Nancy-Université
Université Henri Poincaré

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

Formalisation de la dynamique du langage à travers les grammaires d'interaction

MÉMOIRE

présenté et soutenu publiquement le 25 juin 2007

dans le cadre du

Master Informatique de l'Université Henri Poincaré – Nancy 1
(spécialité Traitement Automatique des Langues)

par

Mathieu Morey

Composition du jury

Examinateurs :	Dominique Méry	Professeur à l'Université Henri Poincaré - Nancy 1
	Noëlle Carbonell	Professeur à l'Université Henri Poincaré - Nancy 1
	Didier Galmiche	Professeur à l'Université Henri Poincaré - Nancy 1
	Claude Godart	Professeur à l'Université Henri Poincaré - Nancy 1
	Guy Perrier	Professeur à l'Université Nancy 2
Encadrant :	Guy Perrier	Professeur à l'Université Nancy 2

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

Loria

Résumé

Les grammaires d'interaction sont un formalisme permettant de faire de l'analyse syntaxique et sémantique de la langue naturelle. Les objets syntaxiques de base sont des descriptions d'arbres polarisées qui spécifient partiellement des arbres syntaxiques.
Dynamic Syntax est un formalisme d'analyse sémantique, dans lequel la syntaxe contraint la croissance d'une structure sémantique. Les objets sémantiques sont des descriptions d'arbres avec des ressources qui spécifient partiellement des arbres sémantiques.
Alors que les grammaires d'interaction n'ont pas encore de dimension sémantique fixée, Dynamic Syntax souffre d'une formulation procédurale donc difficile à manipuler. Nous proposons donc une première tentative de formulation déclarative pour Dynamic Syntax en grammaires d'interaction.
Cette tentative met en lumière plusieurs types de différences entre les deux formalismes, dont certaines semblent surmontables. Nous concluons sur les solutions possibles et les perspectives ouvertes par cette étude.

Mots-clés: grammaires d'interaction, Dynamic Syntax, descriptions d'arbres, analyse sémantique

Remerciements

Je tiens à remercier :

– Guy Perrier pour sa disponibilité, son écoute, sa rigueur scientifique et la liberté qu'il m'a laissée dans ce travail de recherche

– Bruno Guillaume, Joseph Le Roux et Yannick Parmentier pour leurs réponses toujours judicieuses, à la fois sur les logiciels qu'ils développent (Leopar et XMG) et sur mon sujet d'étude

– Ruth Kempson et Claire Beyssade, qui ont répondu à mes nombreuses sollicitations et y ont fourni des réponses aussi complètes que possible

– l'ensemble des membres du projet Calligramme, qui m'a accueilli et forme un environnement scientifique et humain de grande qualité

Table des matières

Introduction

Le but ultime du Traitement Automatique des Langues (ci-après TAL) est de donner à la machine la capacité de langage, qui est de savoir associer à une forme (un énoncé oral ou écrit) un sens ou plus exactement une représentation du sens. Cette capacité est traditionnellement séparée en deux processus inverses : l'analyse et la génération. L'analyse correspond à l'application de traitements à un énoncé en langue naturelle pour obtenir une représentation de son sens manipulable par la machine, donc formelle et si possible non-ambiguë. La génération correspond à la transformation d'informations formelles en un énoncé en langue naturelle qui véhicule ces informations. Le passage de l'énoncé à la représentation formelle et réciproquement se fait par une suite de traitements qui concernent différents niveaux : phonétique (pour simplifier, les sons), morphologie (les mots), syntaxe (les groupes de mots), sémantique (le sens) et pragmatique (l'univers réel) [1]. Plus particulièrement, les trois derniers niveaux apparaissent très liés entre eux, dans le sens où, premièrement, à la suite de Montague de nombreuses théories considèrent que la sémantique peut être déduite « passivement » de la syntaxe, deuxièmement, de nombreux phénomènes sont considérés comme se situant à l'interface entre syntaxe et sémantique, troisièmement, la sémantique requiert fréquemment des informations de la pragmatique.

Actuellement, de nombreux formalismes grammaticaux portent sur la syntaxe, c'est-à-dire sur les relations grammaticales existant entre des mots ou groupes de mots. Le principe de compositionnalité établit que le sens d'un groupe de mots est fonction du sens de chacun des mots. Ce principe est un postulat commun à tous les formalismes, mais il peut être réalisé de différentes façons plus ou moins rigides. En conséquence l'analyse syntaxique, terminée ou en cours, guide plus ou moins fortement l'analyse sémantique dans la plupart des extensions sémantiques proposées pour les formalismes existants.
Parmi ces formalismes, les grammaires d'interaction [Per03] (ci-après IG pour Interaction Grammars) sont surtout conçues, développées et expérimentées en analyse syntaxique, et leur dimension sémantique n'est pas encore clairement fixée [Per05]. Les grammaires d'interaction sont fondées sur deux idées forces : une grammaire est un système de contraintes, et la composition syntaxique est contrôlée par un système de polarités.

Ces deux notions sont au coeur d'un autre formalisme grammatical appelé Dynamic Syntax [KMVG01] (ci-après DS), qui permet de faire de l'analyse sémantique et de la génération. Ce formalisme est fondé psycholinguistiquement et vise à être le plus fidèle possible à l'essence même de la capacité de langage, dont une des caractéristiques est la linéarité. La pensée n'est pas linéaire : les concepts sont organisés et reliés les uns aux autres, ce que l'on modélise sous forme de graphe. Mais le langage, lui, est linéaire : un énoncé, qu'il soit oral ou écrit, est une succession de mots (ou d'unités de sens). À l'oral, les sons sont produits et entendus les uns après les autres ; à l'écrit les mots sont tracés et lus les uns après les autres (de gauche à droite, de droite à gauche, de haut en bas voire de bas en haut selon les cultures). Dynamic Syntax reprend cette idée de linéarité en faisant émerger progressivement, au fur et à mesure de la lecture d'un énoncé, une structure sémantique. En fait, c'est l'application successive de règles non lexicalisées et d'actions lexicales qui fait croître et enrichit la structure sémantique correspondant à l'énoncé lu. Ces règles et ces actions lexicales enrichissent la structure émergente avec deux types d'information : des contraintes sur sa forme finale et des requêtes ou apports d'informations, avec pour impératif qu'aucune requête d'information ne reste insatisfaite dans la structure finale.

[1] Pour un panorama clair et plus complet, voir l'introduction au TAL de Jean Véronis, disponible à `http://www.up.univ-mrs.fr/~veronis/cours/INFZ18/veronis-INFZ18.pdf`.

On retrouve bien ici les deux idées forces des IG, et c'est justement cette proximité qui motive le présent travail.

Après avoir présenté les IG dans le **chapitre 1** et DS dans le **chapitre 2**, nous tentons dans le **chapitre 3** de les faire converger (sur la base de leurs similarités) afin de mieux formaliser DS en nous appuyant sur le cadre fourni par les IG. Assez vite, nous constatons qu'une première approche « naïve » ne suffit pas et nous en expliquons les raisons dans le **chapitre 4**.

Chapitre 1

Les grammaires d'interaction

1.1 Grammaires d'interaction et analyse syntaxique

Les grammaires d'interaction (IG) sont un formalisme grammatical initialement conçu pour modéliser la syntaxe des langues [Per02]. Plus précisément, les IG sont un raffinement des grammaires catégorielles (ci-après CG pour Categorial Grammars), raffinement doublé d'un changement de point de vue. Alors que les CG appartiennent à ce que Pullum et Scholz appellent la *Generative-Enumerative Syntax* [PS01] (un énoncé est correct si une suite de dérivations à partir d'axiomes permet de le générer), les IG appartiennent à la *Model-Theoretic Syntax* (un énoncé est correct s'il en existe un modèle qui satisfait toutes les contraintes de la grammaire). Pour cela les IG utilisent la notion de *description d'arbre* suivant ainsi l'idée développée par Vijay-Shanker [VS92] pour les grammaires d'arbres adjoints (ci-après TAG pour Tree Adjoining Grammars).

L'analyse syntaxique est l'étude des relations grammaticales existant entre des mots ou groupes de mots. Il existe de nombreux formalismes d'analyse syntaxique plus ou moins complexes et plus ou moins étroitement liés à différentes théories linguistiques, aussi nous contenterons-nous d'illustrer notre propos dans cette section par un exemple simple : « Jean aime Marie ».

1.1.1 Analyse en constituants

La façon la plus simple de traiter cet exemple est d'effectuer une (traditionnelle) analyse en constituants.

Soit la grammaire hors-contexte définie par les règles de la figure 1.1, où NP signifie Noun Phrase (groupe nominal), VP Verb Phrase (groupe verbal), tV transitive Verb (verbe transitif), S sentence (phrase). Cette grammaire fait partie des grammaires dites *de constituants*, dans lesquelles des règles de

$$
\begin{array}{lcl}
\text{S} & \rightarrow & \text{NP VP} \\
\text{VP} & \rightarrow & \text{tV NP} \\
\text{NP} & \rightarrow & \text{Jean} \\
\text{tV} & \rightarrow & \text{aime} \\
\text{NP} & \rightarrow & \text{Marie}
\end{array}
$$

FIG. 1.1 – Règles de grammaire hors-contexte pour « Jean aime Marie »

grammaire (souvent nombreuses) indiquent quelle catégorie peut se combiner avec quelle autre catégorie pour former une troisième catégorie.

Pour la phrase exemple « Jean aime Marie », on a donc l'arbre d'analyse syntaxique hors-contexte de la figure 1.2.

Cette analyse arborescente met en évidence que « Jean » est en position de sujet et « Marie » d'objet du verbe transitif « aime ».

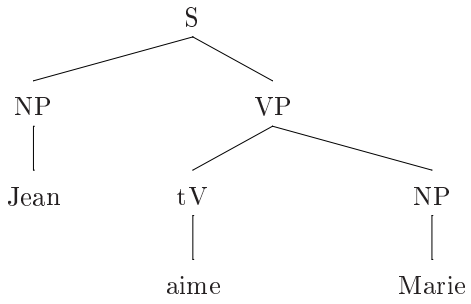

FIG. 1.2 – Arbre d'analyse syntaxique hors-contexte pour « Jean aime Marie »

1.1.2 Analyse en grammaires catégorielles

Le problème majeur des grammaires hors-contexte est la présence de règles non lexicalisées, c'est-à-dire de règles qui ne contiennent que des non-terminaux. Si la grammaire est complètement lexicalisée (un mot pouvant lexicaliser plusieurs règles), pour un énoncé donné on peut calculer le nombre maximum d'analyses possibles en faisant le produit du nombre de règles associées à chaque mot par le nombre de mots de l'énoncé. Si la grammaire n'est pas lexicalisée (en fait, si elle n'est pas lexicalisable), on ne peut plus être certain du nombre d'analyses possibles, car on ne peut pas savoir avant l'analyse de quelles règles de la grammaire on aura besoin au cours de celle-ci. La non-lexicalisation introduit de l'indéterminisme. La solution est donc d'écrire des grammaires lexicalisées. Pour cela, Bar-Hillel et Adjukiewicz [Adj35] ont développé les grammaires catégorielles, dont le principe est d'associer à chaque entrée lexicale une ou plusieurs catégories syntaxiques. Une catégorie est vue comme une fonction qui s'applique à une catégorie pour en former une troisième. Toute l'information sur les possibilités de combinaison de chaque mot est donc contenue intégralement dans la ou les catégories qui lui sont affectées, en tenant compte de l'ordre des mots dans la phrase.

Corollairement, les CG — et c'est cette caractéristique qui en guide le processus de composition — considèrent les syntagmes comme des ressources consommables, certaines ressources étant attendues (on modélise cela par une polarité négative) et d'autres disponibles (polarité positive). Dans cette perspective, une phrase est bien formée si toutes les ressources attendues par chacun des syntagmes qui la composent sont rendues disponibles par les autres syntagmes, et si toutes les ressources offertes par chacun des syntagmes sont consommées par les autres syntagmes. Alors, la bonne formation d'une phrase se résume à la neutralisation des polarités opposées. Intuitivement, un verbe transitif direct a besoin, pour former une phrase correcte, d'un groupe nominal (ci-après GN) à gauche et un à droite. Donc, si une phrase est constituée d'un verbe transitif direct et de deux GN (l'un à gauche et l'autre à droite), les ressources fournies par ces GN sont consommées par le verbe transitif direct et les ressources attendues par ce dernier sont disponibles : les polarités sont neutralisées.

En CG, on définit la catégorie de chaque mot à partir des catégories de base *s* pour *sentence* (phrase), *np* pour *noun phrase* (groupe nominal), ainsi que *n* pour *noun* (dont nous n'avons pas besoin pour notre exemple). On a donc pour les mots de notre exemple les catégories :

aime	(np \ s) / np
Jean	np
Marie	np

Un verbe transitif comme « aime » doit se combiner à sa droite avec un GN (/ np), puis à sa gauche avec un autre GN (np \), pour donner une phrase (s).

L'analyse de la phrase exemple se fait naturellement, comme le montre la figure 1.3.

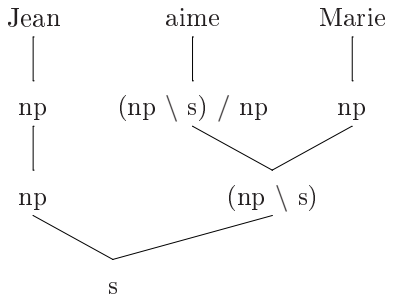

FIG. 1.3 – Analyse catégorielle de « Jean aime Marie »

1.1.3 Analyse en grammaires d'interaction

Les IG reprennent cette idée de polarités en la descendant du niveau des syntagmes à celui des traits grammaticaux utilisés pour décrire les syntagmes (les parties du discours, le genre, la fonction syntaxique...).
Ainsi dans l'exemple précédent, chacun des deux GN demande une fonction syntaxique ($funct \leftarrow ?$), et le verbe demande deux GN et fournit à l'un la fonction sujet ($funct \rightarrow subj$) et à l'autre la fonction objet ($funct \rightarrow obj$). C'est ce qui est représenté par les descriptions d'arbres rassemblées dans la figure 1.4, le résultat de l'analyse étant l'arbre neutralisé 1.5. Dans la section qui suit, nous présentons un peu plus en détail la notion de description d'arbre et le système de polarités.

1.2 Descriptions d'arbres polarisées et recherche de modèles

1.2.1 Descriptions d'arbres

Une description d'arbre est un ensemble de noeuds et de relations de domination et de précédence entre ces noeuds. Dans le cas d'un arbre syntaxique, les noeuds représentent des syntagmes et les relations des contraintes relatives entre ces syntagmes. Les propriétés morphosyntaxiques de ces syntagmes sont exprimées par des structures de traits attachées aux noeuds, comme nous l'avons fait dans la figure 1.4.

Les relations entre les noeuds d'une description peuvent être de quatre types, distingués graphiquement par le style de traits les représentant :

domination immédiate : $A > B$ signifie que le noeud A est le père du noeud B, ce qui est représenté par un trait continu. Syntaxiquement, cela signifie que le syntagme B est un constituant immédiat de A. On peut figer l'ensemble des fils d'un noeud A à l'aide de la relation $A > A_1, A_2, \ldots, A_p$. Cette relation signifie que le noeud A a exactement p fils, et donc syntaxiquement que A a exactement p constituants immédiats.

domination sous-spécifiée : $A >^* B$ signifie que le noeud B est dans le sous-arbre issu de A à une profondeur indéterminée, ce qui est représenté par un trait discontinu. À la limite, B peut s'identifier à A. Syntaxiquement, cela signifie que le syntagme B est inclus dans A à une profondeur indéterminée, ce qui permet d'exprimer à la fois une dépendance syntaxique non bornée et la possibilité d'appliquer des modifieurs à un syntagme. De plus amples détails sont exposés dans [Per05].

précédence immédiate : $A \prec B$ signifie que le noeud A précède immédiatement le noeud B dans l'ordre linéaire des noeuds, ce qui est représenté par une flèche continue. Syntaxiquement, cela signifie que le syntagme A précède immédiatement le syntagme B dans l'ordre linéaire des mots de la phrase.

précédence sous-spécifiée : $A \prec^* B$ signifie que le noeud A précède le noeud B dans l'ordre linéaire des noeuds, ce qui est représenté par une flèche discontinue. Syntaxiquement, cela signifie que le syntagme A précède le syntagme B dans l'ordre linéaire des mots de la phrase.

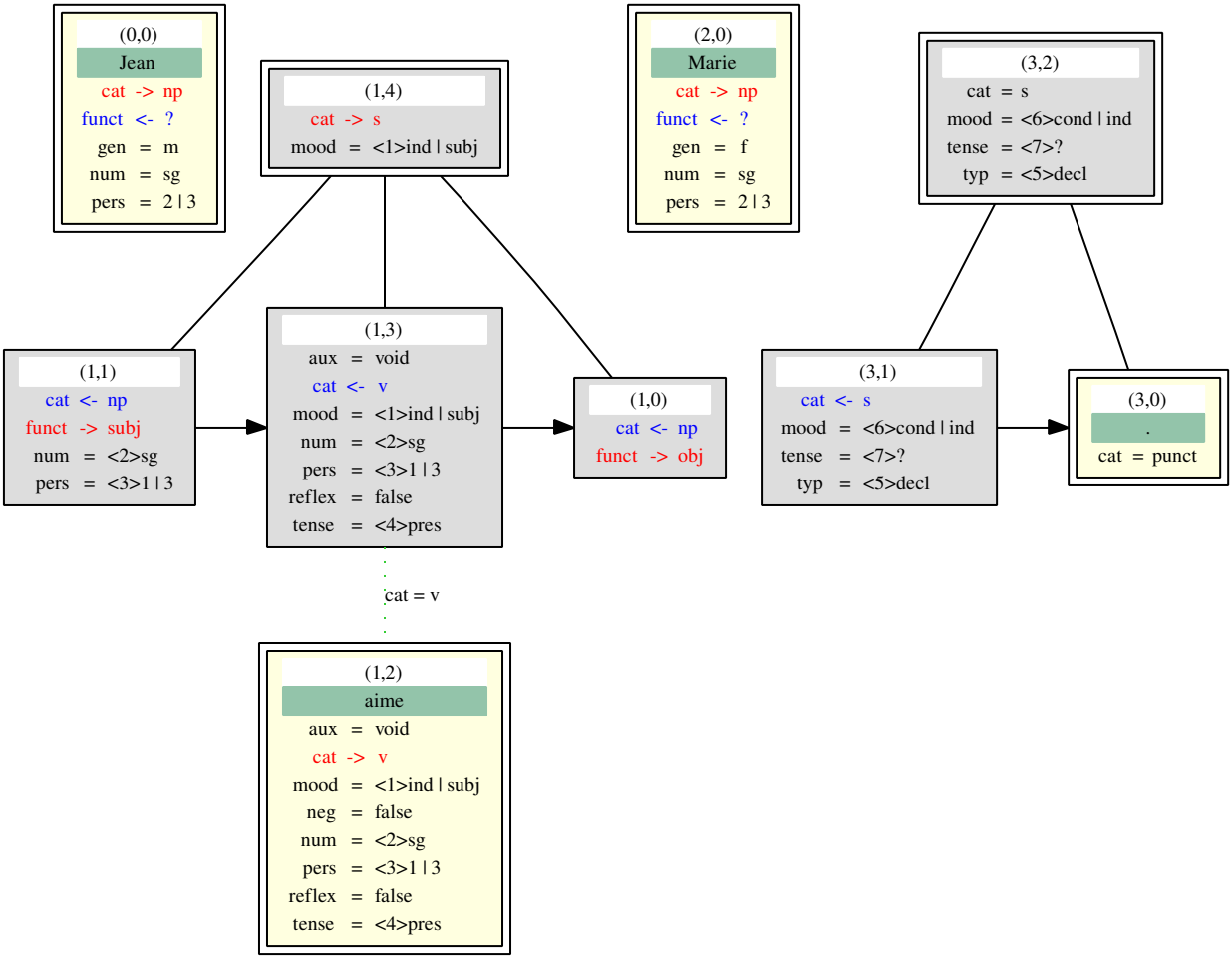

FIG. 1.4 – Description d'arbres associée à « Jean aime Marie. »

La figure 1.4 présente plusieurs relations de domation immédiate, une relation de domination sous-spécifiée (entre les noeuds d'identifiant (1,3) et (1,2)) et des relations de précédence immédiate.
Comme de nombreux éléments de ce formalisme, ce système de relations n'est pas figé et rien n'interdit de l'enrichir pour traduire certaines spécificités d'une langue.

1.2.2 Polarités

L'originalité des IG est que les traits morpho-syntaxiques associés aux syntagmes sont polarisés. Les structures de traits sont un élément de base des formalismes du TAL, et habituellement un trait est associé à une valeur dans un couple $(trait, valeur)$. En IG, on y ajoute une polarité pour obtenir un triplet $(trait, polarit, valeur)$. Une polarité peut être $\leftarrow, \rightarrow, =, \leftrightarrow$ *et* $\sim$ pour dire qu'un trait est *négatif, positif, neutre ordinaire, saturé* ou *virtuel*. Un trait négatif $f \leftarrow v$ représente une ressource attendue, un trait positif $f \rightarrow v$ une ressource disponible et un trait neutre ordinaire $f = v$ une propriété qui ne se comporte pas comme une ressource consommable. Un trait saturé $f \leftrightarrow v$ est un trait neutre particulier qui correspond à la neutralisation d'un trait négatif par un trait positif. Contrairement à un trait neutre ordinaire, il ne peut pas s'unifier avec un trait positif ou négatif. Un trait virtuel $f \sim v$ est un trait neutre particulier qui doit impérativement s'unifier avec un autre trait polarisé non virtuel. L'unification de deux noeuds polarisés se fait suivant le tableau de polarités 1.1.

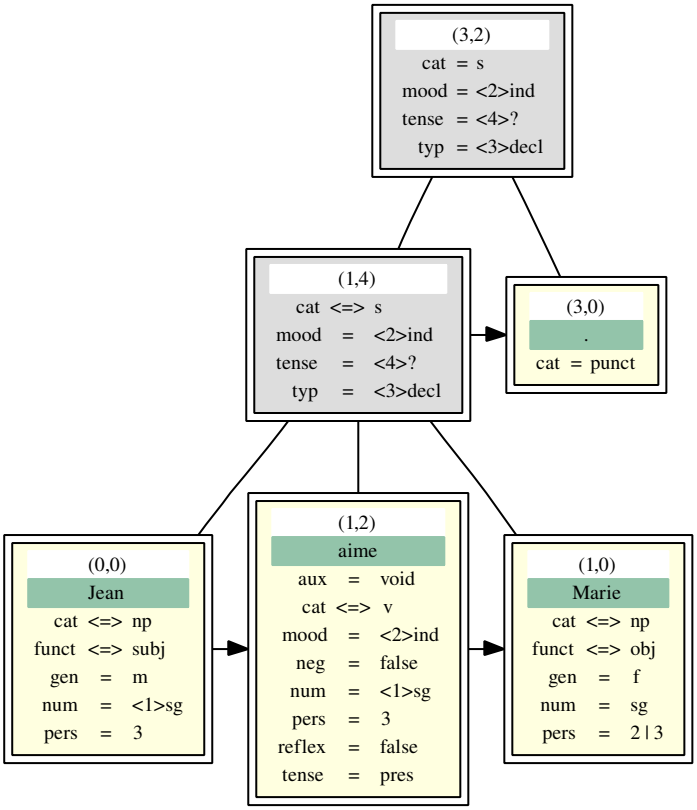

FIG. 1.5 – Arbre saturé pour « Jean aime Marie. »

	$\leftarrow$	$\rightarrow$	$=$	$\leftrightarrow$
$\leftarrow$	$\bot$	$\leftrightarrow$	$\leftarrow$	$\bot$
$\rightarrow$	$\leftrightarrow$	$\bot$	$\rightarrow$	$\bot$
$=$	$\leftarrow$	$\rightarrow$	$=$	$\leftrightarrow$
$\leftrightarrow$	$\bot$	$\bot$	$\leftrightarrow$	$\bot$

TAB. 1.1 – Résultats de l'unification de deux polarités

1.2.3 Recherche de modèles

Une description d'arbre polarisée (ci-après DAP) peut être vue comme un ensemble de contraintes. Cet ensemble de contraintes définit un ensemble d'arbres syntaxiques, chaque arbre syntaxique étant un modèle de la DAP correspondante.
Dans le lexique, un mot est associé une ou plusieurs DAP, qui représentent ses différents emplois. Par exemple, le verbe « manger » peut être intransitif (« Jean mange. ») ou transitif direct (« Jean mange une pomme. »). On a donc dans le lexique une DAP pour l'emploi intransitif (le verbe manger requiert un GN sujet), et une autre pour l'emploi transitif direct (le verbe manger requiert un GN sujet et un GN objet).
On obtient une DAP d'une phrase en prenant une DAP par mot de la phrase et en faisant l'union des DAP choisies. Il y a donc autant de DAP d'une phrase que de combinaisons possibles des DAP des entrées lexicales.
Le processus d'analyse en IG consiste à rechercher les modèles minimaux et neutres des DAP d'une phrase. La minimalité est nécessaire car les seuls modèles intéressants d'un point de vue syntaxique sont ceux qui ajoutent un minimum d'informations à la DAP qu'ils représentent. La recherche de modèles est guidée

par les polarités : toute polarité positive doit fusionner avec une polarité négative et réciproquement, alors qu'une polarité saturée ne peut fusionner ni avec l'une ni avec l'autre. Une phrase n'est grammaticale que si toutes ses polarités sont saturées (i.e. qu'elle est neutre). La construction de modèles se fait en superposant pas à pas des morceaux d'arbres, le choix des morceaux d'arbres à superposer étant guidé par les polarités.

[Per05] contient une présentation plus exhaustive et plus formelle des grammaires d'interaction, qui complète et englobe les idées rapidement reprises ici.

1.3 Fonctionnement concret

Les idées développées dans le formalisme des IG sont mises en oeuvre dans un analyseur syntaxique électrostatique appelé Leopar. Pour analyser une phrase, Leopar utilise un lexique syntaxique (à chaque mot sont associées ses DAP). Pour écrire et générer ce lexique plus efficacement et plus facilement, on utilise XMG, un compilateur de métagrammaires. Nous présentons donc maintenant ces deux outils.

1.3.1 XMG, le compilateur de métagrammaires

XMG est un compilateur de métagrammaires développé au Loria et disponible en ligne à l'adresse `http://sourcesup.cru.fr/xmg/`. Une métagrammaire est une description réduite basée sur les généralisations linguistiques qui apparaissent dans les arbres de la grammaire. Ce concept vient de M.-H. Candito qui à la fin des années 90 a écrit la première métagrammaire TAG. Au lieu de décrire tous les arbres de la grammaire, on va décrire des fragments d'arbres qui apparaissent plusieurs fois, puis décrire la façon dont ces fragments se combinent pour donner les arbres de la grammaire. À partir de ces informations le compilateur génère la grammaire correspondante. Actuellement XMG peut générer des grammaires TAG et IG. [DLP05] présente beaucoup plus largement XMG, son architecture et ses perspectives.

1.3.2 Leopar, l'analyseur syntaxique électrostatique

Leopar est un analyseur syntaxique développé au sein de l'équipe Calligramme et disponible en ligne à l'adresse `http://www.loria.fr/equipes/calligramme/leopar/`. Il permet d'analyser une phrase suivant 3 algorithmes : un algorithme incrémental, un CKY et un Earley. Pour analyser une phrase, Leopar utilise un lexique syntaxique qui est organisé suivant le schéma 1.6. Pour rendre le lexique plus facile à maintenir et développer, les lemmes sont classés par familles. Une famille apporte les informations communes aux lemmes qui la composent. Les lemmes n'apportent donc que les informations qui ne sont pas communes à leur famille. Quand on réalise la jointure des lemmes et des familles, on obtient un lexique syntaxique, non fléchi et non arborescent. Ce lexique, joint avec les formes fléchies (contenues dans le lexique morphologique), génère le lexique morpho-syntaxique. Ce lexique morpho-syntaxique, joint avec la grammaire non ancrée (de préférence générée par XMG), produit la grammaire ancrée qui contient les arbres utilisés par Leopar.

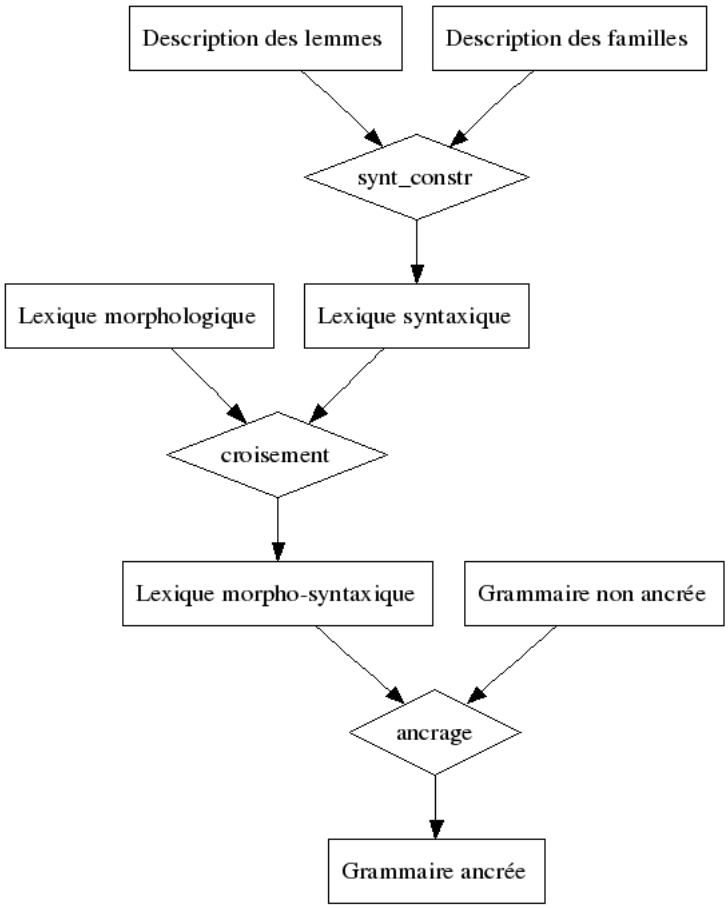

FIG. 1.6 – Organisation du lexique syntaxique de Leopar

Chapitre 2

Dynamic Syntax

2.1 Intuition générale et motivation

Qu'est-ce que la connaissance d'une langue naturelle ?

La plupart des formalismes actuels (parmi lesquels les grammaires catégorielles) considèrent l'ensemble des langues naturelles comme un sous-ensemble des langages formels, à la suite des travaux de Richard Montague compilés dans [Mon74], dont la démarche est résumée par cette phrase :

> La relation entre la syntaxe et la sémantique d'une langue naturelle est essentiellement de même nature que la relation entre syntaxe et sémantique dans un langage formel (e.g. la logique du premier ordre).

En d'autres termes, une langue est définie par une grammaire qui permet de générer l'ensemble de ses phrases, et leur analyse sémantique dérive — par divers mécanismes — de l'analyse syntaxique faite selon les règles de la grammaire. Connaître une langue équivaut strictement à *connaître sa grammaire*.

Si l'on cherche à répondre à cette question de façon plus intuitive, la réponse est sensiblement différente. La connaissance d'une langue est (simplement) la capacité à comprendre et à générer des énoncés compréhensibles par les autres locuteurs de cette langue. La syntaxe n'est alors qu'un moyen de contraindre et de structurer la communication du sens. Connaître une langue équivaut à *savoir associer à un énoncé son sens*, en *s'aidant* des contraintes grammaticales. Ces dernières sont un système de contraintes ; la sémantique tient compte de ces contraintes mais n'est pas une *déduction passive* de la syntaxe.

On retrouve ici pour la sémantique l'opposition évoquée au début du chapitre 1 pour la syntaxe entre *Generative-Enumerative Syntax* et *Model-Theoretic Syntax*. De la même façon que nous l'avons fait pour l'analyse syntaxique, nous allons examiner de plus près les conséquences de l'écart entre ces deux conceptions du point de vue de l'analyse sémantique.

Analyse sémantique montagovienne

L'analyse sémantique est le passage d'un énoncé (éventuellement ambigu) à une représentation formelle non ambiguë. Il existe une longue tradition d'analyse sémantique des langues naturelles, mais c'est Richard Montague qui a laissé l'empreinte la plus forte sur la sémantique computationnelle actuelle. En liant sémantique des langues naturelles et théorie des modèles, son approche permet de considérer que déterminer la signification d'une phrase P équivaut à déterminer les conditions de vérité de P dans l'ensemble des mondes possibles.

Comme nous l'avons vu précédemment pour les CG, Montague lexicalise sa grammaire et donc associe à chaque mot une ou plusieurs catégories syntaxiques fonctionnelles, avec s pour *sentence* (phrase), np pour *noun phrase* (groupe nominal) et n pour *noun* (nom) comme catégories de base.
À chaque mot, il associe également son interprétation sémantique sous la forme d'un λ-terme typé. Les types sémantiques sont eux aussi fonctionnels, basés sur les types atomiques t pour *truth value* (valeur

de vérité) et e pour *entity* (entité).
Pour notre exemple, les catégories syntaxiques et interprétations sémantiques sont indiquées à la figure 2.1. On se donne deux constantes J et M de type $(e \to t) \to t$, et un symbole de relation LOVE de type $((e \to t) \to t) \to (((e \to t) \to t) \to t)$.

Jean	np	$[\![\text{Jean}]\!]$	=	J
Marie	np	$[\![\text{Marie}]\!]$	=	M
aime	$np \to (np \to s)$	$[\![\text{aime}]\!]$	=	λx y . LOVE y x

FIG. 2.1 – Grammaire de Montague pour « Jean aime Marie »

Un principe fondamental de l'analyse sémantique est le principe de compositionnalité, selon lequel le sens d'une phrase est fonction du sens des mots qui la composent et de la façon dont ils se composent. Ce principe peut être formulé de différentes façons, plus ou moins strictes, et Montague définit une version forte de la compositionnalité : il considère la syntaxe et la sémantique comme des algèbres et l'interprétation sémantique comme un homomorphisme entre ces deux algèbres.

À chaque catégorie syntaxique de base, il associe son interprétation qui est un type sémantique, comme la figure 2.2 le montre. Et grâce à l'homomorphisme entre syntaxe et sémantique, on obtient l'interprétation des autres catégories syntaxiques ($[\alpha \to \beta] = [\alpha] \to [\beta]$).

$$[\text{s}] = t \quad [\text{n}] = e \to t \quad [\text{np}] = (e \to t) \to t$$

FIG. 2.2 – Interprétation fonctionnelle des catégories syntaxiques de base de la Grammaire de Montague

Finalement, si l'on cherche à calculer la sémantique de notre exemple, on obtient le λ-terme de la figure 2.3.

$$(\lambda\text{x y . LOVE y x) M J} \to_\beta \text{LOVE J M}$$

FIG. 2.3 – λ-terme et sa β-réduction pour « Jean aime Marie »

De la sémantique de Montague à Dynamic Syntax

Nous pouvons maintenant examiner plus en détail les conséquences respectives des deux conceptions esquissées précédemment de la connaissance d'une langue : d'un côté le langage naturel comme langage formel, de l'autre le langage naturel comme ensemble de contraintes sur la communication d'un sens.
Si l'on adopte le premier point de vue (« classique montagovien »), un énoncé doit de préférence être considéré dans sa globalité, car c'est la seule façon de dériver une structure syntaxique complète qui permettra de calculer la ou les structures sémantiques correspondantes. Si l'on voulait établir une correspondance directe avec le fonctionnement de la faculté de langage d'un être humain, cela signifierait qu'il vérifie d'abord que l'énoncé qu'on lui fournit est correct syntaxiquement, puis qu'il en déduise le sens.
A contrario, si l'on adopte le second point de vue (« intuitionniste »), un énoncé est vu comme une suite de mots qui, linéairement et un par un, apportent une « petite quantité de sens ». La syntaxe n'est alors rien d'autre que le processus par lequel l'être humain utilise et organise les nouveaux éléments de sens qui lui parviennent pour compléter, enrichir et préciser le sens global de l'énoncé qui lui est fourni. Reprenant notre parallèle avec la faculté de langage d'un être humain, cela signifierait que celui-ci essaie en permanence, au fur et à mesure, d'organiser de façon cohérente les informations qu'il reçoit pour faire émerger un sens global, la syntaxe lui donnant des indices sur la façon de le faire. C'est cette perspective

qui semble la plus fondée psycholinguistiquement, et c'est elle qui sert de point de départ à Dynamic Syntax [KMVG01].

Ce point de départ a de nombreuses conséquences. Délaisser une vision statique, globale[2], « verticale » de l'énoncé pour une vision dynamique, partielle, linéaire permet de mieux rendre compte de phénomènes habituellement problématiques car intrinsèquement liés à l'ordre des mots tels que les crossovers ou les phénomènes d'alignement constatés dans les situations de dialogue. En DS la sémantique est l'objet premier de l'étude, la syntaxe n'étant qu'un ensemble de contraintes guidant l'émergence de la structure sémantique. C'est parce qu'elle voit la syntaxe comme un ensemble de contraintes que DS fait partie, comme les IG, des formalismes de la « Model-Theoretic Syntax »[PS01].

Considérer la sémantique comme centre d'intérêt premier conduit également à accepter plus naturellement l'idée de sous-spécification. En effet la sous-spécification est partie intégrante de la sémantique, par exemple un pronom a parfois plusieurs antécédents potentiels. Les ambiguïtés sur le sens d'un mot sont levées soit par le contexte gauche du mot (les mots qui le précèdent), soit par son contexte droit (les mots qui le suivent), soit par la pragmatique. La sous-spécification est impliquée dans la résolution d'anaphore (détermination de l'antécédent d'un pronom), le traitement des subordonnées relatives mais également dans tous les phénomènes qui se situent à la périphérie (gauche ou droite) des phrases, comme la topicalisation ou la dislocation.

La sous-spécification est au coeur du processus d'analyse sémantique de Dynamic Syntax, couplée avec les notions (complémentaires l'une de l'autre) de requête et de fourniture d'information. Par exemple, dans la construction « X donne Y à Z », le verbe « donner » requiert que deux individus X et Z et l'objet Y soient définis par le reste de la phrase pour remplir les différents rôles sémantiques de ce verbe. Les groupes nominaux correspondants fournissent l'information sémantique requise par le verbe. Si les rôles sémantiques du verbe ne sont pas remplis, la phrase n'est sémantiquement pas complète.

L'analyse sémantique de DS consiste en la construction progressive d'un arbre sémantique. Au cours du processus de construction, cet arbre contient des requêtes et des fournitures d'informations, et il peut être sous-spécifié. Nous allons maintenant détailler ce processus.

2.2 Principes et déroulement d'une analyse

2.2.1 Point de départ et principes généraux

Une analyse en DS a toujours le même point de départ, qui est son but : établir une valeur de vérité. On introduit donc un noeud qui est la racine du futur arbre. Sur ce noeud une valeur de vérité est requise, qui est celle de la formule représentant la sémantique de la phrase. La règle d'*initialisation*, qui n'est pas ancrée lexicalement, nous permet d'introduire l'arbre de la figure 2.4. Une analyse en DS est en fait

$$?Ty(t), \Diamond$$

FIG. 2.4 – Arbre associée à la règle d'initialisation

une succession d'applications, à un arbre sémantique en construction, de règles non ancrées lexicalement (comme la règle d'initialisation) et de procédures associées à des entrées lexicales.
Afin de savoir en quel point de l'arbre il faut appliquer une règle ou procédure, il y a à tout instant de l'analyse un noeud et un seul de l'arbre en construction qui porte un marqueur spécial appelé *focus*.
À tout instant de l'analyse, au noeud pointé par le focus, il est possible que plusieurs règles ou procédures puissent s'appliquer. Les différentes constructions d'arbres doivent donc être menées en parallèle, afin de ne garder in fine que les arbres dont le contenu sera complet (une procédure a été exécutée pour chaque mot de l'énoncé à analyser) et correct (l'arbre ne contient plus de requête, et le focus est revenu au noeud racine). Évidemment, cela crée une explosion combinatoire du nombre d'analyses à mener qui explique

[2] Ce propos doit être nuancé : certains formalismes permettent de considérer des structures partielles, comme les IG ou les CCG.

la difficulté à en réaliser une implantation qui soit plus qu'un « jouet ».

À la fin de l'analyse, tout noeud contient une formule typée. Ainsi, un noeud contient au moins deux informations : sa formule et le type de sa formule.
Les différents types couramment utilisés en DS sont exposés dans la figure 2.5. Ils sont composés à partir de trois types atomiques : e pour *entity* (entité), t pour *truth value* (valeur de vérité) et cn pour *common noun* (nom commun)[3].
À la fin de l'analyse, tout noeud est fixé dans l'arbre, c'est-à-dire qu'il ne reste aucune relation de domi-

Type	Description	Exemple
Ty(e)	Terme individuel	Fo(Mary'), Fo(ϵ,x,Student'(x))
Ty(t)	Proposition	Fo(Sing'(John')), Fo(Upset'(Hilary')(Joan'))
Ty(e→t)	Prédicat à 1 place	Fo(Upset'(Hilary')), Fo(Run')
Ty(e→(e→t))	Prédicat à 2 places	Fo(Upset'), Fo(Give'(John'))
Ty(e→(e→(e→t)))	Prédicat à 3 places	Fo(Give'), Fo(Put')
Ty(t→(e→t))	Prédicat propositionnel	Fo(Believe'), Fo(Say')
Ty(cn)	Nom commun	Fo(x,Student'(x)), Fo(y,Father'(John')(y))
Ty(cn→e)	Quantificateur	Fo($\lambda P.\epsilon, P$)

FIG. 2.5 – Types les plus fréquemment utilisés en Dynamic Syntax

nation large qui n'ait pas été résolue comme une suite finie éventuellement nulle de dominations strictes. Le contrôle de cet aspect est réalisé par l'ajout à chaque noeud de son adresse de Gorn dans l'arbre. Les adresses de Gorn ont un autre avantage : elles permettent de représenter « à plat » une structure arborescente, en listant les noeuds, car leur adresse permet de reconstruire la structure arborescente.

L'arbre qui est construit est un arbre fini binaire, où le fils droit d'un noeud est noté $\langle\downarrow_1\rangle$et a le rôle de foncteur et le fils gauche est noté $\langle\downarrow_0\rangle$et a le rôle d'argument de ce foncteur. La seule opération requise sur les types est le *modus ponens* et la seule opération requise sur les formules est l'*application fonctionnelle*.
L'exemple présenté à la section qui suit illustre ces différents points.

2.2.2 Déroulement d'une analyse simple : « Marie dort. »

Voyons maintenant, via l'analyse d'une phrase simple (« Marie dort. »), le déroulement d'une analyse en DS. Par souci de clarté, nous ferons donc toujours les bons choix de règle et de procédure à appliquer afin de ne mener qu'une seule analyse.
Toujours par souci de simplicité, nous n'exposerons pas les règles non lexicales, et une seule fois une procédure lexicale : nous nous contenterons d'en montrer les effets sur notre exemple. Ces règles sont identiques à celles qui sont exposées pour l'anglais dans [CKM05] et [KMVG01].
Enfin, afin de ne pas surcharger les arbres, nous ne mettrons pas toutes les adresses de Gorn des noeuds. Pour notre cas particulier d'un arbre binaire, le noeud racine d'un arbre a l'adresse 0, son fils gauche a l'adresse 00, son fils droit l'adresse 01, et ainsi de suite.

Le point de départ de toute analyse est, nous l'avons dit, la volonté d'établir une valeur de vérité ($?Ty(t)$) pour l'énoncé, ce qui correspond à la figure 2.6. Le noeud a l'adresse de Gorn 0, car il est la racine de l'arbre à construire : $Tn(0)$, et c'est naturellement sur lui que vont avoir lieu les prochaines opérations de croissance de l'arbre : il porte le focus matérialisé par le $\Diamond$.

La structure de phrase la plus courante en français est formée d'un sujet et d'un prédicat. La règle non lexicalisée nommée *introduction sujet - prédicat* ajoute donc des informations à la racine d'un arbre qui n'a pas encore de fils, c'est la figure 2.7.

[3] Ce nom de type est abusif, *cn* dénote plutôt une propriété, ce qui est aussi le cas des noms propres.

$$Tn(0), ?Ty(t), \Diamond$$

FIG. 2.6 – Arbre d'initialisation

$$Tn(0), ?Ty(t), ?\langle\downarrow_0\rangle Ty(e), ?\langle\downarrow_1\rangle Ty(e \rightarrow t), \Diamond$$

FIG. 2.7 – Arbre après introduction sujet - prédicat

Au noeud racine on a ajouté des requêtes qui précisent le type que devront avoir ses fils, qui ne sont pas encore construits. C'est la règle de *prédiction sujet - prédicat* qui construit les fils d'un noeud à partir des requêtes les concernant.

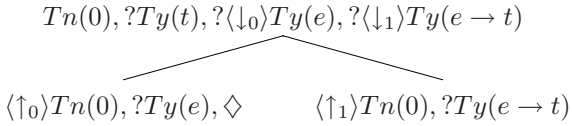

FIG. 2.8 – Arbre après prédiction sujet - prédicat

La règle de *prédiction sujet - prédicat* a également déplacé le focus sur le sujet, c'est à partir de ce noeud que l'arbre croîtra. Nous pouvons maintenant lire le mot « Marie », auquel est associée par un lexique la procédure de la figure 2.9. Après exécution de cette procédure on obtient l'arbre 2.10. Cette procédure peut s'exécuter correctement car le focus est sur un noeud auquel un type e est requis, ce qui remplit la condition d'application de la procédure.

Dans cet arbre et cette procédure, on introduit une variable fraîche x. On introduit également des formules sur lesquelles nous reviendrons rapidement par la suite.

Avant de pouvoir lire « dort » il faut remonter à la racine toutes les informations concernant « Marie ». Pour cela, il faut appliquer, à deux reprises, trois règles non lexicalisées différentes : *complétion*, *élimination* et *thinning*. La première série d'applications produit successivement les figures A.1 à A.3 données en annexe.

La règle de *complétion* a pour effet de remonter au noeud père le focus et le type du fils (qui avait auparavant le focus), ce qui se manifeste ici par $\langle\downarrow_0\rangle Ty(e)$. Cette information est utile pour enlever par *thinning* les requêtes modales sur les types des fils ajoutées au noeud père par la règle d'introduction (comme c'est le cas au noeud racine : $?\langle\downarrow_0\rangle Ty(e)$, $?\langle\downarrow_1\rangle Ty(e \rightarrow t)$). S'il n'y a pas eu *introduction*, comme ici, la remontée du type du fils n'est pas d'une grande utilité[4]. En conséquence, pour la suite de l'exemple, nous omettrons ce genre d'informations dès que possible.

La règle d'*élimination* donne à un noeud son type et sa formule, en fonction de ceux de ses fils : le type est le modus ponens des types des fils, et la formule l'application fonctionnelle du fils droit au fils gauche. La règle de *thinning* enlève la requête de type sur le noeud grâce au type fourni par la règle précédente.

La même série d'applications de règles permet de remonter au noeud racine, avec un thinning supplémentaire (pour enlever l'une des deux requêtes modales sur les fils : $?\langle\downarrow_0\rangle Ty(e)$). On obtient alors la figure 2.11.

Ensuite on peut exécuter la règle d'*anticipation*, qui permet de descendre le focus d'un noeud à un de ses fils qui présente au moins une requête, pour descendre au noeud prédicat (de type requis $e \rightarrow t$). On obtient alors l'arbre 2.12, qui est prêt à recevoir le verbe.

On exécute ensuite la procédure associée à « dort »exposée à la figure 2.13, et comme précédemment on fait remonter les informations concernant ce noeud au noeud racine (application de *complétion*, *élimination* et *thinning*). On remonte également le focus, et alors les conditions de succès de l'analyse sont

[4] Les autres règles pourraient être écrites de façon à tirer profit de cette remontée d'information, mais alors il faudrait remonter d'autres informations, ce qui alourdirait encore le déroulement de l'algorithme.

if	$?Ty(e)$
then	$\text{make}(\langle\downarrow_1\rangle)$; $\text{go}(\langle\downarrow_1\rangle)$; $\text{put}(Fo(\lambda P.\iota, P), Ty(cn \rightarrow e))$; $\text{go}(\langle\uparrow_1\rangle)$; $\text{make}(\langle\downarrow_0\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$; $\text{put}(?Ty(cn))$; $\text{make}(\langle\downarrow_0\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$; $\text{freshput}(x, Fo(x))$; $\text{put}(Ty(e), [\downarrow]\ \bot)$; $\text{go}(\langle\uparrow_0\rangle)$; $\text{make}(\langle\downarrow_1\rangle)$; $\text{go}(\langle\downarrow_1\rangle)$; $\text{put}(Fo(\lambda X.X, Marie'(X)), Ty(e \rightarrow cn))$; $\text{go}(\langle\uparrow_1\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$
else	ABORT

FIG. 2.9 – Procédure associée à l'entrée lexicale « Marie »

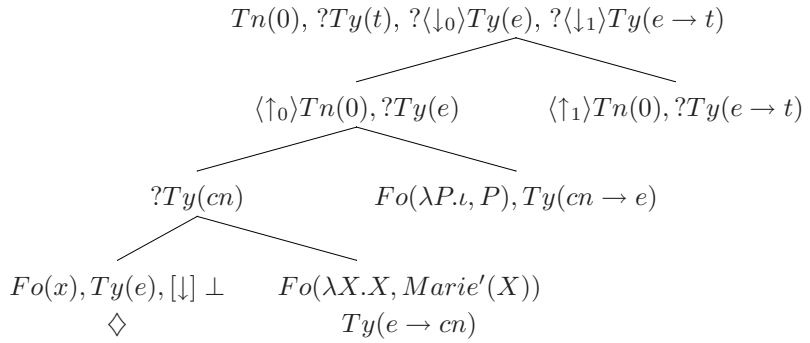

FIG. 2.10 – Arbre après l'exécution de la procédure associée à « Marie »

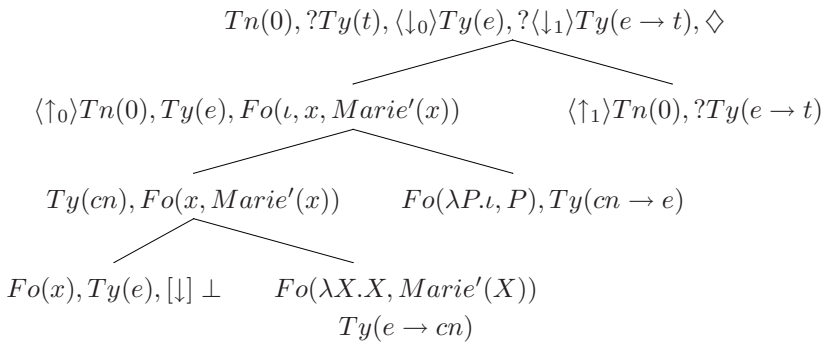

FIG. 2.11 – Arbre après application de complétion, élimination, thinning et thinning

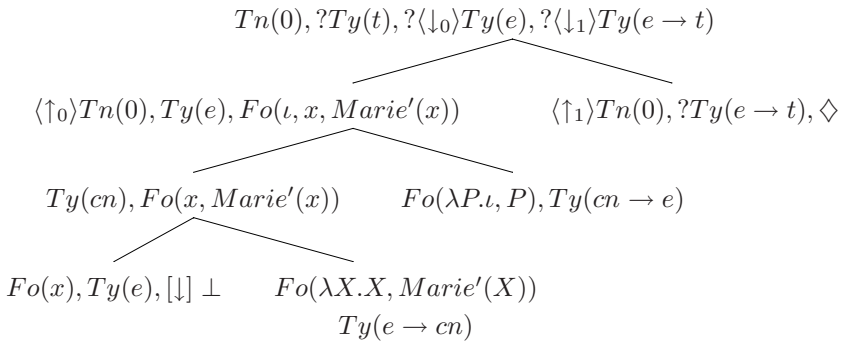

FIG. 2.12 – Arbre après anticipation, avant la lecture de « dort »

remplies et on obtient l'arbre final de la figure 2.14.

if	$?Ty(e \rightarrow t)$
then	$\text{put}(Ty(e \rightarrow t), Fo(\lambda P.Dormir'(P)), [\downarrow] \bot)$
else	ABORT

FIG. 2.13 – Procédure associée à l'entrée lexicale « dort »

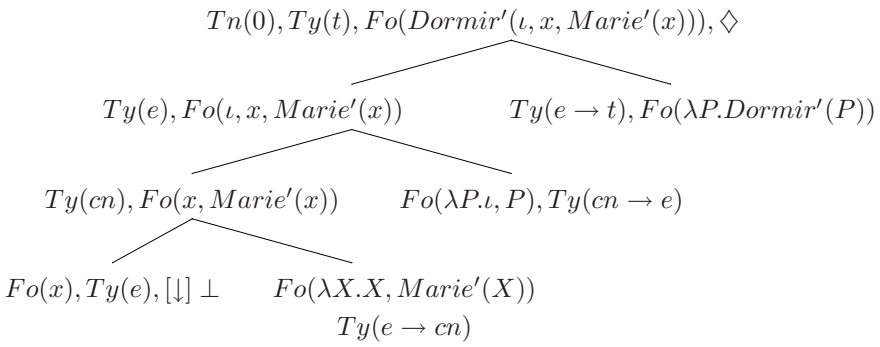

FIG. 2.14 – Arbre final pour « Marie dort »

Revenons maintenant sur le terme $\iota, x, Marie'(x)$. C'est un terme du ϵ-calcul, que nous présentons dans la section suivante. Intuitivement, il désigne l'unique personne appelée Marie à laquelle le locuteur peut faire référence.

Nous reviendrons sur les effets de chacune des règles non lexicalisées dans le chapitre suivant, lorsque nous tenterons d'en transcrire les effets dans une grammaire d'interaction.

2.2.3 Compléments pour traiter des phénomènes plus complexes

Dynamic Syntax a été conçu — dès le départ — pour fournir un cadre au traitement de nombreux phénomènes linguistiques. Pour cela, le formalisme comporte quelques mécanismes supplémentaires à ceux vus dans l'exemple précédent.

Résolution d'anaphores

Lorsqu'un pronom remplace un groupe nominal dans une phrase, on lui attribue le rôle d'*anaphore* ou *substitut du nom*, dont le groupe nominal est l'*antécédent*. En analyse sémantique, lorsqu'un pronom est rencontré, il faut retrouver l'antécédent de cette anaphore (on parle de *résolution de l'anaphore*) afin de pouvoir calculer la formule sémantique de la phrase. L'antécédent d'une anaphore est retrouvé à partir du contexte de l'anaphore, donc des phrases qui la précèdent ou du début de la phrase elle-même. Tous les groupes nominaux du contexte ne sont pas des antécédents possibles, car les pronoms anaphoriques véhiculent des marques de genre et de nombre.

Par exemple, dans les phrases « Pierre aime Marie. Elle le hait. », « elle » porte une marque de genre féminin (nombre singulier), et « le » de genre masculin (nombre singulier). C'est ce qui nous permet de dire que « Pierre » est l'antécédent de « le » et « Marie » de « elle ».

Dynamic Syntax ne comporte pas d'algorithme de résolution des anaphores, car c'est un problème indépendant du formalisme adopté et qui n'a pas encore trouvé de réponse définitive. Cependant, DS doit permettre, une fois qu'un algorithme quelconque a résolu l'anaphore, de donner au pronom la formule de son antécédent. DS réalise cela par une règle non lexicalisée nommée *substitution* qui est formulée comme dans la figure 2.15, où α est l'antécédent trouvé pour U. Cette règle vérifie que la formule de l'antécédent n'est pas déjà présente dans la partie de l'arbre qui constitue le domaine de localité du pronom (sinon,

il y a un problème et l'analyse échoue). Ce domaine de localité est exprimé par la suite de modalités $\langle\uparrow_0\rangle\langle\uparrow_1^*\rangle\langle\downarrow_0\rangle$, dont nous expliquerons le sens par la suite. Par exemple, dans la phrase « Jean le voit », « le » ne peut pas référer à « Jean ». A contrario, les pronoms réflexifs doivent trouver leur antécédent dans leur domaine de localité.

if	$Ty(e), ?Fo(U)$	
then	if	$\langle\uparrow_0\rangle\langle\uparrow_1^*\rangle\langle\downarrow_0\rangle Fo(\alpha)$
	then	ABORT
	else	put$(Fo(\alpha))$
else	ABORT	

FIG. 2.15 – Procédure associée à la règle *substitution(α)*

Structures en tandem

Une subordonnée relative a, tout comme une phrase, sa valeur de vérité. Une subordonnée relative R insérée dans une phrase P a pour originalité d'avoir dans son contexte non seulement le contexte de P, mais aussi le début de P (i.e. la partie de P qui précède l'antécédent de R).

En Dynamic Syntax, R a son propre arbre d'analyse, qui est lié à celui de P par un *lien de type L*, dont l'origine est l'antécédent de la relative. On a ainsi les arbres schématiques de la figure 2.16 pour la phrase « Marie aime un homme qui mange une pomme. ».

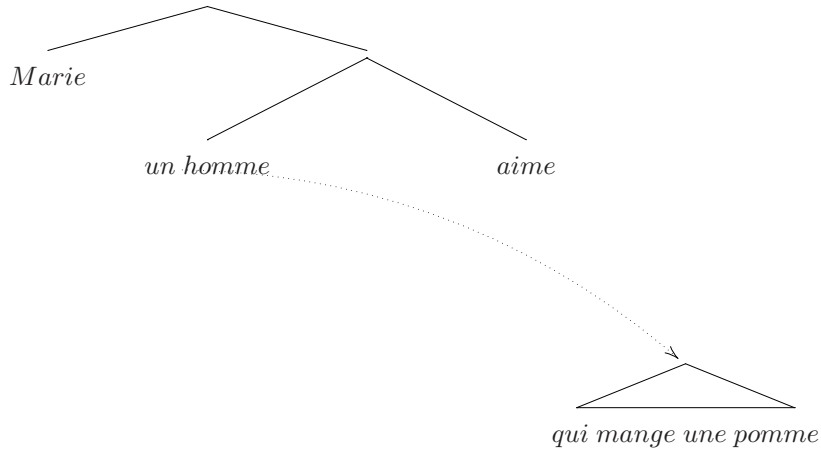

FIG. 2.16 – Arbres en tandem, reliés par un lien L

Dislocation gauche

En anglais, une phrase comme « Joan, Hilary upset. » est tout à fait acceptable. C'est un exemple simple de dislocation gauche : l'objet direct « Joan » est mis en périphérie gauche de la phrase, afin de le mettre en valeur. En DS, une règle non lexicalisée de **-adjonction* permet d'ajouter un noeud A non fixé dans l'arbre en construction. Tout ce qu'on sait de A, c'est qu'il sera dans l'arbre à une profondeur encore indéterminée. La construction de l'arbre se poursuit, et lorsqu'on arrive à l'objet direct manquant, on fixe le noeud non fixé à cet endroit, comme le montrent les figures 2.17 et 2.18. La règle de *fusion*, qui unifie deux noeuds, est appliquée pour fixer le noeud.

On remarque ici que cette dislocation gauche n'a aucune incidence sur la structure finale de l'arbre. En DS, la structure sémantique reste inchangée. Si l'on veut déceler une dislocation gauche a posteriori,

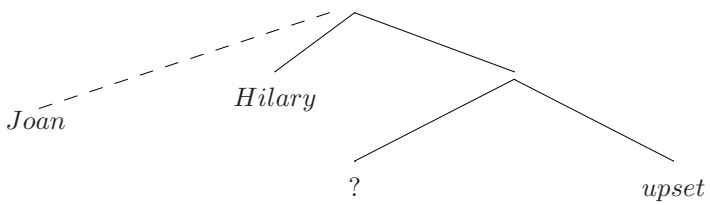

FIG. 2.17 – Arbre avec le noeud portant *Joan* non fixé

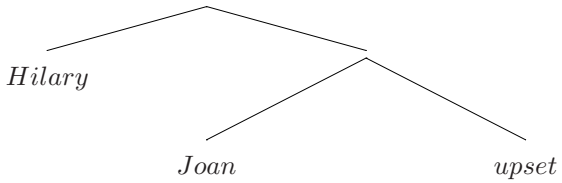

FIG. 2.18 – Arbre final pour la phrase « Joan, Hilary upset »

il faut examiner quelles règles ont été appliquées pour produire l'arbre final.

Nous venons de voir qu'il existe en DS quelques règles non lexicalisées supplémentaires, un nouveau type de lien entre noeuds, et la possibilité de spécifier des sous-arbres non encore fixés. Ces points seront importants par la suite, lorsque nous examinerons plus en détail les convergences et points d'achoppement entre DS et les IG.

2.3 Outils formels

Dynamic Syntax repose pour la description d'arbres sur une logique d'arbres binaires nommée LOFT, qui lui est particulièrement adaptée. Mais DS est un formalisme d'analyse sémantique et en tant que tel doit traiter des problèmes de quantification. Pour cela, le choix fait à ce jour en DS s'est porté sur une variante du *epsilon*-calcul de Hilbert. Nous allons donc maintenant présenter succinctement ces deux outils formels, ce qui nous sera utile à l'heure de comparer DS et les IG.

2.3.1 La logique d'arbres finis (LOFT)

Dynamic Syntax utilise la logique d'arbres finis (ci-après LOFT pour Logic of Finite Trees) [BMvdR95] pour représenter les descriptions d'arbres. C'est une logique modale qui décrit la structure interne d'arbres binaires finis ordonnés. Comme Dynamic Syntax s'inscrit absolument dans la perspective des formalismes qualifiés de *Model-Theoretic Syntax*, ce sont des structures partielles qui sont le plus souvent décrites, et elles sont décrites du point de vue du noeud qui porte le focus. C'est donc un point de vue descriptif interne qui est adopté, plutôt qu'externe. Les modalités de LOFT permettent d'exprimer en DS entre autres les concepts suivants :

- noeud père $\langle\uparrow\rangle$, noeud fils $\langle\downarrow\rangle$
- noeud fils argument $\langle\downarrow_0\rangle$, noeud fils foncteur $\langle\downarrow_1\rangle$
- clôture transitive et réflexive $\langle\uparrow^*\rangle$
- $\langle\downarrow_1^*\rangle$, $\langle\uparrow_0\rangle\langle\uparrow_1^*\rangle\langle\downarrow_0\rangle$ (le domaine de localité présenté précédemment pour la résolution d'anaphores)...

Ces modalités sont combinables avec des formules, pour exprimer qu'une propriété à un autre noeud est requise : $\langle\uparrow\rangle?\phi$ ou établie : $\langle\uparrow\rangle\phi$. Leur utilité a déjà été montrée, entre autres pour la résolution

d'anaphores où $\langle\uparrow_0\rangle\langle\uparrow_1^*\rangle\langle\downarrow_0\rangle$ était le domaine de localité dans lequel $Fo(\alpha)$ ne devait pas être vraie si on voulait que α soit établi comme antécédent d'un pronom. [BMvdR95] contient la définition complète de LOFT, ainsi que les preuves de sa correction, de sa complétude et de sa décidabilité.

2.3.2 L'epsilon-calcul

Pour gérer les termes quantifiés, DS utilise l'ϵ-calcul, défini par Hilbert, dans son « Grundlagen der Mathematik » en 1923. L'intérêt du ϵ-calcul réside dans ses opérateurs de description ϵ, τ et ι, qui permettent de définir explicitement les quantificateurs existentiel ($\exists$)et universel ($\forall$), et plus généralement les symboles mathématiques à l'intérieur de systèmes déductifs formels, comme le quantificateur d'existence et d'unicité ($\exists\,!$).

Descriptions indéfinies

Un epsilon-terme est un terme de la forme $\epsilon_x.A(x)$, avec x une variable et A un prédicat. Ce terme s'interprète, du moins dans la version retenue en DS, comme « un x satisfaisant $A(x)$ s'il existe, sinon un objet arbitraire », grâce à l'axiome *transfini* : $A(y) \Rightarrow A(\epsilon_x.A(x))$.
Les quantificateurs de la logique classique peuvent être encodés en ϵ-termes par les deux équivalences énoncées par Hilbert :
$\exists x, A(x) \Leftrightarrow A(\epsilon_x.A(x))$
$\forall x, A(x) \Leftrightarrow A(\epsilon_x.\neg A(x))$
L'opérateur ϵ permet donc de dissocier la spécification d'un objet de sa preuve d'existence. Cette propriété semble tout indiquée pour traiter les descriptions indéfinies et la résolution d'anaphores dans les langues naturelles, c'est du moins la théorie développée dans la thèse de W.P.M. Meyer-Viol [MV95], ainsi que dans les travaux d'auteurs comme K. von Heusinger, ou en philosophie par H. Slater.

L'opérateur de description τ est défini comme $A(\epsilon_x.\neg A) \equiv A(\tau_x.A)$. On utilise des τ-termes pour rendre compte de la quantification universelle, plutôt que des *epsilon*-termes complexes.

En DS, au final, on trouvera un ϵ-terme au lieu d'un terme quantifié existentiellement et un τ-terme au lieu d'un terme quantifié universellement.

Descriptions définies

Quant aux descriptions définies, elles sont représentées par des ι-termes : $\iota_x.A(x)$ est interprété comme « le x tel que $A(x)$ s'il existe et s'il est unique, sinon un objet arbitraire ». L'opérateur ι se dérive d'ϵ en posant $\iota x.A(x) := \epsilon_x.(unique(A))(x)$, avec $unique := \lambda A.\lambda x.[A(x) \wedge (\forall y, A(y) \Rightarrow x = y)]$.

En DS, on trouvera un ι-terme au lieu d'un terme quantifié existentiellement avec une condition d'unicité.

[Cas07] présente un état des lieux assez complet sur ces opérateurs et traite de leur définition en Coq.

Les epsilon-termes sont souvent assimilés aux fonctions de choix, et une littérature relativement nombreuse leur a été consacrée ces dernières années.

Chapitre 3

Dynamic Syntax à la mode des grammaires d'interaction

3.1 Motivation

Dans les deux formalismes présentés précédemment, IG et DS, l'analyse (syntaxique dans un cas, sémantique dans l'autre) d'un énoncé construit un arbre à partir de descriptions d'arbre partielles éventuellement sous-spécifiées. Ces descriptions d'arbre se superposent, leurs informations se complétant et se précisant (rien n'est détruit) : la construction de l'arbre est un processus monotone croissant. La première condition nécessaire au succès de l'analyse est que tous les noeuds soient fixés (i.e. que les modèles ne contiennent plus aucune relation de domination large non résolue en une suite finie de relations de domination stricte).
Dans les deux formalismes, il y a une notion d'information fournie et d'information requise. En DS, une requête d'information est « effacée » lorsque l'information attendue est fournie. En IG, une requête de ressource (polarité négative) est neutralisée lorsque la ressource attendue est fournie (polarité positive). La seconde condition nécessaire au succès de l'analyse est que plus aucune requête de ressource ne subsiste.
À première vue, Dynamic Syntax et les grammaires d'interaction partagent donc certaines caractéristiques fortes, qui sont précisément celles qui guident leur processus d'analyse et en conditionnent le succès.

En vertu de ce constat, comme les grammaires d'interaction n'intègrent pas encore de dimension sémantique, il paraît judicieux — pour ne pas dire incontournable — du point de vue des IG d'étudier la possibilité d'un rapprochement entre Dynamic Syntax et les IG.
D'autre part, la formulation procédurale de DS, si elle est adaptée à la machine, est particulièrement lourde à appréhender et à manipuler pour un être humain. Les règles computationnelles telles qu'elles sont formulées introduisent de la redondance d'information, avec de nombreuses formules modales à tous les noeuds ou presque, qui sont introduites lors du déploiement de l'arbre (« à la descente ») et enlevées lors de la vérification du contenu des noeuds (« à la remontée »). Enfin, la recherche en DS étant plus axée sur l'explication de phénomènes linguistiques variés dans de nombreuses langues que sur le développement d'une grammaire à large couverture pour une seule langue, le lexique n'est aucunement factorisé. Ces points tendent à rendre Dynamic Syntax peu abordable et constituent donc autant de freins au développement et à la popularisation de ce formalisme. Au final, alors que DS permet de rendre compte très simplement et naturellement de nombreux phénomènes linguistiques habituellement présentés comme problématiques, son temps d'apprentissage est long et le lecteur en retire l'impression que le gain en simplicité du traitement linguistique est annulé par la lourdeur de la logique manipulée. Dynamic Syntax de son côté aurait donc également beaucoup à gagner en étant formulée d'une façon plus déclarative, avec un lexique structuré et factorisé, et en éliminant certaines redondances des règles computationnelles.

3.2 D'une formulation procédurale à une formulation déclarative

3.2.1 Intuition et première approche

Les procédures de DS sont fastidieuses à lire et à manipuler. Pour résoudre ce problème, les membres du Dynamic Syntax Research Group les introduisent sous cette forme, mais parfois aussi sous la forme de la règle de transformation d'arbre correspondante. Nous proposons ici de systématiser cette approche en ne présentant plus de procédure, mais également d'aller plus loin en passant d'une règle de transformation d'arbre (à un sous-arbre *reconnu* on associe le sous-arbre qui le remplace après application de cette règle) à un unique arbre. Pour cela, le système de polarités des IG semble tout à fait adéquat puisqu'il comporte une polarité *virtuelle* qui permet la superposition de noeuds par reconnaissance (et non apport) d'information.

Une analyse « DS façon IG » d'une phrase construit un arbre sémantique à l'aide de descriptions d'arbres polarisées (DAP) des IG selon les principes suivants :

1. l'arbre sémantique en construction se présente sous la forme d'une DAP
2. à chaque entrée lexicale sont associées une ou plusieurs DAP ancrées
3. à chaque règle computationnelle est associée une DAP
4. un algorithme incrémental construit la DAP de la phrase en ajoutant à la DAP en construction soit une DAP du prochain mot non lu de la phrase, soit une DAP de règle
5. le choix de la DAP à ajouter à la DAP en construction est contraint par la position du focus dans la DAP en construction

Maintenant que les grandes lignes sont tracées, nous allons détailler certaines correspondances entre des procédures et règles DS et leurs DAP.

Règles computationnelles

Nous présentons ici les DAP des règles computationnelles dont nous aurons besoin pour traiter notre exemple « Marie dort. ». Nous commençons par le point de départ de toute analyse, qui en DS est la règle d'*initialisation*, modélisée trivialement sous la forme de la DAP 3.1 qui contient un unique noeud.

$$\begin{array}{c} \mathrm{Ty} \leftarrow t \\ \mathrm{focus} \rightarrow m \end{array}$$

FIG. 3.1 – DAP associée à la règle d'initialisation

Ensuite, les règles d'*introduction* et *prédiction* sujet - prédicat seront appliquées. Nous les regroupons en une seule règle, donc une DAP (présentée à la figure 3.2), pour des raisons que nous évoquerons par la suite. Cette DAP comporte aussi des traits à valeurs partagées ($\langle 1 \rangle$ et $\langle 2 \rangle$).

Les deux DAP présentées comportent un trait spécial : le trait *focus* ayant comme seule valeur possible m. Nous reviendrons par la suite sur ce trait, mais les déplacements du focus nécessitent des règles dédiées, comme *complétion* et *anticipation*, auxquelles on ajoute une règle de *finalisation* pour la fin de l'analyse (que l'on pourrait éventuellement ancrer par le point final de la phrase).

Entrées lexicales

Autre exemple plus complexe de transcription, la procédure associée à l'entrée lexicale « Marie » donnée à la figure 3.6 peut être remplacée par la DAP 3.7. La DAP de « Marie » contient un noeud avec

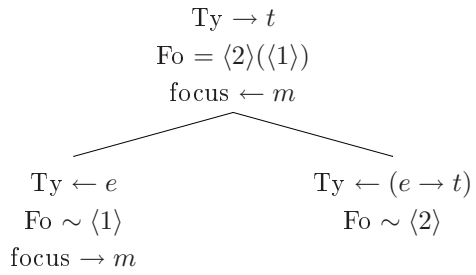

FIG. 3.2 – DAP associée à la règle introduction et prédiction sujet - prédicat

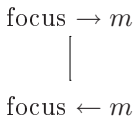

FIG. 3.3 – DAP associée à la règle complétion

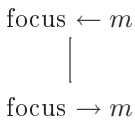

FIG. 3.4 – DAP associée à la règle anticipation

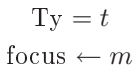

FIG. 3.5 – DAP associée à la règle de finalisation

le trait *prop* (pour property) qui a ici la valeur *closed*, ce qui signifie que le noeud est clos (son nombre de fils est fixé, en l'occurrence à 0).

La DAP de « dort » est plus simple, comme le montre la figure 3.8.

Représentation du focus en IG

Le trait *focus* permet la gestion du focus de DS, selon une idée simple : on donne le focus à un noeud en polarisant positivement son trait focus, on le lui retire en le neutralisant (par ajout d'une polarité négative). Une analyse en DS part de la DAP d'*initialisation*, où le focus est sur le noeud racine (qui est le seul noeud de la DAP à ce moment), et termine lorsque le focus est revenu sur ce noeud racine (et qu'il ne reste aucune requête dans la DAP). À tout instant de l'analyse, un noeud et un seul porte le focus. Pour préserver ces propriétés, la grammaire d'interaction correspondante doit vérifier les propriétés suivantes :

– la DAP d'*initialisation* contient un focus positif sur le noeud racine
– la DAP de *finalisation* contient un focus négatif sur le noeud racine
– toute autre DAP contient un focus positif sur un noeud et un focus négatif sur un autre noeud

La DAP de finalisation est un ajout nécessaire pour respecter la condition de neutralité de la structure à la fin de l'analyse en IG. Immédiatement un problème apparaît, qui est lié au système de polarités présenté précédemment en IG : lorsqu'un trait a été neutralisé on ne peut plus le positiver à nouveau.

if	$?Ty(e)$
then	$\text{make}(\langle\downarrow_1\rangle)$; $\text{go}(\langle\downarrow_1\rangle)$; $\text{put}(Fo(\lambda P.\iota, P), Ty(cn \rightarrow e))$; $\text{go}(\langle\uparrow_1\rangle)$; $\text{make}(\langle\downarrow_0\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$; $\text{put}(?Ty(cn))$; $\text{make}(\langle\downarrow_0\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$; $\text{freshput}(x, Fo(x))$; $\text{put}(Ty(e), [\downarrow]\ \bot)$; $\text{go}(\langle\uparrow_0\rangle)$; $\text{make}(\langle\downarrow_1\rangle)$; $\text{go}(\langle\downarrow_1\rangle)$; $\text{put}(Fo(\lambda X.X, Marie'(X)), Ty(e \rightarrow cn))$; $\text{go}(\langle\uparrow_1\rangle)$; $\text{go}(\langle\downarrow_0\rangle)$
else	ABORT

FIG. 3.6 – Procédure associée à l'entrée lexicale « Marie »

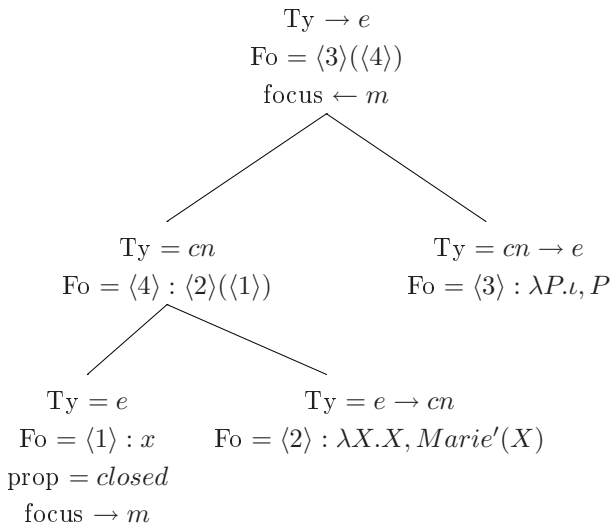

FIG. 3.7 – DAP associée à l'entrée lexicale « Marie », avec valeurs partagées

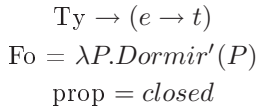

FIG. 3.8 – DAP associée à l'entrée lexicale « dort »

Or les déplacements du focus l'amènent à passer plusieurs fois par le même noeud. Il faudrait donc avoir un système de polarités où la neutralité n'est pas uniquement obtenue par la superposition d'un positif et un négatif, mais par celle de n positifs et n négatifs. Cet ajout au système de polarités ne pose pas de problème majeur donc est tout à fait envisageable pour les futures évolutions des IG, d'autant plus qu'il serait utile dans d'autres situations. Si en théorie on peut considérer ce point comme acquis, en pratique Leopar ne gère pas encore ce type de polarités et donc ne peut gérer le focus.

Clôture d'un noeud

Le trait *prop* associé à la valeur *closed* permet de spécifier qu'un noeud est clos, c'est-à-dire qu'il n'a pas de fils. Cela correspond donc à la restriction $[\downarrow]\ \bot$ de DS, sauf que cette restriction en DS ne s'applique pas aux liens L. Nous y reviendrons dans le chapitre suivant, car ce n'est pas faisable en IG actuellement.

3.2.2 Exemple : analyse de « Marie dort »

La DAP de la règle d'*initialisation* sert de point de départ à l'analyse, c'est la figure 3.9.

$$\begin{array}{c}\text{Ty} \leftarrow t\\ \text{focus} \rightarrow m\end{array}$$

FIG. 3.9 – DAP au début de l'analyse de « Marie dort »

Puis on lui ajoute la DAP d'*introduction et prédiction sujet - prédicat*, ce qui donne la DAP de la figure 3.10.

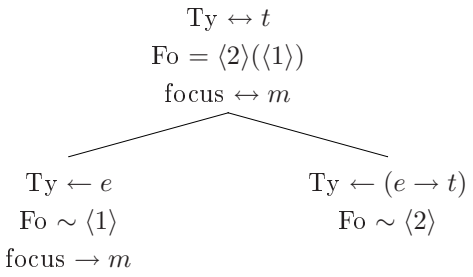

FIG. 3.10 – DAP après introduction et prédiction sujet - prédicat

Pour ne pas surcharger les figures, toutes les suivantes ne mentionneront le trait *focus* qu'au noeud qui le porte à ce moment de l'analyse.
On peut lire l'entrée lexicale « Marie », ce qui donne la figure 3.11.

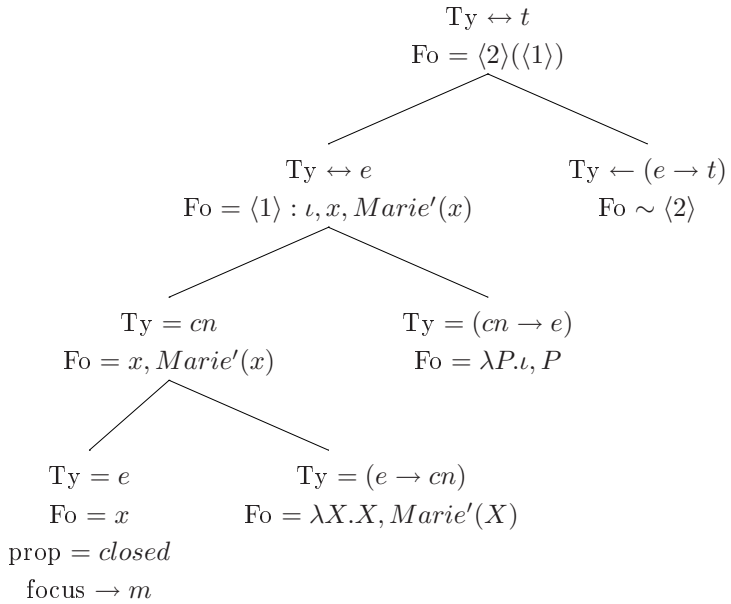

FIG. 3.11 – DAP après lecture de « Marie »

On peut maintenant remonter le focus à la racine, par 3 applications successives de la règle de *complétion*, ce qui donne la figure 3.12.

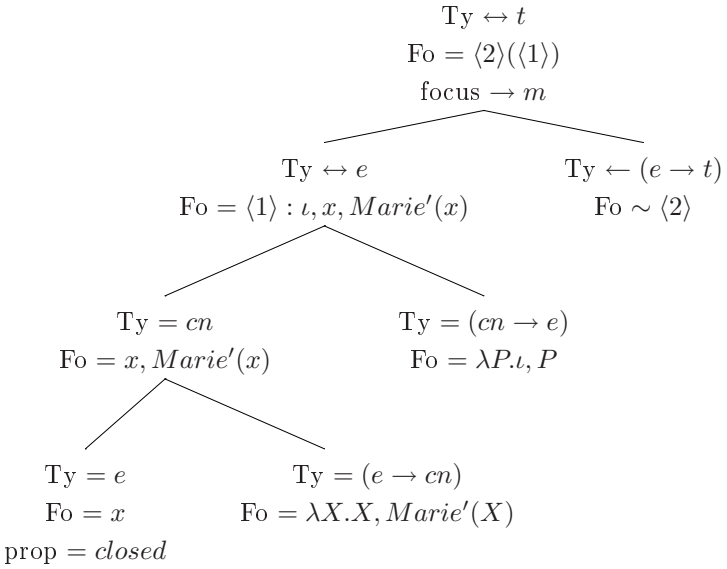

FIG. 3.12 – DAP après 3 complétions

Puis une application d'*anticipation* permet de déplacer le focus au noeud prédicat, ce qui donne la figure 3.13.

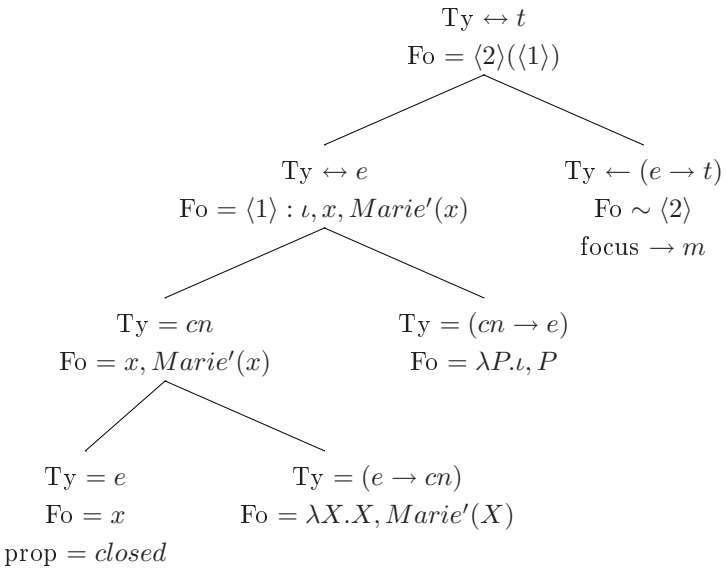

FIG. 3.13 – DAP après anticipation

On lit « dort » et on obtient la DAP de la figure 3.14, qui ne fait pas apparaître les co-indexations par souci de clarté.
Ensuite on remonte le focus à la racine à l'aide de la règle de *complétion* et on obtient la DAP de la figure 3.15.
Enfin on termine l'analyse à l'aide de la règle de *finalisation* et on obtient l'arbre neutralisé de la figure 3.16.

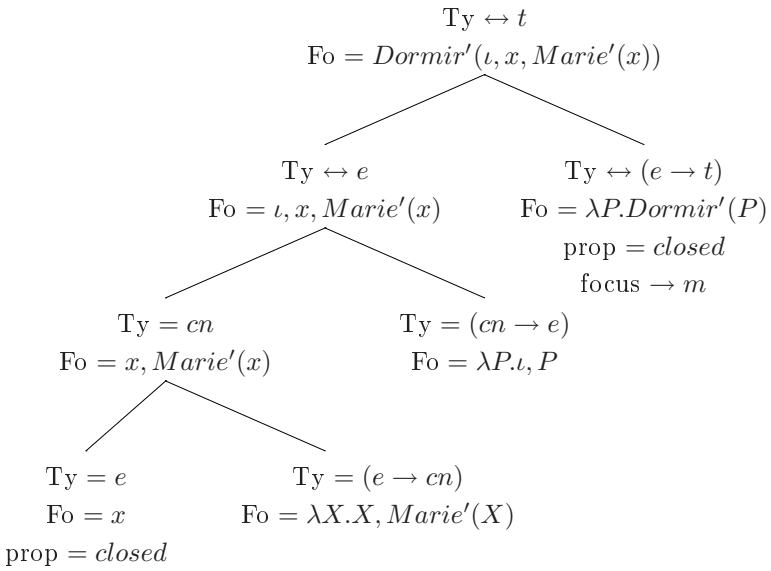

FIG. 3.14 – DAP après lecture de « dort »

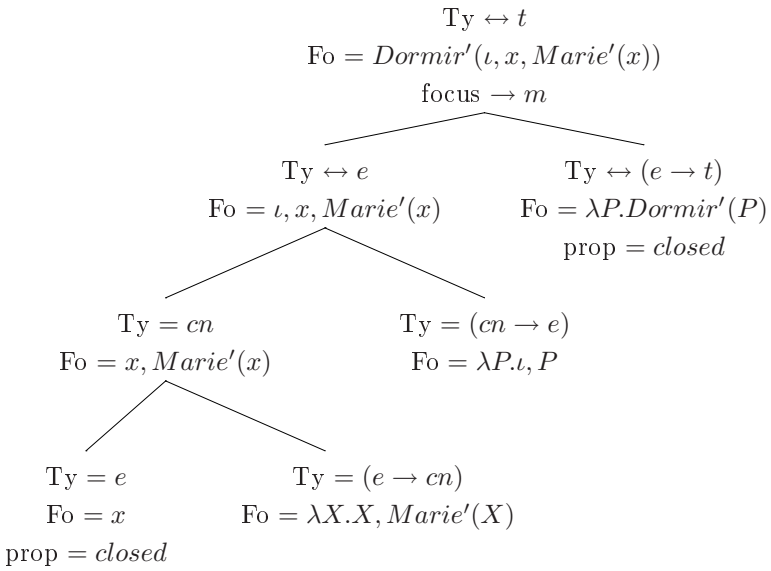

FIG. 3.15 – DAP après complétion

3.2.3 Le problème de redondance, ses solutions et leurs conséquences

La solution présentée jusqu'à présent n'est pas une simple transcription de DS en IG, car elle s'attaque au problème de la redondance d'informations et de règles appliquées au cours d'une analyse en DS.

Le problème de redondance

Dans un arbre DS complet (donc plus chargé que les arbres présentés dans le chapitre 2 qui ne montraient que les informations nouvelles et importantes), pendant la croissance de l'arbre, tant qu'un sous-arbre n'est pas complet, il contient toujours des requêtes à sa racine. Quand on ajoute des fils à un

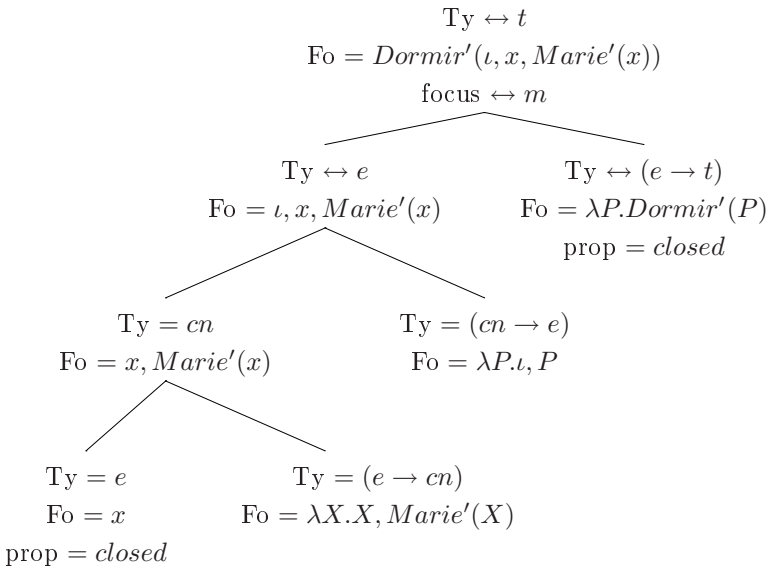

FIG. 3.16 – DAP après finalisation

noeud, on ajoute à ce noeud des requêtes modales sur les types de ses fils et on ne fixe pas le type de ce noeud. Les requêtes sur les types des fils sont utilisées pour construire ceux-ci.
Plus tard dans l'analyse, les informations sur les types des fils remonteront au père, ce qui permettra de supprimer ces requêtes, de fixer le type du père et de lui donner sa formule. On a besoin pour cela des règles transcrites pour les besoins de l'exemple (*introduction sujet-prédicat*, *prédiction sujet-précidat*, *complétion*, *élimination*) et une que nous n'avons sciemment pas transcrite (*thinning*).

On voit ici clairement que les requêtes de type sont « en double » car elles sont écrites à la fois sur le noeud père et sur le noeud fils. Ce type ne peut pas être modifié, car l'arbre croît de façon monotone, donc celui qui sera fixé au fils est exactement celui qui était requis au père et au fils. De plus la remontée du focus en DS est conçue pour empêcher que le focus ne remonte d'un noeud vers son père tant que le type de ce noeud n'a pas été fixé [5]. Donc, si le focus remonte depuis un noeud, c'est que tout le sous-arbre de ce noeud est correctement typé.

Solutions proposées

Pour éviter la redondance que nous venons de mettre en évidence, qui complique inutilement le déroulement d'une analyse, nous proposons de fixer le type des noeuds « à la descente » : dès que l'on ajoute des fils B et C à un noeud A, on fixe le type de A, car celui-ci sera correct (et fourni) si les types de B et C le sont.
S'il y a un problème, il sera dans le sous-arbre de B ou de C et de toute façon l'analyse ne pourra donc pas réussir.

Quant à la formule de A, sa valeur est définie au même moment comme étant le résultat de l'application fonctionnelle de la formule de C à la formule de B. Un mécanisme de valeurs partagées permet de faire remonter les valeurs exactes des formules de C et B dès qu'elles sont fixées.

Ainsi, toutes les informations sont fixées à la descente. Ne pas avoir à remonter toutes ces informations permet d'avoir un arbre moins chargé et le déroulement de l'analyse s'en trouve facilité.

Cela a les conséquences suivantes sur les règles computationnelles de DS :
– les règles d'*introduction* et de *prédiction* sont fusionnées
– la règle de *thinning* est naturellement modélisée en IG par la neutralisation des polarités

[5] C'est une condition indispensable pour respecter la « right roof constraint » qui semble être un invariant linguistique.

- la règle de *complétion* est modélisée par un arbre qui déplace le focus du noeud courant vers son père, si le type du noeud courant est fixé (la remontée d'informations est rendue inutile par la fixation du type du père à la descente)
- la règle d'*élimination* est modélisée par les valeurs partagées et la fixation du type du père à la descente

Le seul problème que nous avons est que nous ne pouvons pas en IG modéliser la condition imposant que le type d'un noeud soit fixé avant qu'on puisse lui appliquer la règle de *complétion*. Nous reviendrons sur ce point au chapitre suivant.

3.3 Factorisation du lexique par écriture d'une méta-grammaire

Les recherches du Dynamic Syntax Research Group portent sur la façon de traiter de nombreux phénomènes linguistiques issus de langues très différentes dans le cadre du formalisme Dynamic Syntax. Le but est autant de valider le formalisme en mettant en évidence la simplicité du traitement du phénomène étudié en DS que de montrer que l'on peut traiter ce phénomène dans l'absolu (et comment on peut le traiter). Dans chaque publication utilisant DS, le formalisme est présenté, avec les règles et procédures nécessaires à l'analyse des exemples fournis (et adaptés à la langue cadre). Chaque publication contient ainsi un petit fragment de grammaire, mais il n'existe à ce jour aucune grammaire de taille significative pour une langue.

Les IG, quant à elles, ont vocation à servir de cadre au développement de grammaires à large couverture ancrées lexicalement. Il a donc fallu réfléchir à la meilleure façon d'organiser le lexique syntaxique, ce qui a donné l'organisation présentée à la figure 1.6. La même organisation est tout à fait transposable à notre proposition de Dynamic Syntax façon grammaires d'interaction, sans que cela ne pose de problème particulier. Au contraire, cela permet d'ajouter facilement et rapidement de nouvelles entrées lexicales.

Ce qui permet une telle facilité d'écriture et de maintenance, c'est la *représentation modulaire* que permet la traduction du lexique de DS en IG. Les IG manipulent des descriptions d'arbres et les descriptions d'arbres sont beaucoup plus faciles à factoriser que des procédures (qui en DS ne sont pas factorisées). Par exemple, pour ajouter l'entrée lexicale « Jean » en IG, il suffit d'ajouter le lemme *Jean* à la liste des lemmes, avec son unique forme fléchie *Jean* et d'associer ce lemme à une famille de lemmes (par exemple celle des noms propres masculins singuliers désignant des êtres animés). Entre les arbres de « Pierre » et « Jean », seuls l'ancre lexicale et le prédicat changeront.

L'écriture de la méta-grammaire XMG correspondante est similaire dans son principe à celle des grammaires syntaxiques actuelles des IG, mais les besoins sont légèrement différents notamment à cause des types récursifs et de l'application fonctionnelle qui sont omniprésents dans la construction des arbres d'analyse en DS. Ces besoins nouveaux ne sont pas insurmontables en l'état, mais devraient être traités pour pouvoir écrire avec un réel confort une méta-grammaire DS-IG.

Il serait intéressant de déterminer dans quelle mesure on pourrait réutiliser les ressources existantes en IG et avoir un lexique « unique » syntaxe IG/sémantique à la DS, mais cela dépasse le cadre de cette étude et la pertinence même de cette démarche est discutable, comme nous le verrons par la suite.

3.4 Limites de l'expérimentation concrète de cette approche

Le travail théorique décrit précédemment n'a pu être réellement testé, pour plusieurs raisons que nous exposons dans cette section.

3.4.1 Problèmes d'implantation

Les règles en DS ne sont pas lexicalisées et sont optionnelles, ce qui entraîne un grand nombre d'analyses à mener en parallèle. Or Leopar ne sait gérer que des DAP lexicalisées, et ne sait pas gérer des DAP

optionnelles. Nous avons tout de même écrit une grammaire jouet dans laquelle les règles étaient ancrées. Cela force à inclure les règles dans la phrase et donc à prévoir l'analyse (parmi les différentes possibles éventuellement) qui sera faite. On perd alors l'intérêt des analyses de DS. De plus, comme nous avons pu le signaler, certaines modalités de DS ne sont pas exprimables en IG. Toute une partie des informations nécessaires à l'analyse de DS est orthogonale aux IG et nécessiterait l'ajout d'une nouvelle dimension dans la grammaire. C'est le cas lorsqu'on veut tester que le type d'un noeud a été fixé avant de remonter le focus à son père mais aussi si l'on veut tester qu'un trait d'un noeud n'a pas encore eu de valeur fixée jusqu'à présent, ou si l'on veut forcer que ce trait acquière une valeur atomique avant la fin de l'analyse. Nous reviendrons sur ces problèmes dans le dernier chapitre.

Il n'existe pas de moyen de gérer un mécanisme d'adresses de Gorn dans Leopar, actuellement. Ce n'est pas forcément indispensable si on a cette dimension supplémentaire dans la grammaire, dans laquelle on peut exprimer des modalités. Il n'y a pas de lien L en IG et donc dans Leopar, ce qui empêche de traiter les subordonnées relatives comme nous le verrons dans le dernier chapitre.

Enfin, nous avons dû ajouter plusieurs DAP juste pour gérer le focus. De la même façon, cette information devrait être gérée dans une autre dimension, car elle ne concerne que le déroulement de l'analyse et pas l'arbre final.

Il faudrait donc ajouter une dimension à XMG sur laquelle un algorithme d'analyse implanté dans Leopar pourrait s'appuyer.

3.4.2 Couverture de la grammaire

Les règles de grammaire de DS étudiées traitent de nombreux phénomènes linguistiques. Parmi ceux-ci, on trouve les liens anaphoriques, les subordonnées relatives (restrictives et non restrictives), et donc les phénomènes de crossover (qui résultent de l'interaction d'une anaphore avec une subordonnée relative) auxquels une abondante littérature est consacrée. Une grande attention est également portée aux phénomènes situés aux périphéries gauche et droite de la phrase : Clitic Left Dislocation, Hanging Topic Left Dislocation, topicalisation, effet de focus, extraposition, pronoms résomptifs, effet de récapitulation, dislocation droite, inversion du sujet... Les problèmes de portée de quantificateurs rencontrés dans les fameuses *donkey sentences* légitiment l'emploi des ϵ-termes. Les langues à *scrambling* sont également étudiées.

Il est difficile de définir précisément la couverture de la grammaire produite. Tous ces phénomènes sont traités en DS avec un grand nombre de règles et d'entrées lexicales, qui dépendent des langues étudiées. Il ne nous a pas été possible de les rassembler toutes pour définir, langue par langue, la couverture de la grammaire.

De plus, de nombreux phénomènes nécessitent des traitements supplémentaires à la construction d'un arbre sémantique. C'est le cas de la résolution d'anaphores et des problèmes de portée de quantificateurs (et à travers eux la dépendance des groupes nominaux), pour n'en citer que les principaux. L'arbre est construit de manière à fournir les informations requises par l'algorithme de résolution proposé pour le phénomène en DS. Ces algorithmes ne sont pas à modéliser en IG, mais font partie du « framework » DS.

Chapitre 4

Entre Dynamic Syntax et grammaires d'interaction : marges de progrès, divergences fortes et questions en suspens

À la lumière de l'approche « naïve » ébauchée au chapitre précédent, Dynamic Syntax et les grammaires d'interaction sont des formalismes suffisamment éloignés l'un de l'autre pour qu'une approche IG de DS préserve tout DS en y ajoutant les avantages des IG. Ce serait évidemment la situation idéale au terme de ce travail, mais nous allons détailler ici tout ce qui nous en empêche.

4.1 Descriptions d'arbres : LOFT versus IG

Pour les descriptions d'arbres, DS utilise LOFT alors que les IG ont leur propre langage. Or LOFT permet d'exprimer plusieurs notions que le langage des IG est impuissant à modéliser correctement.

Premier problème, les liens L relient un noeud d'un arbre à la racine d'un autre arbre qui est construit en tandem avec le premier. On peut déjà remarquer qu'il n'existe entre des noeuds, en IG, que des relations de domination et de précédence, et donc qu'on ne peut exprimer simplement les liens L. Une solution pourrait être de *modéliser un lien L par une relation de domination*, et d'ajouter un nouveau trait *link* avec trois valeurs possibles : *start* si le noeud est point de départ d'un lien L, *end* si le noeud est point d'arrivée d'un lien L, et *none* sinon. Mais cette solution est vite mise en échec lorsqu'on tente de l'intégrer à une analyse de type DS, car trop rigide. Par exemple, comment clore un noeud (dire que la liste de ses fils directs est fixée définitivement) tout en laissant ouverte la possibilité qu'il soit antécédent d'une relative, si le lien L est modélisé par une relation de domination ?
Il faudrait donc ajouter la relation de lien L entre deux noeuds à la liste des relations des IG.

Deuxième problème, comment spécifier qu'un noeud n'a pas de père en IG ? Leopar reconnaît la racine de l'arbre d'après sa catégorie syntaxique. LOFT permet de l'exprimer par la modalité $[\uparrow]\,\bot$, qui signifie que pour tous les noeuds qui domineraient ce noeud, $\bot$ est vrai. Donc aucun noeud ne domine ce noeud.

Troisième problème, de nombreuses modalités ne peuvent pas être écrites en IG. Le langage de descriptions d'arbres des IG ne contient aucun mécanisme pour exprimer des modalités fines (telles que celle du domaine de localité : $\langle\uparrow_0\rangle\langle\uparrow_1^*\rangle\langle\downarrow_0\rangle$). Or, en DS, avec des suites de modalités on peut décrire des placements très précis, ou sous-spécifiés mais contraints assez finement, pour des noeuds. Avec une autre dimension pour ces modalités, on pourrait également faire la différence entre deux types de noeuds définis par une relation de domination large : d'un côté, les noeuds encore non fixés dans la DAP en construction, de l'autre, les noeuds déjà fixés dans la DAP mais dont on sait qu'ils sont dominés largement par un autre noeud.

Pour notamment ces trois raisons, IG est moins expressif que LOFT, et moins expressif que le sous-ensemble de LOFT utilisé par DS. En enrichissant le langage de descriptions d'arbres des IG de façon à résoudre au moins les trois problèmes soulevés, on devrait pouvoir faire des descriptions d'arbres « statiques » aussi complètes et complexes en IG qu'en DS.

DS a une dernière différence statique avec les IG : tous les arbres des IG sont lexicalisés, ce qui n'est pas le cas des arbres DS. On peut tout à fait considérer des IG non lexicalisées, mais les conséquences en terme de calculabilité ne sont pas négligeables.

4.2 Linéarité de Dynamic Syntax

Pour qu'une règle puisse s'appliquer en DS, il faut que certaines conditions d'application de la règle soient remplies. Mais ces conditions peuvent porter sur l'état de l'arbre en construction, qui ne seront plus vérifiées dans l'arbre final. Par exemple, certaines règles pour s'appliquer requièrent qu'à un noeud donné, un trait n'ait pas encore reçu de valeur associée, ou qu'un noeud donné ne domine pour le moment aucun autre noeud. Bien sûr, cette condition ne sera plus valable à un autre moment de l'analyse et alors la règle ne pourra pas s'appliquer.
Cela signifie que l'ordre d'application des règles au cours de l'analyse a de l'importance, éventuellement pour rendre compte de certains effets linguistiques, mais également parce que ces règles forment un système de réécriture non associatif et non commutatif.

La linéarité de DS, couplée au mécanisme d'adresses de Gorn, permet également de faire la différence entre un noeud A encore non fixé mais dont on sait qu'il est dominé largement par un noeud B, et un noeud A déjà fixé qui est dominé largement par B.

En IG par contre, les contraintes sont monotones : toute contrainte fournie par une DAP lexicalisée est ajoutée aux contraintes de la DAP de la phrase, et cette contrainte restera vraie dans la DAP finale.

En résumé, il faudrait ajouter à la grammaire IG une dimension de contraintes sur l'arbre en construction alors que pour le moment on ne peut poser de contraintes que sur la description d'arbre polarisée. L'autre solution, qui paraît moins complète et moins élégante, serait d'associer à chaque règle et procédure deux arbres : si le premier est reconnu, le second est ajouté à la DAP. On aurait alors une grammaire de couples d'arbres.

4.3 Les zones d'ombre de DS

Malgré l'abondante littérature produite par le Dynamic Syntax Research Group (DSRG), de nombreux pans du formalisme restent à bâtir.

Ainsi, comme dans les théories exposées par Steedman, Beyssade ou Winter, les groupes nominaux indéfinis et définis sont considérés comme dépendants d'un événement, d'une situation, d'un autre groupe nominal... Mais aucun modèle d'événements clair n'a été défini pour le moment, ce qui empêche d'avoir un ensemble cohérent permettent de tester cette théorie sur quelques phrases.
Sans modèle d'événements clair, il est également difficile de définir les différentes β-réductions qui doivent s'appliquer aux λ-termes construits pour générer toutes les interprétations voulues et seulement elles. [MV95] en énonce quelques-unes, mais elles ne traitent pas tous les cas possibles.
Sans modèle d'événements clair, il est également impossible de gérer la temporalité, marquée entre autres par le mode et le temps des verbes.

Un peu de la même façon, aucune décision n'a été prise pour choisir la variante du ϵ-calcul à adopter : celle de Meyer-Viol, ou celle, indicée, de von Heusinger ? Le choix même du ϵ-calcul prête à discussion.
Selon ses promoteurs, différentes observations linguistiques mettent en évidence le fait qu'un groupe nominal doit être de type e, ce qui est le cas des termes construits avec les opérateurs de description. De plus, ces termes permettraient de rendre compte naturellement de l'interprétation « E-type » de certains pronoms et évite certains problèmes de portée dans les *donkey sentences*. Les ϵ-termes seraient en outre équivalentes aux fonctions de choix, auxquelles toute une autre littérature est consacrée.
Mais selon certains de ses détracteurs, ces termes ne sont pas nécessaires, car on peut construire des

termes beaucoup plus simples (Beyssade), ou des fonctions au moins aussi expressives à la sémantique claire comme les fonctions de Skolem (Steedman), ou un « type raising » est indispensable (Winter) ...

Les problèmes liés aux événements et aux ϵ-termes ne sont pas simples à trancher, car on touche rapidement au coeur des problèmes des formalismes d'analyse sémantique actuels.

Un autre problème complexe est celui de la non lexicalisation de certaines règles. Peut-être est-il possible de définir un algorithme d'analyse permettant de limiter le nombre d'applications successives de règles non lexicalisées. Dans le cas contraire, une implantation réaliste à large couverture de DS serait infaisable. Cette implantation devrait a priori se faire sous la forme d'un algorithme de type *coin gauche*, qui paraît le plus adapté à DS.

Un autre obstacle à une implantation réaliste à large couverture est le nombre important de branchements possibles dans l'analyse, à chaque étape. Cela implique de mener de nombreuses analyses en parallèle, ce qui est toujours fastidieux. Les éléments permettant d'éliminer certaines options sont extérieurs, donc compliqués à intégrer. Ainsi, pour faire la différence entre une relative restrictive et une non-restrictive, la ponctuation (à l'écrit) et l'intonation (à l'oral), mais également la pragmatique sont des indices possibles.

Mais d'autres zones d'ombre de DS sont elles beaucoup plus abordables.
Ainsi, aucune structure de traits n'est rigoureusement définie pour les pronoms alors qu'un tel ajout n'aurait pas d'impact sur le reste du formalisme, et que les informations de genre et de nombre contenues par les pronoms sont indispensables à des mécanismes comme la résolution d'anaphores.

Conclusion

Nous nous proposions dans cette étude d'examiner les deux formalismes que sont Dynamic Syntax et les grammaires d'interaction, afin de tenter de modéliser le premier dans le deuxième. Leurs caractéristiques communes, bien qu'intéressantes, ne sont pas suffisantes pour permettre un tel rapprochement à l'heure actuelle.

Ce travail a permis de mettre en évidence le manque d'expressivité du langage de descriptions d'arbres des grammaires d'interaction, comparé à un langage comme celui de LOFT. Ce n'est pas rédhibitoire, car les IG pourraient évoluer dans ce sens. Une étude plus approfondie des conséquences qu'auraient de tels changements serait évidemment requise.

L'aspect dynamique (linéaire) du traitement en DS pourrait être géré dans une nouvelle dimension de la grammaire, qui permettrait d'exprimer notamment des contraintes sur l'arbre d'analyse en cours. Mais le principal obstacle est constitué par le flou qui entoure de nombreux points de Dynamic Syntax. Ce formalisme est ambitieux, donc incomplet à plusieurs égards et difficile à automatiser réellement. Il a cependant de réelles qualités que sa formulation actuelle masque en partie, et gagnerait indubitablement à être formulé comme nous avons tenté de le faire, de façon déclarative et avec un lexique structuré comme en IG.

L'élément de DS qui est le plus polémique est sans aucun doute l'utilisation d'une variante du ϵ-calcul, qui n'est pas totalement définie et semble pouvoir être remplacée par plusieurs approches concurrentes plus claires sans trop impacter le reste du formalisme.

Les publications sur DS présentent deux problèmes majeurs. Premièrement, DS n'est pas fortement lexicalisé, ce qui rend chaque étape de l'analyse plus complexe à gérer. Deuxièmement, les phénomènes linguistiques traités en DS sont de réels problèmes du domaine, mais pour être pleinement convaincu des bénéfices que DS apporte, il faut déjà avoir une certaine connaissance du phénomène traité. C'est la combinaison de ces deux facteurs qui rend DS aussi difficile à aborder, et les propositions que nous faisons ici ne peuvent atténuer que légèrement le premier problème.

Travailler sur une formulation de Dynamic Syntax en grammaires d'interaction aura permis d'ouvrir un certain nombre de questions sur les deux formalismes, qui ne peuvent être que bénéfiques à leur développement respectif.

Annexe A

Arbres intermédiaires de l'analyse de « Marie dort »

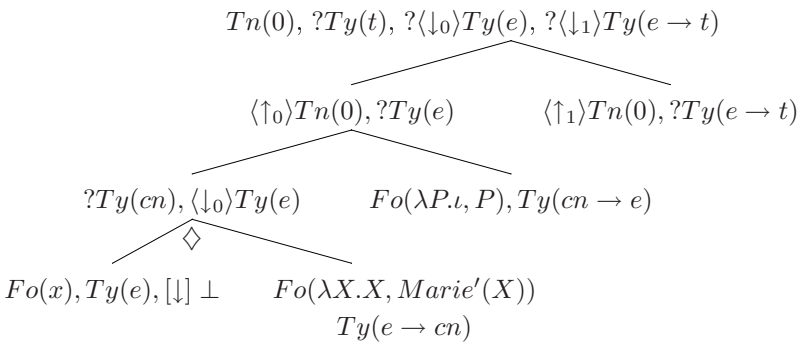

FIG. A.1 – Arbre après complétion

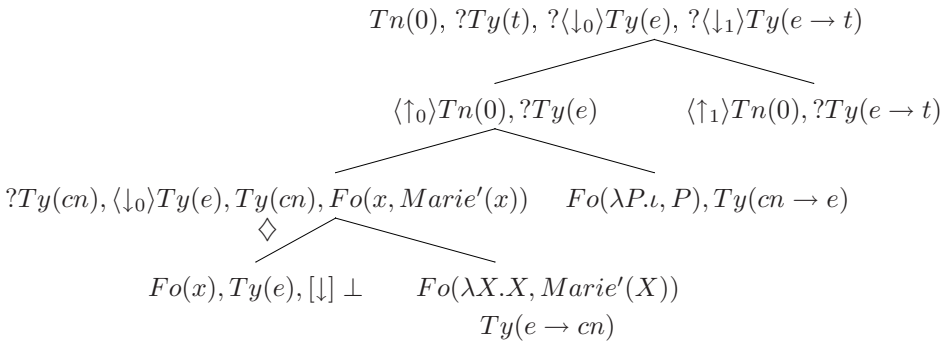

FIG. A.2 – Arbre après élimination

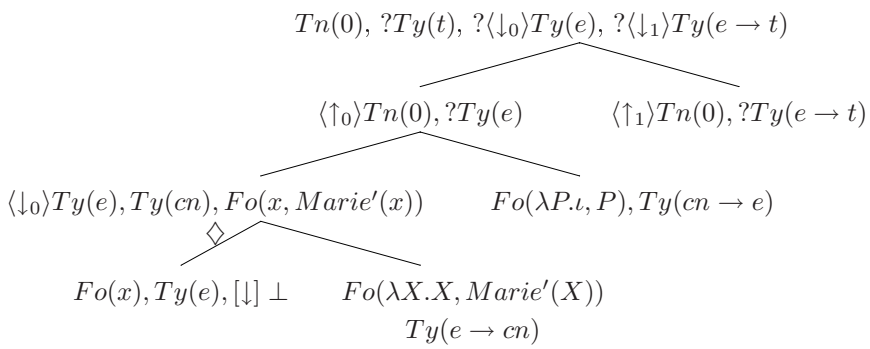

FIG. A.3 – Arbre après thinning

Bibliographie

[Adj35] K. Adjukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1 :1–27, 1935.

[BMvdR95] Patrick Blackburn, Wilfried Meyer-viol, and Maarten de Rijke. A proof system for finite trees. In *125*, page 19, ISSN 0169-118X, 30 1995. Centrum voor Wiskunde en Informatica (CWI).

[Cas07] Pierre Castéran. Utilisation en Coq de l'opérateur de description. In *Journées Francophones des Langages Applicatifs - JFLA07*, 2007.

[CKM05] Ronnie Cann, Ruth Kempson, and Lutz Marten. *The Dynamics of Language–an Introduction*. Elsevier Academic Press, 2005.

[DLP05] D. Duchier, J. Le Roux, and Y. Parmentier. XMG : un Compilateur de Métagrammaire Extensible. In *Conference Traitement Automatique des Langues Naturelles (TALN'2005)*, Dourdan, 2005.

[KMVG01] Ruth Kempson, Wilfried Meyer-Viol, and Dov Gabbay. *Dynamic Syntax–The Flow of Language Understanding*. Blackwell, 2001.

[Mon74] Richard Montague. *Formal Philosophy*. Yale University Press, 1974.

[MV95] W.P.M. Meyer-Viol. *Instantial Logic : an investigation into reasoning with instances*. PhD thesis, Universiteit Utrecht, 1995.

[Per02] Guy Perrier. Descriptions d'arbres avec polarités : les grammaires d'interaction. In *9ème Conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'02), Nancy, France, 2002*, 2002.

[Per03] Guy Perrier. Les grammaires d'interaction. In *Habilitation à diriger des recherches*. Université Nancy 2, 2003.

[Per05] Guy Perrier. La sémantique dans les grammaires d'interaction. *Traitement Automatique des Langues (T.A.L.)*, 45(3) :123–144, 2005.

[PS01] Geoffrey K. Pullum and Barbara C. Sholz. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In P. de Groote, G.Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics : 4th International Conference, LACL 2001, Le Croisic, France, June 27-29, 2001, Proceedings*, pages 17–43. Springer Berlin / Heidelberg, 2001.

[VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Comput. Linguist.*, 18(4) :481–517, 1992.