\texttt{ind} = \texttt{X}]\,(1.top)$$

Enfin, lorsqu'il n'y a plus de dérivation fille, c'est-à-dire lorsque l'on accumule les informations sémantiques ($HeadSem + DtrsSem$), les structures `top` *et* `bot` *de chacun des nœuds sont unifiées, en particulier le noeud 1 de l'arbre dérivé dénoté par la règle $r1$ (7.26) :*

$$(1.bot)[\texttt{ind} = \texttt{j}]\ \sqcup\ [\texttt{ind} = \texttt{X}](1.top)$$

Le résultat est que la variable X prend la valeur j, la formule sémantique finale devient :

```
{ L:dormir(e,j), Lj:jean(j) }
```

7.3.4 Implantation au moyen du système DyALog

Pour réaliser l'implantation de ce second procédé de construction sémantique, nous avons utilisé un système permettant de créer un analyseur syntaxique tabulaire produisant une forêt de dérivation : le système DyALog (Villemonte de la Clergerie, 1993).

DyALog Le terme DyALog réfère (a) à un langage de programmation logique (compatible avec Prolog), et (b) à un compilateur pour ce langage.

Initialement développé dans le cadre de l'application de la tabulation à l'évaluation de programmes logiques, DyALog est adapté à la compilation d'analyseurs syntaxiques pour grammaires à base d'unification (Villemonte de la Clergerie, 2002).

Dans ce contexte, le système DyALog convertit la grammaire sous forme de programme logique (règles encodées sous forme de clauses), et compile ce programme, ce qui produit l'analyseur syntaxique. Les formalismes grammaticaux supportés par DyALog sont les grammaires de clauses définies (DCG), les grammaires TAG, les grammaires d'insertion d'arbres (*Tree Insertion Grammars – TIG*), et les grammaires à concaténation d'intervalles (*Range Concatenation Grammars – RCG*). DyALog supporte également les structures de traits typées à la (Carpenter, 1992).

Le procédé de compilation utilisé par DyALog utilise le schéma présenté en section 7.3.2, à savoir, (1) utilisation d'un automate pour encoder la stratégie d'analyse et (2) évaluation tabulaire de cette automate pour extraire les dérivations menant à une analyse. Notons que, pour le cas des grammaires TAG, l'automate utilisé est un automate à deux piles, et que la stratégie d'analyse utilisée par défaut est une stratégie par appels / retours modulés (Barthélemy et Villemonte de la Clergerie, 1998).

En outre le système DyALog permet d'extraire naturellement la forêt de dérivation à partir des objets tabulés lors de l'analyse.

Enfin, ce système[180], est accompagné d'un ensemble d'outils de conversion de grammaires et de visualisation des forêts de dérivation. DyALog est écrit en DyALog (il est compilé par auto-amorçage – *bootstrapping*), et est interfaçable avec des librairies en langage C.

Le module de construction sémantique – SemConst L'implantation du procédé de construction sémantique présenté ici correspond au logiciel SemConst [181] [182]. Celui-ci propose une interface unique permettant de réaliser les trois étapes de la construction sémantique mentionnées ci-dessus, à savoir (1) extraire le lexique sémantique, (2) analyser la phrase pour produire la forêt de dérivation, la reformater, et (3) réaliser la construction sémantique proprement dite. Ce module prend en entrée trois ressources : une grammaire (compilée à partir d'une métagrammaire au format xmg), un lexique de lemmes et un lexique de formes fléchies (tous deux dans un format texte).

La construction sémantique est alors exécutée comme suit :

Etape 1. Utilisation d'un programme XSLT pour nommer les nœuds puis extraire l'information sémantique des arbres de la grammaire. Cette extraction nous permet de construire un pré-lexique sémantique. Ce lexique est alors instancié à partir du lexique de lemmes au moyen d'un programme Perl[183]. Ce qui nous donne le lexique sémantique introduit en section 7.3.1.

Etape 2. Extraction d'une sous-grammaire à partir des mots de la phrase (ou du corpus) au moyen d'un programmer Perl. Conversion de cette sous-grammaire de XML vers le langage DyALog au moyen des outils de conversion du système DyALog [184]. Utilisation de DyALog pour compiler hors-ligne un analyseur syntaxique à partir de la sous-grammaire. Récupération de la forêt de dérivation au format XML et reformattage au moyen d'un programme Oz travaillant sur l'arbre XML. Ce programme retourne la forêt de dérivation sous la forme présentée en section 7.3.2.

Etape 3. Utilisation d'un programme Prolog qui, à partir du lexique sémantique et de la forêt de dérivation reformatée, produit la représentation sémantique plate dans un format texte. Etant donné que l'opération de composition sémantique correspond à l'unification, l'utilisation d'un programme logique pour faire ce calcul sémantique est relativement naturelle.

Ce module a trois modes de fonctionnement, un mode interactif pour le déboguage, où il est possible de procéder pas à pas à la construction sémantique d'une phrase (avec interface

[180] DyALog est disponible librement à l'adresse `http://dyalog.gforge.inria.fr/`.

[181] Voir aussi annexe A.2 page 231.

[182] SemConst est disponible librement à l'adresse `http://trac.loria.fr/~semconst`.

[183] On rappelle que la grammaire produite par xmg contient des schèmes, *i.e.*, des arbres non-ancrés. Il faut donc pour créer notre lexique sémantique, instancier les informations manquantes (prédicat, rôles thématiques) à partir des lexiques de lemmes et de formes fléchies (comme c'est le cas pour la syntaxe).

[184] Outils du paquet tag_utils.

graphique, voir figure 7.12), un mode corpus pour produire la construction sémantique pour les phrases d'un corpus et enfin un mode *batch* pour procéder au traitement séquentiel d'un ensemble de corpus.

7.4 Conclusion

Nous avons présenté deux procédés de construction sémantique pour grammaire TAG, ainsi que leur implantation au moyen d'analyseurs syntaxiques existants, à savoir LLP2 et DyALog.

7.4.1 Comparaison des deux procédés

Dans le premier procédé, la représentation sémantique associée aux arbres de la grammaire est intégrée dans le nœud ancre de l'arbre sous forme d'une structure de traits complexe. Le premier constat que nous pouvons émettre est que le fait d'avoir des structures de traits complexe nous fait sortir du cadre formel des *Feature-Based TAG* (Vijay-Shanker et Joshi, 1988). Cependant, nous n'utilisons pas de structures de traits cycliques, et hormis durant l'ancrage, deux structures complexes ne peuvent être unifiées. Ainsi, en pratique, les performances des analyseurs syntaxiques ne sont pas altérées.

Le second constat, initialement décrit par (Kallmeyer et Romero, 2004), concerne l'utilisation des étiquettes et variables de prédicats sous forme de traits dans les arbres élémentaires. Ces étiquettes et variables de prédicats augmentent le nombre de traits utilisés, de plus nous ne connaissons pas le nombre d'étiquettes et de variables nécessaires. Ainsi l'utilisation d'un nombre potentiellement infini d'étiquettes et de variables étend la capacité générative du formalisme, et nous sortons là encore du cadre formel des *Feature-Based TAG*.

Dans le second procédé, la représentation sémantique ainsi que les traits sémantiques contenus dans les arbres sont extraits de la grammaire, et servent à construire la représentation sémantique par unification *a posteriori*, c'est-à-dire à partir du résultat de l'analyse. En outre, ce résultat correspond à une représentation factorisée des dérivations menant aux analyses réussies. Ce procédé implémente la proposition de (Kallmeyer et Romero, 2004). Nous l'avons appliqué à des grammaires produites automatiquement et utilisé dans le cadre d'une analyse syntaxique tabulaire pour laquelle nous obtenons les dérivations sous forme compacte. Cette double factorisation (grammaire et dérivations) nous permet de réaliser la construction sémantique au moyen de ressources de taille importante, avec des résultats en terme de temps de traitement encourageants (voir chapitre 8).

Remarque. Pourquoi ne pas utiliser un analyseur tabulaire avec des arbres contenant l'information sémantique dans les traits associés au nœud ancre ? Tout d'abord, il faut remarquer que les traits sémantiques ne vont en aucun cas pouvoir guider l'analyse syntaxique

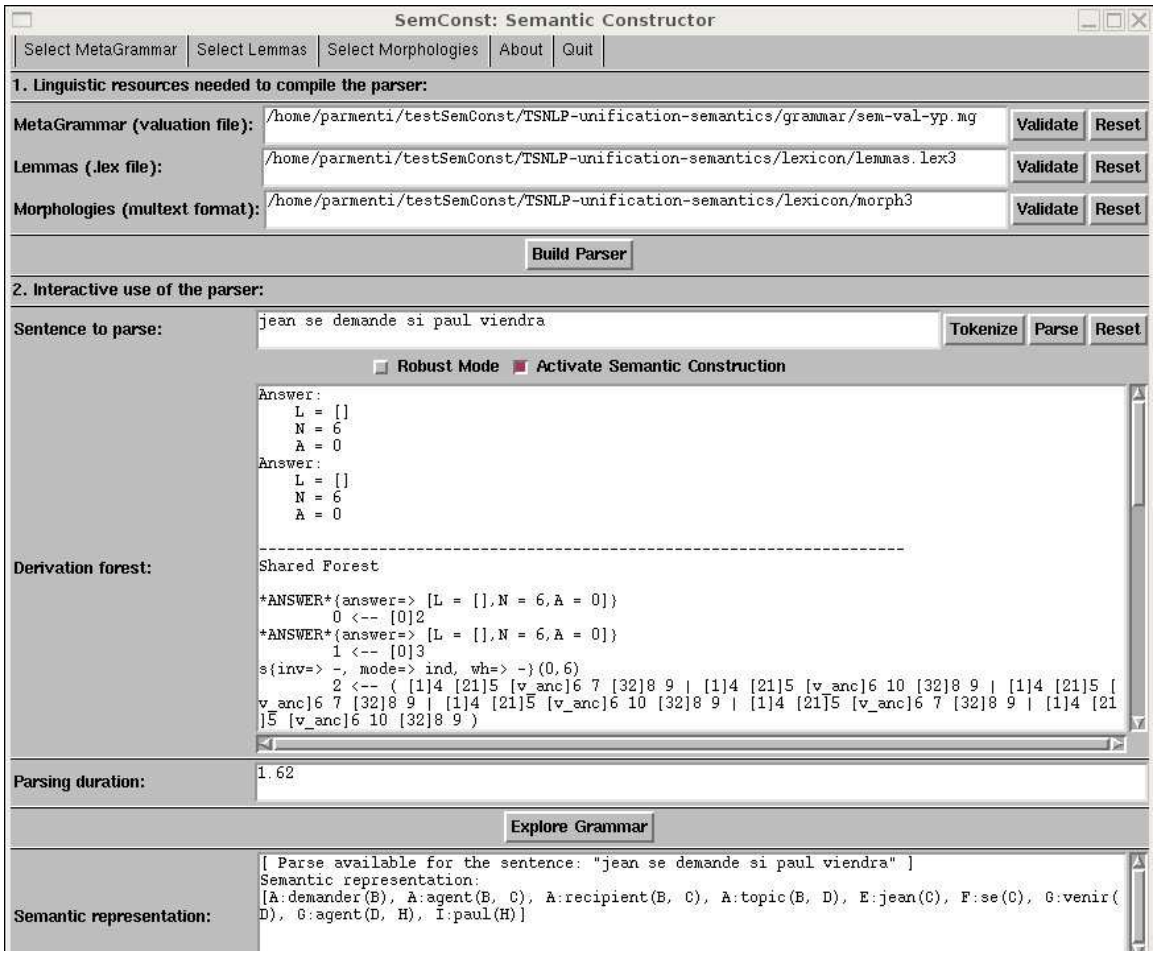


FIG. 7.12 – Interface du module de construction sémantique.

(et n'apporteront donc aucun gain en terme de temps d'analyse). En outre, le fait d'avoir un nombre important de traits dont les valeurs varient fortement dans la grammaire peut avoir un impact néfaste sur le partage de calculs lors de l'analyse syntaxique.

7.4.2 Perspectives

Plusieurs pistes peuvent être imaginées concernant la construction sémantique basée sur l'analyse syntaxique. La première concerne une extension du second procédé, ayant pour but de construire la représentation sémantique directement pendant l'analyse, plus précisément, lors de la construction de la forêt de dérivation. Cela devrait être possible à condition de pouvoir lier les objets tabulés lors de l'analyse syntaxique aux informations de notre lexique sémantique.

Une autre piste de recherche concerne une extension de nos deux techniques de construction sémantique à d'autres types de représentations sémantiques que les formules prédicatives, et à l'utilisation d'autres mécanismes de composition sémantique que la seule unification.

Chapitre 8

Evaluation du procédé de construction sémantique

Sommaire

Dans ce chapitre, nous donnons une évaluation de la plate-forme de construction sémantique proposée. Dans un premier temps, nous introduisons l'objet de cette évaluation. Nous présentons ensuite la méthodologie d'évaluation et finalement commentons les résultats.

8.1 Objet de l'évaluation

L'objectif principal de cette thèse est de fournir des outils permettant de réaliser la tâche de construction sémantique au moyen du formalisme TAG. Dans ce contexte, comment évaluer la pertinence des outils proposés (XMG et SemConst) ?

L'évaluation que nous proposons ici consiste à vérifier l'adéquation des outils à la tâche qui leur incombe. En d'autres termes, il s'agit de vérifier qu'une grammaire à portée sémantique produite par XMG permet de décrire la syntaxe d'une portion significative du français[185] et également d'en construire une représentation sémantique correcte.

Dans ce cadre, les points suivants ont été évalués :

[185] Par significatif, nous entendons une portion couvrant les phénomènes linguistiques courants décrits en section 8.2 (accord, passif, etc).

- capacité de la grammaire à analyser des phrases grammaticales ;
- capacité de la grammaire à rejeter des phrases agrammaticales ;
- calcul du taux d'ambiguïté (nombre d'analyses par phrase),
- capacité de la grammaire à construire une représentation sémantique correcte des phrases grammaticales.

Concernant ce dernier point, il convient de préciser ce que nous entendons par représentation sémantique *correcte*.

L'idéal serait de disposer d'un corpus de test associant à une phrase sa (ou ses) représentation(s) sémantique(s) plate(s). Malheureusement, ce n'est pas le cas. Nous avons donc choisi de réaliser une validation manuelle des représentations sémantiques construites. Cette validation se fait au moyen d'un logiciel développé pour l'occasion (voir procédure d'évaluation présentée en section 8.2.2 ci-dessous). Notons que le résultat de cette validation manuelle constituera un corpus de test associant syntaxe et sémantique, et pourra être utilisé en tant que tel pour l'évaluation de générateurs comme celui de (Gardent et Kow, 2005).

8.2 Méthodologie

Pour réaliser notre évaluation, nous avons réutilisé une partie du travail de (Crabbé, 2005b). Ce dernier a évalué la couverture syntaxique de la grammaire noyau dont nous nous inspirons ici.

Pour son évaluation, (Crabbé, 2005b) a créé un corpus de test dédié à partir de la *Test Suite for Natural Language Processing* (Lehmann et al., 1996). Nous allons dans un premier temps présenter cette suite de tests, puis nous introduirons notre procédure d'évaluation.

8.2.1 Test Suite for Natural Language Processing

La *Test Suite for Natural Language Processing* (TSNLP) est un projet de la Communauté Européenne, mené entre décembre 1993 et mars 1996[186]. Le but de ce projet a été de fournir un environnement de développement de suites de tests *spécifiques* et *réutilisables* pour l'évaluation d'outils pour le TAL (analyseurs syntaxiques, traducteurs, correcteurs orthographiques, etc).

Le cœur de l'environnement fourni par la TSNLP correspond à une base de données de jeux de phrases tests pour l'anglais, l'allemand et le français (la TSNLP contient plus de 5000 phrases tests pour chacune de ces langues). Cette base de données contient des phrases annotées par un certain nombre d'informations :

1. des informations centrales (l'auteur de la phrase, son origine, sa catégorie syntaxique, sa grammaticalité, ses principaux constituants et leur dépendances),

[186] Voir aussi `http://cl-www.dfki.uni-sb.de/tsnlp/`.

2. des informations concernant le phénomène linguistique couvert par la phrase (et éventuellement sur l'interaction avec d'autres phénomènes),
3. l'appartenance à une série de tests,
4. des informations sur l'utilisation de la phrase et l'application évaluable par la phrase de test.

L'idée de la TSNLP est, à partir de cette base de données, de permettre à l'utilisateur de construire une suite de tests spécifique à son application et à ses critères d'évaluation (phénomènes linguistiques étudiés, fonctions grammaticales désirées, longueur des phrases considérées, etc). Pour cela, la TSNLP a été développée avec un ensemble d'applications facilitant la maintenance et l'utilisation de la base de données centrale (notamment pour en extraire des phrases de tests pertinentes pour une certaine évaluation). Ces applications correspondent au logiciel ($tsdb_1$) (*test-suite-data-base 1*), dont la version maintenue actuellement est le `[incr_tsdb()]` (Oepen, 2001). C'est cette version qui a été utilisée pour l'évaluation présentée ici.

Notons enfin qu'un apport majeur des suites de tests par rapport aux corpus de textes classiques est qu'elles permettent de disposer de données contrôlées, systématiques, exhaustives et incluant des phrases négatives (données invalides afin de vérifier la capacité de l'outil à détecter des phrases incorrectes).

8.2.2 Procédure d'évaluation de la plate-forme SemTAG

A présent, nous présentons la procédure mise en place pour évaluer la couverture sémantique de notre grammaire[187]. Dans un premier temps, il a fallu constituer un corpus de test (extrait donc à partir de la TSNLP), puis associer aux mots contenus dans ce corpus des lexiques morphologiques et syntaxiques afin de pouvoir ancrer les arbres de la grammaire (*cf* chapitre 6). Dans un second temps, il a fallu réaliser la construction sémantique sur l'ensemble du corpus de test, et valider les représentations sémantiques produites.

8.2.2.1 Production du corpus de test et des lexiques

La production du corpus s'est faite par interrogation du TSNLP, en considérant les phénomènes suivants :

- complémentation,
- accord,
- passif,
- modification,
- temps - aspect - modalité,
- coordination,
- négation.

[187] Ou plutôt de notre métagrammaire !

Les lexiques ont été produits à partir de ceux de (Crabbé, 2005b), eux-mêmes créés au moyen du logiciel `[incr_tsdb()]` sus-cité. Plus précisément, ce logiciel a permis d'extraire les mots utilisés dans les phrases tests. Le vocabulaire ainsi constitué a ensuite été croisé avec le lexique morphologique libre *Multext* (Ide et Véronis, 1994)[188]. Ainsi a été créé le lexique morphologique (*cf* chapitre 6).

Le lexique syntaxique a, quant à lui, été développé à la main à l'aide d'un concordancier. Rappelons que les entrées de ce lexique syntaxique sont des lemmes associés à des cadre de sous-catégorisation (correspondant aux familles d'arbres décrites dans la métagrammaire). Dans ce contexte, chaque mot (forme fléchie) est associée à un lemme via le lexique morphologique et peut ainsi servir à ancrer un arbre produit par la métagrammaire via le lexique syntaxique. Rappelons que c'est également dans le lexique syntaxique que les informations sémantiques permettant d'instancier les prédicats de la formule sémantique associée à l'arbre élémentaire sont définies (*cf* section 6.4 page 178).

Notons enfin que, comme (Crabbé, 2005b), (i) il nous a fallu ajouter certains segments dans les phrases tests, car l'analyse syntaxique prend comme axiome la catégorie syntaxique p (phrase), et (ii) le corpus a du être pré-traité pour supprimer la ponctuation[189].

8.2.2.2 Validation des représentations sémantiques

Une fois les lexiques et corpus de tests construits. Il est possible d'exécuter la construction sémantique en mode *batch* sur l'ensemble des corpus et de récupérer les représentations sémantiques correspondantes. L'étape finale de l'évaluation consiste alors à valider les représentations sémantiques calculées pour chaque phrase test par SemConst. Pour rendre cette étape de validation plus facile, nous avons développé un outil graphique permettant de sélectionner les représentations graphiques valides parmi les représentations produites par SemConst, et de constituer ainsi un nouveau corpus de test associant une phrase avec ses représentations sémantiques[190]. Cette tâche de validation manuelle reste cependant fastidieuse sur les corpus que nous avons utilisés ici, du fait de leur grande taille. C'est pourquoi les statistiques données ci-dessous correspondent à la sortie du SemConst. En d'autres termes, la seule garantie de la correction des représentations sémantiques produites est la vérification manuelle *non-exhaustive* que nous avons appliquée (nous revenons sur ce point en conclusion du présent chapitre).

8.3 Résultats et commentaires

Avant de donner les résultats chiffrés de cette évaluation, précisons que la métagrammaire utilisée décrit près de 6800 arbres. Cette grammaire est fortement inspirée de la

[188] Voir aussi `http://aune.lpl.univ-aix.fr/projects/multext/`.

[189] A la différence de (Crabbé, 2005b), les désamalgames sont gérés par l'analyseur syntaxique et non par pré-traitement du corpus de test.

[190] Dans un format utilisable directement par le réalisateur GenI (Gardent et Kow, 2005).

grammaire du français de (Crabbé, 2005b).

Trois corpus de test ont été utilisés :

- un corpus d'items grammaticaux (capacité de la grammaire à analyser des phrases grammaticales et à en construire une représentation sémantique),
- un corpus d'items agrammaticaux (capacité de la grammaire à rejeter des phrases agrammaticales, ce qui donne une idée de la *sur-génération* de la grammaire),
- un corpus d'items ambigüs (calcul de l'ambiguïté moyenne en fonction de la longueur de la phrase).

8.3.1 Statistiques

Pour le premier corpus (items grammaticaux), les résultats sont les suivants :

- Nombre de phrases : 1495
- Nombre de phrases ayant au moins une analyse *syntaxique* : 940
- Nombre de phrases ayant au moins une analyse *sémantique* : 933
- Couverture syntaxique globale : 62.88 %
- Couverture sémantique globale : 62.41 %
- Ambiguïté sémantique moyenne : 2.46

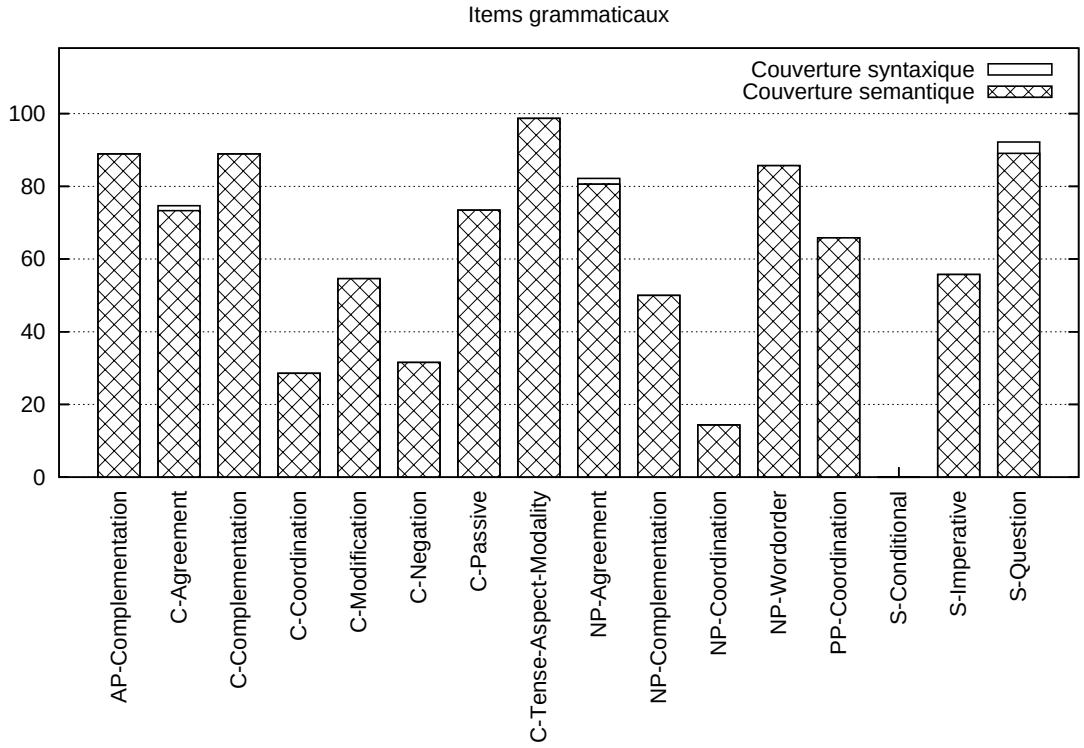
Nous remarquons que la couverture sémantique est quasi-identique à la couverture syntaxique (seules 7 phrases ont une forêt de dérivation et aucune représentation sémantique). La couverture globale reste cependant relativement faible, près de 63 % des phrases sont analysées syntaxiquement (calcul d'une forêt de dérivation) et sémantiquement (calcul d'une ou plusieurs représentations sémantiques). Nous revenons sur ces chiffres en section 8.3.2. Le détail des résultats de l'analyse sémantique du corpus d'items grammaticaux, triés par phénomène linguistique, est donné figure 8.1 page 216.

Le second corpus correspond aux items agrammaticaux. Les résultats de l'analyse de ce corpus sont donnés ci-dessous :

- Nombre de phrases : 1956
- Nombre de phrases ayant au moins une analyse *syntaxique* : 298
- Nombre de phrases ayant au moins une analyse *sémantique* : 291
- Couverture syntaxique globale : 15.24 %
- Couverture sémantique globale : 14.88 %
- Ambiguïté sémantique moyenne : 2.31

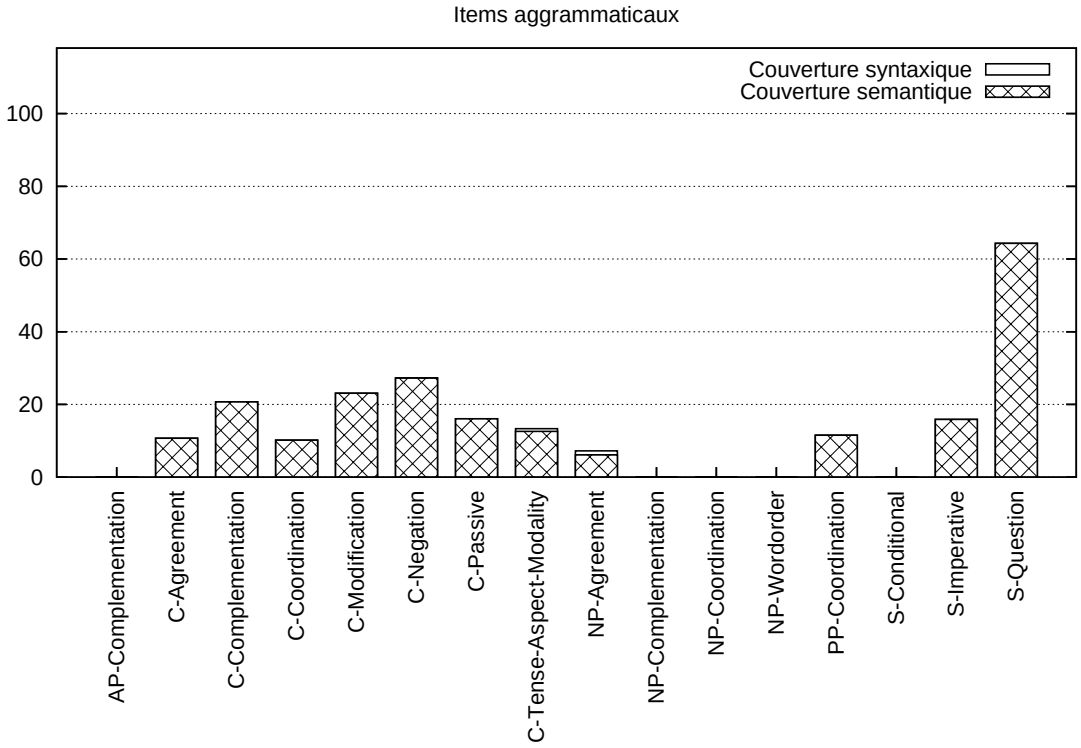
En d'autres termes, la grammaire permet de rejeter près de 85 % des phrases agrammaticales. Le détail des résultats, classé par phénomène linguistique est donné figure 8.2 page 217.

Le troisième corpus concerne les phrases syntaxiquement ambiguës. Le nombre moyen de représentations sémantiques calculées en fonction de la longueur de la phrase est donné figure 8.3 page 218.



Phénomène	Syntaxe	Sémantique	Ambiguïté	Nbre items
AP_Complementation	88.89	88.89	1.88	9
C_Agreement	74.67	73.33	1.15	75
C_Complementation	88.89	88.89	3.17	180
C_Coordination	28.57	28.57	2.03	105
C_Modification	54.61	54.61	1.52	293
C_Negation	31.58	31.58	1.33	57
C_Passive	73.46	73.46	3.29	162
C_Tense	98.70	98.70	1.38	77
NP_Agreement	82.17	80.62	2.02	129
NP_Complementation	50.00	50.00	4.00	6
NP_Coordination	14.37	14.37	1.67	167
NP_Wordorder	85.71	85.71	2.33	7
PP_Coordination	65.85	65.85	5.33	41
S_Conditional	0.00	0.00	0.00	7
S_Imperative	55.77	55.77	1.86	52
S_Question	92.19	89.06	3.61	128

FIG. 8.1 – Résultats sur le corpus d'items grammaticaux.



Phénomène	Syntaxe (%)	Sémantique (%)	Ambiguïté	Nbre items
AP_Complementation	0.00	0.00	0.00	22
C_Agreement	10.69	10.69	1.21	131
C_Complementation	20.74	20.74	2.49	516
C_Coordination	10.20	10.20	1.60	49
C_Modification	23.08	23.08	2.00	13
C_Negation	27.27	27.27	1.00	11
C_Passive	16.07	16.07	1.67	112
C_tense	13.33	12.59	1.09	270
NP_Agreement	7.24	6.14	1.96	456
NP_Complementation	0.00	0.00	0.00	19
NP_Coordination	0.00	0.00	0.00	119
NP_Wordorder	0.00	0.00	0.00	7
PP_Coordination	11.54	11.54	3.11	78
S_Conditional	0.00	0.00	0.00	11
S_Imperative	15.91	15.91	1.71	44
S_Question	64.29	64.29	3.32	98

FIG. 8.2 – Résultats sur le corpus d'items agrammaticaux.

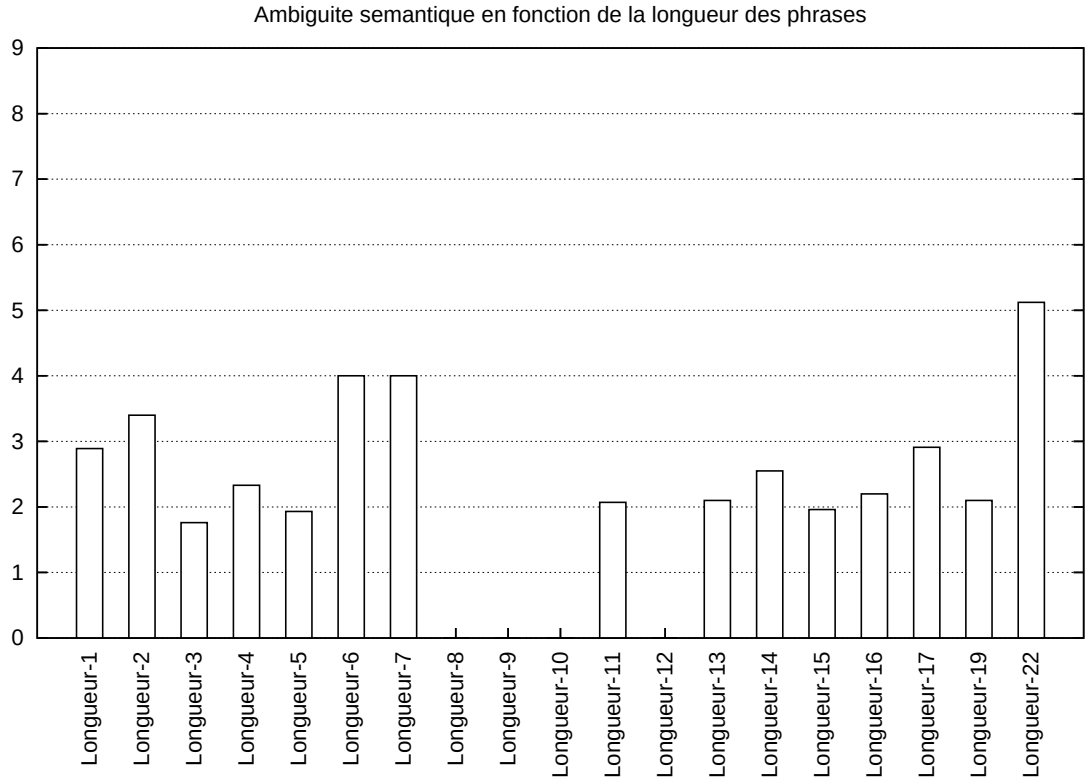


FIG. 8.3 – Evolution de l'ambiguïté sémantique en fonction de la longueur de la phrase.

Une autre statistique intéressante concerne la distribution de l'ambiguïté dans le corpus de phrases grammaticales. En d'autres termes, nous souhaitons savoir, parmi les phrases analysées sémantiquement, combien de phrases ont une représentation sémantique, combien en ont 2, etc. Cette information est donnée figure 8.4 page 220.

8.3.2 Commentaires

Donnons quelques explications sur les limites de la grammaire, d'un point de vue couverture syntaxique tout d'abord, puis d'un point de vue couverture sémantique. Notons que nous parlons ici du corpus d'items grammaticaux.

Comme annoncé précédemment, la grammaire que nous avons utilisée est fortement inspirée de celle de (Crabbé, 2005b)[191]. Etant donné que nous n'avons (actuellement) pas travaillé à son extension, celle-ci souffre des mêmes limites, à savoir[192] :

- l'insertion d'adverbes entre arguments(utilisée dans la section C_Modification) n'est pas supportée par la grammaire,
- les phénomènes de coordination sont fortement représentés dans la TSNLP, mais mal supportés dans le formalisme TAG en général, et dans notre grammaire en particulier (seuls les phénomènes de coordination entre constituants sont traités),
- il existe des limites d'implantation par choix de l'auteur, ainsi le causatif, les principiales et le comparatif ne sont pas traités dans la grammaire,
- il existe également des limites d'implantation de la négation, seul le cas canonique *ne ... pas* est traité,
- enfin, la suite de test semble contenir un certain nombre d'erreurs (phrases idiomatiques douteuses, grammaticalité incertaine).

Notons que nous constatons également un problème technique dans l'affichage de la forêt de dérivation[193]. Ce problème concerne 109 phrases de l'ensemble du corpus d'items grammaticaux[194].

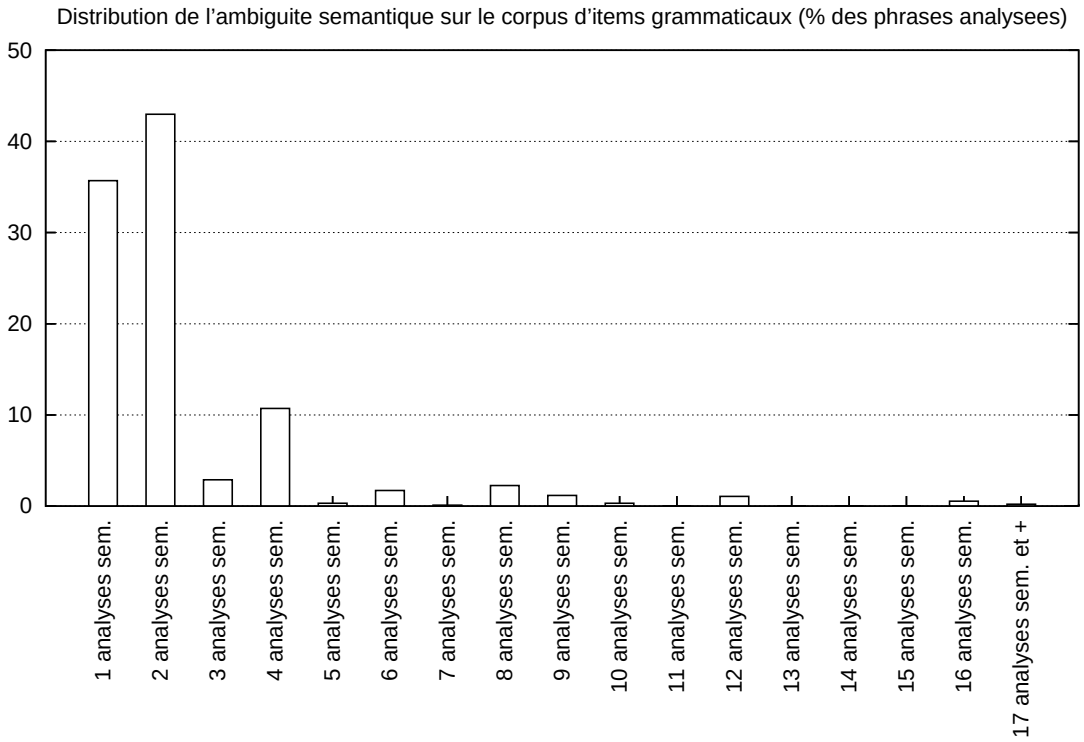
Concernant la couverture sémantique, il est tout d'abord encourageant de noter que cette couverture est très proche de la couverture syntaxique. En d'autres termes, pour quasiment chaque phrase dont nous disposons d'une structure syntaxique, il nous est possible de construire une (ou plusieurs) représentation(s) sémantique(s). Ainsi, sur 940 phrases analysées syntaxiquement, seules 7 structures syntaxiques ne nous permettent pas de construire de représentation sémantique. Ces échecs sont liés au traitement des désamalgames (*e.g.*, conversion de *au* en *à + le*). Les désamalgames sont gérés par l'analyseur syntaxique. Ce-

[191] Cependant, la grammaire de (Crabbé, 2005b) a une couverture de 76 % sur le corpus d'items grammaticaux. Il semblerait que cette différence par rapport à notre évaluation provienne de la version de la grammaire sur laquelle nous nous sommes basés.

[192] Pour une présentation plus détaillée de ces limites, voir (Crabbé, 2005b).

[193] L'analyseur syntaxique calcule une forêt de dérivation, mais ne parvient pas à la convertir en XML, ce problème a été signalé à l'auteur du système DyALog que nous utilisons et devrait être résolu prochainement.

[194] Ce problème explique en partie la différence de couverture par rapport à la grammaire de (Crabbé, 2005b).



Nombre d'analyses sémantiques	Pourcentage des phrases analysées
1	35.69
2	42.98
3	2.89
4	10.72
5	0.32
6	1.71
7	0.11
8	2.25
9	1.18
10	0.32
11	0.00
12	1.07
13	0.00
14	0.00
15	0.00
16	0.54
17 et +	0.21
TOTAL	100.00

FIG. 8.4 – Distribution de l'ambiguïté sémantique sur le corpus d'items grammaticaux.

pendant, la phrase d'entrée contient le mot de départ, il en est de même pour les lexiques. Ce sont donc ces mots qui contiennent l'information sémantique. Par contre, dans la forêt de dérivation retournée par l'analyseur syntaxique, ce sont les formes « dépliées » qui étiquettent les nœuds. Nous ne parvenons donc plus à faire le lien avec l'information sémantique contenue dans le lexique sémantique. Une solution à cela consisterait à traiter les désamalgames en pré-traitement du corpus d'entrée comme le fait (Crabbé, 2005b). Une autre solution serait d'étendre l'analyseur syntaxique afin qu'il utilise la forme compacte dans la forêt de dérivation.

8.4 Conclusion

Dans ce chapitre, nous avons présenté les premiers résultats d'une évaluation de la grammaire du français développée au moyen du système SemConst. Les résultats sont encourageant (couverture sémantique proche de la couverture syntaxique). Ce travail s'appuie sur les travaux de (Crabbé, 2005b) et permet de s'approcher de l'objectif fixé : disposer d'une grammaire à large couverture du français et intégrant une information sémantique.

Ce travail d'évaluation nous a permis de constater la difficulté de la détection d'erreur dès lors que l'on travaille avec des ressources de taille importante. C'est pourquoi nous nous sommes efforcés de rendre l'environnement de travail SemConst le plus adapté possible, notamment en produisant des rapports d'exécution au format pdf. Ces rapports contiennent (a) des listings détaillés mettant en relief les phrases en échec d'analyse (syntaxique et / ou sémantique) et les temps de traitements, et (b) des résumés pour chaque phénomène traité (statistiques globales).

Concernant le problème de la vérification de la correction des représentations sémantiques produites mentionné précédemment, nous constatons qu'il y a plusieurs causes possibles à la production d'une représentation sémantique incorrecte. Les principales sont les suivantes :

1. le lexique contient une information sémantique erronée,
2. l'analyse syntaxique n'est pas celle attendue (ce qui se produit par exemple en cas de sur-génération, *cf* figure 8.5 où la représentation # 2 est incorrecte car elle traite le patient comme un modificateur).

Les effets de ces erreurs sur les représentations sémantiques produites sont diverses. Tout d'abord, (1) il est possible que certaines coindexations entre variables sémantiques soient erronées, ensuite (2) il est possible que des prédicats inappropriés soient utilisés car de mauvais arbres élémentaires apparaissent dans la forêt de dérivation.

Le cas 1 devrait pouvoir être détecté automatiquement en cherchant à construire le graphe sémantique correspondant à une représentation donnée. Le cas 2 est plus difficile à détecter. Il nous faudrait pourvoir caractériser les cas de sur génération, et retirer les représentations problématiques, par exemple en utilisant des heuristiques. Nous pensons

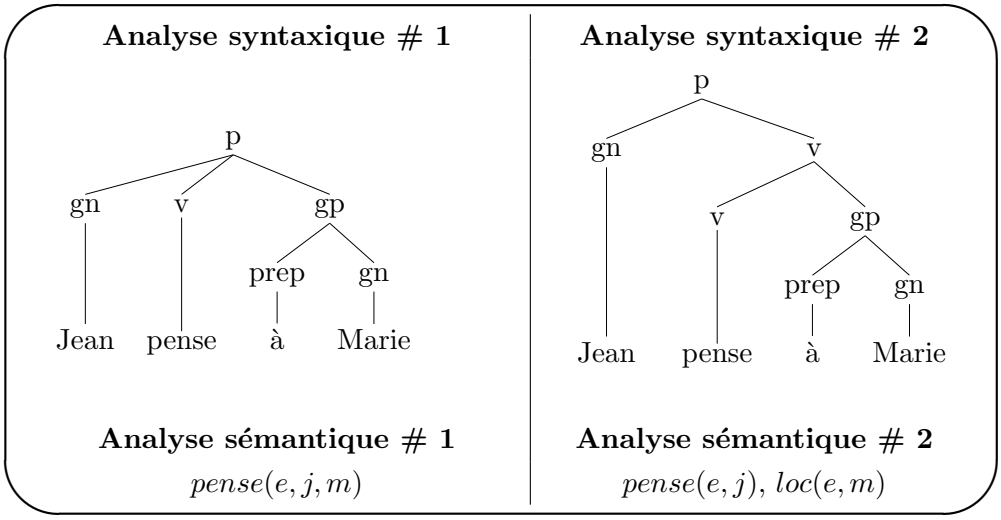


FIG. 8.5 – Représentation sémantique inappropriée due à une sur génération.

également que la génération pourrait apporter une certaine aide. Par exemple, en générant des énoncés à partir de représentations incorrectes, nous pourrions déterminer des critères de malformation dans la grammaire pour un type de représentation sémantique erronée.

Enfin, nous pouvons mentionner le fait que nous avons utilisé un lexique de taille très restreinte dans notre évaluation, il serait intéressant de confronter les performances du système à des lexiques de taille importante. Malheureusement, de tels lexiques *intégrant une information sémantique* ne sont pas disponible actuellement. De plus, il n'est pas envisageable de réaliser l'interface grammaire - lexique manuellement avec des lexiques de taille importante. Il convient donc de chercher des moyens d'automatiser la définition de cette interface (utiliser d'autres informations que le seul nom de famille d'arbres, comme par exemple des structures de traits de type *hypertag* ou encore un langage booléen de requête permettant d'extraire les arbres à associer à un lemme donné).

Conclusion générale

Dans ce travail, nous nous sommes intéressés à la tâche du calcul sémantique pour un formalisme spécifique, à savoir les grammaires d'arbres adjoints (TAG). A la différence d'autres formalismes tels que les grammaires syntagmatiques guidées par les têtes (HPSG) ou grammaires fonctionnelles lexicales (LFG), il n'existe à l'heure actuelle aucun consensus sur une interface syntaxe / sémantique pour TAG. Dans ce contexte, nous avons étudié les approches existantes et avons sélectionné celle qui nous semblait la plus pertinente (aussi bien en terme de couverture que d'implémentabilité). En effet, l'approche sélectionnée, (Gardent et Kallmeyer, 2003), offre plusieurs avantages, plus précisément : (i) un traitement homogène des phénomènes linguistiques habituellement problématiques (quantificateurs, verbes à contrôle, etc) à partir d'une interface syntaxe / sémantique basée sur les structures de traits étiquetant les nœuds des arbres, (ii) un calcul sémantique compositionnel à base d'unification, et (iii) une représentation élégante des ambiguïtés de portée au moyen d'une sémantique plate telle que (Bos, 1995).

Dans l'optique de proposer une architecture supportant des grammaires de taille réelle, nous nous sommes également intéressés aux théories de production semi-automatique de grammaires : les *métagrammaires*. En collaboration avec Benoît Crabbé, Denys Duchier et Joseph Le Roux, nous avons proposé un formalisme métagrammatical permettant de décrire des grammaires d'arbres et également de les annoter avec des informations sémantiques. Dans le cas des TAG, les limites des approches existantes concernaient essentiellement (i) le contrôle des combinaisons des fragments d'arbres et (ii) la gestion de la portée des variables de nœud. Le premier point a été traité[195] au moyen d'un système de polarisation des nœuds entraînant les identifications de nœuds nécessaires à la combinaison des fragments. Le second point a été résolu au moyen de variables de nœuds à portée locale augmentée d'un système d'import / export inspiré de la programmation orienté objets. Cette gestion de la portée des variables offre une flexibilité importante dans la définition de la métagrammaire. En plus de la définition du formalisme, nous avons procédé à l'implantation du compilateur pour le langage correspondant. Cette implantation a été utilisée dans le cadre de (Crabbé, 2005b), où l'auteur fournit une méthodologie pour le développement d'une grammaire noyau du français, et valide la proposition par une évaluation en termes de couverture syntaxique, non-surgénération, et ambiguïté.

[195] La réponse proposée est toujours sujette à discussion, *cf* section perspectives.

Nous avons également implanté un procédé de calcul sémantique basé sur l'interface syntaxe / sémantique de (Gardent et Kallmeyer, 2003), et sur l'analyse syntaxique tabulaire du système DyALog (Villemonte de la Clergerie, 1993). Cette implantation (SemConst) constitue une première proposition d'environnement logiciel pour le calcul sémantique pour TAG.

A partir du compilateur de métagrammaire XMG, et du constructeur sémantique, nous avons développé une plate-forme (SemTAG) permettant :

- de produire des grammaires TAG de taille réelle *à portée sémantique*,
- de vérifier le calcul sémantique de (Gardent et Kallmeyer, 2003) sur des données réalistes.

Enfin, nous avons procédé à l'évaluation de notre plate-forme par rapport à une suite de tests de référence. Cet évaluation nous a permise de constituer une suite de tests syntaxico-sémantique utilisable en génération.

Plusieurs pistes de recherches émergent de ces travaux, tant dans le domaine du développement de grammaires que dans celui du calcul sémantique pour TAG.

Perspectives par rapport aux métagrammaires Nous avons vu au chapitre 4 diverses extensions du procédé de compilation de métagrammaires, l'une consistant en un système de dimensions (niveaux de description), l'autre en un système de contraintes de bonne formation des structures produites. Chacune de ces extensions s'inscrivent dans deux axes de recherches complémentaires, à savoir la définition et le traitement de descriptions métagrammaticales multilingues et multi-formalismes.

Par descriptions métagrammaticales multilingues, nous entendons des métagrammaires permettant de produire des grammaires pour différentes langues. Pour ce point, l'extension dimensionnelle est particulièrement intéressante, puisqu'elle permet d'encoder un niveau de description représentant le lien entre le lexique et les structures arborescentes. A supposer que les fragments élémentaires puissent être communs à deux langues (au moins entre deux langues proches), cette dimension pourrait permettre d'encoder les combinaisons de fragments d'arbres à associer à un mot donné dans une langue donnée. En allant plus loin, cette description pourrait permettre d'émettre des généralisations entre des lexiques de deux langues données (*e.g.*, corrélations entre verbes transitifs en français et en allemand).

Par descriptions métagrammaticales multi-formalismes, nous entendons la possibilité de définir une métagrammaire décrivant des grammaires pour différents formalismes cibles (métagrammaire unique permettant de produire à la fois une grammaire TAG, une grammaire HPSG, etc). Dans ce sens, nous pouvons citer l'approche de (Clément et Kinyon, 2003) qui proposent de produire une grammaire LFG à partir d'une métagrammaire pour TAG. Cependant dans leur proposition, les auteurs produisent une grammaire TAG via un compilateur de métagrammaire pour TAG, et *interprètent* la grammaire ainsi produite sous forme de structures LFG. Ici, nous proposons d'aller encore plus loin, en définissant une métagrammaire *commune* directement à différents formalismes grammaticaux. Dans ce contexte, la

seconde extension (contraintes) paraît adéquate. En effet les contraintes permettent d'agir sur la forme des structures grammaticales produites. Nous pourrions donc imaginer une bibliothèque de contraintes proposant un ensemble de traitements des structures, parmi lesquels l'utilisateur pourrait choisir ceux qui correspondent à son formalisme cible particulier. Notons, qu'à ce jour, nous nous sommes intéressés principalement aux grammaires TAG et aux grammaires d'interaction (Perrier, 2003), et que récemment, nous avons débuté une réflexion sur l'extension de XMG à d'autres formalismes grammaticaux[196].

Un autre thème de recherche a trait au langage de contrôle inclus dans le formalisme métagrammatical. En effet, nous avons vu que le contrôle de la combinaison des descriptions contenues dans la métagrammaire était un problème sensible dans les approches existantes. Nous avons également vu que l'approche proposée ici reposant sur une polarisation des nœuds n'était pas sans danger. Dans ce contexte, il paraît important de poursuivre les recherches d'un système de contrôle à la fois flexible, efficace et sûr. Des travaux de ce type sont menés actuellement dans le cadre des grammaires HPSG (Cohen-Sygal et Wintner, 2006b). Une collaboration sur cette thématique serait bénéfique.

Une autre piste de recherche concerne le typage des données contenues dans la métagrammaire. Comme nous l'avons vu au chapitre 6, le typage actuellement utilisé est un typage faible. Plus précisément, il n'a vocation qu'à déterminer les constantes globales[197]. Il serait intéressant de voir dans quelles mesures ce système de typage pourrait être utilisé pour :

- vérifier la cohérence des données (vérification des domaines de définition des valeurs données aux différents traits décorant les nœuds),
- permettre une économie de notation (utilisation de types élaborés permettant d'ajouter des traits sur certains nœuds en fonction des traits déjà présents),
- définir des contraintes entre structures de traits étiquetant des nœuds d'un même arbre (*e.g.*, principe des traits têtes en HPSG pour contraindre la valeur de la catégorie syntaxique des nœuds mère et tête, principe de projection des traits sémantiques tel que présenté au chapitre 6, etc).

Enfin, un autre thème de recherche intéressant concerne le lien entre la grammaire et le lexique. Nous avons vu au chapitre 6 que le compilateur de métagrammaire produisait en réalité des arbres non-ancrés (*schèmes*). Ces schèmes se voient associé une forme fléchie à partir des informations associées aux lemmes (lexique syntaxique). Actuellement l'ancrage passe soit par l'utilisation d'une famille d'arbre, soit par l'unification entre structures de traits associées aux lemmes et structures de traits globales associées aux schèmes (*e.g.*, *Hypertags* (Kinyon, 2000)). Un autre possibilité intéressante pour réaliser cet ancrage, et

[196] Dans ce contexte, l'équipe Langue Et Dialogue est impliquée avec d'autres partenaires dans l'Action de Recherche Coopérative INRIA Mosaïque sur les formalismes syntaxiques de haut niveau, voir `http://mosaique.labri.fr`.

[197] Ce qui permet d'éviter d'adopter une convention à la Prolog où les variables commencent obligatoirement par une majuscule.

notamment pour rendre compte des exceptions (*e.g.*, les verbes non passivables), serait d'utiliser, un langage booléen pour extraire les schèmes pertinents pour un mot donné du lexique. Ce langage booléen pourrait servir à déterminer quels traits de l'hypertags doivent être associés à quelle valeur (par exemple "passive = + AND sujet = hum"). Chaque expression (associée à un mot du lexique) serait appliquée à l'*hypertag* de chaque arbre élémentaire, et toute évaluation de cette expression retournant la valeur booléenne *vrai* entraînerait la sélection (et l'ancrage) de l'arbre en question. Il faudrait cependant porter attention à la complexité d'un tel langage en terme de temps d'extraction des schèmes correspondants (rappelons qu'une grammaire de taille réelle contient plusieurs milliers de schèmes). Peut-être une restriction appropriée du langage de requête devrait être recherchée.

Perspectives par rapport au calcul sémantique pour TAG Au niveau du calcul sémantique également nous avons des pistes de poursuites de recherches. Actuellement, nous utilisons le système DyALog pour l'analyse syntaxique en « boîte noire ». En d'autres termes, nous utilisons la forêt de dérivation produite à partir de l'analyse syntaxique pour effectuer le calcul sémantique *a posteriori*. Il doit être possible d'étendre le système DyALog pour qu'il réalise le calcul sémantique directement lors de la construction de la forêt de dérivation. Plus précisément, ce système pourrait en cours d'analyse, récupérer les formules sémantiques associées aux arbres impliqués dans une dérivation, et réaliser l'unification des indices sémantiques en fonction du résultat de l'unification des structures de traits décorant les nœuds. Il convient cependant d'étudier l'impact que l'unification des index sémantiques aurait sur le partage de calculs de ce système tabulaire (impact sur les performances notamment).

Une autre piste de recherche concerne l'étude d'autres calculs sémantiques pour les TAG, tels que le λ-calcul ou la sémantique à Glue, notamment dans un contexte de métagrammaire (ce qui présuppose que ces calculs font intervenir de l'information issue de la grammaire, ce qui est en particulier le cas des approches fondées sur l'arbre dérivé). Cette étude viserait à déterminer quels calculs permettent à la fois de construire une représentation correcte du sens pour un maximum de phénomènes linguistiques, et également dans quelles mesures l'information issue de la grammaire nécessaire à ce calcul est intégrable dans un système métagrammatical tel que XMG (définition de nouvelles dimensions sémantiques ?).

Un autre extension enfin concerne l'application du calcul sémantique pour des tâches de raisonnement, c'est-à-dire les tâches où intervient l'*inférence*. En effet, Nous avons vu comment construire une représentation sous-spécifiée du sens d'énoncés. Il serait intéressant de compléter la chaîne de traitement sémantique ainsi constituée en explorant les possibilités d'application, c'est-à-dire les contextes applicatifs dans lesquels ces représentations sont manipulées, par exemple pour le traitement du discours (résolutions d'implications textuelles), ou encore dans le cadre du développement de systèmes de Questions / Réponses (détermination des énoncés contenant une réponse à une question donnée).

Annexes

Annexe A

Informations complémentaires sur les différents outils

A.1 Compilateur de métagrammaire XMG

Le compilateur XMG a été développé en langage Oz-Mozart (Oz-Mozart, 2005), et est disponible sous licence CeCILL[198] pour les environnements Linux-Unix, MacOS et MS Windows.

Ce compilateur prend en entrée une description métagrammaticale sous forme de classes contenant soit une description syntaxique (ou sémantique), soit une conjonction / disjonction d'appels de classes. De plus, les classes peuvent être définies au sein d'une relation d'héritage afin de faciliter la gestion des espaces de noms de variables. La syntaxe concrète du langage métagrammatical est fournie en annexe C. En sortie le compilateur produit une grammaire au format XML (DTD fournie en annexe D) ou via une interface graphique Qtk (voir figure A.1).

La documentation de ce compilateur est disponible en ligne sous forme d'un *WIKI* à l'adresse suivante :

`http://wiki.loria.fr/wiki/XMG/Documentation`

Le compilateur peut être téléchargé soit via un dépôt Subversion, sous formes d'archives contenant les sources, ou encore sous forme de paquet Oz à l'adresse suivante :

`http://sourcesup.cru.fr/xmg`

Il existe également des outils permettant :

- de faire une recherche parmi les arbres d'une grammaire (module VIEWTAG),
- d'éliminer les doublons éventuellement contenus dans la grammaire si la métagrammaire est redondante (module CHECKTAG),

[198] `http://www.cecill.info`

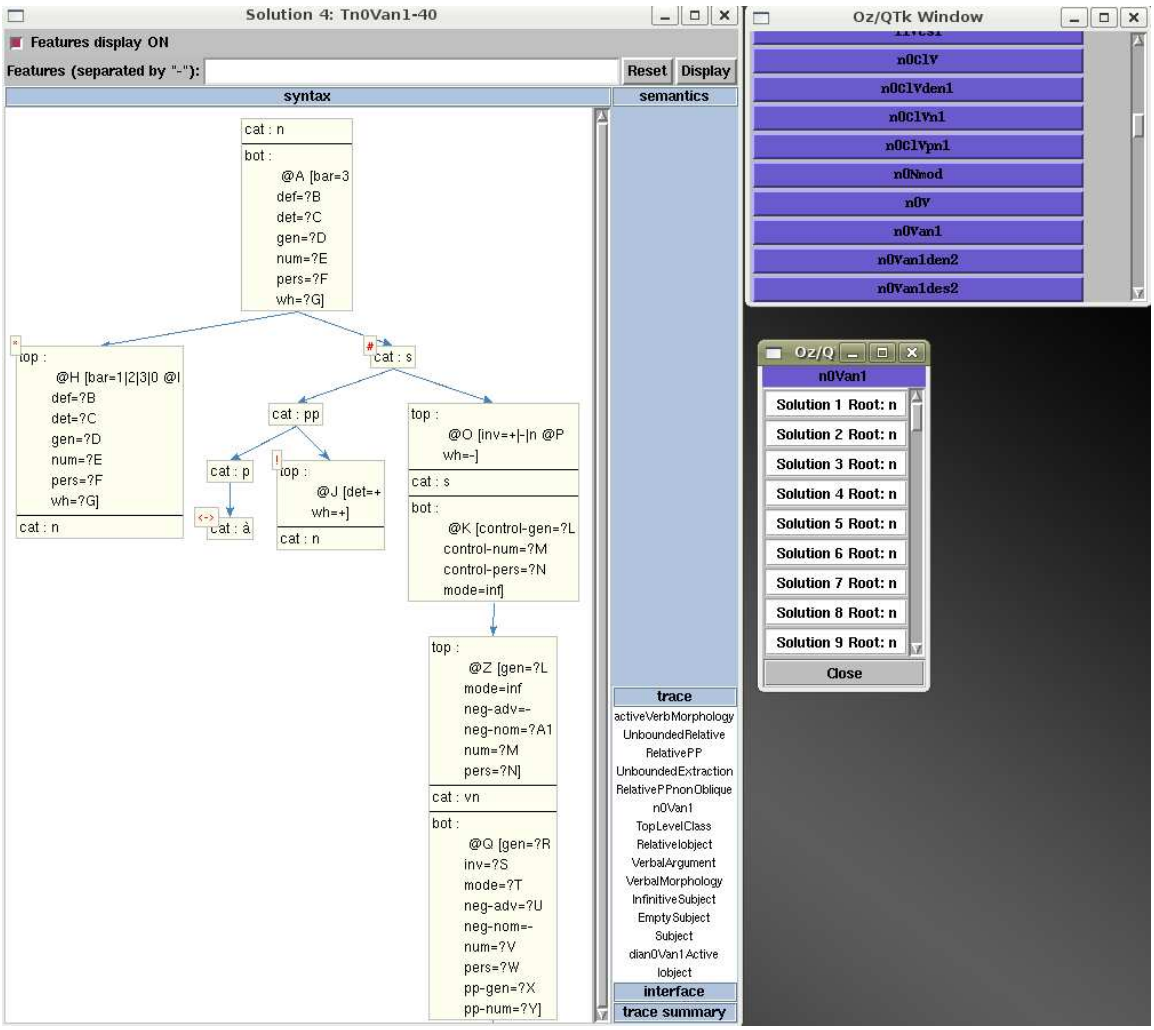


FIG. A.1 – Interface graphique du compilateur XMG

– de réaliser un ancrage des arbres de la grammaire à partir d'un lexique de lemmes (module SELECTOR),
– de réaliser un export des hiérarchies contenues dans la métagrammaire au format pdf (contribution d'Ulrike Fleury).

Ces outils sont regroupés dans le paquet XMG-TOOLS disponible dans le dépôt Subversion mentionné ci-dessus.

Ce travail a été réalisé en collaboration étroite avec Benoît Crabbé, Denys Duchier et Joseph Le Roux.

A.2 Analyseur sémantique SemConst

Le logiciel SemConst, introduit au chapitre 7, a été développé principalement en langage Oz-Mozart (squelette, interface graphique et module de traitement de la forêt de dérivation), et en langage Prolog (module d'unification sémantique). Il fait également appel à un ensemble de scripts Perl et XSLT pour les conversions de données dans différents formats. Il utilise également le système DyALog pour générer un analyseur syntaxique tabulaire pour la grammaire TAG d'entrée[199].

En entrée, l'utilisateur fournit un corpus et une métagrammaire (au format xmg), accompagnée des lexiques correspondant (lexique syntaxique et lexique morphologique). Le SemConst calcule alors les représentations sémantiques sous-spécifiées pour chaque phrase du corpus d'entrée. De plus, il est possible de produire un fichier de statistiques sur la couverture syntaxique et sémantique (dans un format pdf compilé par LaTeX).
NB Ce module s'intègre dans une architecture plus importante, intégrant le compilateur de métagrammaire xmg et le réalisateur de surface GenI (Gardent et Kow, 2005).
L'architecture de ce logiciel est représentée sur la figure A.2.
La documentation de ce logiciel est disponible en ligne sous forme de *WIKI* :

`http://wiki.loria.fr/wiki/SemConst/Documentation`

Le logiciel peut être téléchargé librement sous licence CeCILL soit sous forme de sources via un dépôt *darcs*[200], soit sous forme d'archive *tar* disponible en ligne à l'adresse :

`http://trac.loria.fr/~semconst`

Ce travail a été réalisé en collaboration étroite avec Claire Gardent (module Prolog d'unification sémantique notamment) et Eric Kow (outils de conversion de lexiques). Nous remercions Eric De La Clergerie pour son aide dans l'utilisation du système DyALog.

A.3 Outils existants

Enfin, voici quelques informations sur les outils existants dans la communauté tag.

[199] Voir `http://trac.loria.fr/~semconst` pour la liste des dépendances du logiciel SemConst.
[200] `http://www.darcs.net`

URS: underspecified Semantic Representations

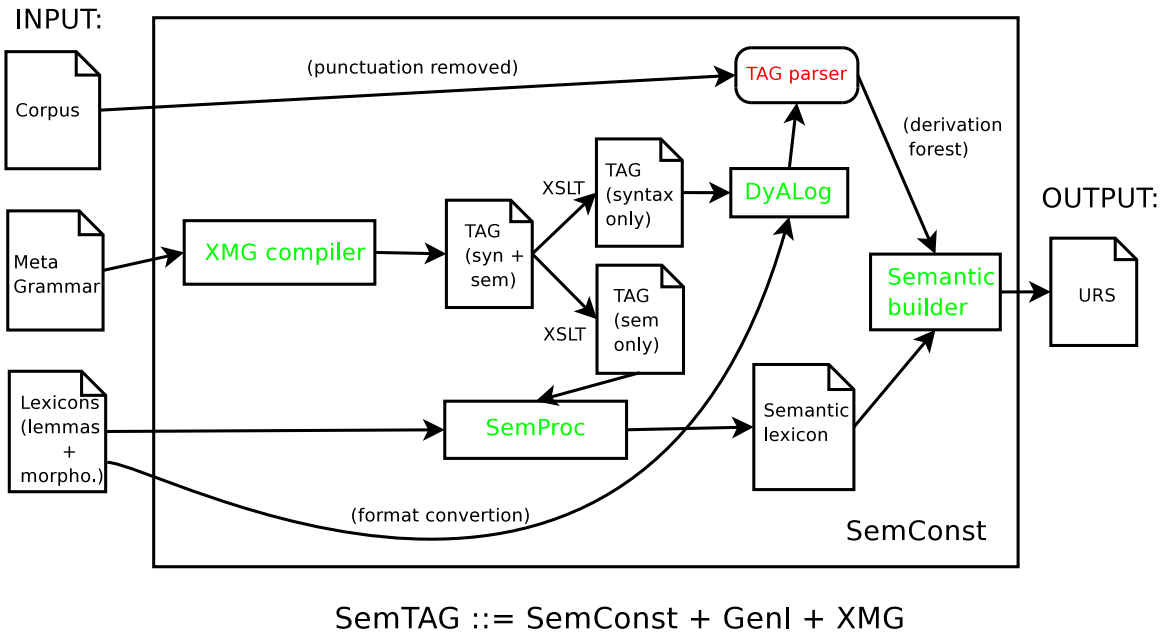


FIG. A.2 – Architecture du logiciel SemConst.

A.3.1 Compilateurs de métagrammaire

Compilateur de Paris 7 Le compilateur de métagrammaire de (Candito, 1999) a été développé en langage Common-LISP. Cet outil a servi aux travaux de thèse de (Barrier, 2006).

Compilateur de l'Université de Pennsylvanie Le compilateur de métagrammaire de (Xia, 2001), appelé LEXORG, a été développé en Prolog et est équipé d'une interface X-window.

Compilateur MGC Le compilateur de métagrammaire de (Gaiffe et al., 2002), appelé MGC, a été développé en Java. Il a été utilisé dans le cadre d'expérimentations sur la production de grammaires lexicales fonctionnelles à partir d'une métagrammaire (Clément et Kinyon, 2003). Cet outil n'est plus maintenu, cependant une version est encore disponible en ligne à l'adresse :

`http://led.loria.fr/outils/mgc/mgc.html`

Compilateur du projet INRIA ATOLL Le compilateur de métagrammaire de (Thomasset et Villemonte de la Clergerie, 2005) a été développé en langage DYALOG. Il est livré avec un ensemble d'outils (visualisateurs, etc) facilitant son utilisation. Il est disponible sous licence GPL à l'adresse :

`http://mgkit.gforge.inria.fr/`

A.3.2 Analyseurs syntaxiques pour TAG

Analyseur XTAG La référence en terme d'environnement pour l'analyse en TAG correspond au projet XTAG de l'université de Pennsylvanie (XTAG-Research-Group, 2001). Cet environnement inclut un analyseur syntaxique développé en langage C, et disponible sous licence GPL à l'adresse suivante :

`http://www.cis.upenn.edu/~xtag/`

Loria LTAG Parser version 2 (LLP2) L'analyseur LLP2 a été présenté en section 7.2.2 page 193. Rappelons que cet analyseur a été développé en Java et est disponible librement via la page :

`http://www.loria.fr/~azim/LLP2/help/fr/`

Système DyALog Le système DyALog (Villemonte de la Clergerie, 1993) est un compilateur tabulaire de programmes logiques. L'une de ses applications est la compilation d'analyseurs syntaxiques pour grammaires TAG. Ce système a été introduit en section 7.3.4 page 205. Pour plus d'informations, voir la page :

`http://dyalog.gforge.inria.fr/`

Annexe B

Syntaxe abstraite du formalisme XMG

Nous récapitulons ici les différents opérateurs du langage abstrait du formalisme métagrammatical XMG.

B.1 Abstraction (classe)

$$Classe \quad ::= \quad Nom \rightarrow \{\, Contenu \,\}$$

B.2 Contenu d'une classe (dimensions et combinaisons)

$$\begin{aligned} Contenu \quad ::= \quad & Dimension \mathrel{+}= Description \mid Nom \mid \\ & Contenu \vee Contenu \mid Contenu \wedge Contenu \end{aligned}$$

B.2.1 Dimension syntaxique (description d'arbres)

$$\begin{aligned} Description \quad ::= \quad & X \rightarrow Y \mid X \rightarrow^{+} Y \mid X \rightarrow^{*} Y \mid \\ & X \prec Y \mid X \prec^{+} Y \mid X \prec^{*} Y \mid \\ & X = Y \mid X[f{:}E] \mid X(p{:}E) \mid \\ & Description \vee Description \mid Description \wedge Description \end{aligned}$$

B.2.2 Dimension sémantique (formules de sémantique plate)

$$Description \; ::= \; \ell{:}p(E_1, ..., E_n) \mid \neg\ell{:}p(E_1, ..., E_n) \mid E_i \ll E_j$$

B.3 Héritage multiple (hiérarchie de classes)

$$\begin{aligned} \mathit{Classe} \quad ::= \quad & \mathit{Nom} \quad \angle \quad C_1 \wedge C_2 \wedge \ldots C_n \\ & \rightarrow \{\, \mathit{Contenu} \,\} \end{aligned}$$

B.4 Gestion des espaces de noms de variables (export)

$$\mathit{Classe} \quad ::= \quad \langle E_1, \ldots, E_n \rangle \Leftarrow \mathit{Nom} \quad \rightarrow \quad \{\, \mathit{Contenu} \,\}$$

B.5 Gestion des espaces de noms de variables (import paramétré)

$$\begin{aligned} \mathit{Classe} \quad ::= \quad & \langle E_1, \ldots, E_n \rangle \Leftarrow \mathit{NomB} \quad \angle \quad \mathit{NomA}[I_1, \ldots, I_m] \\ & \rightarrow \quad \{\, \mathit{Contenu} \,\} \end{aligned}$$

B.6 Gestion des espaces de noms de variables (import avec renommage)

$$\begin{aligned} \mathit{Classe} \quad ::= \quad & \langle E_1, \ldots, E_n \rangle \Leftarrow \mathit{NomB} \quad \angle \quad \mathit{NomA}[I_1 = J_1, \ldots, I_m = J_m] \\ & \rightarrow \quad \{\, \mathit{Contenu} \,\} \end{aligned}$$

B.7 Evaluation de classes (accumulation de descriptions et résolution)

$$\mathit{Evaluation} \quad ::= \quad \mathit{evalue} \quad \mathit{Nom}$$

Annexe C

Syntaxe concrète du formalisme XMG

Une métagrammaire est définie (dans cet ordre) par :
- une liste de principes,
- un ensemble de définitions (types, properties, features)
- une liste d'items (spécifications de classes ou évaluations)

```
principles(P) definitions(D) items(I)
```

C.1 Principes

Les principes sont utilisés pour vérifier la bonne formation des structures produites. Ils sont optionnels.

```
use <Principe> with (  propriétés ) dims ( dimensions )
```

où `propriétés` est une liste de propriétés définies plus bas, `dimensions` vaut `<syn>` ou `<sem>`.

C.2 Définitions

C.2.1 Types

Les types de données peuvent être définis par une énumération, une définition structurée ou une définition « anonyme » (via le symbole `!` créant un nouveau nom de type).

```
type ID = {id1,...,idn}
type ID = [int .. int]
type ID = [ id1 : T1 ,
            id2 : T2 ,
            ...
            idn : Tn ]
type ID !
```

C.2.2 Propriétés

Les propriétés sont typées, et peuvent utiliser des abréviations (*i.e.*, `+` pour la valeur booléenne `true`).

```
property ID : Type
property ID : Type { val = abbrev }
```

C.2.3 Traits

Les traits associés aux nœuds sont typés.

```
feature ID : Type
```

C.3 Items

C.3.1 Ensembles d'exclusion mutuelle

```
mutex ID -- déclare un ensemble d'exclusion mutuelle
mutex ID += Class -- ajoute une classe à cet ensemble
```

C.3.2 Définition de classe paramétrée

```
class ID[X1,...,Xn]
```

C.3.2.1 Imports

```
import C1
| import C1 as [X1,...,Xn]         -- import paramétré
| import C1 as [...,Xi=Yi,..]      -- import avec renommage
```

C.3.2.2 Exports

```
export
?ID                                -- export d'une variable
?ID1 = ?ID2                        -- export avec renommage
```

C.3.2.3 Déclaration de variables

```
declare ?ID1 ...
```

C.3.2.4 Contenu de classes

```
{ STMT }
```

où `STMT` représente un *statement* défini comme suit :

```
STMT ::=  { STMT }
     |  S1;S2                  -- conjonction
     |  S1|S2                  -- disjonction
     |  S*=E                   -- interface
     |  E=E                    -- equation (unification)
     |  <ID> { ... }           -- dimension (ID = syn ou ID = sem)
     |  ID[P1,...,Pn]          -- instanciation de classe (crochets obligatoires)
```

où E est une expression :

```
E  ::=  INT                    -- entier
     |  STRING                 -- chaîne de caractères
     |  BOOL                   -- booléen
     |  ID                     -- identificateur (atome)
     |  ?ID                    -- variable
     |  !ID                    -- skolem constant
     |  FEATS                  -- structure de traits
     |  E(E,...,E)             -- application
     |  E1.E2                  -- opérateur dot
     |  ID[P1,...,PN]          -- instantiation de classe
     |  (E|..|E)               -- disjonction
```

C.3.2.5 Dimension syntaxique

```
<syn> { SYNLIT }

SYNLIT  ::=  SYNLIT | SYNLIT                -- disjonction
          |  SYNLIT ; SYNLIT                -- conjonction
          |  (SYNLIT)
          |  E ->  E                        -- dominance immédiate
          |  E ->+ E                        -- dominance stricte
          |  E ->* E                        -- dominance "large"
          |  E >>  E                        -- précédence immédiate
          |  E >>+ E                        -- précédence stricte
          |  E >>* E                        -- précédence "large"
          |  E = E                          -- équation de noeuds
          |  NODE                           -- définition de noeud


NODE  ::=  node (VAR|PROPS|FEATS)* CHILDREN
VAR   ::=  ?ID
PROPS ::=  [?X =] (PROP,...,PROP)
PROP  ::=  ABBREV | FEAT

FEAT  ::=  ID=E
```

```
FEATS ::=  [?X =] [FEAT,...,FEAT]

CHILDREN  ::=  { CHUNK ;;; ... ;;; CHUNKn }
CHUNK     ::=  CHILD SIBLING
CHILD     ::=     NODE
    | ... NODE

SIBLING   ::=     CHILD
    | ,,, CHILD

CONSTR ::= FEAT
```

C.3.2.6 Dimension sémantique

```
<sem> { SEMLIT }

SEMLIT  ::=
             SEMLIT ; SEMLIT           -- conjonction
          |  SEMLIT | SEMLIT           -- disjonction
          |  (SEMLIT)                  -- groupement
          |  E : SEMPRED               -- étiquette de formule sémantique
          |  SEMPRED                   -- prédicat
          |  E <* E                    -- contrainte de dominance

SEMPRED ::=  NEG SEMPRED               -- négation d'une formule sémantique
          |  E

NEG     ::= ~                          -- négation
```

C.4 Evaluation de classe

```
value ID
```

Remarque Il est possible d'inclure des fichiers via la commande :

```
include myFile.mg
```

Pour plus d'informations, la documentation est disponible à l'adresse

`http://wiki.loria.fr/wiki/XMG/Documentation`

Annexe D

DTD du format XML pour les grammaires produites par XMG

```
<!-- XML Document Type Definition

     Authors:
       Yannick Parmentier <Yannick.Parmentier@loria.fr>
       Joseph Le Roux <Joseph.Leroux@loria.fr>

     Description:
       XML Encoding of semi-automatically generated Tree Adjoining Grammars

     Acknowledgment:
       This DTD derives from one written by Denys Duchier <Duchier@loria.fr>

     Version: 1.1

     Typical usage:

            <?xml version="1.0" encoding="iso-8859-1"?>
            <!DOCTYPE grammar SYSTEM "xmg-tag.dtd,xml">
            <grammar>
            ...
            </grammar>

     Notes:
       A compiled Tree Adjoining Grammar consists of a list of entries, each
       representing a (not yet anchored) tree of the grammar.
       More precisely, an entry contains the following data :

       - a family name determining to which tree-family it belongs,
```

```
       - a trace containing the names of the meta-grammar classes that were
         combined,

       - a syntactic description giving the structure of the tree,

       - a semantic description expressed in flat semantics (set of literals),

       - an interface consisting of an attributes-values matrix whose features
         are shared with features in the tree (used for anchoring).

-->

<!-- Compiled grammar -->

<!ELEMENT  grammar  (entry* | subgrammar*)>

<!-- Eventual subgrammar (anchoring) -->

<!ELEMENT  subgrammar  (entry*)>
<!ATTLIST  subgrammar  id  CDATA #REQUIRED>

<!-- Entries -->

<!ELEMENT  entry  (family,trace,tree,semantics,interface)>
<!ATTLIST  entry  name  ID  #REQUIRED>

<!-- Values (tree families) -->

<!ELEMENT  family  (#PCDATA)>

<!-- Traces of compilation -->

<!ELEMENT  trace  (class*)>

<!ELEMENT  class  (#PCDATA)>

<!-- syntactic descriptions (the templates) -->

<!ELEMENT  tree  (node)>
<!ATTLIST  tree  id  CDATA  #REQUIRED>

<!ELEMENT  node  (narg? , node*)>
<!ATTLIST  node  type ( nadj | std | subst | lex | anchor | coanchor | foot ) #REQUIRED>
<!ATTLIST  node  name  CDATA #IMPLIED>
```

```
<!ELEMENT  narg (fs)>

<!ELEMENT  fs  (f*)>
<!ATTLIST  fs  coref CDATA #IMPLIED>

<!ELEMENT  f  (sym | vAlt | fs)>
<!ATTLIST  f  name  CDATA #REQUIRED>

<!ELEMENT  vAlt  (sym+)>
<!ATTLIST  vAlt  coref CDATA "">

<!ELEMENT  sym EMPTY>
<!ATTLIST  sym
           value   CDATA #IMPLIED
           varname CDATA #IMPLIED>

<!-- semantic representations -->

<!ELEMENT  semantics  (literal | sym | semdominance)*>

<!ELEMENT  literal   (label? , predicate, arg*)>
<!ATTLIST  literal   negated  CDATA  "no">

<!ELEMENT  label  (sym)>

<!ELEMENT  predicate  (sym)>

<!ELEMENT  arg  (sym | fs)>

<!ELEMENT  semdominance  (arg+)>
<!ATTLIST  semdominance  op   CDATA   "ge">

<!-- Interfaces -->

<!ELEMENT interface  (fs?)>
```

Bibliographie

(Abeillé et al., 1999) A. Abeillé, M.H. Candito, et A. Kinyon. 1999. FTAG : current status and parsing scheme. Dans *VEXTAL, Venice, Italy*.

(Abeillé, 1990) Anne Abeillé. 1990. Lexical and syntactic rules in a tree adjoining grammar. Dans *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, pages 292–298, Morristown, NJ, USA. Association for Computational Linguistics.

(Abeillé, 1993) A. Abeillé. 1993. *Les Nouvelles Syntaxes - Grammaires d'unification et analyse du français*. Editions Armand Colin, Linguistique.

(Abeillé, 1998) . Abeillé. 1998. Verbes à montée et auxiliaires dans une grammaire d'arbres adjoints. Dans *Linx, ISSN 0246-8743, N° 39, numéro spécial dédié aux Modèles linguistiques : convergences, divergences*, pages 119–158.

(Abeillé, 2002) A. Abeillé. 2002. Une grammaire électronique du français. CNRS Editions, Paris.

(Ait-Kaci, 1991) H. Ait-Kaci. 1991. Warren's abstract machine : A tutorial reconstruction. Dans K. Furukawa, éditeur, *Actes de la Huitième Conférence Internationale en Programmation Logique*. MIT Press, Cambridge, MA.

(Alonso et al., 2000) Miguel A. Alonso, Djamé Seddah, et Éric Villemonte de la Clergerie. 2000. Practical aspects in compiling tabular TAG parsers. Dans *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 27–32, Université Paris 7, Jussieu, Paris, France, Mai.

(Alonso et al., 2004) Miguel A. Alonso, Éric Villemonte de la Clergerie, Vítor J. Diaz, et Manuel Vilares, 2004. *Relating Tabular Parsing Algorithms for LIG and TAG*, volume 23 de *Text, Speech and Language Technology*, pages 157–184. Kluwer Academic Publishers. revised notes of a paper for IWPT2000.

(Alshawi et Crouch, 1992) Hiyan Alshawi et Richard S. Crouch. 1992. Monotonic semantic interpretation. Dans *Meeting of the Association for Computational Linguistics*, pages 32–39.

(Bar-Haim et al., 2006) Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, et Idan Szpektor. 2006. The Second PASCAL Recognising Textual Entailment Challenge. Dans *Proc. of the Pascal RTE Challenge Workshop*.

(Barrier, 2006) S. Barrier. 2006. *Une métagrammaire pour les noms prédicatifs du français : développement et expérimentations pour les grammaires TAG*. Thèse de Doctorat, Université Paris 7.

(Barthélemy et Villemonte de la Clergerie, 1998) François Barthélemy et Éric Villemonte de la Clergerie. 1998. Information flow in tabular interpretations for generalized push-down automata. *Theoretical Computer Science*, 199 :167–198.

(Barwise et Cooper, 1981) K. Jon Barwise et Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy*4() :159–219.

(Becker, 1994) T. Becker. 1994. A new automaton model for TAGs : 2-SA. *Computational Intelligence*, 10(4) :422–431, November.

(Billot et Lang, 1989) Sylvie Billot et Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. Dans *ACL*, pages 143–151.

(Blackburn et al., 2006) P. Blackburn, J. Bos, et K. Striegnitz. 2006. Learn Prolog Now !, June. Texts in Computer Sciences, Volume 7, College Publications.

(Blackburn et Bos, 2005) P. Blackburn et J. Bos. 2005. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI.

(Bonhomme et Lopez, 2000) P. Bonhomme et P. Lopez. 2000. Resources for Lexicalized Tree Adjoining Grammars and XML encoding : TagML. Dans *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC, 2000), Athens*.

(Bos, 1995) J. Bos. 1995. Predicate Logic Unplugged. Dans *Proceedings of the tenth Amsterdam Colloquium, Amsterdam*.

(Büttcher, 2002) S. Büttcher. 2002. A Java implementation of Warren's Abstract Machine. Available at `http://stefan.buettcher.org/cs/wam/wam.pdf`.

(Candito et Kahane, 1998) M. Candito et S. Kahane. 1998. Can the tag derivation tree represent a semantic graph ? an answer in the light of the meaning-text theory. Dans *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks*.

(Candito, 1996) M.H. Candito. 1996. A principle-based hierarchical representation of LTAGs. Dans *Actes de COLING'96, Kopenhagen*.

(Candito, 1999) M.H. Candito. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au français et à l'italien*. Thèse de Doctorat, Université Paris 7.

(Carpenter, 1992) Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, England.

(Chomsky, 1956) N. Chomsky. 1956. Three models for the description of language. *Information Theory, IEEE Transactions on*2() :113–124.

(Church, 1940) A. Church. 1940. A formulation of a simple theory of types. *Journal of Symbolic Logic*, 5 :56–68.

(Clément et Kinyon, 2003) L. Clément et A. Kinyon. 2003. Generating LFGs with a MetaGrammar. Dans *Proceedings of the 8th International Lexical Functional Grammar Conference, Saratoga Springs, NY*.

(Cohen-Sygal et Wintner, 2006a) Y. Cohen-Sygal et S. Wintner. 2006a. Influence des couleurs sur la description métagrammaticale (communication privée). Juillet.

(Cohen-Sygal et Wintner, 2006b) Y. Cohen-Sygal et S. Wintner. 2006b. Partially Specified Signatures : A Vehicle for Grammar Modularity. Dans *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia*, July.

(Cohen-Sygal et Wintner, 2007) Y. Cohen-Sygal et S. Wintner. 2007. The Non-Associativity of Polarized Tree-Based Grammars. Dans *Proceedings of the Eighth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2007), Mexico City, Mexico*.

(Copestake et al., 2005) Ann Copestake, Dan Flickinger, Carl Pollard, et Ivan A. Sag. 2005. Minimal Recursion Semantics : An introduction. *Research on Language and Computation*, 3.4 :281–332.

(Crabbé et Duchier, 2004) B. Crabbé et D. Duchier. 2004. Metagrammar Redux. Dans *Proceedings of Constraint Solving in Language Processing (CSLP 2004), Copenhagen*.

(Crabbé, 2005a) B. Crabbé. 2005a. Grammatical development with XMG. Proceedings of the 5th International Conference on the Logical Aspects of Computational Linguistics (LACL05).

(Crabbé, 2005b) B. Crabbé. 2005b. *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. Thèse de Doctorat, Université Nancy 2.

(Dalrymple et al., 1995) M. Dalrymple, J. Lamping, F. Pereira, et V. Saraswat. 1995. Linear logic for meaning assembly. Proceedings of Computational Logic for Natural Language Processing, Edinburgh (revised version of 2002 available at `http://citeseer.ist.psu.edu/dalrymple02linear.html`).

(de Groote, 2001) Philippe de Groote. 2001. Towards abstract categorial grammars. Dans *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

(de Groote, 2002) Philippe de Groote. 2002. Tree-adjoining grammars as abstract categorial grammars. Dans *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia.

(Duchier et al., 2005) D. Duchier, J. Le Roux, et Y. Parmentier. 2005. XMG : Un Compilateur de Métagrammaire Extensible. Dans *Actes de La 12ème édition de la conférence sur le TALN (TALN 2005), Dourdan, France*.

(Duchier et Gardent, 1999) D. Duchier et C. Gardent. 1999. A constraint based treatment of descriptions. Dans *Proceedings of the 3rd International Workshop on Computational Semantics (IWCS), Tilburg*.

(Duchier et Niehren, 2000) D. Duchier et J. Niehren. 2000. Dominance constraints with set operators. Dans *Proceedings of Computational Logic 2000*, volume 1861 de *Lecture Notes in Computer Science*, pages 326–341. Springer.

(Duchier, 1999) D. Duchier. 1999. Set constraints in computational linguistics - solving tree descriptions. Dans *Workshop on Declarative Programming with Sets (DPS'99), Paris, pp. 91 - 98*.

(Duchier, 2000) D. Duchier. 2000. Constraint programming for natural language processing. Lecture Notes, ESSLLI 2000. Available at `http://www.ps.uni-sb.de/Papers/abstracts/duchier-esslli2000.html`.

(Duchier, 2003) D. Duchier. 2003. A metagrammatical formalism for lexicalized tags. Dans *Proceedings of the Lorraine/Saarland workshop on Prospects and Advances in the Syntax/Semantics Interface*.

(Earley, 1970) Jay Earley. 1970. An efficient context-free parsing algorithm13() :94–102, Février.

(Ebert, 2005) C. Ebert. 2005. *Formal Investigations of Underspecified Representations*. Thèse de Doctorat, Department of Computer Science, King's College London.

(Egg et al., 2001) Markus Egg, Alexander Koller, et Joachim Niehren. 2001. The Constraint Language for Lambda Structures. *Journal of Logic, Language and Information*10() :457–485.

(Erbach et Uszkoreit, 1990) Gregor Erbach et Hans Uszkoreit. 1990. Grammar engineering : Problems and prospects – report on the saarbrücken grammar engineering workshop. Rapport Technique 1, Saarbrücken, Germany.

(Evans et al., 1995) R. Evans, G. Gazdar, et D. Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. Dans *Proceedings of the 33rd Annual Meeting of the ACL, 77-84*.

(Evans et Gazdar, 1996) R. Evans et G. Gazdar. 1996. DATR : a language for lexical knowledge representation. Computational Linguistics, vol.22.2, 167–216.

(Frank et Van Genabith, 2001) A. Frank et J. Van Genabith. 2001. GlueTag - Linear Logic based Semantics for LTAG – and what it teaches us about LFG and LTAG –. Dans *Proceedings of LFG01, Hong Kong*.

(Frank, 2002) Robert Frank. 2002. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Boston.

(Fuchss et al., 2004) Ruth Fuchss, Alexander Koller, Joachim Niehren, et Stefan Thater. 2004. Minimal Recursion Semantics as Dominance Constraints : Translation, Evaluation, and Analysis. Dans *Proceedings of the 42nd ACL*, Barcelona.

(Gaiffe et al., 2002) B. Gaiffe, B. Crabbé, et A. Roussanaly. 2002. A new metagrammar compiler. Dans *Proceedings of the TAG+6 workshop on TAG and related formalisms, Venice, Italy*.

(Gamut, 1991) L.T.F. Gamut. 1991. *Logic, Language, and Meaning, (Volume I : Introduction to Logic, Volume II : Intensional Logic and Logical Grammar)*. University of Chicago Press.

(Gardent et Kallmeyer, 2003) C. Gardent et L. Kallmeyer. 2003. Semantic construction in FTAG. Dans *Proceedings of the European chapter of the Association for Computational Linguistics (EACL'03), Budapest*.

(Gardent et Kow, 2005) C. Gardent et E. Kow. 2005. Generating and selecting grammatical paraphrases. *European conference on Natural Language Generation, Aberdeen, Scotland*.

(Gardent et Parmentier, 2006) C. Gardent et Y. Parmentier. 2006. Coreference Handling in XMG. Dans *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia*, July.

(Gardent, 2006) C. Gardent. 2006. Intégration d'une dimension sémantique dans les grammaires d'arbres adjoints. Dans *Actes de La 13ème édition de la conférence sur le TALN (TALN 2006)*.

(Girard, 1987) Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50 :1–102.

(Gregory, 2000) H. Gregory. 2000. Montague semantics – course materials. Available at `http://wwwuser.gwdg.de/~hgregor1/docs/montague2e.pdf`.

(Groenendijk et Stokhof, 1991) Jeroen Groenendijk et Martin Stokhof. 1991. Two theories of dynamic semantics. Dans *JELIA '90 : Proceedings of the European workshop on Logics in AI*, pages 55–64, New York, NY, USA. Springer-Verlag New York, Inc.

(Hobbs, 1983) Jerry R. Hobbs. 1983. An improper treatment of quantification in ordinary english. Dans *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 57–63, Morristown, NJ, USA. Association for Computational Linguistics.

(hye Han et al., 2000) Chung hye Han, Juntae Yoon, Nari Kim, et Martha Palmer. 2000. A feature-based lexicalized tree adjoining grammar for korean. Rapport Technique IRCS-00-04, IRCS, University of Pennsylvania.

(Ide et Véronis, 1994) Nancy Ide et Jean Véronis. 1994. MULTEXT : Multilingual text tools and corpora. Dans *Proceedings of the 15th. International Conference on Computational Linguistics (*Coling 94*)*, volume I, pages 588–592, Kyoto, Japan.

(Joshi et al., 1975) A. Joshi, L. Levy, et M. Takahashi. 1975. Tree adjunct grammars. Journal of Comput. Syst. Sci., Vol. 10-1.

(Joshi et Schabes, 1997) A. Joshi et Y. Schabes. 1997. Tree-adjoining grammars. Dans G. Rozenberg et A. Salomaa, éditeurs, *Handbook of Formal Languages*, volume 3, pages 69 – 124. Springer, Berlin, New York.

(Joshi et Vijay-Shanker, 1999) A. Joshi et K. Vijay-Shanker. 1999. Compositional Semantics with Lexicalized Tree Adjoining Grammar (LTAG) : How Much Underspecification is Necessary ? Dans *Proceedings of the Third International Workshop on Computational Semantics, IWCS-03, Tilburg, The Netherlands*.

(Kallmeyer et Joshi, 1999) L. Kallmeyer et A. Joshi. 1999. Factoring Predicate Argument and Scope Semantics : Underspecified Semantics with LTAG. Dans *Proceedings of 12th Amsterdam Colloquium*, pages 169–174, Dec.

(Kallmeyer et Joshi, 2003) L. Kallmeyer et A. Joshi. 2003. Factoring Predicate Argument and Scope Semantics : Underspecified Semantics with LTAG. Dans *Research on Language and Computation*, volume 1 :1-2, pages 3–58.

(Kallmeyer et Romero, 2004) L. Kallmeyer et M. Romero. 2004. LTAG Semantics with Semantic Unification. Dans *Proceedings of TAG+7. Vancouver*, pages 155–162, May.

(Kallmeyer, 2002a) L. Kallmeyer. 2002a. Enriching the TAG Derivation Tree for Semantics. Dans *Proceedings of KONVENS 2002. 6. Konferenz zur Verarbeitung natürlicher Sprache*, pages 67–74, Sep.

(Kallmeyer, 2002b) L. Kallmeyer. 2002b. Using an Enriched TAG Derivation Structure as Basis for Semantics. Dans *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 127–136. Università di Venezia.

(Kamp, 1981) Hans Kamp. 1981. A theory of truth and semantic representation. Dans Jeroen Groenendijk, Theo Janssen, et Martin Stokhof, éditeurs, *Formal Methods in the Study of Language*, pages 277–322. Mathematisch Centrum, Amsterdam.

(Kasami, 1965) T. Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Rapport technique, Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

(Keller, 1988) W. R. Keller. 1988. Nested cooper storage : The proper treatment of quantification in ordinary noun phrases. Dans U. Reyle et C. Rohrer, éditeurs, *Natural Language Parsing and Linguistic Theories*, pages 432–447. Reidel, Dordrecht.

(Kinyon et Prolo, 2002) Alexandra Kinyon et Carlos A. Prolo. 2002. A classification of grammar development strategies. Dans *Proceedings of the Workshop on Grammar Engineering and Evaluation*, pages 43–49, Taipei, Taiwan.

(Kinyon, 2000) Alexandra Kinyon. 2000. Hypertags. Dans *Proceedings of COLING-2000*, Saarbruecken.

(Koller et al., 1998) Alexander Koller, Joachim Niehren, et Ralf Treinen. 1998. Dominance constraints : Algorithms and complexity. Dans *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France. To appear in LNCS.

(Koller et al., 2003) Alexander Koller, Joachim Niehren, et Stefan Thater. 2003. Bridging the Gap Between Underspecification Formalisms : Hole Semantics as Dominance Constraints. Dans *Proceedings of the 11th EACL*, Budapest.

(Koller et al., 2004) A. Koller, A. Burchardt, et S. Walter. 2004. Computational semantics. Lecture Notes, ESSLLI 2004. Available at `http://www.coli.uni-saarland.de/projects/milca/courses/esslli04/`.

(Koller, 2004) Alexander Koller. 2004. *Constraint-based and graph-based resolution of ambiguities in natural language*. Thèse de Doctorat, Universität des Saarlandes.

(Kow et al., 2006) E. Kow, Y. Parmentier, et C. Gardent. 2006. SemTAG, the LORIA toolbox for TAG-based Parsing and Generation. Dans *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8), Sydney, Australia*.

(Kroch et Joshi, 1985) A. Kroch et A. Joshi. 1985. The linguistic relevance of tree adjoining grammars. Rapport technique, MS-CIS-85-16, University of Pennsylvania, Philadelphia.

(Lang, 1988) B. Lang. 1988. Complete evaluation of horn clauses : An automata theoretic approach. Rapport technique, Rapport de Recherche 913, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France.

(Lang, 1991) Bernard Lang, 1991. *Towards a Uniform Formal Framework for Parsing*, pages 153–171. Kluwer Academic Publishers.

(Lang, 1994) Bernard Lang. 1994. Recognition can be harder then parsing. *Computational Intelligence — Intelligence Informatique*.

(Lehmann et al., 1996) Sabine Lehmann, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, Veronika Lux, Judith Klein, Kirsten Falkedal, Frederik Fouvry, Dominique Estival, Eva Dauphin, Hervé Compagnion, Judith Baur, Lorna Balkan, et Doug Arnold. 1996. TSNLP — Test Suites for Natural Language Processing. Dans *Proceedings of COLING 1996*, Kopenhagen.

(Lopez, 1999) Patrice Lopez. 1999. *Analyse d'énoncés oraux pour le dialogue homme-machine à l'aide de grammaires lexicalisées d'arbres*. Thèse de Doctorat, Université Henri Poincaré – Nancy 1.

(Montague, 1970) R. Montague. 1970. Pragmatics and intensional logic. *Synthèse*, 22 :68–94.

(Montague, 1974a) Richard Montague. 1974a. English as a formal language. *Formal Philosophy. Selected papers of Richard Montague, pages 188-221.*

(Montague, 1974b) Richard Montague. 1974b. *Formal Philosophy : Selected Papers of Richard Montague*. Yale University Press, New Haven, CT. Edited and with an introduction by Richmond H. Thomason.

(Montague, 1988) R. Montague. 1988. The proper treatment of quantification in ordinary english. Dans J. Kulas, J. H. Fetzer, et T. L. Rankin, éditeurs, *Philosophy, Language, and Artificial Intelligence : Resources for Processing Natural Language*, pages 141–162. Kluwer, Boston.

(Nesson et Shieber, 2006) R. Nesson et S. Shieber. 2006. Simpler TAG Semantics through Synchronization. Dans *Proceedings of Formal Grammars'06, Malaga, Spain*.

(Oepen, 2001) Stephan Oepen. 2001. [incr tsdb()] — competence and performance laboratory. User manual. Technical report, Computational Linguistics, Saarland University, Saarbrücken, Germany.

(Oz-Mozart, 2005) Oz-Mozart. 2005. The Oz-Mozart Programming System. `http://www.mozart-oz.org`.

(Parmentier et Le Roux, 2005) Y. Parmentier et J. Le Roux. 2005. XMG : an Extensible Metagrammatical Framework. Dans *Proceedings of the 17th European Summer School in Logic, Language and Information (ESSLLI), Edinburgh, United Kingdom*.

(Pereira et Warren, 1980) F. Pereira et D. Warren. 1980. Definite clause grammars for language analysis —a survey of the formalism and a comparison to augmented transition networks. *Artificial Intelligence*, 13 :231–278.

(Perlmutter, 1970) David Perlmutter. 1970. Surface structure constraints in syntax. *Linguistic Inquiry*, 1 :187–255.

(Perrier, 2003) Guy Perrier. 2003. Les grammaires d'interaction. Habilitation à Diriger des Recherches en informatique, Université Nancy 2.

(Player, 2004) N. Player. 2004. *Logics of Ambiguity*. Thèse de Doctorat, University of Manchester.

(Pogodalla, 2004a) Sylvain Pogodalla. 2004a. Computing semantic representation : Towards ACG abstract terms as derivation trees. Dans *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 64–71, May.

(Pogodalla, 2004b) Sylvain Pogodalla. 2004b. Using and extending ACG technology : Endowing categorial grammars with an underspecified semantic representation. Dans *Proceedings of Categorial Grammars 2004, Montpellier*, pages 197–209, June.

(Rambow et al., 1995) Owen Rambow, K. Vijay-Shanker, et David J. Weir. 1995. D-tree grammars. Dans *Meeting of the Association for Computational Linguistics*, pages 151–158.

(Rogers et Vijay-Shanker, 1992) J. Rogers et K. Vijay-Shanker. 1992. Reasoning with descriptions of trees. Dans *Proceedings of the 30th annual meeting of the Association for Computational Linguistics (ACL 92)*, pages 72–80.

(Rogers et Vijay-Shanker, 1994) J. Rogers et K. Vijay-Shanker. 1994. Obtaining trees from their descriptions : An application to tree-adjoining grammars. *Computational Intelligence, 10 :401–421*.

(Rogers, 1994) James Rogers. 1994. Capturing CFLs with tree adjoining grammars. Dans *32nd Meeting of the Association for Computational Linguistics, Las Cruces, New Mexico*, pages 155–162.

(Romero et al., 2004) M. Romero, L. Kallmeyer, et O. Babko-Malaya. 2004. LTAG Semantics for Questions. Dans *Proceedings of TAG+7. Vancouver*, pages 186–193, May.

(Romero et Kallmeyer, 2005) M. Romero et L. Kallmeyer. 2005. Scope and Situation Binding in LTAG using Semantic Unification. Dans *Proceedings of the Sixth International Workshop on Computational Semantics IWCS-6, Tilburg*, Jan.

(Sarkar, 2000) A. Sarkar. 2000. Practical experiments in parsing using tree adjoining grammars.

(Schabes et Joshi, 1988) Yves Schabes et Aravind K. Joshi. 1988. An earley-type parsing algorithm for tree adjoining grammars. Dans *ACL*, pages 258–269.

(Schabes et Waters, 1995) Yves Schabes et Richard C. Waters. 1995. Tree insertion grammar : A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*21().

(Schuler et al., 2000) William Schuler, David Chiang, et Mark Dras. 2000. Multi-component TAG and notions of formal power. Dans *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00), Hong Kong, China*, pages 448–455.

(Schulte, 2000) Christian Schulte. 2000. Programming deep concurrent constraint combinators. Dans Enrico Pontelli et Vítor Santos Costa, éditeurs, *Practical Aspects of Declarative Languages. PADL 2000*, volume 1753 de *Lecture Notes in Computer Science*, pages 215–229, Boston, MA, USA. Springer-Verlag.

(Schulte, 2002) C. Schulte. 2002. *Programming Constraint Services*, volume 2302 de *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

(Seddah et Gaiffe, 2005a) D. Seddah et B. Gaiffe. 2005a. Des arbres de dérivation aux forêts de dépendance : un chemin via les forêts partagées. Dans *Actes de la conférence sur le Traitement Automatique des Langues Naturelles (TALN) 2005, Dourdan, juin*.

(Seddah et Gaiffe, 2005b) D. Seddah et B. Gaiffe. 2005b. How to Build Argumental graphs Using TAG Shared Forest : a view from control verbs. Dans *Proceedings of the Fifth Internation Conference on Logical Aspect of Computional Linguistic (LACL) 2005, Bordeaux*, april.

(Seddah et Sagot, 2006) Djamé Seddah et Benoît Sagot. 2006. Modélisation et analyse des coordinations elliptiques via l'exploitation dynamique des forets de dérivation. Dans *Proc. of TALN'06 (poster)*, pages 609–618.

(Seddah, 2004) D. Seddah. 2004. *Synchronisation des connaissances syntaxiques et sémantiques pour l'anlyse d'énoncés en langage naturel à l'aide des grammaires d'arbres adjoints lexicalisées*. Thèse de Doctorat, Université Henri Poincaré – Nancy 1.

(Shieber et al., 1986) Stuart M. Shieber, Fernando C. N. Pereira, Lauri Karttunen, et Martin Kay. 1986. Compilation of papers on unification-based grammar formalisms. Rapport Technique CSLI-86-48, Center for the Study of Language and Information, Stanford, California, Avril.

(Shieber et Schabes, 1990) Stuart M. Shieber et Yves Schabes. 1990. Synchronous tree-adjoining grammars. Dans *Proceedings of the 13th conference on Computational linguistics*, pages 253–258, Morristown, NJ, USA. Association for Computational Linguistics.

(Shieber, 1984) Stuart M. Shieber. 1984. The design of a computer language for linguistic information. Dans *COLING-84*, pages 362–366.

(Smolka, 1995) G. Smolka. 1995. The Oz Programming Model. Dans LNCS series, éditeur, *Computer Science Today*, volume 1000, pages 324 – 343. Springer, Berlin, New York.

(Stalnaker, 1998) Robert Stalnaker. 1998. On the representation of context. *Journal of Logic, Language and Information*7() :3–19.

(Tarski, 1939) Alfred Tarski. 1939. On undecidable statements in enlarged systems of logic and the concept of truth. *J. Symb. Log.*4() :105–112.

(Thomasset et Villemonte de la Clergerie, 2005) François Thomasset et Éric Villemonte de la Clergerie. 2005. Comment obtenir plus des méta-grammaires. Dans *Actes de TALN'05*, Dourdan, France, Juin. ATALA.

(Tomita, 1987) Masaru Tomita. 1987. An efficient augmented-context-free parsing algorithm. *Computational Linguistics*13() :31–46.

(Van Benthem, 1988) J. Van Benthem. 1988. *A Manual of Intensional Logic*. Stanford : CSLI Publications.

(Van Roy, 1990) P. Van Roy. 1990. Extended dcg notation : A tool for applicative programming in prolog. Rapport technique, Technical Report UCB/CSD 90/583, Computer Science Division, UC Berkeley.

(Van Roy, 1993) P. Van Roy. 1993. 1983-1993 : The Wonder Year of Sequential Prolog Implementation. Rapport technique, Paris Research Laboratory Research Report 36, Digital Equipment Corporation, Rueil-Malmaison, France, december.

(Vijay-Shanker et Joshi, 1985) K. Vijay-Shanker et A. Joshi. 1985. Some computational properties of Tree Adjoining Grammars. Dans *ACL*.

(Vijay-Shanker et Joshi, 1988) K. Vijay-Shanker et Aravind K. Joshi. 1988. Feature structures based tree adjoining grammars. Dans *COLING*, pages 714–719.

(Vijay-Shanker et Schabes, 1992) K. Vijay-Shanker et Y. Schabes. 1992. Structure sharing in lexicalized tree adjoining grammars. Dans *Proceedings of COLING'92, Nantes, France*, pages 205–212.

(Vijay-Shanker et Weir, 1993) K. Vijay-Shanker et D. Weir. 1993. The use of shared forest in Tree Adjoining Grammar parsing. Dans *Proceedings of EACL*.

(Villemonte de la Clergerie et Alonso, 1998) Eric Villemonte de la Clergerie et Miguel Alonso. 1998. A tabular interpretation of a class of 2-stack automata. Dans *Proceedings of the 17th international conference on Computational linguistics*, pages 1333–1339, Morristown, NJ, USA. Association for Computational Linguistics.

(Villemonte de la Clergerie, 1993) Éric Villemonte de la Clergerie. 1993. *Automates à Piles et Programmation Dynamique. DyALog : Une application à la programmation en Logique*. Thèse de Doctorat, Université Paris 7.

(Villemonte de La Clergerie, 2000) Éric Villemonte de La Clergerie. 2000. Créer, extraire et manipuler des forêts partagées avec DyALog. Dans Philippe Blache et Éric Villemonte de La Clergerie, éditeurs, *Actes de la Journée ATALA "Représentation et traitement de l'ambiguité pour l'analyse syntaxique"*. ATALA, INRIA, Janvier.

(Villemonte de la Clergerie, 2002) Éric Villemonte de la Clergerie. 2002. Construire des analyseurs avec DyALog. Dans *Actes de TALN'02*, Juin.

(Villemonte de la Clergerie, 2005a) Éric Villemonte de la Clergerie. 2005a. DyALog : a tabular logic programming based environment for NLP. Dans *Proceedings of CSLP'05*, Barcelona.

(Villemonte de la Clergerie, 2005b) Éric Villemonte de la Clergerie. 2005b. From metagrammars to factorized TAG/TIG parsers. Dans *Proceedings of IWPT'05 (poster)*, pages 190–191, Vancouver, Canada, Octobre.

(Villemonte de la Clergerie, 2006) Éric Villemonte de la Clergerie. 2006. Designing tabular parsers for various syntactic formalisms. The association for Logic, Language and Information (FOLLI), Malaga, Spain, Juillet-Août. Tutorial delivered at the 18th European Summer School in Logic, language and information (ESSLLI'06).

(Xia et al., 1998) F. Xia, M. Palmer, K. Vijay-Shanker, et J. Rosenzweig. 1998. Consistent grammar development using partial-tree descriptions for lexicalized tree adjoining grammar. *Proceedings of TAG+4*.

(Xia et al., 1999) F. Xia, M. Palmer, et K. Vijay-Shanker. 1999. Toward semi-automating grammar development. Dans *Proc. of NLPRS-99, Beijing, China*.

(Xia, 2001) Fei Xia. 2001. *Automatic Grammar Generation from two Different Perspectives*. Thèse de Doctorat, University of Pennsylvania.

(XTAG-Research-Group, 2001) XTAG-Research-Group. 2001. A lexicalized tree adjoining grammar for english. Rapport Technique IRCS-01-03, IRCS, University of Pennsylvania. Available at `http://www.cis.upenn.edu/~xtag/gramrelease.html`.

Résumé

Dans cette thèse, nous proposons une architecture logicielle (SemTAG) permettant de réaliser un calcul sémantique pour Grammaires d'Arbres Adjoints. Plus précisément, cette architecture fournit un environnement permettant de construire une représentation sémantique sous-spécifiée (*Predicate Logic Unplugged* (Bos, 1995)) à partir d'une grammaire et d'un énoncé.

Afin de faciliter la gestion de grammaires de taille réelle, la plate-forme SemTAG intègre un compilateur de métagrammaires. Le rôle de ce compilateur est de produire semi-automatiquement une grammaire à partir d'une description factorisée. Cette description correspond à (a) une hiérarchie de fragments d'arbres et (b) des combinaisons de ces fragments au moyen d'un langage de contrôle. De plus, chaque arbre ainsi produit peut être équipé d'une interface syntaxe / sémantique à la (Gardent et Kallmeyer, 2003).

La construction sémantique est réalisée à partir du résultat de l'analyse syntaxique. Cette analyse est fournie par un analyseur syntaxique tabulaire généré automatiquement à partir de la grammaire d'entrée au moyen du système DyALog (De La Clergerie, 2005). Cet analyseur produit une forêt de dérivation, qui encode toutes les dérivations, et à partir desquelles les unifications des indexes sémantiques sont extraites.

Cette plate-forme a été évaluée en termes de couverture sémantique sur la test-suite TSNLP.

Mots-clés: calcul sémantique, grammaires d'arbres adjoints, métagrammaires, sémantique sous-spécifiée, analyse syntaxique profonde.

Abstract

In this thesis, we propose a software architecture (namely SemTAG) allowing for semantic construction with Tree Adjoining Grammars (TAG). SemTAG provides with an environment allowing to build an underspecified semantic representation (Predicate Logic Unplugged (Bos, 95)) from a TAG lexicon and a sentence.

In order to facilitate lexicon management, the SemTAG platforms uses a metagrammar compiler. The goal of the compiler is to automatically produce a TAG from a reduced description. This description corresponds to (a) a hierarchy of tree fragments and (b) combinations of these fragments defined using a control language. Furthermore, each tree produced may be equipped with a syntax / semantic interface following (Gardent and Kallmeyer, 03).

The semantic construction is based on the result of the syntactic parsing. Parsing is done by a tabular TAG parser generated automatically from the input TAG by the DyALog

system of (De La Clergerie, 05). This parser outputs a derivation forest, which encodes all TAG derivations, and from which the unifications of semantic indices are extracted.

This platforms has been evaluated in terms of semantic coverage on the TSNLP test-suite.

Keywords: semantic construction, tree adjoining grammars, metagrammars, underspecified semantics, deep parsing.