La coordination dans les grammaires d'interaction

Joseph Le Roux

Département de formation doctorale en informatique

Institut National Polytechnique de Lorraine

École doctorale IAEM Lorraine

La coordination dans les grammaires d'interaction

THÈSE

présentée et soutenue publiquement le 17 octobre 2007

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Joseph Le Roux

Composition du jury

Rapporteurs :	Professeur Alain Lecomte,	Université de Paris 8
	Professeur Aarne Ranta,	Université de Göteborg
Examinateurs :	Professeur Denys Duchier,	Université d'Orléans
	Professeur Owen Rambow,	Université de Columbia
	Professeur Françoise Simonot-Lion,	École des Mines de Nancy
	Professeur Guy Perrier,	Nancy Université (directeur de thèse)

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

Mis en page avec la classe thloria.

Résumé

Cette thèse présente une modélisation des principaux aspects syntaxiques de la coordination dans les grammaires d'interaction de Guy Perrier [Per03]. Les grammaires d'interaction permettent d'expliciter la valence des groupes conjoints. C'est précisément sur cette notion qu'est fondée notre modélisation.

Nous présentons également tous les travaux autour de cette modélisation qui nous ont permis d'aboutir à une implantation réaliste : le développement du logiciel XMG et son utilisation pour l'écriture de grammaires lexicalisées, le filtrage lexical par intersection d'automates et l'analyse syntaxique.

Mots-clés: linguistique computationnelle, traitement automatique des langues, syntaxe formelle, coordination, grammaires d'interaction, analyse syntaxique, théorie des automates

Abstract

This thesis presents a modelisation of the main syntactical aspects of coordination using Guy Perrier's Interaction Grammars [Per03] as the target formalism. Interaction Grammars make it possible to explicitly define conjuncts' valencies. This is precisely what our modelisation is based upon.

We also present work around this modelisation that enabled us to provide a realistic implementation : lexicalized grammar development (using our tool XMG), lexical disambiguation based on automata intersection and parsing.

Keywords: computational linguistics, natural language processing, formal syntax, coordination, interaction grammars, parsing, automata theory

Remerciements

Je tiens à remercier en premier lieu mon encadrant, Guy Perrier, qui m'a toujours soutenu et encouragé durant ces quatre années. Il m'a beaucoup aidé dans mes réflexions et je lui dois beaucoup. Je le remercie également d'avoir relu avec attention les versions successives de ce document et d'avoir à chaque fois supporter les approximations, les erreurs et les fautes d'orthographe.

Je remercie également les rapporteurs, Alain Lecomte et Aarne Ranta, d'avoir accepté de s'intéresser à mon travail et d'avoir indiqué avec une grande rigueur leurs observations. Je veux aussi remercier les autres membres du jury. En particulier, je remercie Denys Duchier qui a initié le travail sur XMG. Son trop bref passage à Nancy a donné lieu à trois thèses, dont celle-ci. Je remercie Owen Rambow d'avoir spontanément joué le rôle de cobaye en installant et en utilisant XMG. Ses courriels nous ont souvent permis de trouver — et dans la plupart des cas, de corriger — des bogues.

Durant quatre ans, j'ai pu apprécier la compagnie des membres du projet Calligramme. Je remercie en particulier les développeurs de `leopar` de *première génération* : Bruno Guillaume, Sylvain Pogodalla et Guillaume Bonfante. C'est en utilisant leur logiciel que j'ai compris la mécanique des grammaires d'interaction. Le chapitre 6 et l'annexe A doivent énormément à Guillaume Bonfante.

Au cours de ce doctorat, j'ai participé au développement de deux logiciels. Je tiens à remercier chaleureusement les personnes avec qui j'ai collaboré étroitement : Yannick Parmentier, le développement de XMG a vraiment été fait *en tandem*, et Jonathan Marchand, qui s'est occupé de la partie vraiment difficile de l'algorithme de Earley (la gestion des environnements).

Une thèse, c'est aussi des discussions et des échanges passionnants avec les autres doctorants ou ingénieurs. Je tiens donc à saluer tous ceux qui m'ont fait aimer Nancy et le LORIA : Sylvain, Benjamin, Benoît, Eric, Hacène, Laïka, Sébastien, Dmitry, Mathieu, Sarah, Karen... et tous ceux que j'oublie.

Je remercie mes amis, là bas en Bretagne, dont la compagnie m'a si souvent manqué durant ces long hivers lorrains, pour leur joie de vivre communicative et leur amitié toujours intacte à chaque visite malgré les années et la distance.

Je remercie mes parents qui ont toujours cru en moi quand moi même je n'y croyais plus.

Enfin, je remercie Gwen de me rendre heureux en complétant l'irréalité du réel.

Many of the sentences I wanted to analyze have been far from easy, and I do not claim to have always found the best solution of the difficulties. Still I thought it better to give my own attempts for what they are worth than to shirk the task.

– Otto Jespersen, Analytic Syntax

Table des matières

Introduction

Dans cette thèse, nous proposons de modéliser le phénomène de coordination dans les grammaires d'interaction. Qu'entendons nous par cela ? Il s'agit d'une part d'un travail de modélisation d'un phénomène linguistique, c'est-à-dire de proposer un cadre théorique qui permette de prédire la grammaticalité (ou l'agrammaticalité) de certaines constructions syntaxiques. Nous utiliserons les grammaires d'interaction pour réaliser cette modélisation. Ces grammaires nous permettront d'exprimer facilement deux aspects de notre modélisation : la valence des groupes conjoints, à travers les polarités, et le domaine de localité étendu d'un groupe conjoint, grâce aux les descriptions d'arbres. Notre proposition prendra donc la forme d'une grammaire d'interaction, en fait d'un fragment d'une telle grammaire qui s'insèrera dans une grammaire générale du français développée par Guy Perrier, de manière à pouvoir vérifier notre modélisation expérimentalement sur corpus. La démarche expérimentale nous semble très importante pour valider les modélisations linguistiques. D'autre part, nous voulons intégrer cette grammaire dans un système de traitement automatique des langues naturelles, ce qui suppose de disposer d'outils efficaces qui mettront en œuvre notre proposition. C'est pourquoi une partie importante de la thèse est consacrée à des aspects qui vont au-delà de notre modélisation, dans le but de la confronter à des corpus. Nous avons travaillé sur les techniques de conception de grammaires lexicalisées, sur le filtrage lexical en vue de l'analyse, ainsi que sur des algorithmes d'analyse syntaxique. Ainsi, nous proposons une architecture complète pour valider notre proposition théorique.

Nous nous consacrons entièrement à l'aspect syntaxique de la coordination, pour la simple raison qu'il n'existe pas encore de consensus sur la manière de représenter la sémantique dans les grammaires d'interaction. Évidemment, nous sommes conscients que notre modélisation doit pouvoir s'étoffer d'une composante sémantique et nous avons pris soin de modéliser la coordination avec la sémantique comme ligne de mire.

1 La coordination : pourquoi ?

La coordination est un phénomène syntaxique omniprésent, quel que soit le registre de langue considéré. Cela amène deux réflexions. Premièrement, une grammaire du français ne peut être complète sans prendre en compte ce phénomène. Passer sous silence la coordination revient à ne modéliser qu'une partie limitée de la langue. Deuxièmement, et pour les mêmes raisons, un système de traitement automatique de la langue ne peut pas faire l'économie de la coordination. Aucun corpus réel n'en est exempt. Donc, pour des raisons à la fois linguistiques et pragmatiques, il nous faut tenir compte de la coordination.

La coordination est un phénomène difficile à modéliser. Elle remet en cause la notion de constituance, puisque l'on peut coordonner des non-constituants, c'est-à-dire des constituants auxquels une partie fait défaut. Elle remet également en cause la notion d'arbre syntaxique. On peut voir les conjonctions de coordination comme des éléments qui autorisent un partage se sous-arbres et les graphes acycliques semblent alors plus adaptés que les arbres. De plus, l'ellipse est souvent associée à la coordination et il est souvent difficile de déterminer, de la coordination ou de l'ellipse, qui commande l'autre.

En conséquence, la plupart des formalismes grammaticaux peinent à en fournir une analyse convenable. Dans la plupart des cas, on étend les formalismes avec des opérations spécialisées ou on autorise des structures exceptionnelles pour modéliser les groupes coordonnés. Nous rejetons cette approche car elle nous semble en contradiction avec l'omniprésence du phénomène.

Nous insistons donc sur le fait que le principe de notre proposition peut se formuler dans les grammaires d'interaction, sans ajout d'opération spécifique. Bien sûr, nous avons souvent été confrontés durant la conception de notre modélisation aux rigidités ou aux manques dont souffrent ces grammaires. En particulier, un système de composition syntaxique plus souple, nous aurait permis d'exprimer plus facilement certains aspects de la coordination.

2 La modélisation proposée

Nous ne prétendons pas dans cette thèse présenter une théorie linguistique novatrice de la coordination : ceci n'est pas une thèse de linguistique. Nous reprenons l'idée déjà développée que les conjoints sont au même niveau par rapport à la conjonction de coordination et que le conjonction de coordination n'est pas la tête de la construction. Cette modélisation peut être exprimée selon un point de vue lexicaliste, qui convient parfaitement aux grammaires d'interaction.

Cependant, notre proposition fait tout de même preuve d'originalité dans le sens où la valence des conjoints y joue un grand rôle. Nous rejoignons en cela la modélisation de la coordination dans les grammaires catégorielles, tout en explicitant complètement l'importance de la valence exprimée à l'aide des polarités des grammaires d'interaction.

2.1 Le principe

Deux groupes peuvent être conjoints s'ils ont le même comportement syntaxique. Ainsi, nous rendons compte de la loi de Wasow qui stipule que deux groupes peuvent être coordonnés si chacun d'entre eux peut être utilisé dans le contexte offert par le reste de la phrase, voir les exemples 1. Dans les grammaires d'interaction, cela veut dire que les deux conjoints ont la même interface. L'interface est la partie de la structure syntaxique qui n'est pas encore saturée et qui peut donc interagir avec le reste de la phrase. Nous pouvons ainsi déterminer si deux groupes peuvent être coordonnés.

(1) (a) Jean accompagne [Marie].
(b) Jean accompagne [le frère de Pierre].
(c) Jean accompagne [Marie et le frère de Pierre].

(d) [Jean aime] Chomsky.
(e) [Marie déteste] Chomsky.
(f) [Jean aime mais Marie déteste] Chomsky.

Le groupe constitué de la conjonction et des deux conjoints doit encore interagir avec le reste de la phrase pour donner une structure finale grammaticale. Le segment coordonné doit interagir avec le reste de la phrase comme le ferait chacun des conjoints. La conjonction doit en quelque sorte fusionner les interfaces des conjoints pour le reste de la phrase.

Comme nous n'avons pas voulu faire évoluer les grammaires d'interaction, nous avons dû traduire cette idée de modélisation dans le formalisme existant. La conjonction a alors deux rôles dans les grammaires d'interaction. D'une part elle doit s'assurer que les conjoints-candidats présentent la même interface et d'autre part offrir une interface unifiée au reste de la phrase. La première étape se fait grâce à l'opération de composition syntaxique propre aux grammaires d'interaction, la superposition partielle de descriptions d'arbres. Si chaque conjoint peut se superposer complètement sur la conjonction en donnant une structure saturée, alors la première étape est réalisée. La deuxième étape indique que la description associée à une conjonction de coordination dispose, en plus des deux parties chargées de saturer les conjoints, d'une troisième partie qui doit se superposer complètement sur le contexte offert par le reste de la phrase.

Il faut donc que les descriptions d'arbres associées aux conjonctions de coordination répondent à un schéma assez strict. Elles sont composées de trois parties : une partie dite haute et deux parties dites basses. La partie haute est chargée d'interagir avec le reste de la phrase. Cette partie correspond donc à l'interface des conjoints que cette conjonction doit coordonner. Les deux parties basses ont deux rôles : vérifier que les conjoints ont la même interface et empêcher ces conjoints d'interagir avec le reste la phrase en les saturant complètement. Ces deux parties basses sont donc identiques pour vérifier que les conjoints sont semblables et elles correspondent à la description duale de la partie haute. C'est-à-dire que la partie haute et les parties basses ont la même forme arborescente mais que les polarités sont inversées entre la première et les dernières.

Cette proposition permet déjà de rendre compte de nombreux cas de coordinations : la coordination de constituants et la coordination de non-constituants avec montée de nœuds. Ces deux sous-phénomènes constituent la majorité des cas de coordination.

2.2 Extension

Pour d'autres phénomènes, comme la coordinations de séquences ou la coordination avec trou verbal, notre proposition ne suffit pas.

(2) (a) Il a donné raison [à Jean pour sa perspicacité] et [à Pierre pour sa franchise].
(b) [Jean viendra toute la semaine] mais [Marie seulement le mercredi].

Pour la coordination de séquences, exemple (2a) , où chaque conjoint est formé par plusieurs constituants, nous sommes limités par la forme arborescente des descriptions d'arbres des grammaires d'interaction. Quant au trou verbal, exemple (2b), il contredit l'hypothèse qui établit que les conjoints ont la même valence, si on l'interprète dans un

sens elliptique. Mais les grammaires d'interaction sont suffisamment expressives pour permettre de modéliser ces phénomènes, mais en abandonnant la simplicité de la première modélisation. Nous utiliserons alors les polarités virtuelles et neutres des grammaires d'interaction. On pourra ainsi reconstruire le contexte manquant aux conjoints pour retrouver la symétrie de la première proposition. Pour ces phénomènes, les grammaires d'interaction possèdent déjà l'expressivité adéquate et il n'est pas nécessaire de les étendre pour en tenir compte.

Jusqu'à présent, nous avons supposé que les conjoints sont semblables. Tout du moins, ils doivent avoir exactement la même interface. Ce n'est pas toujours le cas : c'est ce que l'on appelle la coordination disparate. C'est un phénomène assez rare en français mais plus courant dans les langues à cas. L'exemple 3 illustre la coordination de deux verbes, l'un requérant un complément à l'indicatif, l'autre au subjonctif. Nous proposons pour modéliser ce phénomène, tout en gardant notre proposition, d'enrichir le système de traits des grammaires d'interaction, de manière à modéliser le syncrétisme et l'ambiguïté de valeurs pour un trait. C'est donc uniquement pour ce dernier cas que nous étendons les grammaires d'interaction pour donner plus de souplesse à l'opération de superposition.

(3) a Le sénateur déclare et regrette tout à la fois qu'une nouvelle loi reste envisageable.
a *Le sénateur déclare et regrette tout à la fois qu'une nouvelle loi est/soit envisageable.

2.3 Résultats

Nous pouvons prédire la grammaticalité des coordinations de constituants et de leurs modificateurs, des coordinations de non-constituants avec montée de nœuds, à droite comme à gauche. Nous pouvons aussi vérifier la correction des coordinations de séquences et des coordinations avec trou verbal, bien que dans ce cas l'implantation ne couvre pas tous les cas existant.

Notre traitement des coordinations disparates n'est qu'une proposition théorique. Le nouveau système de traits demande un travail de réécriture important de l'analyseur `leopar`.

Nous couvrons donc la plupart des cas de coordinations. Il nous reste cependant quelques constructions non supportées. C'est en partie dû au fait que notre grammaire de la coordination s'insère dans une grammaire générale du français, et que notre modélisation est donc dépendante de cette grammaire. Par exemple, puisque les superlatifs et les comparatifs ne sont pas modélisés dans la grammaire principale, leur coordination ne l'est pas non plus. Pour d'autres aspects, comme la coordination de verbes à des temps composés (sous la forme de participes passés), la grammaire principale évolue assez vite et leur modélisation est trop récente pour que nous ayons eu le temps de proposer une modélisation de leur coordination.

D'un point de vue quantitatif, nous avons confronté notre modélisation à la TSNLP, qui est un jeu de phrases tests pour les analyseurs syntaxiques, comprenant des phrases positives et des phrases négatives. Nous acceptons 85% des phrases positives contenant des coordinations et nous rejetons l'intégralité des phrases négatives. Les 15% restants qui

ne sont pas analysés sont des coordinations soit de phénomènes qui ne sont pas pris en compte par la grammaire principale (les superlatifs par exemple), soit des coordinations qui contredisent l'hypothèse faite sur le régime des verbes par la grammaire principale (par exemple, *parler à ou de l'entreprise* n'est pas modélisable pour l'instant car les constructions *parler à* et *parler de* sont deux entrées différentes du verbe *parler*).

3 Travaux relatifs à l'implantation

Pour développer notre modélisation et la tester sur des corpus, nous avons dû étendre le domaine de nos recherches à toute la chaîne qui va de la conception de grammaires jusqu'à l'analyse syntaxique. Comme nous voulons insister sur l'aspect expérimental de notre approche, ces travaux ont toute leur place dans ce document.

Un des traits distinctifs des grammaires d'interaction est certainement leur lexicalisation complète. Il n'existe qu'une opération de composition syntaxique, la superposition de descriptions d'arbres et toute la connaissance linguistique est reportée dans le lexique, c'est-à-dire dans l'ensemble des descriptions d'arbres qui peuvent être associées aux mots pour en décrire les différents usages. Cette lexicalisation apporte deux problèmes pour le traitement automatique des langues : la préservation de la cohérence de la grammaire en cours de développement et l'accroissement de la taille du lexique qui a un impact important sur les performances de l'analyse.

La coordination complique l'analyse syntaxique. Tout d'abord, les coordinations allongent les phrases. Ensuite, le polymorphisme des conjonctions rend l'analyse très ambiguë. Nous avons donc développé un nouvel algorithme d'analyse inspiré de l'algorithme d'Earley pour les grammaires hors-contexte. Cet algorithme rend plus facile la décomposition de l'analyse d'une phrase en plusieurs sous-analyses. Nous utiliserons cette propriété pour proposer d'analyser les groupes coordonnées avant le reste de la phrase.

3.1 Écriture de grammaires lexicalisées

Le premier problème vient de l'écriture des grammaires elles-mêmes. En effet, puisque la connaissance linguistique est reportée dans le lexique, c'est à dire dans des centaines de descriptions d'arbres, cette connaissance est présente de manière extrêmement redondante. L'extension ou la modification d'une telle grammaire devient très difficile à mesure que le lexique contient de nombreuses descriptions. Il faut s'assurer à chaque modification que la cohérence interne de la grammaire est préservée. En particulier, il faut s'assurer qu'un phénomène linguistique est traité de manière uniforme à travers le lexique.

Il faut donc pouvoir vérifier la cohérence de la grammaire. La tendance la plus courante dans le domaine de l'écriture de la grammaire consiste à ne plus écrire directement la grammaire mais plutôt d'écrire une description de la grammaire qui ne soit pas redondante, est de générer automatiquement la grammaire finale. Pour les grammaires lexicalisées, cette description comprend des fragments de structures syntaxiques qui correspondent à la modélisation de phénomènes syntaxiques et des indications pour combiner ces fragments pour produire les structures syntaxiques complètes.

C'est dans ce contexte que nous avons participé au développement de l'outil XMG. Cet outil permet justement de générer des grammaires d'interaction ainsi que des grammaires d'arbres adjoints à partir des descriptions des structures de la grammaire finale. C'est grâce à l'utilisation de ce formalisme que nous avons développé notre grammaire de la coordination. C'est aussi avec cet outil qu'est écrite la grammaire du français dans laquelle s'insère notre grammaire. Une partie de la thèse est consacrée à la présentation de cet outil et des concepts qui lui sont propres.

3.2 Filtrage lexical

Grâce à XMG, nous pouvons nous affranchir des problèmes liés à la taille de la grammaire lors de sa conception. En revanche, la taille de la grammaire garde une importance lors de l'analyse syntaxique. Plus le nombre de structures syntaxiques qui peuvent être associées à un mot est grand, plus il est difficile de déterminer quelle est la structure qui doit lui être associée dans le contexte d'une phrase à analyser. L'étape qui consiste à choisir pour chaque mot d'une phrase la structure s'appelle la sélection lexicale. Le nombre de mots dans une phrase et le nombre de structures syntaxiques associées par le lexique à chaque mot de la phrase sont les deux facteurs qui compliquent cette étape. En effet, le nombre de sélections lexicales pour une phrase est le produit du nombre de descriptions d'arbres associées à chaque mot de la phrase par le lexique.

Or les phrases qui présentent le phénomène de coordination sont en général plus longues que les autres — la coordination peut être vue comme un moyen d'exprimer plusieurs phrases en une. De plus, à cause de notre modélisation entièrement lexicaliste de la coordination, on associe de très nombreuses structures syntaxiques aux conjonctions de coordinations. L'étiquetage est donc une étape difficile pour les phrases avec coordination.

Nous avons donc été amenés à nous intéresser au moyen d'éliminer le plus d'étiquetages qui ne peuvent pas donner d'analyse. C'est ce que l'on appelle le filtrage lexical ou désambiguïsation. Nous avons repris les techniques efficaces de filtrage déjà définies dans les grammaires d'interaction dont nous avons montré la complexité, tout en proposant des les étendre de deux manières. D'une part nous proposons une méthode pour désambiguïser plus efficacement les conjonctions de coordination fondée sur la valence des conjoints qui est la base de notre modélisation syntaxique. Cette méthode permet également de distinguer précisément la position des groupes coordonnés en fonction de cette valence. D'autre part, nous avons généralisé cette approche qui consiste à donner de l'information syntaxique pour désambiguïser le plus tôt possible sous la forme de ce que nous appelons les *patrons interdits*, qui correspondent à des sélections lexicales qui ne peuvent aboutir à une analyse et qui doivent donc être éliminés.

3.3 Analyse syntaxique

Nous avons également travaillé sur l'amélioration de l'analyse syntaxique dans les grammaires d'interaction, pour deux raisons liées à notre modélisation de la coordination.

Premièrement, les algorithmes d'analyse déjà existant comme le *shift-reduce*, sont très sensibles à la longueur de la phrase et au nombre de polarités porté par les structures

syntaxiques. Ce sont justement deux caractéristiques des phrases qui présentent des coordinations. Nous avons donc développé un nouvel algorithme qui y est moins sensible. C'est un algorithme de type descendant inspiré de l'algorithme d'Earley des grammaires hors-contexte. Bien que le principe de cet algorithme soit relativement simple, son exposition est rendue ardue car elle ne correspond pas à l'intuition derrière les grammaires d'interaction qui est de superposer partiellement des descriptions d'arbres. Cet algorithme utilise les descriptions d'arbres comme des guides pour construire l'arbre d'analyse directement par la racine.

Deuxièmement, notre méthode de filtrage lexical nous permet de distinguer les limites des segments coordonnés. D'après notre modélisation, ces segments sont autonomes, c'est-à-dire que l'on peut les analyser hors-contexte. Notre algorithme a donc été envisagé de manière à pouvoir assembler des sous-analyses correspondant à des parties autonomes pour construire l'analyse complète. Nous développons une extension de notre algorithme pour ce faire. Bien que pour l'instant ces sous-analyses portent exclusivement sur des groupes coordonnés, il est tout à fait envisageable de généraliser cette approche.

4 Plan de la thèse

En tenant compte des travaux effectué, nous avons structurés le document de la façon suivante :

Chapitre 1. Les grammaires d'interaction. Nous présentons dans ce premier chapitre le formalisme des grammaires d'interaction. En effet, ce formalisme est jeune et encore mal connu. Il évolue vite, et c'est pourquoi nous avons essayé de donner un présentation générale de la version la plus récente. Ce formalisme permet de décrire facilement et finement la notion de valence en syntaxe. La grammaire y est envisagée comme un système de contraintes imposées par chaque mot du langage. La structure syntaxique associée à une phrase (l'arbre syntaxique) doit alors vérifier l'ensemble des contraintes apportées par les mots qui la composent.

Chapitres 2, 3 et 4. L'écriture de grammaires lexicalisées et XMG. Avant de nous attaquer à la modélisation proprement dite, nous présentons la problématique liée à l'écriture des grammaires fortement lexicalisées, en nous intéressant plus particulièrement aux grammaires d'arbres adjoints et aux grammaires d'interaction. La redondance de l'information linguistique dans ces grammaires rend très difficile la conception et la maintenance des grammaires à large couverture. Dans ce cadre, nous présentons également la solution qui a été proposée par Benoît Crabbé, Denys Duchier, Yannick Parmentier et nous-mêmes, XMG, qui permet d'exprimer une grammaire fortement lexicalisée sous une forme concise, et donc sans redondance. Nous terminerons cette partie par un exemple-jouet de grammaire de la coordination pour illustrer les possibilités offertes par l'outil.

Chapitre 5. La modélisation de la coordination Dans ce chapitre, nous présenterons notre modélisation de la coordination qui est la principale contribution de ce document. Ce chapitre est relativement long, à la mesure de l'ampleur de la tâche. Pour

commencer, nous donnerons un certain nombre d'exemples de coordinations significatifs de certaines propriétés intrinsèques à ce phénomène. Ainsi, en nous fondant sur ces exemples, nous pourrons proposer notre proposition de modélisation qui est fondée sur la valence des groupes coordonnés.

Dans une seconde partie, nous montrerons comment le principe de notre proposition doit s'adapter à la réalité de la grammaire dans laquelle notre modélisation doit s'insérer. Nous détaillerons ainsi sous-phénomène par sous-phénomène comment nous modélisons la coordination.

Ensuite nous présenterons une extension des structures de traits polarisées des grammaires d'interaction pour tenir compte des coordinations disparates. Cette extension n'est pas encore implantée et n'a donc pas pu faire l'objet d'évaluation. Cette partie permet de modéliser des coordinations disparates dans le cas du syncrétisme de cas que l'on observe dans des langues à déclinaisons, comme les langues slaves par exemple.

Ensuite, nous évoquons d'autres modélisations de la coordination dans d'autres formalismes, les grammaires catégorielles, les grammaires syntagmatiques dirigées par les têtes et les grammaires lexicales fonctionnelles. Nous expliquons leur principe et nous les comparons à notre approche.

Finalement, nous expliquons en détail comment nous avons écrit notre grammaire de la coordination sous la forme d'une métagrammaire traitée par XMG. Nous détaillons la méthodologie et l'organisation de notre proposition.

Chapitre 6. Filtrage lexical. À partir de ce chapitre, nous développons les travaux qui ont eu pour but de pouvoir tester notre proposition sur corpus. Tout d'abord, nous aborderons le problème de la désambiguïsation lexicale dans les grammaires d'interaction. Des techniques originales ont été élaborées pour améliorer le filtrage lors de l'étiquetage. Nous en caractériserons la complexité. Malheureusement, ces techniques ne sont pas suffisantes pour les phrases qui présentent le phénomène de coordination. En effet, notre grammaire de la coordination associe une soixantaine de descriptions aux conjonctions de coordinations et les phrases concernées sont plus longues en moyenne que les phrases sans coordination.

Nous présentons donc deux nouvelles méthodes de désambiguïsation. La première se concentre sur les coordinations et permet de d'améliorer l'efficacité du filtrage tout en délimitant les positions des groupes conjoints. Cette propriété nous servira dans la partie consacrée à l'analyse syntaxique. L'originalité de cette méthode est qu'elle se fonde sur la modélisation syntaxique que nous proposons. Des connaissances linguistiques permettent donc de filtrer plus efficacement. La deuxième méthode est plus générale, mais elle repose aussi sur des connaissances linguistiques. Si l'on sait que des structures syntaxiques ne peuvent pas être présentes simultanément dans l'étiquetage d'une phrase, alors on peut éliminer directement ces étiquetages pour ne pas avoir à les considérer lors des étapes de filtrages ultérieures.

Chapitres 7 et 8. Analyse syntaxique. La dernière partie de ce document est consacrée à l'analyse syntaxique dans les grammaires d'interaction.

Nous présenterons tout d'abord l'algorithme historique d'analyse dans les grammaires

d'interaction que l'on nomme *shift-reduce*. Cet algorithme repose sur la notion de raffinement de descriptions d'arbres. On cherche en effet à simplifier une description d'arbres pour qu'elle se rapproche petit à petit d'un arbre qui sera le résultat de l'analyse. Cet algorithme de base est rendu efficace par l'utilisation d'une heuristique psycho-linguistique. Le nombre de polarités non-saturées est borné. On lit la phrase de gauche à droite en accumulant les descriptions lues dans la mémoire. Tant que la borne n'est pas atteinte, on lit des mots de la phrase d'entrée (*shift*). Ensuite, on cherche à saturer les polarités des mots déjà lus (*reduce*). Ces deux étapes sont réitérées tant qu'il reste des mots à lire et que la mémoire n'est pas vide.

Il y a deux problèmes pour cet algorithme et notre modélisation. Tout d'abord, cet algorithme est sensible au nombre de polarités présentes dans les descriptions d'arbres. Ensuite, il est difficile de trouver une notion de sous-analyse qui permette de construire une analyse complète à partir de sous-analyses ou d'analyses partielles.

Pour ces raisons, nous présenterons un nouvel algorithme d'analyse syntaxique, qui est moins sensible au nombre de polarités présentes et qui permette de présenter des sous-analyses que l'on peut ensuite combiner pour obtenir une analyse complète. Cet algorithme s'inspire de celui d'Earley pour les grammaires hors-contextes. Cet algorithme a fait l'objet d'une implantation et ces résultats, bien qu'encourageants, doivent encore être améliorés.

Nous proposons donc pour finir une modification de notre algorithme pour décomposer une analyse en plusieurs sous-analyses indépendantes que l'on peut combiner. Pour l'instant nous n'envisageons de l'appliquer qu'aux groupes conjoints, mais il est tout à fait possible de l'appliquer à tous les types de segments qui peuvent être analysés indépendamment du reste de la phrase.

Chapitre 1

Les grammaires d'interaction

Sommaire

Dans ce premier chapitre, nous allons présenter le formalismes des grammaires d'interaction (GI) [Per03]. Ce formalisme est assez récent mais il s'ancre à la fois dans la tradition des grammaires catégorielles (GC) et celles des grammaires syntagmatiques. Des premières les GI reprennent la notion de valence des structures qui guide la composition syntaxique. Des secondes on retrouve la notion d'arbre d'analyse qui représente les relations de constituance entre syntagmes.

En revanche, les GI apportent un certain nombre de nouveautés. La notion de valence est ici explicite. Elle est exprimée à l'aide de polarités attachées aux traits morphosyntaxiques des structures syntaxiques. On peut directement lire les polarités sur ces structures. La composition syntaxique elle aussi est originale. Tout d'abord les structures syntaxiques que l'on manipule ne sont pas des arbres mais des descriptions d'arbres, que l'on présente généralement comme des arbres sous-spécifiés. Ensuite l'opération de composition elle-même est nouvelle : c'est l'opération de superposition partielle de descriptions d'arbres.

Nous allons dans un premier temps souligner l'importance de la valence en syntaxe qui a amené à une première version des GI. Puis dans un second temps, nous présenterons les GI dans leur version actuelle qui ont évolué d'une théorie logique vers un formalisme grammatical à part entière.

1.1 Un formalisme polarisé

L'importance de la valence en syntaxe ne date pas des grammaires d'interaction. Nous revenons brièvement sur cette histoire pour aboutir aux premières GI. Nous appelons valence d'un élément syntaxique, par analogie avec la valence des éléments chimiques, le fait que ces éléments fournissent ou demandent une contribution au reste de la phrase pour former des éléments plus importants.

1.1.1 La notion de valence en syntaxe

Tesnière [Tes59] a développé une théorie de la valence pour expliquer la bonne formation syntaxique. Dans sa théorie, il dénomme valence le nombre d'arguments, ou *actants*, dont a besoin un verbe, un nom ou un adjectif. C'est ce que l'on appelle couramment le régime ou le cadre de sous-catégorisation. Tesnière distingue les verbes monovalent, bivalent, trivalent ou quadrivalent. Les verbes impersonnels ont une valence nulle, ils sont dits avalents. La valence correspond donc au nombre d'actants mis en jeu dans l'action décrite par un verbe (*le petit drame* de Tesnière).

Le concept de valence s'étend aux noms prédicatifs ainsi qu'aux adjectifs. On peut donc voir un nom comme une entité ayant besoin de ces actants pour à son tour fournir un actant à un verbe de la phrase qui lui-même attend un certain nombre d'actants pour fournir une proposition.

Depuis, la valence est devenue un concept de première importance en syntaxe. En témoigne les dernières avancées des grammaires transformationnelles qui se fondent sur la valence : le programme minimaliste de [Cho95], aussi formalisé par [Sta96].

Dans la tradition des grammaires catégorielles, on fait généralement remonter à Ajdukiewicz [Ajd35] la première modélisation de la valence des éléments syntaxiques. On peut aussi y voir une généralisation de l'approche de Tesnière où tous les mots ont une valence qui exprime à la fois leurs besoins et leurs contributions. À chaque mot du langage, on associe un ensemble de catégories syntaxiques atomiques, notamment gn pour les groupes nominaux, n pour les noms communs et s pour les phrases, ou des *fractions* construites de manière inductive à partir de ces catégories. Le dénominateur représente le besoin exprimé par l'élément syntaxique et le numérateur indique ce que l'élément fournit au reste la phrase. Pour vérifier qu'une phrase appartient au langage, il faut dans un premier temps attribuer à chaque mot une fraction. Il faut ensuite que le produit de ces fractions se simplifie vers la catégorie s. Par exemple, à la phrase «*Le petit chat est noir*», on associe la structure suivante :

$$\frac{gn}{n} \times \frac{n}{n} \times n \times \frac{\dfrac{s}{gn}}{\dfrac{n}{n}} \times \frac{n}{n}$$

qui se réduit en s. Cette phrase appartient au langage.

La valence donne des informations linguistiques pertinentes, mais elle ne suffit pas. Elle ne dit rien sur *l'intérieur* d'un élément syntaxique, mais se contente de dire comment il peut interagir, c'est-à-dire son interface extérieure, ce qu'il attend du *monde extérieur* et ce qu'il lui fournit.

Dans ce premier calcul par exemple, toute permutation d'une phrase correcte est également correcte. On a donc développé des formalismes, les grammaires catégorielles (on peut citer par exemple Bar-Hillel ou Steedman [BH53; SB07]), qui ont repris cette idée fondamentale de valence, tout en ajoutant un contrôle toujours plus fin de la composition syntaxique, notamment en contrôlant l'associativité et la commutativité du produit.

1.1.2 La logique linéaire intuitionniste implicative

Une autre proposition, celle de Lambek [Lam58], consiste à présenter les grammaires catégorielles comme un calcul logique. Mais il faudra attendre une trentaine d'années pour comprendre le lien entre cette logique particulière et la logique mathématique.

Girard [Gir87] a défini la logique linéaire, qui est une logique sensible aux ressources : les hypothèses sont consommées pour produire les conclusions et leur nombre est donc significatif, contrairement à ce qui se passe dans les logiques plus usuelles que sont la logique classique et la logique intuitionniste. En effet, ces logiques ont été largement utilisées pour modéliser le raisonnement mathématique dans lequel une hypothèse peut être utilisée *ad libitum*.

Le fragment multiplicatif implicatif de cette logique a donné un cadre formel général aux grammaires catégorielles. La logique linéaire a permis également de donner un paradigme pour l'analyse syntaxique, l'analyse comme déduction (*parsing as deduction*). L'analyse d'une phrase est vue comme une démonstration logique. L'interface entre syntaxe et sémantique devient naturel grâce à l'isomorphisme de Curry-Howard, qui définit une correspondance entre la démonstration d'un séquent intuitionniste, qui représente l'analyse syntaxique, et un lambda-terme typé, qui représente la sémantique de la phrase. Pour notre exemple, *«Le petit chat est noir.»*, il faut donc donner une démonstration du séquent :

$$n \multimap gn, n \multimap n, n, (n \multimap n) \multimap (gn \multimap s), n \multimap n \vdash s$$

Mais cette logique est commutative et associative, et le problème des permutations de mots est toujours présent. Pour pallier ce problème, plusieurs solutions ont été proposées que l'on peut classer en deux catégories :

- ajouter des modalités à la logique utilisée pour contrôler l'associativité et la commutativité des connecteurs logiques comme le fait [Moo96],
- déléguer la gestion de l'ordre des mots à un niveau subalterne, comme dans les GC *modernes* que sont les grammaires catégorielles abstraites [de 01], les λ-grammaires de [Mus03], les grammaires catégorielles minimalistes [AL06] ou l'environnement de développement de grammaires (*grammatical framework*) de [Ran07].

Les grammaires d'interaction présentent une troisième solution à ce problème : l'ordre des mots et leur valence sont indiqués sur les mêmes structures mais on peut parler d'un

type d'information indépendamment de l'autre. En particulier, on peut établir une relation de domination sans préciser si le dominé est en périphérie droite ou gauche.

1.1.3 Grammaires d'interaction primitives

Les grammaires d'interaction naissent de l'observation faite dans [Per01] qu'une démonstration dans la logique linéaire intuitionniste implicative, qui est souvent représentée sous la forme d'un réseau de preuve, revient à superposer des descriptions d'arbre qui correspondent à la formule à démontrer pour obtenir un arbre. Les nœuds de ces descriptions d'arbres sont polarisés, c'est-à-dire qu'ils sont décorés d'une polarité + ou −. La superposition des descriptions d'arbres doit vérifier que chaque nœuds étiqueté par + est superposé à un nœud étiqueté par −.

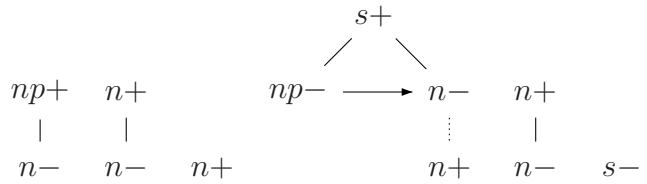


FIG. 1.1 – Descriptions d'arbre associées aux mots de la phrase *«Le petit chat est noir.»*.

Cette équivalence est fondamentale : elle permet de passer d'une vision opératoire issue de la théorie de la preuve à une vision déclarative issue de la théorie des modèles dans laquelle on peut envisager la grammaire comme un système de contraintes.

Comme cette observation est directement issue de la logique linéaire intuitionniste implicative, l'ordre des fils (*daughter nodes*) d'un nœud dans les descriptions d'arbres est libre, ce qui correspond à l'associativité et la commutativité de cette logique. L'idée naturelle pour tenir compte de l'ordre des mots est donc d'ajouter un ordre entre ces fils pour contraindre l'ordre des dépendances.

À chaque mot de la langue, ces premières GI [Per00a] associent un ensemble de descriptions d'arbres dont les nœuds sont polarisés. Analyser une phrase revient à construire un arbre en superposant les descriptions d'arbres associées aux mots de la phrase. Si un tel arbre existe, alors la phrase appartient au langage engendrée par la gramaire. C'est donc l'idée essentielle des grammaires d'interaction : superposer des descriptions d'arbres qui indiquent la contribution des mots à la construction d'une phrase. Pour obtenir les grammaires d'interaction actuelles, il faut étendre cette première proposition. Tout d'abord il est nécessaire d'enrichir les relations qui peuvent exister entre les nœuds d'une description pour notamment tenir compte de l'ordre des mots. Ensuite, plutôt que de polariser les nœuds, il a été décidé de munir ces nœuds de structures de traits qui eux sont polarisés. On obtient ainsi un système de polarité très fin mais toujours facile à comprendre. Enfin, on étend le système des polarités pour ajouter plus de souplesse à l'opération de superposition.

1.2 Les descriptions d'arbres polarisées

En section 1.1.3, nous avons décrit les premières grammaires d'interaction. Dans le reste de ce chapitre nous présentons les grammaires d'interaction dans leur version actuelle qui ont évolué d'une théorie de la logique linéaire [Per00b] vers un formalisme pour modéliser la syntaxe et la sémantique des langues naturelles [Per04].

Les structures de base manipulées par notre formalisme sont les descriptions d'arbres polarisées, notées DAP dans la suite.

1.2.1 Utilisation de descriptions

Avant de donner une définition précise des DAP, nous essayons de motiver leur utilisation. Pour l'instant, on se contentera de dire que ce sont des arbres sous-spécifiés ou incomplets. L'intérêt des polarités dans le traitement des langues a été vu précédemment, dans la section 1.1.1. Celui de manipuler des descriptions d'arbres au lieu de simples arbres est présenté ici.

Tout d'abord, l'utilisation de descriptions comme structures syntaxiques dont les arbres modèles seront des résultats d'analyse, nous permettra de nous inscrire dans un cadre formel connu, la syntaxe vue à travers la théorie des modèles (*model theoretic syntax*) [CR98; PS01]. Outre ce changement de paradigme, on peut s'intéresser à l'intérêt concret des descriptions en ce qui concerne l'expression de faits linguistiques.

D'une part, une description d'arbre peut décrire une *famille* d'arbres proches. L'utilisation de descriptions nous permet donc de factoriser la grammaire, c'est-à-dire ici de réduire la taille du lexique, notamment sa redondance. Nous verrons que le problème de redondance n'est pas spécifique aux GI. Plus généralement, c'est un problème des grammaires fortement lexicalisées que nous avons tenté de résoudre avec XMG (voir le chapitre 3). Linguistiquement, des phénomènes proches peuvent être représentés par les mêmes DAP. En d'autres termes, le formalisme évite ainsi de créer artificiellement trop de différences que l'on ne retrouve pas au niveau linguistique. Il crée moins d'artefacts.

D'autre part, l'utilisation de descriptions aux relations sous-spécifiées permet d'étendre le domaine de localité. Ainsi on va pouvoir décrire des phénomènes linguistiques qui ne sont ni simplement des rapports de constituants à sous-constituants directs, ni des rapports de concaténation entre syntagmes. C'est ce que [MHF83] appelle l'approche déterministe[1] de certains phénomènes comme le rattachement prépositionnel ou, dans le cas qui nous intéressera plus tard, la coordination. C'est la même structure qui indique tous les rattachements possibles, et sans transformation indéterministe. Pour la coordination de non-constituants (cf. partie II), cette propriété nous permettra de modéliser les dépendances non-bornées des conjoints sans multiplier les structures.

Ces avantages par rapport aux arbres simples sont à mettre en balance avec une plus grande complexité des algorithmes de manipulation. On verra dans le chapitre 7 que le problème de l'analyse est un problème difficile, plus précisément NP-complet.

[1] Le terme *déterministe* de [MHF83] nous semble peu opportun. On lui préfèrerait aujourd'hui le terme *monotone*. Aucune transformation de structures ne contredit une information présente dans ces structures.

$+$	$\rightarrow$	$\leftarrow$	$=$	$\approx$	$\leftrightarrow$	$\perp$
$\rightarrow$	$\perp$	$\leftrightarrow$	$\rightarrow$	$\rightarrow$	$\perp$	$\perp$
$\leftarrow$	$\leftrightarrow$	$\perp$	$\leftarrow$	$\leftarrow$	$\perp$	$\perp$
$=$	$\rightarrow$	$\leftarrow$	$=$	$=$	$\leftrightarrow$	$\perp$
$\approx$	$\rightarrow$	$\leftarrow$	$=$	$\approx$	$\leftrightarrow$	$\perp$
$\leftrightarrow$	$\perp$	$\perp$	$\leftrightarrow$	$\leftrightarrow$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

TAB. 1.1 – Somme de polarités

1.2.2 Polarités

Le formalisme des grammaires d'interaction est fondé sur la notion de valence des éléments syntaxiques. Cette valence est représentée par les polarités, non au niveau des syntagmes mais au niveau des traits morphosyntaxiques attachés à ces syntagmes. Une polarité est un élément de $\mathcal{P} = \{\rightarrow, \leftarrow, =, \approx, \leftrightarrow, \perp\}$. Plus précisément, il y a un premier groupe de polarités utilisé dans les descriptions que nous nommerons les polarités élémentaires :

- $\rightarrow$, la polarité positive, indique que le trait fournit sa valeur ;
- $\leftarrow$, la polarité négative, indique que le trait requiert sa valeur ;
- $=$, la polarité neutre indépendante, indique que le trait ne se comporte pas comme une ressource consommable, qu'il suffit à donner une information linguistique valide comme un trait dans une grammaire d'unification, ou en d'autres termes qu'il est autonome, et enfin
- $\approx$, la polarité virtuelle, indique que le trait n'est pas autonome, qu'il a besoin d'un autre trait de *confirmation* de l'information morpho-syntaxique muni d'une polarité différente de $\approx$.

D'autre part, on distingue un second groupe de polarités qui n'apparaîtra que dans les calculs de sommes de polarités. La somme de polarités nous permettra de donner un statut en terme de valence aux structures intermédiaires de l'analyse syntaxique (cf. chapitre 7). Ce second groupe comprend :

- $\leftrightarrow$, cette polarité indique la saturation du trait. On ne peut plus consommer ou fournir ce trait. Elle provient d'une somme dans laquelle il existe exactement un terme $\rightarrow$ et exactement un terme $\leftarrow$.
- $\perp$ est la polarité incohérente. Elle indique que les ressources que constituent les polarités n'ont pas été gérées correctement. Elle provient d'une somme comportant plus d'une polarité $\rightarrow$ ou plus d'une polarité $\leftarrow$.

Nous pouvons maintenant définir la somme de polarités, notée $+$. Cette opération permet de gérer les ressources portées par les polarités. Le résultat de cette opération est présentée dans la table 1.1. On remarque que cette opération est commutative et associative. Cette propriété nous donnera une plus grande liberté pour l'analyse.

On dira qu'une polarité est saturée si elle est égale à $\leftrightarrow$ ou $=$.

Le système de polarités n'est pas figé et rien n'empêche d'étendre les GI en en ajoutant d'autres, comme par exemple les polarités absorbantes des grammaires d'unification polarisées proposées par [Kah04].

1.2.3 Traits, valeurs et environnements

Nous voulons munir les nœuds de nos descriptions de structures de traits et nous voulons être capables de partager des valeurs de traits entre plusieurs nœuds d'une même description : notre modélisation de la coordination repose en partie sur cette possibilité. C'est pourquoi, nous empruntons aux langages de programmation la notion d'environnement d'évaluation, ou simplement d'environnement. Les traits n'auront de valeur que relativement à un environnement. Intuitivement, un environnement est une liste d'associations faite de couples *noms de variables/valeurs*. Différents traits pourront faire mention à des mêmes noms de variables de l'environnement. Ainsi, ils partageront les valeurs associées à ces variables.

Soit $\mathcal{F}$ un ensemble fini de noms de traits (morpho-syntaxiques). Chaque nom de trait f de $\mathcal{F}$ représente un type d'information porté par un nœud. À chaque nom de trait f est associé un ensemble de valeurs atomiques $\mathcal{V}_f$. Les valeurs que peut prendre un trait sont définies pour chaque nom f sur le domaine $\mathcal{D}_f$ construit à partir de $\mathcal{V}_f$.

Dans notre version des grammaires d'interaction[2], une valeur du domaine D_f est un sous-ensemble d'éléments de $\mathcal{V}_f$, soit $\mathcal{D}_f = 2^{V_f}$. L'ensemble vide $\emptyset$ est appelée valeur incohérente. Les ensembles sont utiles car ils permettront de représenter la disjonction des éléments qui la composent. La valeur de D_f constituée de tous les éléments de V_f est écrite par abus de notation « ? », quel que soit V_f. Les singletons $\{v\}$ seront notés v.

Exemple 1. *On peut par exemple prendre* $\mathcal{F} = \{cat, funct\}$ *et* $\mathcal{V}_{cat} = \{np, n, s, adj\}$, $\mathcal{V}_{funct} = \{subj, obj, attr, deobj\}$. *Dans ce cas* $\{np, s\} \in \mathcal{D}_{cat}$ *et* $subj \in \mathcal{D}_{funct}$.

Pour pouvoir donner une valeur partageable à un trait, nous devons d'abord définir la notion d'environnement d'évaluation d'un trait. Nous aurons besoin des ensembles de variables dénombrables $\mathcal{X}_f$ pour chaque nom de trait f et $\mathcal{X} = \bigcup_{f \in \mathcal{F}} \mathcal{X}_f$.

Définition 1 (Environnement)**.** *Un environnement* Γ *est défini par*

- *son support* $\mathcal{S}$*, c'est-à-dire une partie finie de* $\mathcal{X}$
- *une fonction d'assignation qui associe à toute variable* $x \in \mathcal{S} \cap \mathcal{X}_f$ *un élément de* $\mathcal{D}_f$ *noté* $\Gamma.x$
- *une relation d'équivalence sur le support, compatible avec les noms de traits associés. Si deux variables sont équivalentes alors elles ont la même image par la fonction d'assignation*[3]*. La classe d'équivalence à laquelle appartient* x *est notée* $\langle x \rangle$*. Enfin, on étend la fonction d'assignation aux classes d'équivalence :* $\Gamma.\langle x \rangle = \Gamma.x$

Exemple 2. *Nous donnons* Γ*, l'environnement suivant, pour* $x_1, x_2 \in \mathcal{D}_{cat}$ *et* $x_3, x_4 \in \mathcal{D}_{funct}$*. La relation d'équivalence est la relation d'équivalence discrète (la plus petite relation d'équivalence).*

[2] Nous verrons par la suite comment la structure des domaines sera enrichie

[3] Attention : la réciproque n'est pas vraie. Deux variables peuvent avoir la même valeur sans être équivalentes.

x	x_1	x_2	x_3	x_4
$\Gamma.x$	$\{np, n\}$	$\{np, s\}$	$subj$	obj

Nous définissons également une opération sur ces environnements, l'identification de variables.

Définition 2 (Identification)**.** *Soient $x, y \in \mathcal{X}_f$ deux variables appartenant à un même environnement Γ. L'identification de x et y dans Γ, notée $\Gamma[x \equiv y]$, produit un nouvel environnement de même support que Γ, tel que :*

- *la relation d'équivalence est la plus petite extension de la relation d'équivalence sur Γ telle que x et y soient équivalentes, et*
- *la fonction d'assignation est celle de Γ mais : $\Gamma[x \equiv y].\langle x\rangle = \Gamma[x \equiv y].\langle y\rangle = \Gamma\langle x\rangle \cap \Gamma\langle y\rangle$*

L'opération d'identification est associative et commutative. On peut donc définir l'identification de n variables dans un environnement, notée $\Gamma[\equiv \{x_1, \ldots, x_n\}]$.

Exemple 3. *Si l'on reprend l'exemple précédent, $\Gamma[x_1 \equiv x_2]$ donne :*

$\Gamma[x_1 \equiv x_2]$:

x_1, x_2	x_3	x_4
np	$subj$	obj

On note les variables équivalentes dans la même case.

Deux environnements peuvent aussi être concaténés pour créer un nouvel environnement, à condition que les ensembles de variables des deux environnements soient disjoints. La concaténation revient à juxtaposer les deux environnements.

Nous avons désormais tous les éléments pour définir les traits polarisés propres aux grammaires d'interaction.

Définition 3 (Trait polarisé)**.** *Un trait polarisé t sur un environnement Γ est un triplet constitué d'un nom de trait, d'une polarité et d'une variable $t = (f, p, \langle x\rangle) \in \mathcal{F} \times \mathcal{P} \times 2^{\mathcal{X}_f}$ où x est une valeur du support de Γ.*

Une structure de traits S est un ensemble de traits $\{(f_1, p_1, \langle x_1\rangle), \ldots, (f_n, p_n, \langle x_n\rangle)\}$ où les f_i sont tous différents. Une structure de trait est dite saturée si la polarité associée à un nom de trait est saturée, pour chaque nom de trait.

À chaque structure de traits S sur un environnement Γ, on peut associer la structure de traits effective $S!$ qui remplace les (classes d'équivalence des) variables par leur image par la fonction d'assignation de Γ, c'est-à-dire :

$$S! = \{(f_1, p_1, \Gamma.\langle x_1\rangle), (f_2, p_2, \Gamma.\langle x_2\rangle), \ldots, (f_n, p_n, \Gamma.\langle x_n\rangle)\}$$

Ces traits polarisés sont très semblables aux traits habituellement manipulés par les formalismes linguistiques à base d'unification [Car92]. D'ailleurs, nous définissons une opération proche de l'unification sur ces structures de traits, la superposition.

Définition 4 (Superposition). *Soient deux structures de traits T_1 et T_2 définies sur un même environnement Γ :*

$$\begin{array}{rl} T_1 = & \{(f_1, p_1, \langle x_1\rangle), (f_2, p_2, \langle x_2\rangle), \ldots, (f_n, p_n, \langle x_n\rangle)\} \quad \cup T'_1 \\ T_2 = & \{(f_1, q_1, \langle y_1\rangle), (f_2, q_2, \langle y_2\rangle), \ldots, (f_n, q_n, \langle y_n\rangle)\} \quad \cup T'_2 \end{array}$$

où les noms de traits de T'_1 (resp. T'_2) sont absents de T_2 (resp. T_1).

La superposition de T_1 et T_2 est une structure de traits notée $T_1 + T_2$ sur un environnement Γ' telle que :

$T_1 + T_2 = \{(f_1, p_1 + q_1, \langle x_1\rangle), (f_2, p_2 + q_2, \langle x_2\rangle), \ldots, (f_n, p_n + q_n, \langle x_n\rangle)\} \cup\ T'_1 \cup\ T'_2$
et $\Gamma' = \Gamma[x_1 \equiv y_1][x_2 \equiv y_2] \cdots [x_n \equiv y_n]$

Attention, la superposition n'est pas une unification, notamment parce qu'elle n'est pas idempotente (la somme des polarités n'est pas idempotente).

On peut généraliser cette opération en une opération n-aire de superposition de n structures de traits.

Une structure de traits qui possède un trait ayant une valeur interprétée par $\emptyset$ dans son environnement ou qui associe à un nom de trait une polarité $\perp$ est dite incohérente. Deux structures de traits S et T sont non superposables si $S + T$ est incohérente.

Exemple 4. *Soient t_1, t_2 et t_3 trois structures de traits polarisées définies respectivement sur Γ_1, Γ_2 et Γ_3.*

$t_1 = \{(cat, \leftarrow, x_1), (funct, \rightarrow, x_2), (nb, =, x_3)\}$

Γ_1 :

x_1	x_2	x_3
$\{np, s\}$	$subj$	pl

$t_2 = \{(cat, \rightarrow, x_4), (funct, \leftarrow, x_5), (nb, =, x_6)\}$

Γ_2 :

x_4	x_5	x_6
np	?	pl

$t_3 = \{(cat, \approx, x_7), (funct, =, x_8)\}$

Γ_3 :

x_7	x_8
$\{np, s\}$	$\{subj, obj\}$

On veut superposer ces trois structures de traits polarisés. Dans un premier temps, on va concaténer les environnements $\Gamma_1, \Gamma_2, \Gamma_3$ en un seul environnement Γ.

Γ :

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
$\{np, s\}$	$subj$	pl	np	?	pl	$\{np, s\}$	$\{subj, obj\}$

Ensuite on va réaliser l'opération $t_1 + t_2 + t_3$. L'opération $+$ étant commutative et associative, on peut réordonner les termes. Ici, on va réaliser $(t_2 + t_3) + t_1$.

$$\begin{array}{rcl} t_2 + t_3 & = & \{(cat, \rightarrow, \langle x_4\rangle), (funct, \leftarrow, \langle x_5\rangle), (nb, =, \langle x_6\rangle)\} + \{(cat, \approx, \langle x_7\rangle), (funct, =, \langle x_8\rangle)\} \\ & = & \{(cat, \rightarrow + \approx, \langle x_4\rangle), (funct, \leftarrow + =, \langle x_5\rangle), (nb, =, \langle x_6\rangle)\} \\ & = & \{(cat, \rightarrow, \langle x_4\rangle), (funct, \leftarrow, \langle x_5\rangle), (nb, =, \langle x_6\rangle)\} \end{array}$$

et l'environnement après superposition est $\Gamma' = \Gamma[x_4 \equiv x_7][x_5 \equiv x_8]$ *c'est-à-dire*

Γ' :

x_1	x_2	x_3	x_4, x_7	x_5, x_8	x_6
$\{np, s\}$	$subj$	pl	np	$\{subj, obj\}$	pl

Continuons avec la dernière somme :

$$\begin{aligned}(t_2 + t_3) + t_1 &= \{(cat, \rightarrow, \langle x_4\rangle), (funct, \leftarrow, \langle x_5\rangle), (nb, =, \langle x_6\rangle)\} \\ &\quad +\{(cat, \leftarrow, \langle x_1\rangle), (funct, \rightarrow, \langle x_2\rangle), (nb, =, \langle x_3\rangle)\} \\ &= \{(cat, \rightarrow + \leftarrow, \langle x_4\rangle), (funct, \leftarrow + \rightarrow, \langle x_5\rangle), (nb, = + =, \langle x_6\rangle)\} \\ &= \{(cat, \leftrightarrow, \langle x_4\rangle), (funct, \leftrightarrow, \langle x_5\rangle), (nb, =, \langle x_6\rangle)\}\end{aligned}$$

et l'environnement après superposition est $\Gamma'' = \Gamma'[x_4 \equiv x_1][x_5 \equiv x_1][x_6 \equiv x_3]$ *c'est-à-dire*

Γ'' :

x_1, x_4, x_7	x_2, x_5, x_8	x_3, x_6
np	$subj$	pl

On remarque que la structure de traits finale $t_1 + t_2 + t_3$ *est saturée.*

Nous avons présenté les structures de traits manipulées par les grammaires d'interaction et la mécanique sous-jacente des environnements. Malgré la simplicité de ces environnements, ils nous permettront d'exprimer la coindexation, en d'autres termes que plusieurs nœuds d'une DAP partagent certains traits. Par exemple deux traits $(f, p, \langle x\rangle)$ et $(f, q, \langle y\rangle)$ sont coindexés (ou partagés) si et seulement si $\langle x\rangle = \langle y\rangle$. Cette coindexation nous sera utile pour la modélisation de la coordination.

1.2.4 Les descriptions d'arbres

Nous avons présenté les structures de traits polarisés, c'est-à-dire le contenu des nœuds des DAP. Nous allons maintenant voir la structure de ces graphes.

La définition suivante introduit formellement les descriptions d'arbres.

Définition 5 (Description d'arbre)**.** *Une description d'arbre est un graphe orienté acyclique* (V, E, l_e) *construit sur un ensemble de nœuds* V *et d'arcs* E*. La fonction d'étiquetage des arcs* l_e *a pour image* $\{d, s\}$ *(direct et sous-spécifié). La structure* (V, E) *est un arbre (une seule composante connexe et tous les nœuds ont un arc entrant sauf un appelé racine).*

Sur cette structure, on distingue des ensembles de nœuds frères ou fratries $(F_i)_{i\in I}$*. Une fratrie est constituée de nœuds ayant le même prédécesseur par un arc étiqueté* d*. Ces nœuds frères sont partiellement ordonnés, par les relations* $\prec$ *et* $\prec^*$*.*

Intuitivement, une description d'arbre est un arbre où les relations de parenté (entre nœud père et nœuds fils ou entre nœuds frères) peuvent être sous-spécifiées par $\xrightarrow{s}$ et $\prec^*$. Ces deux relations indiquent qu'il peut exister des nœuds intermédiaires qui ne sont pas donnés dans la description. La définition précédente rapproche notre formalisme des grammaires de descriptions d'arbres ou des grammaires de substitution de descriptions

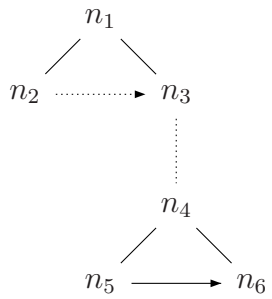


FIG. 1.2 – La description D

d'arbres présentées en [RWVS01]. La différence est que nous forçons les DAP à être des arbres, et non des graphes orientés acycliques. En ce sens, nous nous rapprochons des descriptions d'arbres de [DT99] qui sont des descriptions d'arbres *en forme d'arbre*. Toutefois, il existe une différence avec ce dernier formalisme : ce ne sont pas les nœuds qui seront polarisés mais les traits morpho-syntaxiques attachés à ces nœuds.

Étant donné un ensemble de nœuds $\mathcal{N}$, on notera les descriptions d'arbres soit graphiquement, soit à l'aide de la grammaire suivante[4] :

$$\begin{array}{rcl} \mathcal{D} & =:: & \mathcal{E} \mid \mathcal{D} \wedge \mathcal{D} \\ \mathcal{E} & =:: & \mathcal{N} \xrightarrow{s} \mathcal{N} \\ & & \mid \mathcal{N} \xrightarrow{d} \mathcal{N} \\ & & \mid \mathcal{N} \prec \mathcal{N} \\ & & \mid \mathcal{N} \prec^* \mathcal{N} \end{array}$$

où $\mathcal{D}$ représente les descriptions qui sont des accumulations de descriptions élémentaires $\mathcal{E}$. Chaque description élémentaire définit une relation existante entre deux nœuds de $\mathcal{N}$. Les relations $\xrightarrow{s}$ et $\xrightarrow{d}$ indiquent qu'il existe un arc entre les deux nœuds étiqueté par s ou par d.

Les deux dernières relations concernent des nœuds d'une même fratrie. La première indique que le second nœud est le successeur immédiat du premier dans l'ordre partiel de la fratrie. $n \prec p$ indique que n précède p et qu'aucun nœud successeur de n et prédécesseur de p n'existe. La seconde indique qu'il existe une chaîne entre les deux nœuds, $n \prec^* p$ indique que n précède p.

Exemple 5. *La description d'arbre*

$$\begin{array}{rcl} D & = & n_1 \xrightarrow{d} n_2 \wedge n_1 \xrightarrow{d} n_3 \wedge n_4 \xrightarrow{d} n_5 \wedge n_4 \xrightarrow{d} n_6 \wedge \\ & & n_3 \xrightarrow{s} n_4 \wedge n_2 \prec^* n_3 \wedge n_5 \prec n_6 \end{array}$$

peut se représenter graphiquement comme sur la figure 1.2.

Les descriptions que nous manipulerons seront polarisées, c'est-à-dire que les nœuds seront munis d'une structure de traits polarisés. De plus, de manière à contrôler la sous-spécification de la relation $\xrightarrow{s}$, nous allons aussi munir ces arcs de structures de traits.

[4] Attention cette grammaire permet d'écrire des graphes qui ne sont pas des DAP !

C'est ce que nous appelons les contraintes à longue distance. Elles forceront les nœuds intermédiaires à se superposer avec la structure de traits[5]. Enfin, de manière à donner des informations supplémentaires sur les structures finales d'analyse, on va munir certains nœuds de propriétés, des contraintes, qui seront à vérifier sur la structure finale. C'est ce que nous appelons les marques. Les marques permettent d'étendre les GI pour tenir compte de certains phénomènes ou pour simplifier la grammaire. Par exemple, on utilise des marques pour dire qu'un nœud doit être le plus à gauche ou le plus à droite d'une fratrie, qu'il a un nombre de fils défini par avance, qu'il domine au moins une ancre ou au contraire qu'il n'en domine pas. On peut également définir de nouvelles marques si besoin est. Ce sont en général des contraintes qui ne peuvent être vérifiées qu'au niveau de la phrase, et non simplement au niveau du mot. Nous donnerons quelques types de marques dans les exemples à la fin de cette section.

Il est temps maintenant d'assembler les pièces pour définir les descriptions d'arbres polarisées.

Définition 6 (Description d'arbre polarisée)**.** *Étant donné un environnement Γ construit sur des noms de traits $\mathcal{F}$ et des valeurs de traits $\bigcup_{f\in\mathcal{F}} \mathcal{V}_f$, une description d'arbre polarisée est un quintuplet $(D, N, C, A, \mathcal{M})$ où :*

- *$D = (V, E, l)$ est une description d'arbre. Nous appelons S l'ensemble des arcs étiquetés par s ;*
- *N est une fonction qui associe à chaque nœud de V une structure de traits polarisés sur Γ ;*
- *Pour modéliser certaines contraintes sur des relations non-bornées (les extractions en particulier), nous introduisons la fonction C qui associe aux arcs de S une structure de traits polarisés. On notera $N_1 \xrightarrow{s}_c N_2$ pour indiquer que cet arc est associé à la structure de traits c définie sur Γ*
- *A est un ensemble de nœuds feuilles distingué, les ancres, porteurs de l'information phonologique ;*
- *$\mathcal{M}$ est un ensemble de marques. Ces marques sont des couples formés d'un prédicat et d'un nœud. Ils indiquent certaines spécificités des nœuds correspondants dans le modèle. Nous verrons leur usage dans la section suivante[6].*

Dans la définition précédente, le nombre d'ancres d'une DAP n'est pas fixé. Une DAP peut avoir aucune, une, ou plusieurs ancres. On dira d'une DAP qu'elle est *lexicalisée* si elle a exactement une ancre[7], $|A| = 1$. Les DAP utilisées par notre plateforme logicielle `leopar` sont toutes lexicalisées. Cette distinction aura une importance pour les algorithmes d'analyse.

[5] C'est ainsi que sont modélisées les barrières à l'extraction dans les GI.

[6] Les ancres pourraient être définies comme des marques spéciales. Cependant, pour les besoins de l'analyse il est plus pratique de les distinguer.

[7] La définition que nous adoptons ici diffère de celle utilisée dans d'autres formalismes où il suffit généralement qu'une structure syntaxique ait au moins une ancre pour être lexicalisée.

1.2.5 Exemple

Nous allons étudier un exemple issu de la grammaire d'interaction du français. Sur la figure 1.3, on peut voir les DAP telles qu'elles sont représentées dans l'analyseur `leopar` pour la phrase *Jean en prend.*

Descriptions initiales

Sur la figure 1.3, on peut voir des DAP telles que `leopar` les représente. Ces DAP proviennent de la grammaire d'interaction du français de Guy Perrier. Nous avons une description initiale par mot de la phrase, le point – comme tout signe de ponctuation – étant considéré comme un mot.

Les nœuds des DAP sont représentés par des rectangles contenant les structures de traits polarisés. La relation $\xrightarrow{d}$ est représentée par un trait plein, la relation $\xrightarrow{s}$ est représentée par un trait pointillé, la relation $\prec$ est représentée par une trait plein fléché et enfin la relation $\prec^*$ est représentée par un trait pointillé fléché.

Le cadre des nœuds ancres est doublé, ce qui indique graphiquement qu'ils sont porteurs de la marque *clos* qui indique que le nombre de fils dans la structure d'analyse est fixé. Pour les ancres, il n'y a aucun fils autorisé. Cela permet d'indiquer que les nœuds ancres des DAP interprètent uniquement des feuilles de l'arbre syntaxique. Les autres nœuds ont un cadre simple, le nombre de fils des nœuds qu'ils interprètent est donc libre. Un nœud de *en* est dessiné en blanc en trait simple. Cela signifie qu'il a la marque *vide*. Le nœud qu'il interprète dans la structure d'analyse ne peut dominer un nœud interprété par une ancre. Cette marque joue en quelque sorte le rôle de trace à la Chomsky dans le sens où elle remplit la position canonique, ici de l'objet, sans qu'il y ait de forme phonologique à cette position.

Les traits coindexés, c'est-à-dire les traits dont les valeurs sont partagées, sont indiqués grâce à des entiers entre chevrons.

Contenu syntaxique

Nous pouvons également présenter le contenu linguistique de ces DAP. La première DAP, associée à *Jean*, contient deux nœuds. Sa racine possède deux traits qui possèdent des polarités autres que $=$. Le premier trait $cat \rightarrow np$ indique que ce trait apporte une ressource : le nom propre fournit une catégorie au reste de la phrase. En revanche le second trait $funct \leftarrow ?$ indique que la fonction de *Jean* doit lui être fournie par le contexte par superposition — en l'occurrence par le verbe.

Pour le verbe, la DAP correspond à l'utilisation canonique de ce verbe transitif. Le sujet est avant le verbe et de catégorie nominale tandis que l'objet est après le verbe et également de catégorie nominale. On note que c'est le verbe qui fournit les fonctions aux groupes nominaux qui eux doivent fournir en échange leur catégorie. Finalement la racine de cette DAP fournit la catégorie s (pour proposition) au reste de la phrase. Cette catégorie sera consommée par le symbole de ponctuation.

Il nous reste à détailler la DAP du pronom *en*. Cette DAP va fournir l'objet de catégorie nominale au verbe. Il est à noter que pour cette construction cliticisée, on garde la DAP

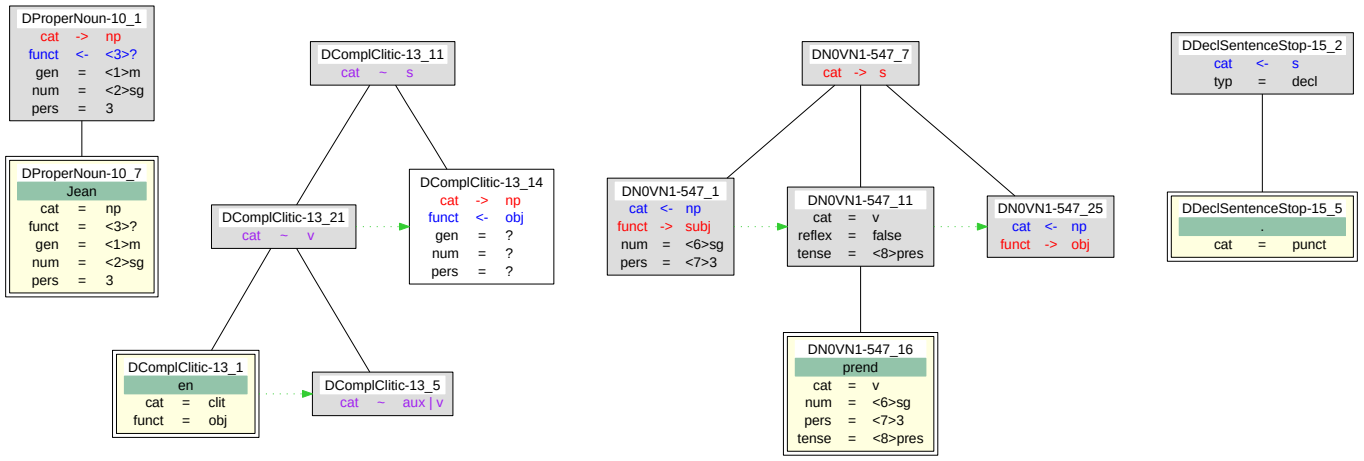


FIG. 1.3 – les DAP initiales pour la phrase *Jean en prend.*

canonique pour le verbe et c'est le pronom qui va s'adapter à cette DAP[8]. Les polarités virtuelles $\approx$ forceront le pronom à se superposer complètement sur le verbe. Les polarités virtuelles servent donc à donner du contexte pour placer les DAP correctement.

Nous allons voir maintenant comment passer d'une suite de DAP à un arbre d'analyse.

1.3 Modèles et interprétations des descriptions

Intuitivement, la composition syntaxique dans les GI se fait par superposition partielle de descriptions d'arbre de manière à obtenir un arbre. Chaque DAP est la contribution de son (ou ses) ancre(s) à la structure associée à la phrase complète.

Ces superpositions respectent les polarités des nœuds. Formellement, il s'agit de définir un morphisme des nœuds d'une suite de descriptions construites sur le même environnement et les mêmes noms de traits vers les nœuds d'un arbre. Nous appelons ce morphisme la *fonction d'interprétation* et l'arbre image de cette fonction *l'arbre syntaxique*.

Si les DAP indiquent la contribution de chaque mot de la phrase à celle-ci, alors la fonction d'interprétation indique comment assembler ces contributions et le résultat de cet assemblage est le modèle syntaxique.

Plus précisément, les arbres que nous considérons sont finis. De plus, les fils d'un nœud sont ordonnés et cet ordre est préservé par héritage aux descendants des fils. Cet ordre permettra de modéliser l'ordre des mots dans la phrase. Soient deux nœuds t_1 et t_2 d'un arbre fini ordonné T. On notera la relation de parenté $t_1 \rightarrow_T t_2$ et la relation de préséance $t_1 \prec_T Nt_2$. L'ensemble des nœuds de T est noté $N(T)$.

Finalement, les nœuds de ces arbres sont étiquetés par des structures de traits classiques : les traits sont de la forme (f, v), où f est un nom de trait de $\mathcal{F}$ et v une valeur de $\mathcal{V}_f$.

Définition 7 (Modèle)**.** *Étant donnée un suite finie de descriptions $D = d_1, \ldots, d_n$ définies sur le même environnement Γ et les mêmes noms et valeurs de traits, un modèle est un couple (T, I) où T est un arbre syntaxique (fini, ordonné et étiqueté) et I la fonction d'interprétation des nœuds de D dans les nœuds de T. I vérifie les propriétés :*

1. *I préserve les relations sur D de la manière suivante :*
   - $n_1 \xrightarrow{d} n_2 \Rightarrow I(n_1) \rightarrow_T I(n_2)$.
   - $n_1 \prec n_2 \Rightarrow I(n_1) \prec_T I(n_2)$
   - $n_1 \xrightarrow{s}_c n_2 \Rightarrow \exists t_1, \ldots, t_n \in N(T)$ tels que $I(n_1) = t_1 \rightarrow_T \cdots \rightarrow_T t_n = I(n_2)$. *D'autre part, si $I(n) = t_i$ alors la structure de traits associée à n doit se superposer à c sans donner d'incohérence.*
   - $n_1 \prec^* n_2 \Rightarrow \exists t_1, \ldots, t_n \in N(T)$ tels que $I(n_1) = t_1 \prec_T \cdots \prec_T t_n = I(n_2)$

   Les deux premières relations s'interprètent à l'identique sur l'arbre syntaxique et les dernières par les fermetures transitives réflexives des premières.
2. *pour tout nœud n de D, si $(f, p, \langle x \rangle)$ est un trait polarisé de la structure de traits associée à n , la structure de traits associée à $I(n)$ contient un trait (f, v) tel que $v \in \Gamma.\langle x \rangle$. En particulier, $\Gamma.\langle x \rangle = \emptyset$ est interdit.*

[8] Ce ne serait pas le cas dans les grammaires d'arbres adjoints qui auraient besoin d'une construction verbale spécifique.

3. *si les structures de traits associées à deux nœuds n et m contiennent respectivement $(f, p, \langle x \rangle)$ et $(f, q, \langle x \rangle)$ alors $I(m)$ et $I(n)$ sont étiquetés par des structures de traits contenant le même trait (f, v).*
4. *la linéarisation des feuilles de T doit inclure la suite de mots $ancre(d_1) \cdots ancre(d_n)$.*
5. *si une DAP définit la marque (P, n) alors on doit vérifier $P(I(n), T)$.*

Le dernier point mérite d'être explicité. Si un nœud n d'une DAP est marqué par P, cela signifie que dans le modèle (T, I) le prédicat P doit être vérifiée par le nœud $I(n)$. Par exemple, si un nœud est marqué comme *feuille*, alors dans le modèle son interprétation doit vérifier le prédicat $feuille$. Le marquage des nœuds permet ainsi d'ajouter de multiples contraintes à la structure finale.

La figure 1.4 représente un modèle pour notre exemple *«Jean en prend.»* On y voit l'arbre syntaxique. Chaque nœud de cet arbre est étiqueté par les noms des nœuds des DAP d'entrée. Par exemple, la racine de cet arbre est l'image par la fonction d'interprétation des racines des DAP pour *en*, *prend* et le point. Grâce au dessin, on retrouve ainsi la fonction d'interprétation. Cette figure illustre un *bon* modèle. C'est exactement le modèle que l'on cherche pour cet phrase.

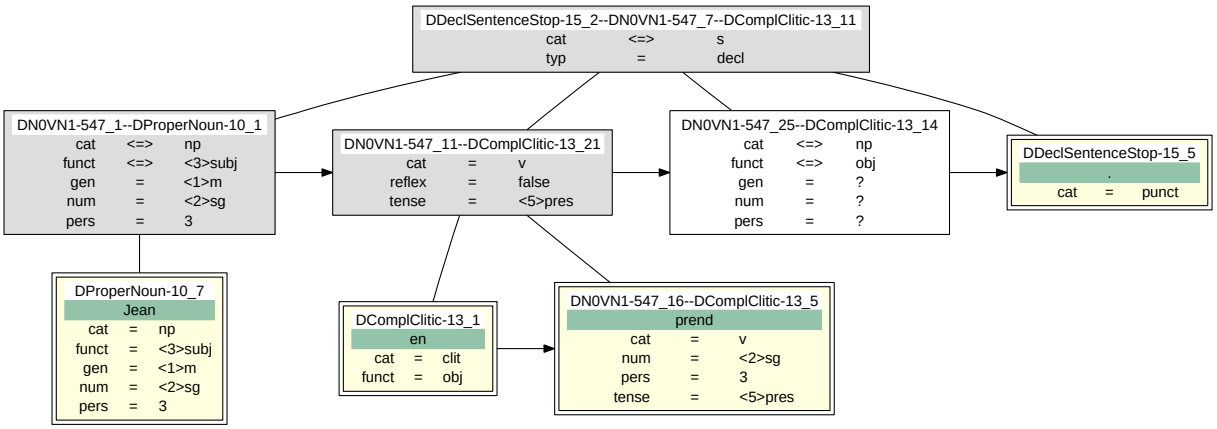


FIG. 1.4 – Le modèle saturé minimal de *Jean en prend.*

Cependant, la définition précédente n'est pas satisfaisante pour définir l'intuition de superpositions partielles de descriptions d'arbres respectant les polarités. Tout d'abord, aucune mention des polarités n'a été faite. Nous allons donc définir une sous classe de modèles dits saturés. De plus, telles que nous les avons définies, les suites finies de descriptions peuvent avoir une infinité de modèles. Nous affinerons là encore les définitions des fonctions d'interprétation et des arbres syntaxiques pour retrouver la notion de modèles minimaux.

1.3.1 Modèles saturés

Nous voulons contrôler les polarités : dans chaque nœud et pour chaque trait, une polarité positive doit rencontrer une et une seule polarité négative et chaque polarité virtuelle doit rencontrer au moins une polarité autre que virtuelle. En d'autres termes, nous voulons rejeter des modèles comme celui de la figure 1.5.

Définition 8 (Modèle saturé)**.** *Un modèle (T,I) d'une description syntaxique D est saturés si pour tout trait de T, les traits de D qu'il interprète sont :*

- *soit tous neutres ou virtuels avec au moins un neutre ;*
- *soit il y a exactement un trait positif, exactement un trait négatif et un nombre quelconque de traits neutres ou virtuels.*

On peut voir cette notion de saturation comme une contrainte supplémentaire que doivent vérifier les modèles ou, d'un point de vue procédural, comme un guide qui aide à la recherche de modèles. Cette dernière intuition servira à développer les algorithmes d'analyse du chapitre 7.

1.3.2 Modèles minimaux

Si une description admet un modèle, elle en admet en vérité une infinité mais nous ne voulons considérer que les modèles qui sont construits exclusivement à partir des DAP sans information supplémentaire. En particulier, le modèle doit être construit uniquement à partir des nœuds des DAP. Cela revient à dire que la fonction d'interprétation est surjective. Mais ces nœuds pourraient contenir des traits qui ne proviendraient pas de la sélection. Il faut également contraindre les structures de traits à être minimales. Enfin, les relations entre nœuds interviennent également dans la notion de minimalité. Les relations $\rightarrow_T$ de parenté dans l'arbre syntaxique doivent toutes être l'image d'une relation $\xrightarrow{d}$ dans une DAP. Pour les relations $\prec_T$ aucune contrainte particulière ne s'applique.

Définition 9 (Modèle minimal)**.** *Un modèle (T, I) d'une suite de descriptions syntaxiques D est minimal si la restriction de I aux nœuds est une surjection, si toute relation de parenté dans T interprète une relation de domination immédiate dans D et si tout trait (t, v) dans T interprète un trait de D.*

Cette notion de minimalité se retrouve très souvent dans l'approche de la syntaxe à travers la théorie des modèles [PS01]. Nous la retrouverons également dans le chapitre consacré à XMG pour la création des grammaires d'arbres adjoints.

Remarque. Maintenant que nous avons présenté d'une part l'opération de superposition de structures de traits et d'autre part les contraintes de saturation et de minimalité des modèles nous pouvons faire le lien entre les deux d'un point de vue opérationnel. Si un nœud n de l'arbre syntaxique de structure de traits T interprète des nœuds $n_1, \ldots, n_m$ de structures de traits respectives $T_1, \ldots, T_m$ (sur un environnement Γ) alors on peut construire la structure de traits $T_p = T_1 + \cdots + T_m$ qui est saturée et si T_p contient le trait $(f, p, \langle x \rangle)$ alors n contient un trait (f, v) tel que $v \in \Gamma.\langle x \rangle$. Et puisque l'opération + est commutative et associative, il sera possible de donner plusieurs stratégies pour arriver à ce résultat. Cette observation fonde les différents algorithmes d'analyse que nous verrons plus loin (chapitre 7).

1.4 Langage engendré

Pour faire des GI un vrai formalisme d'analyse grammaticale, il nous reste à préciser la notion de langage engendré par nos grammaires. De manière à simplifier cette définition, nous allons uniquement nous intéresser aux langages engendrés par des GI lexicalisées qui sont les seules grammaires manipulées par `leopar`. Une grammaire d'interaction est lexicalisée si toutes les DAP qu'elle manipule sont lexicalisées.

Un vocabulaire V est un ensemble de mots. Un lexique L est une application d'un vocabulaire vers des ensembles finis de descriptions initiales lexicalisées. Les descriptions initiales sont les DAP que nous avons présentées.

Définition 10 (Langage engendré)**.** *Soit D un ensemble fini de DAP définies sur un même environnement. Une grammaire d'interaction $G = \{V, D, L\}$ est donnée par son lexique $L : V \rightarrow 2^D$ et définit un langage $\mathcal{L}(G)$ de la manière suivante : étant donnée une liste de mots $w_1, \ldots, w_n$, elle appartient à $\mathcal{L}$ s'il existe une suite de DAP $d_1, \ldots, d_n \in L(w_1) \times \cdots \times L(w_n)$ admettant un modèle saturé et minimal.*

1.5 Conclusion

Nous avons présenté le formalisme des GI. Les structures manipulées, les descriptions d'arbres, peuvent faire penser à la D-théorie [MHF83], aux grammaires de substitution de descriptions d'arbres [RWVS01] ou aux grammaires d'arbres adjoints [VS92]. Dans tous les cas on cherche a étendre le domaine de localité, c'est-à-dire faire en sorte qu'une structure puisse donner des informations sur toutes les contributions d'un mot à la phrase et non pas seulement des informations sur la contribution immédiate.

Nous sommes également proches des grammaires de descriptions d'arbres utilisées dans [DT99]. Il s'agit également d'un formalisme où il n'existe qu'une opération qui ressemble à la superposition des nœuds positifs sur les nœuds négatifs. La différence principale est que dans nos DAP, ce ne sont pas les nœuds qui sont polarisés mais les traits morpho-syntaxiques. Notre opération de superposition est également différente. En effet dans le cas des GI, un nœud de l'arbre syntaxique n'a pas forcément le même nombre de fils que les nœuds qui l'interprètent. En ce sens nous sommes plus proches des grammaires de substitution de descriptions d'arbres de [RWVS01] qui proposent une opération, la *sister adjunction*, qui ajoute des fils à des nœuds et qui est utilisée pour modéliser les modificateurs. Cependant, ce dernier formalisme présente un seconde opération la *subsertion* qui modélise la contribution argumentale et qui elle ne modifie pas l'arité d'un nœud[9]. Les GI n'ont qu'une opération : la superposition.

D'autre part, les polarités rapprochent les GI des grammaires catégorielles où la valence des mots est cruciale. Mais les GI se démarquent des grammaires catégorielles par plusieurs points :

- les polarités sont explicites ;
- les polarités ne sont pas attachées aux syntagmes mais aux traits morpho-syntaxiques ;

[9]Pour être précis, cette opération permet tout de même d'ajouter des fils à une feuille.

- le système de polarités est plus fin avec des traits neutres et virtuels qui permet de traiter certains phénomènes par unification simple ;
- l'utilisation de graphes (les DAP) permet d'avoir une vision plus intuitive de la compositionnalité des structures, au lieu d'avoir recours aux modalités des grammaires catégorielles ;
- enfin et surtout, il existe un important changement de point de vue. Les GC se placent dans un paradigme énumératif/génératif tandis que les GI se placent dans le paradigme de la syntaxe à travers la théorie des modèles. Les GC ne distinguent pas vraiement la composition syntaxique des contraintes sur l'ordre des mots. Il existe une confusion entre les deux notions qui peut compliquer terriblement la modélisation. Les GI permettent de s'affranchir de cette contrainte.

Nous n'avons pas présenté d'algorithme d'analyse pour les grammaires d'interaction. Ce sera l'objet de la partie IV. Nous les avons présentées comme un système de contraintes, inspiré de la *model theoretic syntax*. Nous détaillerons dans la suite un nouvel algorithme d'analyse syntaxique qui permettra d'aborder les grammaires d'interaction selon le point de vue de la syntaxe générative.

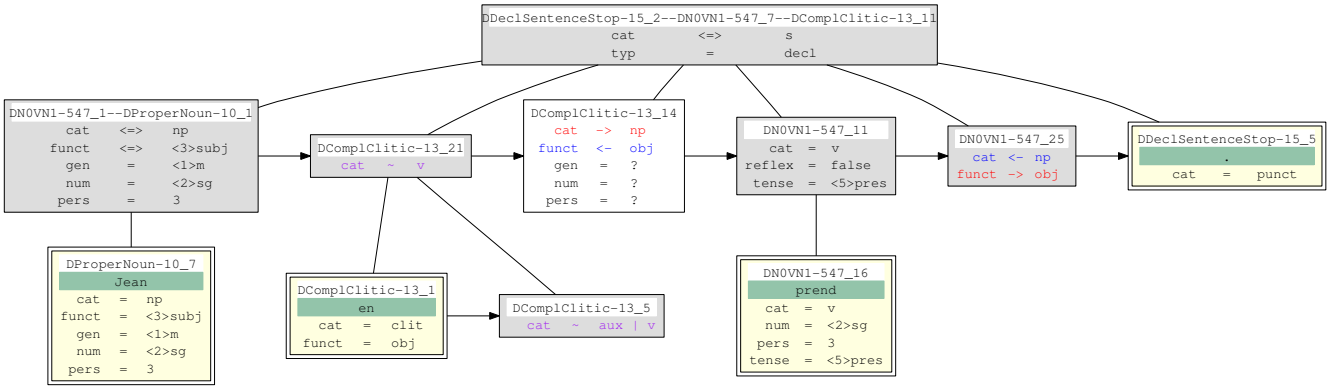


FIG. 1.5 – Un modèle non-saturé de *«Jean en prend.»*

Première partie

Écriture et maintenance des grammaires lexicalisées

Chapitre 2

Des règles lexicales à XMG

Sommaire

2.1 Introduction

Dans le cadre de cette thèse, nous allons être amenés à écrire une grammaire d'interaction de la coordination. Ce travail s'inscrit dans une perspective plus large qui est l'écriture et la maintenance d'une grammaire fortement lexicalisée qui n'est pas un problème propre aux GI. Nous avons pris part au développement d'un outil, XMG, un compilateur de métagrammaires, précisément dans le but de faciliter la production semi-automatique de ce type de grammaires. Dans ce chapitre, nous présentons le problème que constituent l'écriture et la maintenance de grammaires lexicalisées et quelques solutions envisagées pour répondre à ce problème, qui nous amèneront au chapitre suivant à présenter en détail XMG.

2.2 Le besoin de production automatique

Par opposition aux grammaires de règles, les grammaires fortement lexicalisées utilisées en linguistique informatique, comme les grammaires d'arbres adjoints lexicalisées (TAG), les grammaires d'interaction ou les grammaires catégorielles (GC) reportent la connaissance de la langue dans le lexique. Un tel lexique peut être vu comme une fonction associant à un mot de la langue l'ensemble des structures syntaxiques représentant ses usages dans différents contextes. Pour avoir une bonne couverture, il est nécessaire que le lexique associe à chaque mot un maximum de structures. Par exemple pour analyser les expressions ci-dessous, le lexique doit associer au verbe *mange* (en TAG) des structures différentes, représentées en figure 2.1 :

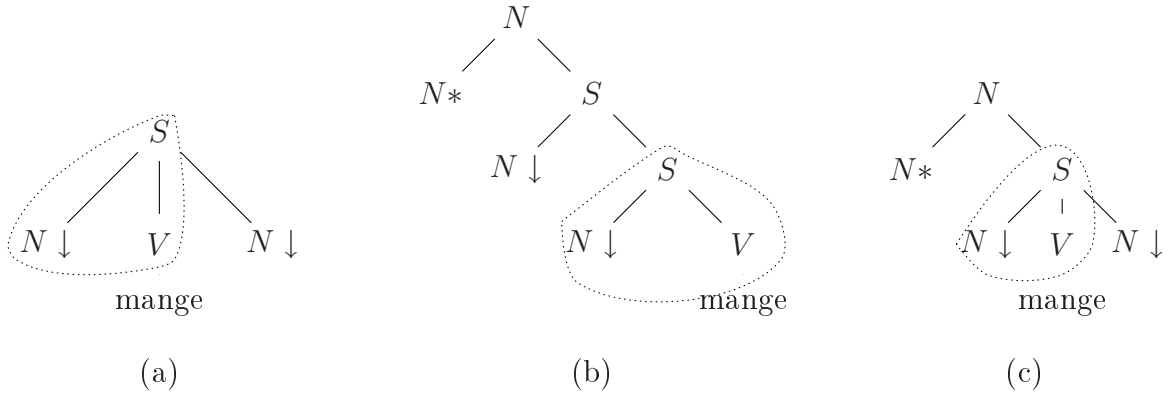


FIG. 2.1 – Arbres TAG pour quelques constructions proches d'un verbe transitif

a) Jean mange la pomme.
b) La pomme que Jean mange ...
c) Jean qui mange une pomme ...

Nous avons également entouré dans chaque arbre de la figure 2.1 la partie responsable de la modélisation de l'accord entre le sujet et le verbe (cet accord s'effectue par coindexation et unification de traits que nous avons omis ici).

Nous avons pris l'exemple des TAG car c'est dans ce formalisme que les problèmes d'écriture de grammaires lexicalisées ont surgi en premier. Les GI ou les GC n'ont pas besoin de structures différentes pour ces exemples, mais le principe est aussi valable pour ces formalismes Pour les GI, ce phénomène est également présent. Sur la figure 2.2, on peut voir quelques exemples de DAP que nous associerons à la coordination dans la grammaire présentée dans le chapitre 5. Nous avons donné quatre exemples de DAP pour la coordination de propositions avec éventuellement montée de nœud périphérique. On peut reconnaître la forme en trois parties autour du nœud ancre que nous développerons au cours de notre modélisation. Mais dans sa présentation au sein des GI lexicalisées, cette proximité de forme n'apparaît pas explicitement.

Nous parlons dans le cas des TAG de lexiques associant à certains verbes des milliers d'arbres et pour les GI de dizaines voire de centaines de DAP. Cette taille importante de lexique ne va pas sans problème. Comme les règles syntaxiques sont éclatées à travers le lexique, l'analyse d'un nouveau phénomène linguistique ou, pire, sa révision peuvent avoir de graves conséquences sur la cohérence de la grammaire. Si les premiers lexiques à large

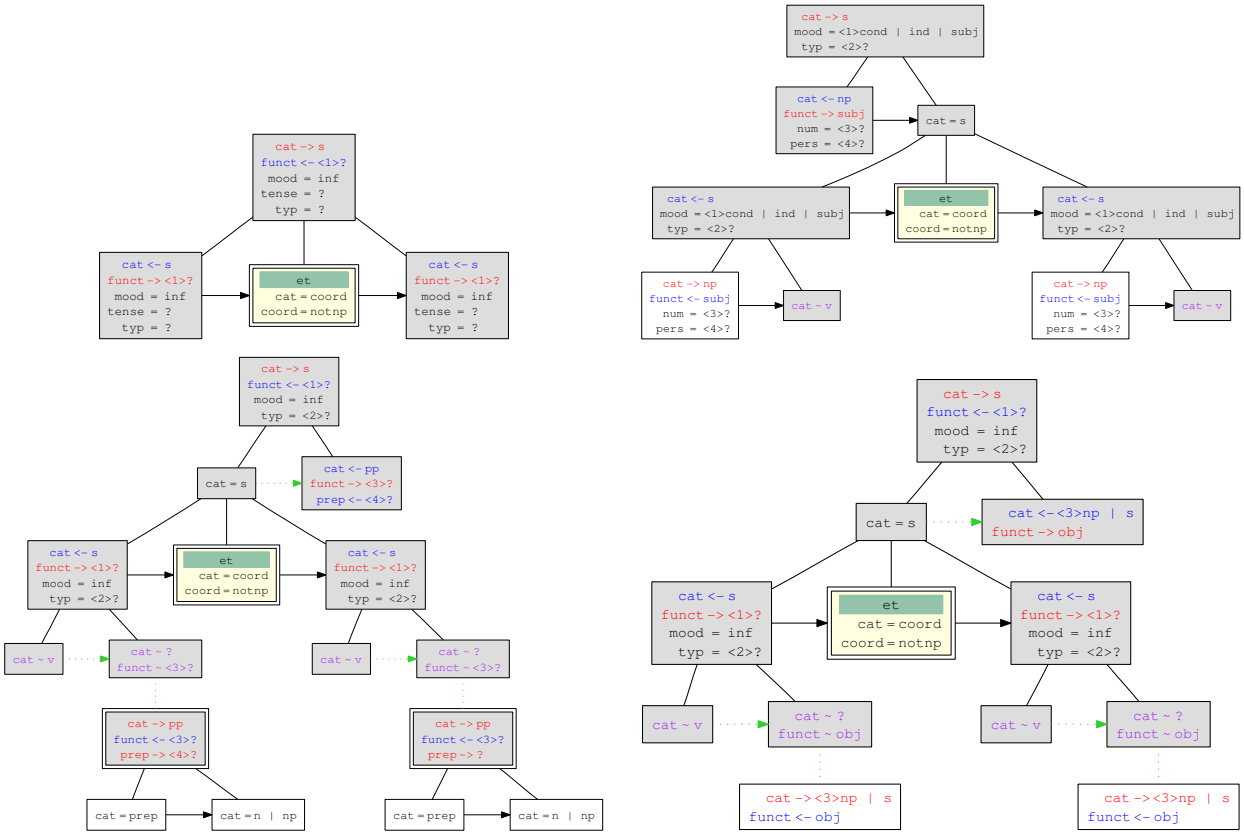


FIG. 2.2 – Quelques DAP associées à la coordination

couverture furent écrits à la main, leur génération automatique devient de plus en plus pressante, de manière à :

1. pouvoir garantir la cohérence de l'ensemble ;
2. pouvoir réviser une grammaire rapidement, même quand le lexique devient important.

Le premier point signifie que les formes *mange, mangeraient, aiment,...* doivent avoir une image similaire par le lexique si l'on considère qu'elles ont le même comportement syntaxique. Une première étape, que l'on retrouve dans toutes les grammaires à large couverture, est l'utilisation de structures syntaxiques non-ancrées, c'est-à-dire de structures qui ne portent plus de feuilles contenant explicitement une forme fléchie. À la place, la feuille dite ancre est marquée comme porteuse de cette forme fléchie et une opération préalable à l'analyse, appelée *ancrage*, doit être effectuée pour chaque mot de la phrase. Elle consiste à vérifier que la forme fléchie peut bien être l'ancre de la structure non-ancrée. Cette opération est le plus souvent fondée sur l'unification ou le filtrage de structures de traits morpho-syntaxiques. Cette première étape permet de réduire grandement la taille du lexique. Cependant, le vrai sens de la cohérence est le suivant : un phénomène linguistique doit être modélisé de la même façon par toutes les structures syntaxiques qui

doivent le faire. Par exemple, l'accord entre sujet et verbe doit être modélisé de manière uniforme par toutes les structures syntaxiques de la grammaire.

Le second point est évidemment lié à cette notion de cohérence interne du lexique. Si l'on veut changer la modélisation d'un phénomène particulier, ce changement doit être propagé à toutes les structures syntaxiques qui y font référence. L'utilisation de schèmes est ici encore nécessaire mais insuffisante. L'idée est de décomposer le lexique en plusieurs niveaux : dans un premier temps on se contente de décrire ces phénomènes de la façon la plus indépendante possible et dans un deuxième temps d'expliquer comment les structures syntaxiques y font référence. Dans l'exemple précédent, on décrirait en premier lieu l'accord entre sujet et verbe puis la manière dont les constructions (par exemple canonique ou avec extraposition) y font référence.

Nous avons développé un formalisme, dont l'implantation est XMG, qui, à partir d'une description dite métagrammaticale concise génère toutes les structures associées aux mots du lexique. Notre approche se veut la plus indépendante possible des formalismes grammaticaux et capable de gérer simultanément les aspects syntaxiques et sémantiques du lexique. Ce n'est pas la première tentative de réponse au problème du développement de grammaires fortement lexicalisées. Nous allons donc dans un premier temps présenter ces premières propositions.

2.3 L'approche métagrammaticale

2.3.1 Un problème majeur : la redondance

Le credo du lexicalisme, *tout est exception*, pose de nombreux problèmes aux grammairiens. D'une part, lors du développement de grammaires, il faut décrire chaque structure syntaxique séparément : puisque chaque structure syntaxique est censée être unique, elle ne partage rien avec ces *proches*, c'est à dire les structures qui modélisent des constructions proches.

Certaines généralités sont donc impossibles à énoncer dans les grammaires lexicalisées. Par exemple, il est probable que les structures syntaxiques associées à un verbe transitif dans les cas d'affirmatives ou d'interrogatives partagent beaucoup d'informations communes. Cette généralisation, que l'on pourrait rendre par une factorisation de cette partie commune, est impossible à établir dans les formalismes fortement lexicalisés. Cette observation est notamment faite par [VSS92]. Dans la cadre de la maintenance de ces grammaires, qui est peut-être une tâche encore plus ardue que l'écriture elle même, si l'on veut modifier l'analyse d'un phénomène présent dans plusieurs constructions, chacune représentée par une structure lexicale distincte, il faut modifier chaque structure séparément. Durant ces modifications, la cohérence générale de la grammaire n'est pas préservée.

Nous venons d'exhiber le problème majeur des grammaires fortement lexicalisées : la redondance de l'information qui rend l'écriture fastidieuse et la maintenance difficile à cause de la gestion manuelle de la cohérence interne.

Un autre problème, moins pressant techniquement mais tout de même important d'un pont de vue linguistique, est l'impossibilité d'énoncer des alternances ou tout au moins

des constructions alternatives, ce qui permettrait d'un point de vue pratique d'avoir une vue plus synthétique de la grammaire et simplifierait le travail d'écriture et d'un point de vue plus théorique d'énoncer l'équivalence de certaines réalisations. Par exemple, on peut dire qu'un sujet se réalise soit sous la forme nominale soit sous la forme phrastique (*« Ce principe est important. »* ou *« Bien manger est important. »*) et que ces deux constructions sont en alternance. Si dans le lexique on fait référence à la notion de sujet, on veut concrètement faire référence à l'une et l'autre de ces constructions. C'est cette notion d'alternance qui différencie les approches que nous allons présenter.

On peut toutefois mettre ces deux problèmes en relation. On parlera de *redondance verticale* pour indiquer que des fragments communs peuvent se trouver présents dans différentes structures, comme les fragments entourés de la figure 2.1, problème qu'une hiérarchisation de la grammaire peut résoudre, et de *redondance horizontale* pour le problème des constructions alternatives, quand des fragments jouent un même rôle dans l'organisation de la grammaire. Le traitement de la redondance verticale fait l'objet d'un consensus : l'héritage entre structures est la solution de choix, voir [Fli87]. Par exemple, les structures de la figure 2.1 hériteront toutes du fragment définissant la relation entre sujet et le verbe qui ne sera écrit qu'une seule fois. Cette notion d'héritage est présentée dans la prochaine section. En revanche, pour la redondance horizontale, différentes approches existent. Avant de décrire la solution adoptée dans cette thèse, la métagrammaire, nous évoquerons les règles lexicales.

2.3.2 Héritage ou réutilisation

Nous nous intéressons ici à la définition de la notion d'héritage entre objets de la grammaire. Plutôt que de décrire directement les structures présentes dans le lexique, on va plutôt se consacrer à l'écriture de fragments réutilisables de telles structures.

Certains fragments pourront donc en réutiliser d'autres, c'est à dire ajouter de l'information à l'information contenue dans ces autres fragments et être à leur tour réutilisables par d'autres. Cette notion, d'entités réutilisables et extensibles est depuis longtemps présente en informatique. La programmation orientée objet, dont la première pierre est sans doute [Hoa65], répond à ce problème d'une manière assez intuitive.

Le point clé est la notion d'héritage entre les classes. Une classe C_1 peut être définie en différence par rapport à une autre classe C_2, c'est à dire que le comportement des instances de C_1 est le même que celui des instances de C_2 sauf précision explicite dans la définition de C_1. On dit que C_1 hérite de C_2, que C_2 est la superclasse de C_1 ou que C_1 est une sous-classe de C_2.

Cette idée fondamentale est reprise dans de nombreuses propositions d'organisation des lexiques. La proposition la plus considérable est sans doute [Fli87; FPT85; FN92] pour les grammaires syntagmatiques dirigées par les têtes (HPSG). Flickinger définit des structures de traits sous-spécifiées qui sont affinées par héritage. C'est à dire que des structures de traits sont définies par réutilisation d'autres structures, en les complétant ou en les modifiant.

Il donne l'argument suivant pour justifier son organisation : une grammaire de type HPSG standard contient entre 30 et 50 noms de traits pour chacun desquels il existe au minimum 2 valeurs possibles. Il existe donc entre $2^{30} = 10^9$ et $2^{50} = 10^{15}$ structures

de traits dans cette hypothèse optimiste. Bien sûr, dans une grammaire réaliste, toutes les possibilités ne sont pas autorisées mais il n'empêche que la taille d'un lexique reste énorme. Il convient de noter que certains traits et certaines valeurs se retrouvent dans plusieurs entrées lexicales. On définit donc des fragments de structures sous-spécifiées qui ne contiennent qu'une partie des traits et qui sont réutilisées en bloc. La taille du lexique se trouve donc réduite considérablement mais il faut maintenant *construire* le lexique réel à partir de sa description : il faut assembler les fragments.

On peut noter que dans cette proposition, on est en présence d'un vrai héritage, au sens de la programmation orientée objet, puisqu'une sous-classe est autorisée à effacer, c'est à dire à contredire, l'information donnée par une superclasse. Ce mécanisme n'est donc pas monotone, un trait présent dans la hiérarchie peut disparaître ou être modifié, l'ordre d'évaluation qui donne le contenu d'une classe a une importance. On le verra par la suite notre proposition diffère car nous voulons un formalisme déclaratif et nous sommes donc contraints d'utiliser une version faible de l'héritage, la spécialisation, où on ne peut qu'ajouter de l'information et non pas contredire l'information déjà présente.

Voyons maintenant comment le problème lié à la redondance horizontale est traité.

2.3.3 Règles lexicales

Le problème du statut et de la structuration des alternances du lexique a d'abord été résolu par l'emploi de règles lexicales [Bre82; SUP+83]. Ces règles sont encore utilisées pour produire des HPSG [CF00] et des TAG [Bec00; Pro02]

Ces règles lexicales indiquent comment transformer une structure correspondant à une certaine construction en la structure d'une construction alternative. Par exemple, pour un verbe transitif on explique comment transformer la structure de la voie active en la structure pour la voie passive. Puis de la voie passive, on passe généralement à la voie passive sans agent en *effaçant* la partie de la structure syntaxique correspondant à l'agent.

$$\text{Passif sans agent} ::= \text{Efface} \circ \mathit{Passive} \circ \mathit{Construit}(\mathit{transitive})$$

L'expression ci-dessus résume la construction du passif sans agent. Tout d'abord on construit la structure canonique complètement spécifiée d'un verbe transitif en parcourant la hiérarchie d'héritage du lexique et en accumulant les structures sous-spécifiées des superclasses. Ensuite on applique la règle de passivation qui va réécrire la structure. Puis la règle d'effacement est appliquée.

Les constructions complexes sont donc rendues par des chaînes de transformations simples. Mais comme certaines structures ont été modifiées, l'ordre d'applications de ces règles est crucial. Dans notre exemple, l'application de *Efface* sur la structure associée à *Construit(transitive)* ne donnerait aucun résultat. C'est l'inconvénient majeur de cette approche car le linguiste, même s'il gagne en concision puisqu'il ne décrit que les constructions canoniques, doit définir des suites de transformations qui s'enchaînent.

D'autre part, ces règles s'appliquent généralement jusqu'à stabilisation de l'ensemble des structures syntaxiques c'est à dire que la génération de nouvelles structures continue jusqu'à ce qu'aucune nouvelle règle ne puisse être appliquée. Rien n'empêche des suites de règles qui produisent des structures de plus en plus importantes sans stabilisation, ce

qui entraîne généralement l'indécidabilité de ce genre de systèmes. Toutefois, comme le précise [Pro02], cet aspect est moins gênant dans le cas de la génération de grammaires que dans le cas de l'analyse, des grammaires transformationnelles en particulier. Les cas *pathologiques* sont connus et sont traités une fois pour toutes lors de la génération et non pas aléatoirement en fonction de l'entrée de l'analyseur. Mais l'auteur de l'article admet que les résultats obtenus ne correspondent pas toujours aux résultats attendus. Dans certains cas, il y a sous-génération car certaines règles ne s'appliquent pas alors que dans d'autres cas les règles s'appliquent de manière trop permissive et sont responsables de sur-génération d'arbres TAG. En pratique, il semble qu'il faille un temps d'ajustement pour maîtriser complètement l'ordre d'applications des règles tout en évitant les règles qui s'appliquent en boucle.

Enfin, l'écriture d'une grammaire à vocation linguistique se réduit à l'écriture de règles de réécriture d'arbres. Bien que ces règles sont pour la plupart naturelles et faciles à énoncer, on perd vite l'intuition linguistique du fait de problèmes techniques à propos de filtrage et de transduction d'arbre. C'est un des reproches adressés à l'encontre de cette approche par [Can96]. Le linguiste doit se préoccuper de bien modéliser la langue. Les problèmes techniques sont du ressort du système utilisé. Plus profondément, c'est ici le statut des alternances qui est en jeu. En effet, à chaque application d'une règle lexicale on s'éloigne un peu plus de la structure canonique. On peut très bien imaginer des règles lexicales qui transforment complètement leur entrée et dépassent donc le cadre des alternances. Les règles lexicales sont en quelque sorte trop puissantes pour les alternances.

Ces inconvénients nous semblent en contradiction avec le but fixé : pouvoir décrire une grammaire sans connaître le processus de génération ni l'ordre dans lequel il effectue ses différentes étapes et disposer d'un formalisme simple qui puisse être traité rapidement par une machine. C'est pourquoi nous serons amenés à rejeter complètement l'approche par réécriture qui impose un formalisme procédural.

2.4 Une solution : la métagrammaire

Les problèmes de hiérarchisation et d'alternance sont également abordés par l'approche métagrammaticale initiée par [Can99]. Le grammairien n'écrit plus directement la grammaire cible, qui équivaut à écrire un ensemble de structures syntaxiques dans un formalisme fortement lexicalisé, mais une description concise, non redondante, de cette grammaire cible que l'on appelle la métagrammaire. On peut faire le parallèle avec la programmation : on passe de l'assembleur aux langages de haut niveau. Cette métagrammaire est constituée de fragments de structures syntaxiques (ou de manière équivalente de structures syntaxiques sous-spécifiés) et, selon les différentes solutions proposées, de règles de combinaisons de ces fragments pour produire les structures complètes. Cette production est effectué par un *compilateur de métagrammaire*.

Toutes les approches proposent comme premier moyen de résoudre le problème posé par la redondance verticale la réutilisation du contenu d'un fragment par d'autres fragments, ce qui revient à une notion d'héritage ou de spécialisation/enrichissement de fragments que nous avons déjà présenté plus haut en section 2.3.2. Par contre, ce sont ces règles de combinaison pour éviter la redondance horizontale qui ont évolué au cours des

différentes propositions qui s'éloignent de plus en plus de l'approche par règles lexicales pour s'orienter vers une voie purement déclarative où seules les alternances sont énoncées.

2.4.1 La proposition originale

Pour [Can96] le grammairien ne donne pas les règles lexicales évoquées plus haut, elles sont *en dur* dans le formalisme proposé. En réalité, dans ce cadre-ci on ne distingue pas encore clairement la métagrammaire du compilateur de métagrammaire. Il existe trois types de fragments, appelés dimensions[10] :

- les fragments décrivant un cadre de sous-catégorisation, c'est à dire le nombre d'arguments et leur catégorie (dimension 1),
- les fragments responsables de la redistribution des fonctions syntaxiques (dimension 2)
- et enfin les fragments de réalisation des fonctions finales (dimension 3).

Les fragments des dimensions 1 et 2 sont essentiellement ce que l'auteur appelle des méta-équations, que l'on pourrait mettre sous forme de structures de traits. Ces fragments sont organisés en hiérarchies d'héritage.

La dimension 3 contient des fragments d'arbres. Ces fragments d'arbres sont en fait des quasi-arbres de [RVS92] dans des hiérarchies d'héritage monotones (ou de spécialisation). Les quasi-arbres sont des descriptions d'arbres spécialisées pour admettre comme modèles des arbres TAG. Cette idée d'utiliser l'héritage monotone et les descriptions d'arbres provient de [VSS92] qui avait traité le problème de la redondance verticale pour les TAG.

Le compilateur va combiner, ou croiser, les fragments finaux de la dimension 1 avec les fragments de la dimension 2. Ce premier croisement va attribuer une fonction aux arguments dont certains peuvent se trouver effacés (ici encore le cas du passif sans agent). Puis dans un second temps, ces nouveaux fragments obtenus vont être croisés avec les fragments de la dimension 3 afin de réaliser les arguments. Cette dernière dimensions contient des descriptions d'arbres. Les croisements avec la dimension 3 sont répétés jusqu'à stabilisation de l'ensemble des structures produites. Finalement les descriptions sont normalisées pour obtenir des arbres. Cette dernière étape consiste essentiellement à transformer des relation de dominance en relation de parenté. La figure 2.3 montre les classes croisées pour une construction passive avec extraction (*l'homme par qui la chanson est interprétée*).

Cette première proposition fut une avancée importante pour le développement de grammaires TAG fortement lexicalisées. Cependant elle souffre de quelques lacunes :

- elle ne permet d'écrire que des grammaires d'arbres adjoints, et rien n'est prévu pour générer d'autres types de grammaires.
- Cette approche fut initiée dans le but d'écrire une grammaire des verbes. Le nombre et le rôle des dimensions sont figés. Il est possible que ces dimensions ne permettent pas d'exprimer les intuitions du linguiste. Il faut donc dans ce cas réécrire un compilateur.
- Les identifiants (noms de variables) sont globaux, ce qui rend difficile l'écriture de la métagrammaire à cause des risque de collision de noms. Ce problème peut

[10] Attention le mot dimension n'a pas le même sens dans la section 3.1.2.

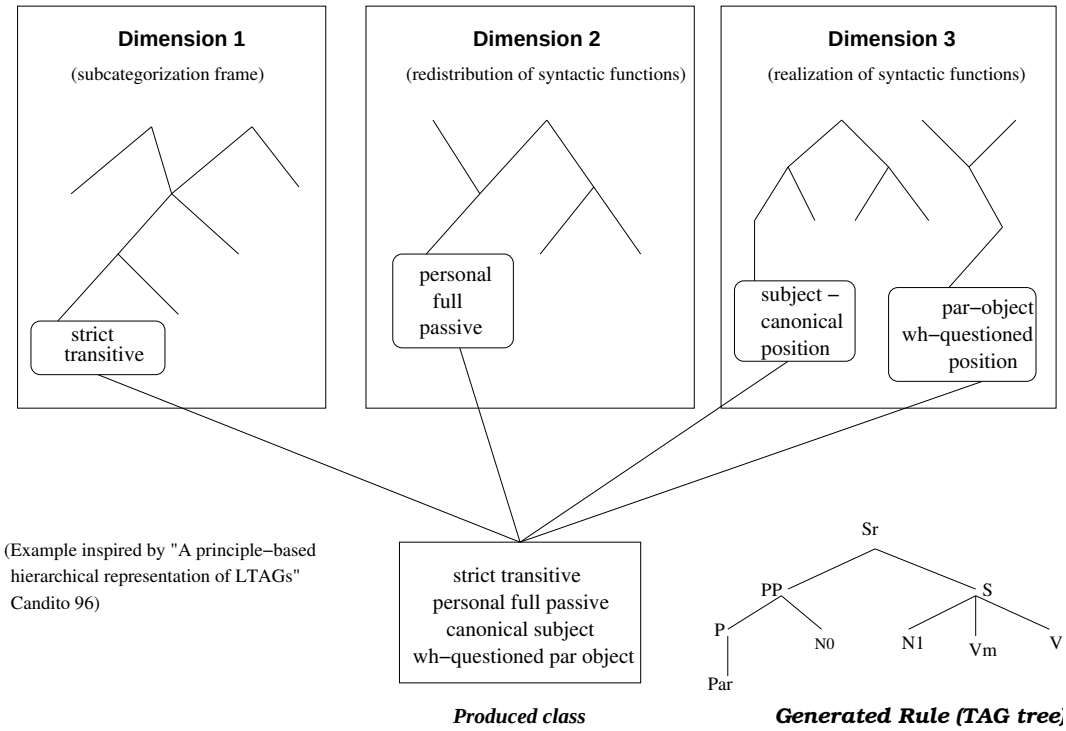


FIG. 2.3 – Croisement de classes pour les passifs avec extraction (Par qui ...)

paraître trivial mais en pratique il est très important. Il est difficile d'écrire une métagrammaire de taille importante et surtout il est quasiment impossible d'étendre une métagrammaire déjà existante. Ce problème empêche de modéliser simplement certains phénomènes. En effet, puisque les noms de nœuds sont globaux, deux exemplaires d'une même classe sont considérés comme strictement identiques puisqu'ils manipulent les mêmes noms. Une classe qui doit être utilisée plusieurs fois dans un croisement doit donc être définie plusieurs fois, avec des noms différents.

– Il est difficile de prévoir le comportement du compilateur. En effet, il faut tester tous les croisements entre des fragments de la dimension 1 et ceux de la dimension 2, dont certains ne vont pas donner de structure cohérente, et croiser le résultat de ce premier croisement avec un nombre quelconque de fragments de la dimension 3 où là encore certains croisements ne vont rien donner. L'existence d'un fragment d'une structure peut être remise en cause par l'application d'une redistribution avec effacement. En d'autres termes, l'ordre dans lequel sont effectués les croisements a une importance sur les structures produites, ce qui ne permet pas de raisonner facilement sur le comportement du programme, ni de l'étendre d'ailleurs. D'un point de vue calculatoire, cette approche est peu performante, car de nombreux croisements sont effectués inutilement, alors que l'on pourrait explicitement spécifier qu'ils sont incohérents, et qu'il n'y a pas lieu de les traiter.

Le dernier point ci-dessus souligne combien cette première proposition est encore influencée par les règles lexicales. La différence avec les approches antérieures est qu'ici le linguiste n'écrit pas de règles de réécriture d'arbres. Les effacements sont réalisés par le croisement de la dimension 1 avec la dimension 2, qui se fait sans arbre.

2.4.2 Une première révision

Les travaux présentés dans [XPVS98; XPVS99] constituent une première amélioration du travail de [Can96], en tout cas un premier pas vers l'abstraction des dimensions, un des principaux reproches faits à cette première approche. Toutefois, cette proposition se rapproche des règles lexicales. Une métagrammaire est constituée

- de blocs, c'est à dire de fragments d'arbres, divisés en deux groupes : les blocs de sous-catégorisation et les blocs de transformation. Les premiers peuvent apparaître dans des constructions canoniques contrairement aux seconds (extraction par exemple),
- de cadres de sous-catégorisation qui sont des structures de traits. Ils permettront de sélectionner des blocs en fonction de leur forme,
- de règles de redistribution lexicales qui sont des règles de réécriture de cadres de sous-catégorisation.

Une règle de redistribution est un couple (r_g, r_d) de cadres de sous-catégorisation qui transforme un cadre s en un nouveau cadre. La première étape vérifie que s est une entrée valide pour cette règle en filtrant s par r_g. Si le filtrage aboutit, alors s et r_d sont combinés pour produire un nouveau cadre de sous-catégorisation. On a donc un certains nombre de cadres canoniques (par exemple la voie active des verbes transitifs) et les règles de distribution qui permettent de générer les cadres des autres constructions (passif et passif sans agent). On écrit donc des règles de réécriture de cadres.

Chaque cadre de sous-catégorisation initial génère par applications de règles de redistribution de nouveaux cadres de sous-catégorisation. À partir de ces nouveaux cadres, on sélectionne des blocs de sous-catégorisation que l'on assemble puis de blocs transformationnels qui seront ajoutés à ce premier assemblage. On obtient ainsi pour chaque cadre, une famille d'arbres TAG.

À la différence de [Can96], dans cette approche les règles de redistribution ne sont plus fixées par les croisements mais elles sont données par l'utilisateur. Par contre, l'ordre d'application des règles de redistribution est toujours significatif, ce qui en fait un formalisme non monotone au détriment de l'efficacité (toutes les combinaisons de règles doivent être essayées) et de la simplicité d'utilisation (l'utilisateur doit comprendre dans quel ordre s'appliquent les règles). Par contre les combinaisons de descriptions d'arbres induites par les cadres de sous-catégorisations sont monotones. Il est également impossible d'énoncer explicitement les alternances, uniquement des transformations.

Par contre, le fait de ne pas utiliser de dimension au sens de Candito, permet d'organiser plus librement la grammaire. L'approche suivante poursuit dans ce sens.

2.4.3 Une approche orientée besoins et ressources

[CGR04; GCR02] ont présenté une généralisation de la première métagrammaire. Le nombre de dimensions est arbitraire. En effet, ils critiquent essentiellement la troisième dimension qui, dans les faits réalise la majorité des croisements. Les première et deuxième dimensions sont cantonnées au rôle d'initialisateurs du processus de croisement. Ils prônent donc une réorganisation de la métagrammaire en plus de classes dont le rôle est bien délimité.

En conséquence, les croisements des fragments finaux de chaque dimension ne sont plus connus à l'avance par le compilateur de métagrammaire. Le linguiste doit pouvoir les expliciter. Chaque fragment d'arbre TAG est inclus dans une classe. Une classe contient également deux ensembles de symboles. Le premier contient les besoins de la classe et le second, les ressources qu'elle fournit. Ce sont donc ces besoins/ressources qui guident le processus de croisement.

Chaque fois que deux classes C_1 et C_2 sont croisées, la nouvelle classe produite C hérite des besoins et des ressources de ses ancêtres mais les symboles qui figurent à la fois en tant que besoin et ressource sont retirés : la classe produite a donc moins de besoins et moins de ressources que C_1 et C_2. Le processus de croisement s'arrête quand aucun croisement n'est plus possible. Les classes finales sont celles qui n'ont ni besoin ni ressource restant : ce sont les classes *équilibrées*. Les descriptions d'arbres de ces classes équilibrées permettent de calculer les arbres de la grammaire cible. Ce processus de croisement, jusqu'à disparition des besoins et des ressources, est exponentiel dans le nombre de classes. En pratique, il faut plusieurs heures pour traiter une métagrammaire *réaliste*.

À partir de cette proposition, on commence à se détacher des règles lexicales. Les alternances ont un statut plus clair, du moins on s'éloigne de la vision transformationnelle des approches précédentes. Les classes qui ont les mêmes besoins et les mêmes ressources définissent un ensemble de classes équivalentes, une alternance. Bien sûr, la question suivante se pose : quel est le statut de deux classes dont les besoins/ressources diffèrent légèrement ? Sont-elles équivalentes ? Sont-elles réellement différentes ? Finalement, cette granularité particulièrement fine n'a pas de correspondance linguistique directe avec la notion d'alternance.

Cette nouvelle approche ne permet toujours pas de spécifier ou d'interdire des croisements directement bien que le mécanisme de besoin/ressources puisse être utilisé indirectement dans ce sens. La gestion des identifiants est toujours globale. C'est sans doute pour cette raison, la difficulté d'écrire une métagrammaire, que cette proposition prometteuse n'a pas été plus utilisée.

Enfin, les TAG restent le formalisme cible unique, bien que [CK03] montrent que l'on pourrait l'adapter pour produire des grammaires lexicales fonctionnelles (LFG).

2.4.4 La nouvelle école : MGCOMP et XMG

Les nouveaux compilateurs de métagrammaires MGCOMP et XMG, présentés dans [TV05; DLP05] tentent de corriger les défauts des approches précédentes. Il est étonnant de voir combien ils convergent dans leur choix de solution. XMG étant l'objet du prochain chapitre, nous présentons ici MGCOMP.

La première évolution marquée est la volonté de générer d'autres formalismes que les TAG. Même si l'implantion actuelle de l'outil ne le permet pas toujours, il est prévu de produire des TAG, des TAG multicomposants (MC-TAG), des grammaires d'arbres insérés (TIG), des grammaires à concaténation d'intervalles (RCG) ou des grammaires de clauses définies (DCG).

Ensuite, il permet également d'ajouter des informations autres que syntaxiques dans les fragments, comme de la sémantique, par exemple, ou des structures de traits pour l'ancrage des structures syntaxiques dans le lexique, les hypertags [Kin00]. Les différents types d'information sont clairement séparés dans chaque classe, chacun définissant une *facette* de la classe.

MGCOMP reprend la notion de besoins et ressources présentée précédemment. Le croisement des classes se fait jusqu'à neutralisation des classes. La différence est ici que les besoins et les ressources sont attachées aux classes mais également aux nœuds des fragments définies dans chaque classe.

Les problème des identificateurs globaux est résolu par l'ajout d'espaces de nommage associés aux classes. Un espace de nommage est un concept venant des langages de programmation. C'est le contexte dans lequel des identificateurs peuvent être définies et dans le quel ces derniers sont visibles. On le retrouve dans les langages de programmation orientée objet et dans le format d'échange XML. Ici, un identificateur n'a de sens que dans une classe. Des primitives permettent de fusionner des espaces de nommage, c'est à dire importer certains identificateurs et les partager. Elles permettent d'interdire la collision de noms inhérente aux approches par nommage global de [Can96] et [CGR04] qui rendaient fastidieuse l'écriture de métagrammaire de taille importante. Une classe peut être utilisée plusieurs fois lors d'un croisement.

MGCOMP permet d'imposer ou d'interdire explicitement certains croisements, de manière à donner plus de contrôle au grammairien, à rendre plus efficace le mécanisme croisement et surtout de modéliser clairement les exceptions. MGCOMP fournit un mécanisme de gardes qui peuvent bloquer certaines constructions d'arbres

Enfin, MGCOMP permet de générer des grammaires factorisées (dans le cas des TAG). Il s'inspire en cela de l'extension des grammaires hors-contexte de [NSS03]. Les arbres TAG sont des arbres finis complètement ordonnés. Dans MGCOMP :

- l'ordre entre fils peut être sous-spécifié grâce à un opérateur d'entrelacement de sous-arbres,
- certaines branches sont optionnelles. Les gardes dont nous avons parlé permettent justement de décider de la présence ou non de la branche au moment de l'analyse,
- certains nœuds sont répétées arbitrairement (par l'étoile de Kleene des langages réguliers)

Chaque structure de la grammaire produite correspond à un ensemble d'arbres TAG conventionnels. Ce type de grammaires est pour l'instant uniquement traité par l'analyseur DYALOG (voir par exemple [Vil04]).

2.5 Conclusion

Les formalismes lexicalisés souffrent donc d'un défaut majeur qui est la redondance et l'éclatement de l'information à travers les structures syntaxiques. Plus personne ne songe à écrire une grammaire lexicalisée qui ait une bonne couverture de la langue à la main. La grammaire réelle est construite à partir d'une description et d'un mécanisme d'assemblage à partir des objets de cette description.

Nous n'avons pas pu évoquer toutes les propositions faites pour résoudre les problèmes de redondance dans les grammaires fortement lexicalisées. Cependant, nous avons essayé de montrer l'évolution de ces propositions. D'une part, l'héritage semble être la solution idéale pour résoudre la redondance verticale. Toutes les propositions que nous avons mentionnées l'utilisent. D'autre part, dans les approches les plus récentes, les alternances ne sont plus énoncées comme des chaînes de transformations mais par des classes qui ont les mêmes besoins et les mêmes ressources.

Nous le verrons dans le chapitre suivant, XMG opte pour une approche différente. L'héritage, même s'il est présent dans le langage utilisateur de ce formalisme, n'est pas une primitive de XMG. À la place, on trouve la relation de conjonction (ou réutilisation ou agrégation). L'héritage est une spécialisation de cette relation. Mais la différence fondamentale, c'est que l'on disposera d'une primitive qui indique clairement l'alternance : la disjonction. Conjonction et disjonction amènent naturellement à voir la métagrammaire comme un programme en logique et le compilateur de métagrammaire comme une machine virtuelle de type PROLOG.

Chapitre 3

XMG

Sommaire

En collaboration avec Benoît Crabbé, Denys Duchier et Yannick Parmentier, nous avons contribué au développement du formalisme *eXtensible MetaGrammar* (XMG) qui définit une organisation des lexiques des grammaires fortement lexicalisées. Nous avons également pris une part active au développement du logiciel du même nom.

Dans ce chapitre nous présentons le formalisme et l'outil qui y est lié. Dans la suite nous verrons comment XMG sera utilisé pour développer notre grammaire. Nous présentons XMG dans la première section de manière générale avant d'en détailler dans les sections suivantes les aspects les plus particuliers.

3.1 Principes généraux

XMG est un formalisme et un outil [Cra05]. Le formalisme permet de décrire de manière concise le contenu d'une grammaire lexicalisée en expliquant la construction des items syntaxiques finaux à partir de fragment initiaux, en accordant un soin tout particulier à l'établissement des alternances, et plus généralement des constructions alternatives, c'est-à-dire des équivalences qui peuvent exister entre fragments. Ce dernier point est l'apport crucial de XMG. La conséquence importante de ces deux concepts que sont la réutilisation et l'alternative est qu'il n'y a plus de notion de croisements. En ce sens, notre approche rompt donc avec ce qui semblait être la tradition dans les métagrammaires. De la même façon, la notion d'héritage est un cas particulier de la relation plus générale de réutilisation. XMG permet de définir des grammaires avec information syntaxique mais aussi sémantique, grâce à la notion de dimension. Surtout, il est extensible, dans le sens où tout est prévu pour pouvoir ajouter d'autres types d'information, c'est-à-dire d'autres dimensions.

L'outil lit une telle description et produit une grammaire cible lexicalisée, c'est à dire une collection de structures syntactico-sémantiques. Dans la pratique ces structures sont non-ancrés, mais rien n'interdit de produire des structures ancrées.

3.1.1 Relation avec la programmation logique

Même si la notion de réutilisation et d'alternative aurait pu être exprimée de diverses manières, nous avons choisi d'utiliser le paradigme de la programmation logique qui nous semblait être en adéquation avec la problématique de l'organisation d'une grammaire.

Cette description concise, que l'on appelle métagrammaire, est à mettre en relation avec un programme logique. Il existe une analogie entre les deux. Au niveau de l'écriture de cette description, la syntaxe concrète est influencée par celle des langages de programmation logique. Il n'y a pas d'affectation, uniquement de l'unification.

Dans un programme logique l'unité de base est la clause et on retrouve cette notion dans la métagrammaire sous la forme de fragment nommé et réutilisable, ou classe dans la terminologie de XMG. Les clauses peuvent énoncer un axiome ou être construites à partir d'autres clauses tandis que dans la métagrammaire on aura des fragments indépendants ou construits à partir d'autres fragments. Ces constructions sont dans les deux cas similaires. On peut indiquer grâce à la conjonction et à la disjonction de clauses/fragments deux types de constructions différentes :

la conjonction : elle permet de dire que l'on réutilise tous les fragments opérandes de la conjonction pour construire le fragment résultat.

la disjonction : elle indique que l'un des fragments opérandes doit être réutilisé pour obtenir le fragment résultat.

Enfin, certaines classes sont annotées pour indiquer qu'elles représentent une structure linguistique complète. C'est l'équivalent des requêtes que l'on peut formuler dans un progamme logique et qui déclenchent une évaluation.

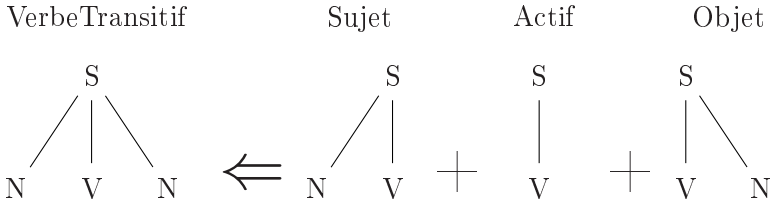
On le voit, dans XMG, le nombre de primitives pour la combinaison de fragments est limité : il n'en existe que deux. La métagrammaire peut donc se représenter graphi-

quement comme un graphe *et/ou*, ou de manière plus compacte comme une grammaire hors-contexte. Les structures complètes correspondent à la réunion des fragments contenus dans les classes rencontrées lors d'un parcours dans un tel graphe ou de manière analogue à une dérivation dans la grammaire correspondante. Le squelette de la métagrammaire peut donc être vu comme une grammaire hors-contexte dont les terminaux ne sont pas des mots mais des fragments de structures. Dans XMG, c'est plus particulièrement l'analogie avec les grammaires de clauses définies (DCG) (équivalentes aux CFG dans la version simple où les clauses n'ont pas d'argument) qui a été retenue pour sa notion d'accumulateur et pour la place prépondérante de l'opération d'unification. À chaque fois qu'une classe est évaluée, on accumule le fragment qu'elle contient et on réalise les unifications indiquées. À la fin de l'évaluation, on dispose d'un accumulateur contenant tous les fragments des classes rencontrées lors d'une évaluation, dont certaines parties ont été unifiées.

Cette façon de voir les métagrammaire, cette interprétation, nous permet de rester dans un cadre formel connu avec des algorithmes de traitement efficaces.

Combinaisons de classes

Par exemple, considérons l'arbre syntaxique associé à une entrée lexicale telle que *voit* (*i.e.* un verbe transitif). Disposant des fragments d'arbres *Sujet*, *Actif* et *Objet*, nous pouvons réécrire l'arbre $VerbeTransitif$ comme la conjonction de ces 3 fragments :



Ce qui s'écrit également comme suit[11] :

$$VerbeTransitif \quad \rightarrow \quad Sujet \wedge Actif \wedge Objet \tag{3.1}$$

Nous nous ramenons ainsi au formalisme des *grammaires de clauses définies* (DCG), dans lequel les terminaux ne seraient pas des mots mais des fragments d'arbres. La conjonction permet donc d'exprimer la construction par réutilisation de fragments. Ces fragments sont nommées. Ce nom permet de réutiliser cet ensemble, d'y référer de manière univoque.

Les DCG fournissent également une autre opération, la disjonction. Elle va nous permettre d'établir les alternances, et plus généralement de nommer des ensembles de constructions alternatives. Plus précisément, la disjonction permet de créer des *classes d'équivalences* que l'on peut nommer et donc réutiliser. Elles servent à expliciter les alternances. Ainsi, nous pouvons préciser l'exemple précédent de l'arbre associé aux verbes transitifs en spécifiant que le sujet peut être sous forme canonique *ou* sous forme relative. Cela s'énonce via la règle suivante :

$$Sujet \quad \rightarrow \quad SujetCan \vee SujetRel \tag{3.2}$$

[11] Attention, dans le cas d'une utilisation réelle avec XMG, il faudrait indiquer que les nœuds S et les nœuds V doivent s'unifier.

En réalité, la conjonction et la disjonction sont des coquilles vides : rien n'a encore été dit au sujet la sémantique de la «réutilisation». Elles permettent uniquement de décrire l'organisation des classes entre elles, en terme de réutilisation et d'alternance.

La conjonction va être selon le contexte instanciée vers une certaine opération logico-algébrique (nous parlerons de la notion de dimension plus-bas) comme par exemple l'unification ou encore la conjonction d'une certaine logique. Il n'y a pas de mécanisme de croisement implicite – en tout cas, dans le langage noyau. Le grammairien doit spécifier tous les croisements[12], ce qui lui donne un grand contrôle sur la grammaire produite, permet d'éviter de générer des structures non-voulues (problème de la surgénération) mais peut s'avérer fastidieux. La section 3.1.3 présente une solution à ce problème pour rendre implicite une partie des croisements ou des unifications.

Gestion des identifiants

Un soin tout particulier a été pris pour assurer la facilité du développement de grammaires importantes. Un des problèmes évoqués au chapitre précédent est celui de la collision de noms qui est inévitable dans les formalismes où les noms sont globaux. Dans XMG, nous avons choisi une approche radicalement différente : les noms n'ont de portée qu'au sein d'une classe.

Dans ce cas, les collisions ne peuvent donc se produire qu'entre identifiants d'une même classe, ce qui n'arrive pas en pratique. Cependant, cette solution est trop restrictive. L'utilisateur a souvent besoin quand il réutilise des classes (par conjonction ou disjonction) de faire explicitement référence à des nœuds qui sont désignés par des variables pour leur ajouter de l'information, les unifier, ou pour toute autre opération. Il convient donc de pouvoir accéder à l'intérieur d'une classe aux nœuds définis dans une autre classe.

Nous avons mis en place un mécanisme d'import ou d'export de noms entre classes. Une classe déclare donc explicitement quelles sont les variables qui seront accessibles aux classes qui la réutiliseront. Quand une classe A réutilise une classe B, elle peut accéder aux variables exportées par B, par l'intermédiaire d'une notation pointée. Elle accède uniquement à ces variables exportées. Les conflits de noms sont donc rares dans XMG.

Héritage

D'autres opérations peuvent être construites à partir des deux primitives. Par exemple, la notion d'héritage entre fragments utilise la conjonction en lui ajoutant un import d'identifiants qui est une opération que nous définirons plus bas.

Intuitivement, si la classe B définit un fragment b et hérite de la classe A qui, elle, définit un fragment a alors le contenu de B est la conjonction des fragments a et b et l'espace de nommage de A est fusionné à celui de B, c'est à dire que les variables locales de la classe A sont ajoutées aux variables locales de B. L'héritage permet de partager des variables entre plusieurs classes.

XMG autorise l'héritage multiple. Une classe peut importer le contenu et l'espace de nommage de plusieurs autres classes. Si plusieurs classes exportent les mêmes noms, le

[12]Nous appelons ici croisement tout emploi de la conjonction ou de la disjonction.

système devrait échouer. Pour des raisons pratiques[13], dans l'implantation seule la dernière classe importée exportera réellement ce nom et un message d'avertissement signale le problème.

Évaluation

Enfin, dans un programme logique, une exécution correspond à une requête. De manière analogue, on peut demander à évaluer un ou plusieurs fragments de la métagrammaire. Cette évaluation va déclencher tous les croisements indiqués par les conjonctions et disjonctions, à partir du fragment soumis à évaluation. Comme notre métagrammaire est un programme logique, ceci revient à retrouver toutes les dérivations de la classe évaluée dans une DCG.

Dans notre exemple si l'on demande l'évaluation de la classe associée aux verbes transitifs (dans notre outil `value VerbeTransitif`), on va expliciter la classe en remplaçant chaque nom par son contenu. Ici, on va remplacer l'appel à *Sujet* par la disjonction associée. Ceci nous donnera :

$$\mathit{VerbeTransitif} \quad \rightarrow \quad (\mathit{SujetCan} \vee \mathit{SujetRel}) \wedge \mathit{Actif} \wedge \mathit{Objet} \tag{3.3}$$

On calcule la forme normale disjonctive de la formule associée à *VerbeTransitif* et pour chaque membre de la disjonction on utilise la sémantique associée à la conjonction pour assembler les fragments.

$$\mathit{VerbeTransitif} \quad \rightarrow \quad (\mathit{SujetCan} \wedge \mathit{Actif} \wedge \mathit{Objet}) \vee (\mathit{SujetRel} \wedge \mathit{Actif} \wedge \mathit{Objet}) \tag{3.4}$$

$$\mathit{VerbeTransitif1} \quad \rightarrow \quad \mathit{SujetCan} \wedge \mathit{Actif} \wedge \mathit{Objet} \tag{3.5}$$

$$\mathit{VerbeTransitif2} \quad \rightarrow \quad \mathit{SujetRel} \wedge \mathit{Actif} \wedge \mathit{Objet} \tag{3.6}$$

XMG va ensuite pour chacune des formules inférées énumérer les modèles. Un modèle est un n-uplet $(D_1, \ldots, D_n)$ où D_i est la restriction d'un modèle pour un type d'information donné, ce que nous appelons une dimension. En pratique, XMG permet d'écrire des classes à trois dimensions. Les modèles sont donc des triplets (D_1, D_2, D_3) où D_1 est une description d'arbres, D_2 est une description de prédicats, et D_3 est une structure de traits.

En pratique, on ne calcule pas la forme normale disjonctive, on utilise une méthode plus efficace. La recherche de modèles s'effectue selon la méthode de résolution qui fonde la programmation logique, définie par [Rob65]. En d'autres termes, on ne calcule pas explicitement la forme normale disjonctive mais on utilise le mécanisme de retour arrière (*backtracking*) pour factoriser les parties communes.

Le but de cette section était de présenter l'organisation des fragments dans notre formalisme et de montrer qu'il y a bien une interprétation dynamique de ces classes. Le comportement de XMG est donc très similaire à un interprète PROLOG.

[13] Nous avons considéré que le compilateur ne devait pas imposer une politique de nommage aux utilisateurs. Ce choix est discutable, il est vrai.

3.1.2 Types d'informations et dimensions

Une grammaire permet d'exprimerdifférents types d'informations (morphologique, syntaxique, sémantique, etc) sur la langue. XMG est conçu pour permettre au linguiste de donner ces différents types dans la même métagrammaire. Ces différents types d'informations sont compartimentés dans des *dimensions*. À chaque dimension correspond un type d'information. Chaque dimension va définir son propre langage de description et une opération d'accumulation, c'est à dire qu'elle va donner une sémantique à l'opération de réutilisation, la conjonction. L'accumulation est l'opération d'assemblage de fragments.

Bien qu'il soit théoriquement possible d'ajouter un nombre de dimensions arbitraire, l'outil logiciel XMG est en standard muni de trois dimensions :

- `<syn>`, la dimension syntaxique qui permet de manipuler des fragments de descriptions d'arbres. L'opération d'accumulation est la juxtaposition de ces fragments (plus formellement, la conjonction dans la logique de description utilisée) ;
- `<sem>`, la dimension sémantique constituée de prédicats. L'accumulation est la conjonction logique ;
- `<dyn>`, la dimension interface, qui permettra d'ancrer les productions[14]. L'accumulation est l'unification de structures de traits. Les structures résultats sont des matrices de traits de type *hypertags* [Kin00].

Chaque dimension dispose de son propre accumulateur. Lors des combinaisons de fragments provenant d'une requête, les différentes dimensions sont traitées en parallèle. Les dimensions ne sont pas indépendantes puisque des identifiants peuvent être partagées, pour réaliser l'ancrage et l'interface syntaxe-sémantique (voir sur ce sujet [GP07]).

3.1.3 Post-traitement de dimension

Pour certains formalismes, comme les TAG (simples ou multi-composants), la dimension `<syn>` est trop permissive puisqu'elle manipule des descriptions d'arbres et non pas des arbres (ou des ensembles d'arbres). À la fin d'une évaluation, la première projection D_1 du résultat de cette évaluation (D_1, D_2, D_3) est une description d'arbres.

D'un autre côté, travailler uniquement avec des arbres ne permettrait pas d'arriver au niveau de sous-spécification souhaité et rendrait fastidieuse l'écriture de la métagrammaire. De plus, les arbres TAG sont formellement des quasi-arbres, c'est à dire des arbres où les nœuds qui peuvent être site d'adjonction sont en réalité deux nœuds en relation de dominance. Le fait d'utiliser des descriptions d'arbres pour décrire les arbres TAG est donc une façon tout à fait naturelle et popularisée par [VS92]. Cependant, d'une part les analyseurs dans leur vaste majorité ne travaille qu'avec des arbres et d'autre part les quasi-arbres sont un cas très particulier de descriptions d'arbres.

Une approche pragmatique a donc été choisie pour les TAG : XMG est muni d'un solveur de contraintes qui à partir de l'accumulateur de la dimension `<syn>` contenant

[14] Cette dimension peut très bien être utilisée comme dimension morphologique. Cependant, dans la plateforme logicielle développée pour les GI nous utilisons un lexique morphologique qui se veut indépendant du formalisme comme de la grammaire. C'est pourquoi un outil supplémentaire va croiser la grammaire produite par XMG qui contient des DAP à l'information morphologique incomplète avec le lexique morphologique pour créer la grammaire complète.

une collections de description d'arbre calcule tous les modèles minimaux correspondant à chaque description, c'est à dire des arbres.

Si l'on n'utilise que les primitives de conjonction et de disjonction de classes que nous avons présentées, le nombre de ces modèles (nous parlons ici des arbres TAG) peut être prodigieusement grand, et certains n'ont pas de réel sens linguistique. De manière à contraindre le nombre de solutions, nous pouvons accéder directement dans la métagrammaire aux nœuds des fragments et obliger certaines unifications de nœuds manuellement. D'autre part, un mécanisme de gestion des besoins et ressources a été mis en place. Ce mécanisme est d'une granularité particulièrement fine comparé aux approches précédentes puisque ces besoins/ressources ne sont pas au niveau de la classe mais au niveau des nœuds eux-mêmes. Le solveur est capable de prendre en compte des besoins/ressources pour éliminer un grand nombre de modèles. D'autres mécanismes de croisements automatiques ont également été mis en place.

Pour d'autres formalismes, en particulier les GI, les descriptions d'arbres sont les structures manipulées par l'analyseur. On n'a donc pas besoin de chercher les modèles[15]. On peut noter toutefois que pour les GI la tâche de recherche de modèle est reportée à la phase d'analyse[16].

On peut donc distinguer deux méthodologies dans l'utilisation de XMG :

1. Utilisation de l'identification de nœuds à chaque conjonction de classes, de manière à contrôler au maximum les structures produites. Il n'y a pas besoin dans ce cas de recourir à un post-traitement. C'est la méthode qui a été utilisée pour le développement de GI.
2. Conjonction de classes sans identification de nœuds. XMG génère alors toutes les structures acceptables. Il faut donc un post-traitement important et des contraintes suffisamment nombreuses pour limiter les structures produites aux structures pertinentes linguistiquement. C'est la méthode qui a été utilisée pour le développement de TAG.

3.2 Langage noyau

Nous définissons dans cette section la syntaxe abstraite de l'outil XMG, celle du langage noyau. La syntaxe concrète sera donnée par les exemples au chapitre suivant.

3.2.1 Combinaison de fragments

Nous avons vu à la section précédente que la compilation d'une métagrammaire correspondait à la combinaison de fragments. Ces fragments sont organisées en classes qui sont des entités nommées porteuses d'une information linguistique.

Dans XMG, la métagrammaire est décrite au moyen d'un langage de représentation, qui définit les opérations suivantes :

[15] En pratique, XMG post-traite également les GI pour vérifier que les structures générées ne sont pas des graphes quelconques mais ont bien une forme d'arbre.

[16] Ce qui explique en partie la différence de complexité des problèmes d'analyse : polynomiale pour les TAG et NP-complète pour les GI.

- la *définition* grâce à laquelle on ajoute une nouvelle classe. Cette opération associe un nom et un contenu linguistique ;
- la déclaration d'informations linguistiques (cf. section suivante) ;
- la réutilisation de classes existantes (du contenu associé à un nom). C'est notre équivalent de l'appel de fonction[17] ;
- l'unification de variables ;
- la *composition conjonctive ou disjonctive* . Ce sont les deux primitives de construction et d'équivalence que nous avons déjà présentées.

On a donc la grammaire suivante :

$$\begin{aligned} \mathit{Classe} &::= \mathit{Nom} \rightarrow \mathit{But} \\ \mathit{But} &::= \mathit{Description} \mid \mathit{Nom} \mid x = y \mid \\ &\quad\ \mathit{But} \vee \mathit{But} \mid \mathit{But} \wedge \mathit{But} \end{aligned}$$

On remarque que XMG fournit un langage incluant la disjonction et de ce fait introduit de l'indéterminisme dans la combinaison des fragments d'arbres.

3.2.2 Contenu des dimensions

Ce contenu peut être syntaxique, sémantique ou permettre l'interface avec le lexique. Chaque type d'information définit une dimension.

La dimension syntaxique

Les fragments dénotés par les classes de notre langage de représentation sont décrits au moyen de nœuds et de contraintes de dominance et de préséance sur ces nœuds. Ces contraintes définissent des descriptions d'arbres :

$$\mathit{Description} ::= x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \mid x[f{:}E] \mid x(p{:}E) \tag{3.7}$$

où x, y représentent des variables de noeuds[18], $\rightarrow$ la parenté ou dominance immédiate, $\rightarrow^{+}$ la dominance stricte ou fermeture transitive (non-réflexive), $\rightarrow^{*}$ la dominance large ou fermeture transitive réflexive, $\prec$ la préséance immédiate, $\prec^{+}$ la préséance stricte (fermeture transitive), $\prec^{*}$ la préséance large (fermeture transitive réflexive), $x[f{:}E]$ l'association du trait f au noeud x et $x(p{:}E)$ l'association de la propriété p à ce même noeud x. Dans ces deux derniers cas E est une expression dénotant un couple composé d'une polarité et d'une valeur de trait, qui elle même peut être une structure de traits.

Les traits comme les propriétés sont regroupés en structures de traits mais les premiers donnent de l'information morpho-syntaxiques tandis que les secondes donnent de l'information propre au formalisme, les nœuds ancres en particulier mais plus généralement toutes les marques sont des propriétés.

[17] Voir la section 4.3 pour une utilisation de classe comme fonction.

[18] Puisque notre langage n'est pas typé statiquement, il convient d'expliciter qu'une variable x fait référence à un nœud. C'est possible grâce à la primitive $\mathit{node}(x)$.

Ainsi, nous disposons d'un langage suffisamment expressif pour supporter les formalismes syntaxiques basés sur les descriptions d'arbres, tels que les grammaires TAG et les GI.

La dimension sémantique

L'extensibilité de XMG se retrouve également dans le fait qu'il offre un support adéquat à l'intégration d'une représentation sémantique dans la métagrammaire. En effet, notre langage de représentation permet non seulement de placer dans les classes des informations syntaxiques (*e.g.* les descriptions d'arbres que nous avons vues précédemment), mais également des représentations sémantiques. A l'heure actuelle, un langage à base de structures prédicatives a été implanté, pour utiliser la sémantique à trous (*hole semantics*) de [Bos96] :

$$Description \ ::= \ \ell{:}p(E_1, ..., E_n) \mid \leftarrow \ell{:}p(E_1, ..., E_n) \mid \ E_i \ll E_j \tag{3.8}$$

Ici, $\ell{:}p(E_1, ..., E_n)$ représente le prédicat p avec les arguments $E_1, .., E_n$, et étiqueté par ℓ, $\leftarrow$ l'opérateur de négation, et $E_i \ll E_j$ indique que E_j est dans la portée de E_i (à la manière de la MRS [CFPS06]).

Un tel langage peut servir, par exemple, à associer aux arbres des informations de type sémantique plate comme dans [GK03].

En reprenant notre comparaison entre métagrammaire et DCG se pose la question de la distinction entre information syntaxique et information sémantique. Cela est rendu possible par l'utilisation de *dimensions*, correspondant chacune à un *accumulateur* spécifique dans le formalisme des *grammaires de clauses définies étendues* (EDCG) [Van90]. Les dimensions sont donc traitées en parallèle dans les différents accumulateurs.

Ainsi, la combinaison de classes (*i.e.* d'informations syntaxiques ou sémantiques) donne lieu à l'accumulation de leur contenu dans des structures dédiées. L'expression (3.2.1) vue précédemment est alors étendue en remplaçant *Description* par :

$$Dimension \mathrel{+}= Description$$

L'opération += se nomme l'accumulation. Le contenu informationnel doit être placé dans l'accumulateur correspondant.

Ce *typage* de l'accumulation permet de traiter plusieurs dimensions. A l'heure actuelle, XMG dispose de 3 dimensions, notées **syn**, **sem** et **dyn**. Les deux premières représentent les dimensions syntaxique et sémantique et la dernière une dimension utilisée pour l'accumulation d'informations lexicales. Les structures que l'on y manipule sont des structures de traits et l'opération de conjonction est définie comme l'unification de ces structures de traits.

Notons que ces dimensions peuvent partager des variables logiques, ce qui permet au niveau métagrammaticale un développement relativement naturel d'une interface entre syntaxe et sémantique (voir [GK03]) ainsi que d'une interface avec le lexique.

Portée des variables

Dans les approches antérieures les noms de variables ont une portée globale à la métagrammaire. Cela pose deux types de problèmes :

1. conflits de noms, passé un certain nombre de classes.
2. l'impossibilité de réutiliser deux fois la même classe.

Nous avons donc intégré un procédé d'export d'identifiants. Ainsi, il est possible de donner avec précision le domaine de visibilité d'une variable. Plus précisément, à chaque classe est associée une structure de traits contenant les identifiants exportés qui forment évidemment un sous-ensemble des identifiants utilisés[19] dans cette classe. La définition (3.2.1) devient :

$$\mathit{Classe} \quad ::= \quad \langle f_1{:}E_1, \ldots, f_n{:}E_n \rangle \Leftarrow \mathit{Nom} \quad \rightarrow \quad \mathit{But} \tag{3.9}$$

Une classe est maintenant définie par son nom, son contenu mais aussi par ces identifiants exportés.

Cette construction indique que le nom f_i fait référence à l'expression E_i (qui est un identifiant). Elle permet donc de renommer les identifiants exportés. C'est extrêmement utile, notamment dans les cas d'imports multiples d'une même classes.

Et l'appel de classe s'accompagne de l'accès à la structure d'export (notée *Var* ici), (3.2.1) est remplacée par :

$$\mathit{But} ::= \mathit{Dimension} \mathrel{+}= \mathit{Description} \mid \mathit{Var} \Leftarrow \mathit{Nom} \mid \mathit{But} \vee \mathit{But} \mid \mathit{But} \wedge \mathit{But} \mid x = y \tag{3.10}$$

On peut accéder alors à un identifiant X via la notation pointée *Var.X*.

XMG fournit donc des primitives de manipulations d'espace de nommage assez limitées. C'est à l'utilisateur de propager petit à petit, de classe en classe, la portée des variables. Bien sûr, l'opération d'héritage, qui ne fait pas partie du langage noyau (c'est un sucre syntaxique) permet de s'affranchir de certaines lourdeurs. Elle combine conjonction (réutilisation) et l'import des identifiants exportés en une instruction.

3.3 Une architecture modulaire inspirée de la programmation logique

En section 3.1, nous avons présenté les caractéristiques de XMG, dont le fait qu'il supporte plusieurs formalismes syntaxiques. Cette caractéristique est fortement liée à l'architecture modulaire du compilateur.

Nous avons également insisté sur l'analogie entre une métagrammaire et un programme logique. C'est en réalité plus qu'une analogie, puisque nous traitons la métagrammaire comme un programme logique. Elle est compilée puis exécutée sur une machine virtuelle.

3.3.1 Des modules dédiés

La compilation d'une métagrammaire se fait en plusieurs phases, dont certaines diffèrent suivant le formalisme. Chacune de ces phases est prise en charge par un module spécifique.

Actuellement XMG comporte 3 modules principaux :

[19] C'est à dire les nouveaux identifiants ainsi que les identifiants eux-mêmes importés.

a) La partie avant (*i.e.* le compilateur proprement dit) traduit la métagrammaire en instructions exprimées dans un langage de plus bas niveau.

b) Ces instructions sont ensuite exécutées par une machine virtuelle (MV) de type *Warren's Abstract Machine* (WAM, voir [War83; AK91]).

Cette MV réalise l'unification des structures de données de la métagrammaire (*i.e.* structures de traits associées aux noeuds, traits polarisés pour les GI, etc), puis l'accumulation des dimensions pour une combinaison de classes donnée. En sortie de la MV, nous disposons de données accumulées dans chaque dimension, dans le cas des TAG, des descriptions d'arbres dont il faut calculer les solutions.

c) En plus de la partie avant et de la MV, qui sont communes aux formalismes des TAG et des GI, XMG intègre un module de résolution de descriptions d'arbres. Ce module est programmé sous forme d'un résolveur de contraintes (voir [DN00]).

3.3.2 Extensibilité

XMG a dès le commencement été pensé pour :
- pouvoir inclure différents types d'informations linguistiques,
- pouvoir générer différents types de grammaires,
- et que ces extensions puissent être facilement incorporées.

C'est cette *propriété* que nous appelons extensibilité. Pour pouvoir traiter un nouveau type d'information, il faut ajouter une nouvelle dimension, c'est à dire ajouter deux choses. Premièrement il faut incorporer un langage de description pour cette dimension et l'analyseur correspondant dans la partie avant de XMG. Ensuite, il faut ajouter les instructions de la machine virtuelle qui permettent de traiter et de combiner ces informations. En particulier, il faut définir la sémantique de la conjonction pour la nouvelle dimension.

Éventuellement, on peut choisir, comme nous l'avons fait, que l'évaluation de la machine virtuelle ne produise qu'une structure sous-spécifiée. Dans ce cas, il peut être nécessaire d'ajouter un solveur spécifique pour obtenir les structures complètement spécifiées.

Dans l'outil, l'utilisateur ne peut pas encore définir ses propres dimensions. Il faut programmer un module spécifique pour les nouvelles dimensions.

3.4 Compilation

Cette première phase, très classique, est elle même divisée en deux étapes. Premièrement, l'analyseur lit la métagrammaire et la transforme en une fermeture [Lan64], c'est à dire un arbre abstrait de syntaxe (*abstract syntax tree* ou AST) et un environnement, c'est à dire une table reliant les variables à leurs valeurs. Par exemple, la figure 3.1 représente une classe et son AST.

Quelques vérifications sont effectuées pour s'assurer que les identifiants manipulées dans une classes sont bien déclarées localement ou qu'ils proviennent d'un export valide. Il s'agit aussi de remplacer l'héritage par les appels de classes correspondant. C'est l'étape d'explicitation. Par exemple, dans la figure 3.2, l'héritage et l'appel aux variables importées est explicité. Ici, `R1` désigne une nouvelle variable stockant la structure d'export de `B`.

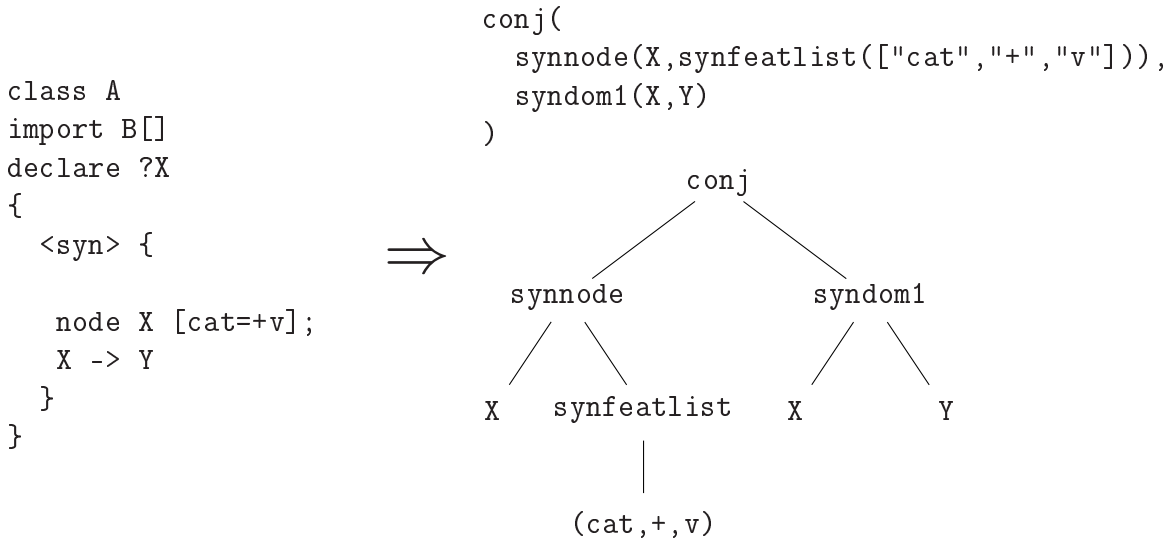


FIG. 3.1 – AST correspondant

Dans cet exemple, nous avons gardé les noms de variables tels quels, mais en réalité nous passons par la notion d'environnement pour traiter uniformément les accès mémoire.

```
class A                          class A
import B[]
declare ?X                       declare ?X
{                                {
  <syn> {                          <syn> {
                                     R1 = B[];
   node X [cat=+v];                  node X [cat=+v];
   X -> Y                            X -> R1.Y
  }                                  }
}                                }
```

FIG. 3.2 – Explicitation des classes

L'AST précédent est enrichie par l'appel à la classe `B` et surtout un environnement pour avoir une fermeture complète,représentée en figure 3.3. Chaque classe a donc son propre environnement qui fait correspondre à un entier soit une valeur soit une adresse dans un autre environnement, notamment pour faire référence à une variable déclarée dans une classe importée.

Ensuite cet AST est parcouru pour générer une suite d'instructions de bas-niveau, comparables à des instructions pour une machine virtuelle de programmation logique.

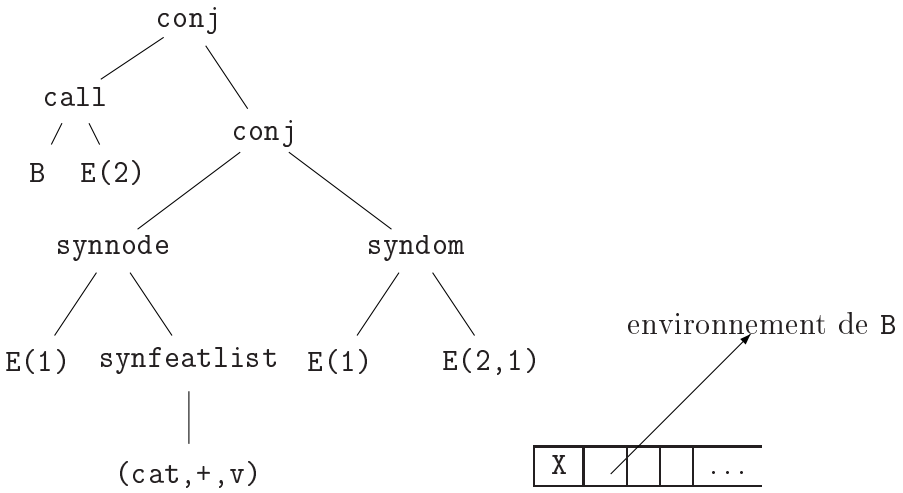


FIG. 3.3 – Fermeture associée à la classe `A`

3.5 Exécution

Les instructions bas-niveau en question sont directement inspirées du jeu d'instructions de la machine abstraite de Warren [War83; AK91] qui est la manière standard d'implanter un langage de programmation en logique [CKC83].

C'est donc naturellement que notre machine virtuelle implante cette machine abstraite. Toutefois, pour minimiser l'espace mémoire utilisé qui est assez conséquent au vu des structures de données manipulées, nous avons choisi le partage de structure pour implanter l'opération d'unification [BM72]. C'est à dire que lors de l'unification de deux termes t_1 et t_2 au lieu de compléter à la fois t_1 et t_2 pour qu'ils incorporent l'information nouvelle apportée par l'autre terme, nous n'en modifions qu'un seul et indiquons que la valeur du second est contenue à l'adresse du premier. L'avantage est évident, on construit beaucoup moins de termes et l'on gagne donc en espace mémoire. Par contre, l'accès à la valeur d'un terme ne se fait plus en temps constant, il est parfois nécessaire de suivre une chaîne de pointeurs, aussi appelée chaîne de déréférencement. On a donc un gain en espace mais une perte en temps.

3.6 Un résolveur extensible

Après les les deux premières parties (compilation et exécution) du traitement, la MV génère une description $(D_1, \ldots, D_n)$ (en supposant n dimensions). Pour chaque dimension i, un résolveur spécialisé S_i cherche les solutions (les modèles) $S_i(D_i)$ de la description D_i correspondante. La sortie finale après résolution des contraintes est un ensemble de structures de la forme $\{(M_1, \ldots, M_n) | M_i \in S_i(D_i), 1 \leq i \leq n\}$. Selon les dimensions et le formalisme cible les résolveurs sont plus ou moins complexes. Pour la dimension `<sem>` le résolveur est trivial et retourne la description elle même. Pour la dimension `<syn>` le résolveur dépend du formalisme cible : pour les TAG nous utilisons un résolveur de domi-

nance qui cherche tous les arbres modèles d'une description. Il s'inspire de l'approche par contraintes sur des ensembles développée par [DN00]. Pour les GI, il se contente de faire quelques vérifications sur l'accumulateur, puisque ce formalisme manipule directement des descriptions d'arbres. Il vérifie entre autres qu'il n'y a qu'une racine et que les nœuds en relation de préséance ont le même père.

Dans la suite de ce chapitre nous allons détailler le solveur utilisé pour produire des TAG car dans ce cas l'énumération des modèles utilise des techniques assez complexes et relativement originales.

3.6.1 Principe du résolveur d'arbres

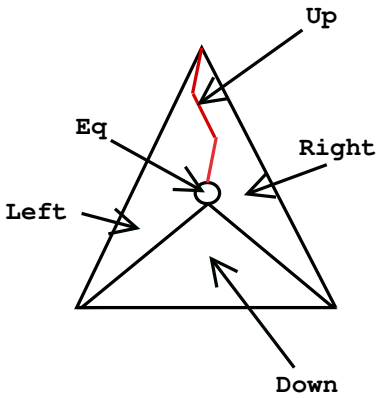


FIG. 3.4 – Positionnement d'un nœud d'un modèle

Si l'on prend un nœud x quelconque d'un arbre solution (un modèle), on peut partitionner l'ensemble des autres nœuds de cet arbre en cinq régions *disjointes* relatives à x : les nœuds identifiés à x, les nœuds au dessus de x (entre la racine et x), les nœuds en dessous, les nœuds à gauche et les nœuds à droite (cf. figure 3.4). On introduit alors pour chaque nœud x cinq ensembles de variables Eq_x, Up_x, $Down_x$, $Left_x$ et $Right_x$ pour coder les ensembles de variables interprétées par des nœuds du modèle. Les relations données par la description nous permettent de poser des contraintes d'inclusion et de disjonction entre ces ensembles. Par exemple si dans la description l'on a la relation $x \rightarrow y$, elle va se traduire par les contraintes suivantes :

- les deux nœuds sont différents, $Eq_x \cap Eq_y = \emptyset$
- les nœuds sous y sont également sous x, $Down_y \subset Down_x$
- les nœuds au dessus de x sont également au dessus de y, $Up_x \subset Up_y$
- les nœuds sur les côtés de x sont inclus dans les nœuds sur les côtés de y, $Left_x \subset Left_y$.

On ajoute ensuite certaines contraintes de bonne formation d'arbre comme le fait qu'il ne puisse exister qu'une seule racine, à choisir uniquement parmi les racines de fragments présents dans la description.

Une fois que toutes les relations de la description d'arbre ont été transformées en contraintes sur les ensembles de variables, on énumère toutes les paires de nœuds (x, y) et que l'on met en relation arbitrairement parmi $x \rightarrow y, x \prec y, y \rightarrow x, y \prec x, x \rightarrow^+ y, x \prec^+$

$y, y \rightarrow^+ x, y \prec^+ x, x = y, x||y$. Cette dernière relation signifie qu'aucune des autres n'est valide. On pose ensuite les contraintes correspondantes. On n'utilise pas les relations $\rightarrow^*$ et $\prec^*$, car elles sont moins précises que les relations que nous utilisons déjà.

Si l'on arrive à mettre tous les nœuds en relation sans incohérence dans les équations d'inclusion et de disjonction d'ensembles alors la structure est un modèle valide. Évidemment, pour une description d'arbre il peut exister plusieurs modèles[20].

3.6.2 Contraintes additionnelles

L'extensibilité de XMG est encore étendue par la programmation de modules additionnels optionnels et paramétrables. Ces modules permettent d'étendre les fonctionnalités du résolveur de descriptions d'arbres pour contraindre les modèles produits par XMG selon des critères spécifiques, appelés aussi *principes* [LCP06]. Les principes instanciables présents dans XMG actuellement sont de 3 types : la sensibilité aux resssources, la validité par rapport à un formalisme et les contraintes linguistiques.

Sensibilité aux ressources

Deux modules permettent de d'ajouter aux arbres la sensibilité aux ressources. Dans le premier cas, les couleurs, c'est une gestion des besoins et des ressources tandis que dans le second cas, c'est l'unicité d'une ressource dans chaque modèle qui est mise en avant.

Besoins et ressources C'est le principe de **couleurs**. Comme annoncé à la section 3.1.3, dans un contexte de développement de grammaires à large couverture par combinaison de fragments, une idée assez courante, nous l'avons vu au chapitre précédent, consiste à contraindre les combinaisons acceptables et à les automatiser en intégrant un système de gestion de ressources. Pour gérer ces ressources, XMG intègre un langage de couleurs. Ce langage permet d'indiquer des informations de saturation des nœuds qui donneront un modèle valide.

Plus précisément chaque nœud de la description peut être muni d'une couleur parmi le rouge, le noir et le blanc. Un nœud rouge ne peut s'identifier à aucun nœud durant la recherche de modèle, un nœud noir ne peut s'identifier qu'à des nœuds blancs et les nœuds blancs doivent s'identifier à un nœud noir.

Donc, dans un modèle valide, il n'y a que des nœuds noirs et des nœuds rouges. Plus précisément il y a dans chaque ensemble *Eq* exactement un nœud rouge ou un nœud noir et plusieurs nœuds blancs. Il y a donc une bijection entre les nœuds rouges et noirs de la description et les nœuds du modèle.

Les nœuds blancs sont absorbés par les nœuds noirs. Notre module ajoute donc au résolveur de description des variables RB_x représentant le nœud rouge (*red*) ou noir (*black*) auquel le nœud x s'identifie, en plus des 5 ensembles associés à x. Les ensembles de nœuds respectivement noirs, rouges et blancs de la description sont notés V_B, V_R et V_W. Les règles d'identification des couleurs donnent les contraintes suivantes :

[20] On peut imaginer le cas extrême d'un ensemble de nœuds sans relation. Tous les arbres construits à partir de cet ensemble de nœuds sont des modèles minimaux valides.

- $x \in V_R \Rightarrow RB_x = x \wedge Eq_x = \{x\}$, si x est rouge, il est égal au nœud rouge ou noir qui lui correspond et aucun autre nœud ne peut lui être égal,
- $x \in V_B \Rightarrow RB_x = x$, si x est noir alors il est égale au nœud noir ou rouge qui lui correspond dans le modèle,
- $x \in V_W \Rightarrow RB_x \in V_B$, si x est blanc alors on sait simplement que le nœud noir ou rouge qui lui correspond dans le modèle est un nœud noir,
- si x et y sont deux nœuds distincts (s'ils sont en relation par $\rightarrow, \rightarrow^+, \prec, \prec^+, ||$) alors $RB_x \neq RB_y$

Une présentation de l'emploi d'un langage de couleurs pour produire une grammaire TAG est donnée dans [CD04].

Unicité des ressources C'est le principe d'**unicité** paramétré par une propriété de noeuds. Ce principe permet de garantir la validité des solutions produites par XMG de telle sorte qu'un nœud au plus de chaque solution soit porteur de la propriété passée en paramètre, par exemple : *dans un arbre* TAG*, il ne peut y avoir deux extractions*[21].

Ce module ajoute au résolveur une variable U. La contrainte correspondante est : si x est porteur de la propriété passée en argument alors $Eq_x = U$. Si plusieurs nœuds sont porteurs de la propriétés, ils sont donc unifiés.

Validité par rapport à un formalisme

Arbres TAG Les arbres TAG sont des arbres particuliers, notamment les arbres auxiliaires qui représentent les modificateurs. Ces derniers possèdent obligatoirement un nœud feuille distingué, appelé le nœud pied, qui porte la même catégorie syntaxique que la racine de l'arbre. L'unicité que nous avons présentée plus haut n'est pas suffisante, puisqu'en plus de l'unicité du nœud marqué comme pied de l'arbre, il faut vérifier la catégorie.

Structure des arbres un principe d'**arité** qui pose une contrainte sur le nombre de fils d'un nœud dans le modèle. Cette contrainte introduit pour chaque nœud x un ensemble $Fils_x$ qui peut contraindre son cardinal.

Contraintes linguistiques

Ordonnancement un principe de **rang** paramétré par un nombre entier, permettant de réaliser l'ordonnancement des clitiques dans les arbres TAG produits.

Si deux nœuds ont le même rang alors soit ils sont identifiés et ont le même père soit ils ont des nœuds pères différents et sont donc distincts. Si deux nœuds x_1 et x_2 ont respectivement des rangs différents r_1, r_2 et $r_1 < r_2$ alors soit ils ont des pères différents soit ils ont le même père et x_1 est à gauche de x_2, c'est à dire $Left_{x1} \subset Left_{x_2}$.

[21] Nous adoptons cette convention dans un contexte métagrammaticale, bien que certains exemples de phrases à double extraction existent (cf [Abe02]).

Nouveaux principes

Ici encore, comme pour les dimensions, rien n'empêche de programmer de nouveaux modules de contraintes.

Pour indiquer que la métagrammaire utilise un principe, le grammairien doit le préciser via la directive `use`. Par exemple s'il faut utiliser le mécanisme de besoins et ressources présenté plus haut, il indiquera `use color`.

3.6.3 Implantation

Le logiciel XMG est programmé en Oz/Mozart [MMVR95; VH04], un langage multi-paradigme construit sur un noyau de programmation par contrainte [Sch02] avec un typage dynamique.

Le compilateur qui analyse la métagrammaire et génère une suite d'instructions bas-niveau utilise un générateur d'analyseur (de type FLEX et BISON) qui produit des structures de données Oz appelé Gump [Kor96].

Ensuite, le résolveur de contraintes utilise la bibliothèque standard de Oz/Mozart qui fournit toutes les fonctions utiles à la programmation par contrainte sur les ensembles d'entiers. L'originalité de ce langage est que sa machine abstraite modélise les contraintes comme des règles de réécriture sur des espaces de calcul qui s'appliquent concurremment à la différence des langages de programmation par contrainte basés sur la programmation logique qui implantent les contraintes par des suites d'essais/retours arrières. Le premier avantage est de distinguer clairement les contraintes de l'algorithme de recherche. C'est aussi un moyen d'être plus efficace : la mise à jour des espaces de calcul est censé être plus légère que celle d'un arbre de recherche.

Finalement, XMG produit la collection de structures sous la forme d'un fichier XML. Pour les TAG , la sortie est au format TagML [BL00].

3.7 Conclusion

Nous avons présenté le formalisme et le logiciel XMG qui constitue l'un des générateurs semi-automatiques de grammaires lexicalisées les plus aboutis à l'heure actuelle. Bien que le formalisme et l'architecture du logiciel permettent théoriquement de produire tout type de grammaire lexicalisée, en pratique l'implantation n'autorise que la génération de GI, de TAG et de MC-TAG. Il reste à combler ce fossé. D'un point de vue théorique, il reste à définir plus spécifiquement le type de contraintes utiles aux linguistes de manière à mieux cerner la notion de principes. Nous aimerions également pouvoir écrire des métagrammaires modulaires réutilisables dans différents contextes à la manière dont sont écrites les grammaires de l'environnement de développement grammatical (*grammatical framework*, GF) de [Ran07]. Dans GF une grammaire se compose de plusieurs modules. L'un des modules est commun à toutes les langues et il est donc utilisé dans chaque grammaire. Les autres sont spécifiques à une langue donnée et ne servent donc que pour une seule grammaire. On peut par ailleurs importer ou étendre des modules déjà existants. Des travaux dans ce sens ont déjà été effectués avec XMG, dans [KRS+06], où des classes sont

réutilisées dans des grammaires pour des langues proches (allemand et yiddish). Toutefois, il faudrait donner un vrai statut à ces modules et à l'opération de composition, en s'inspirant peut-être des signatures sous-spécifiées de [CSW06].

Chapitre 4

Une petite métagrammaire de la coordination

Sommaire

Dans ce chapitre nous allons décrire une petite grammaire d'interaction de la coordination pour illustrer la syntaxe concrète[22]. Nous n'utiliserons pas les principes.

La méthodologie est la suivante :

1. définir les catégories, les traits que nous allons utiliser
2. écrire les fragments de bases, généralement en relation d'héritage
3. définir les alternatives
4. évaluer les classes *terminales* considérées autonomes.

Nous voulons pouvoir dans cet exemple coordonner deux syntagmes nominaux ou deux propositions. Nous ne donnerons que la partie syntaxique et ne nous intéresserons pas aux parties lexicale et sémantique.

Pour commencer, nous définissons une classe décrivant un fragment autonome (section 4.1). Puis nous verrons comment utiliser l'héritage et manipuler les espaces de nommage (section 4.2).

[22]La syntaxe décrite précédemment était la syntaxe abstraite.

4.1 Version initiale

Nous donnons une première version simpliste pour voir le processus général d'utilisation de XMG. Dans la section suivante, nous développerons une organisation de la métagrammaire plus élaborée.

4.1.1 Déclaration des traits

XMG exige de déclarer les traits utilisés dans la grammaire. XMG distingue les traits (*feature*) des propriétés (*property*). La différence entre les traits et les propriétés est que les traits contiennent de l'information morpho-syntaxique tandis que les propriétés contiennent de l'information propre au formalisme utilisé.

Ces traits et propriétés sont construits sur des types. Ici les types sont définis de façon extensionnelle, soit en donnant la liste des valeurs atomiques possibles, soit en donnant un intervalle d'entiers.

```
type MARK  = {anchor,empty,closed}
type CAT   = {adj,adv,aux,clit,coord,cpl,det,n,np,pp,prep,pro,punct,s,v}
type FUNCT = {aobj,attr,dat,deobj,loc,obj,obl,subj}
type PERS  = [1..3]
```

Le premier type définit quelques marques que l'on utilise dans les GI. Un nœud peut être une ancre, un nœud vide, c'est à dire un nœud dont aucun des descendants n'est une ancre ou un nœud clos, c'est à dire que l'arité de ce nœud est fixée.

Les deux types suivants définissent des types relatifs à la catégorie syntaxique et à la fonction grammaticale d'un syntagme.

Enfin le dernier type illustre l'utilisation d'intervalle d'entier. Ici le type `PERS` est l'ensemble des entiers compris entre 1 et 3[23].

Nous pouvons maintenant définir les traits et les propriétés à partir des types ci-dessus :

```
property mark : MARK
feature cat : CAT
feature funct : FUNCT
feature pers : PERS
```

Ici nous indiquons que nous créons une propriété que nous appelons `mark` et qui pourra prendre comme valeur une disjonction atomique d'éléments du type `MARK`. De la même façon nous définissons trois traits `mark`, `funct` et `pers`.

4.1.2 La première classe

Notre première classe définit l'allure générale de la description d'arbre associée à une conjonction de coordination.

[23] L'utilisation d'entiers pour modéliser les personnes n'est considérée ici qu'à titre d'illustration. Dans une grammaire réaliste on utilisera plutôt un type propre.

```
class SimpleCoord
declare ?Top ?LeftConj ?RightConj ?Anc
{
      <syn>{
            node Top;
            node LeftConj;
            node RightConj;
            node Anc;

            Top -> LeftConj;
            Top -> RightConj;
            Top -> Anc;

            LeftConj >> Anc;
            Anc >> RightConj
      }
}
```

Cette première étape de description est associée à un nom, `SimpleCoord`. On pourra donc réutiliser ce fragment en appelant ce nom dans une autre classe.

La classe commence par la déclaration de quatre noms de variables qui n'ont de portée que sur cette classe. Comme ils ne sont pas explicitement marqués comme exportables (nous verrons l'export dans la section suivante), ils ne seront pas jamais modifiables par une autre classe. Ces variables vont nous permettre de créer des nœuds et de placer ces nœuds les uns par rapport aux autres.

Nous ne nous intéressons qu'à la partie syntaxique dans cet exemple, donc nous ne définissons que la dimension `<syn>` de nos descriptions. Ici elle est composée de deux parties. La première associe des nœuds aux noms de variables. Cela signifie que désormais toute référence à une des variables fera référence à un nœud du fragment. La seconde partie dispose les nœuds les uns par rapport aux autres. Le nœud `Top` est la racine et `LeftConj` précède[24] `Anc` qui lui même précède `RightConj`. Pour obtenir le résultat, il suffit de conclure la métagrammaire par `value SimpleCoord`, qui déclenchera l'évaluation de cette classe.

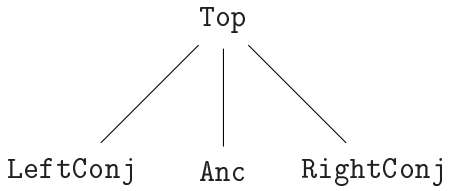


FIG. 4.1 – Le fragment représenté par `Coord`

La figure 4.1 donne l'allure de l'arbre produit par cette évaluation. Comme nous n'avons pas muni nos nœuds de structures de traits, on ne peut pas afficher cette DAP

[24] Le lecteur peut se demander pourquoi on note >> la relation de préséance $\prec$. Il n'y a malheureusement pas de bonne raison...

avec `leopar`. Le nœud `LeftConj` correspond au conjoint à gauche de la conjonction et `RightConj` au nœud à droite. Le nœud `Anc` est le nœud ancre : il portera la conjonction de coordination. Finalement le nœud `Top` est le nœud qui portera la contribution du groupe coordonné au reste de la phrase.

Ce premier exemple est très simple. Nous allons maintenant voir comment utiliser l'héritage, la conjonction et la disjonction.

4.2 Héritage et organisation des classes

Nous allons maintenant étendre notre fragment en ajoutant entre autres des traits aux nœuds. Nous pourrions reprendre notre classe précédente telle quelle et lui ajouter du contenu mais nous avons choisi de la réécrire pour illustrer le mécanisme d'héritage de XMG.

4.2.1 La superclasse

```
class CoordTopo
export ?Top ?LeftConj ?RightConj ?Anc
declare ?Top ?LeftConj ?RightConj ?Anc
{
      <syn>{
            Top -> LeftConj;
            Top -> RightConj;
            Top -> Anc;

            LeftConj >> Anc;
            Anc >> RightConj
      }
}
```

Notre nouvelle classe `CoordTopo` contient uniquement les relations entre les nœuds. Nous précisons également grâce à la directive `export` que les classes qui importeront celle-ci pourront utiliser tous les identifiants. Cette première classe sera la superclasse de notre hiérarchie d'héritage. Nous étendons cette hiérarchie avec la classe suivante.

4.2.2 Ajout de traits

```
class CoordFeatureSharing
import CoordTopo[]
declare ?Xcat
{
      <syn>{
            node Top [cat =+ Xcat](mark=closed);
            node LeftConj [cat =- Xcat];
```

```
            node RightConj [cat =- Xcat];
            node Anc [cat = coord](mark=anchor)
      }
}
```

Notre nouvelle classe `CoordFeatureSharing` réutilise le contenu et les identifiants de `CoordTopo` par l'intermédiaire de la directive `import CoordTopo[]`. Les deux crochets indiquent que la classe `CoordTopo` ne prend pas d'argument. On verra plus loin l'utilisation de classes paramétrées.

Dans cette classe, on précise que les identifiants importés doivent s'unifier avec des nœuds munis d'une structure de traits (entre crochets) et éventuellement de propriétés (entre parenthèses). Le nœud `Anc` a la catégorie `coord` et c'est le nœud qui portera l'ancre (le mot) de la description.

Les autres nœuds n'ont pas de catégorie spécifique, car `Xcat` est une variable, mais ils ont tous la même valeur pour ce trait. C'est ce qu'on appellera la coindexation dans la prochaine partie consacrée à la modélisation de la coordination dans les GI. La racine `Top` a un trait positif tandis que les nœuds `LeftConj` et `RightConj` ont un trait négatif.

Le nœud `Top` est marqué comme clos, c'est à dire que dans tout modèle le nœud interprétant `Top` aura trois fils, plus exactement le même nombre de fils que `Top` dans la classe évaluée. On peut encore dans la métagrammaire ajouter des fils supplémentaires à ce nœud.

On va maintenant distinguer les coordinations selon leur catégorie.

4.2.3 Spécialisation et disjonction

On écrit deux nouvelles classes qui héritent de `CoordFeatureSharing`.

```
class SentenceCoord
import CoordFeatureSharing[]
declare ?N
{
      <syn>{
            node N [cat = s]
      };
      Top=N
}


class NPCoord
import CoordFeatureSharing[]
declare ?Xfunct
{
      <syn>{
            node Top [cat = np,funct =- Xfunct];
            node LeftConj [funct =+ Xfunct];
```

```
            node RightConj [funct =+ Xfunct]
      }

}
```

Les deux classes précédentes spécialisent la classe `CoordFeatureSharing` en instanciant la catégorie des nœuds. On peut remarquer que l'on n'instancie que la catégorie du nœud `Top` et que cette instanciantion sera propagée par coindexation aux autres nœuds. Pour la coordination de syntagmes nominaux, on force également les conjoints à avoir la même fonction syntaxique à nouveau par coindexation.

On peut également remarquer que l'instanciation de cette catégorie se fait de deux manières différentes. Dans la classe `NPCoord` on accède directement aux nœuds importés. Dans `SentenceCoord`, on crée un nouveau nœud `N` qui n'a de portée que dans la classe et qui est ensuite unifier avec `Top`. Comme `N` n'est pas exporté, il ne sera pas visible dans les sous-classes. De l'extérieur, on ne verra pas la différence entre les deux façons de faire. En pratique, on utilise plutôt la seconde méthode dans les hiérarchies d'héritage.

On va maintenant évaluer nos classes.

```
class Coord
{
      SentenceCoord[] | NPCoord[]
}


value Coord
```

On termine par une classe nommée `Coord` qui établit qu'une coordination est soit une coordination de proposition soit une coordination de syntagmes nominaux. Enfin, on demande l'évaluation de cette dernière classe qui va produire toutes les descriptions correspondantes.

4.3 Utilisation d'une classe paramétrée

Nous allons ici donner un exemple d'utilisation des classes paramétrées de XMG pour modéliser la propagation de la personne entre les conjoints syntagmes nominaux et le groupe coordonné[25], comme dans la phrase suivante :

(4.1) Jean et toi venez.

Nous voulons établir que si au moins l'un des conjoints est à la première personne, le groupe coordonné l'est également et que sinon on applique la même règle pour la deuxième personne et qu'en dernier ressort le groupe est à la troisième personne.

On va donc utiliser la classe paramétrée suivante. Cette classe n'appartient pas à la hiérarchie d'héritage que nous avons commencé à construire. Elle est orthogonale à notre hiérarchie.

[25]Nous considérons que les pronoms sujets utilisés dans des coordinations sont des syntagmes nominaux.

```
class PersonAgreement[PCoord,PConj1,PConj2]
{
      {{{PCoord = 1;
            {{PConj1 = 1; PConj2 = @{1,2,3}}
             |
             {PConj1 = @{2,3}; PConj2 = 1}}
      }
      |
      {PCoord = 2;
            {{PConj1 = 2; PConj2 = @{2,3}}
             |
             {PConj1 = 3; PConj2 = 2}}
      }}
      |
      {PCoord = 3; PConj1 = 3; PConj2 = 3}}
}
```

La classe `PersonAgreement` est une classe à trois paramètres. Le premier représente la personne du groupe coordonné, le deuxième la personne du conjoint gauche et le troisième celle du conjoint droit. Cette classe implante la règle énoncée ci-dessus à l'aide de la conjonction notée ; et de la disjonction notée |. Cette classe unifie les 3 variables paramètres avec des entiers ou des disjonctions d'entiers, notées par `@`. Le lecteur remarquera que l'on aurait pu se contenter de dire que la personne du groupe coordonné est égale à la plus petite des personnes des conjoints, et donc utiliser une fonction sur les entiers. Mais ce type de fonction n'est pas encore implanté dans XMG, et nous devons donc passer par ce genre de détour.

```
class NPCoord_Agr
import NPCoord[]
declare ?PTop ?PLConj ?PRConj
{
      <syn>{
            node Top[pers = PTop];
            node LeftConj[pers = PLConj];
            node RightConj[pers = PRConj]
      };
      PersonAgreement[PTop,PLConj,PRConj]
}
```

Ensuite, nous avons étendu la classe `NPCoord` par la classe `NPCoord_Agr`. Dans un premier temps, les nœuds ont une personne ayant une valeur quelconque : une variable non liée est associée à chacune d'elles. Puis ces personnes sont utilisés comme paramètres de l'appel à `PersonAgreement` qui va lier les variables selon la règle énoncée. La figure 4.2 représente la grammaire produite en remplaçant `NPCoord_Agr` par `NPCoord` dans la classe `Coord`. On peut y voir les cinq descriptions pour la coordination de syntagmes nominaux.

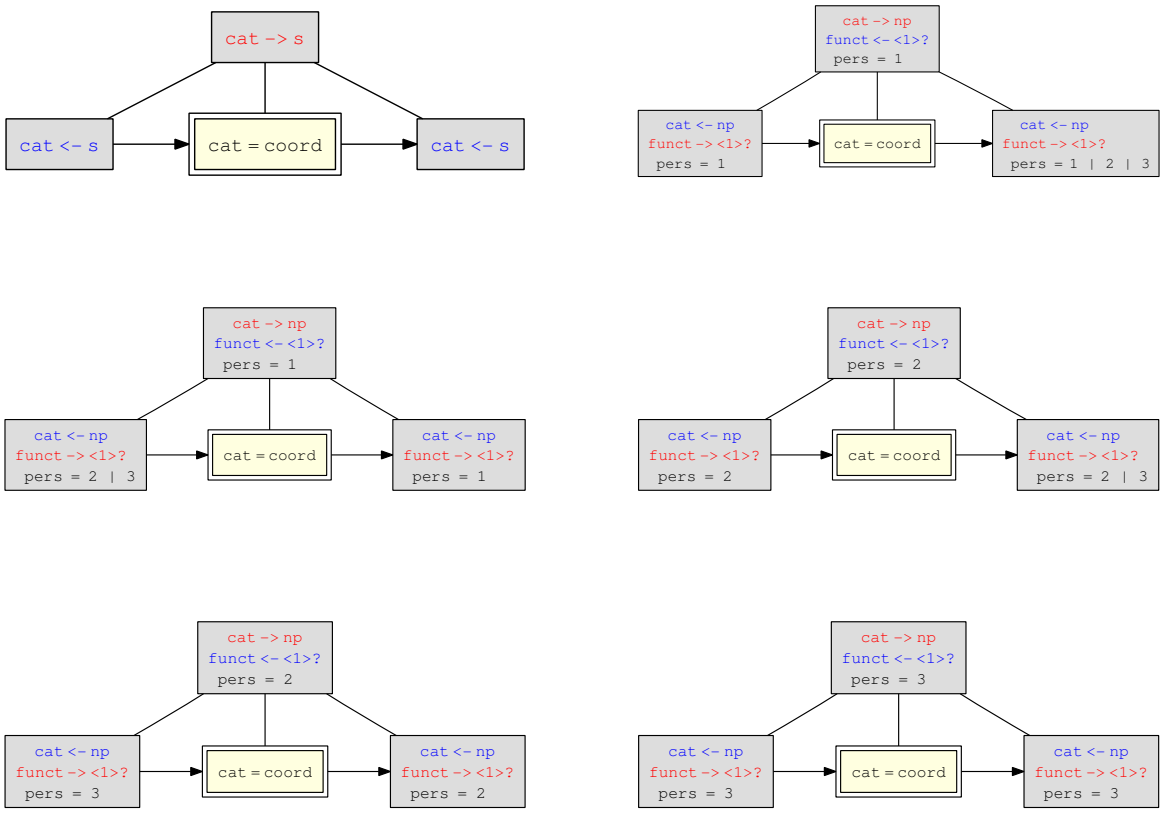


FIG. 4.2 – La grammaire produite

XMG permet donc d'exprimer la dépendance qui existe entre les traits d'accords des conjoints et ceux du groupe coordonné. On génère alors toutes les DAP correspondantes. Malheureusement cette dépendance n'apparaît plus dans les structures produites. Cela pose donc un problème d'ordre linguistique — on ne peut pas faire apparaître la dépendance — mais aussi un problème plus pratique : on génère de nombreuses structures. Voilà pourquoi, il serait bon d'étendre les GI de manière à ce que des valeurs de traits aient une valeur fonctionnelle ou relationnelle. Cela permettrait de résoudre ces deux problèmes.

4.4 Conclusion

Ce petit exemple visait donc à présenter certains aspects de l'écriture de la métagrammaire comme l'héritage, la sous-spécification, la coindexation de traits et les classes paramétrées. D'autres aspects, en revanche, n'ont pas été abordés : l'interface avec le lexique ou la sémantique.

Nous avons écrit six classes pour obtenir six descriptions. Ceci illustre bien le fait que la métagrammaire n'est pas tant une factorisation qu'une organisation rationnelle, une vue synthétique de la grammaire produite. Cette organisation explique également la construction des DAP finales par l'assemblage de DAP partielles.

Deuxième partie

Analyse de la coordination

Chapitre 5

Modélisation de la coordination

Sommaire

5.1 Introduction

La coordination est un phénomène complexe à modéliser. Mis à part les cas simples de coordinations de constituants, qui déjà peuvent poser problèmes si le formalisme distingue les coordinations en fonction de la catégorie des constituants coordonnés, les autres phénomènes de coordinations sont un défi aussi bien aux formalismes qu'aux théories linguistiques sous-jacentes. En effet, la plupart des théories et des formalismes s'accommodent mal des non-constituants ou des ellipses (qui sont généralement deux façons de décrire la même chose) et la coordination regorge de telles constructions. De plus la coordination présente à la fois une grande régularité et une grande disparité : une grande régularité qui permet de généraliser très vite certaines observations comme la similitude entre les conjoints et une grande disparité car là où le régime (ou cadre de sous-catégorisation) d'un verbe est généralement fixé et admet peu de variations, il se pose la question de la définition de l'équivalent d'un cadre de sous-catégorisation pour les conjonctions de coordination, en tous cas d'établir quelles sont les natures et les fonctions des conjoints.

Contrairement aux autres phénomènes syntaxiques difficiles (extractions,...) la coordination est omniprésente. On cite souvent comme exemple le fait que la conjonction *and* est le second mot le plus présent dans le *Wall Street Journal* et il ne serait pas étonnant de faire les mêmes observations dans les quotidiens francophones.

Nous nous inscrivons dans un perspective double : tout d'abord nous voulons proposer une modélisation qui réponde aux principaux défis que pose la coordination, tels la notion de non-constituant, de polymorphisme ou encore d'ellipse. D'autre part, nous voulons disposer à court terme d'une implantation qui puisse être utilisée dans un système de TAL. C'est cette double gageure que nous essayons de tenir dans cette partie de la thèse.

Nous allons décomposer notre étude en trois parties :

- La première sera consacrée à la définition de la problématique. En effet, il serait présomptueux de prétendre répondre à tous les problèmes que pose la coordination. D'autre part nous pouvons partitionner le phénomène de coordination en fonction de la nature des conjoints. Cette partition se retrouvera au niveau de la modélisation. Chaque sous-phénomène sera modélisé par des DAP qui correspondront à un même patron.
- Ensuite nous passerons à la modélisation proprement dite. Nous nous inspirons largement de la modélisation de la coordination dans les grammaires catégorielles, en l'adaptant au cadre spécifique des grammaires d'interaction. Nous proposons également une extension des GI pour traiter un sous-phénomène de la coordination, la coordination disparate que nous définirons plus loin.
- Enfin, nous l'avons déjà dit, nous voulons proposer une implantation de la modélisation proposée qui puisse s'insérer dans la grammaire d'interaction du français développée par Guy Perrier [Per06]. La troisième partie de cette étude est donc consacrée à la métagrammaire de la coordination que nous avons développée à l'aide

de XMG.

5.2 Les phénomènes étudiés

Pour commencer l'étude de la coordination, nous allons distinguer les différents sous-phénomènes auxquels nous nous intéresserons dans ce chapitre. Cette première étape nous permettra de délimiter clairement ce que nous voulons ou ne voulons pas modéliser. Cette classification en sous-phénomènes aura bien sûr des conséquences sur la modélisation elle-même. En effet, pour chaque sous-phénomène distingué, on trouvera un *patron* de DAP particulier.

5.2.1 Coordination de constituants

Le premier de ces sous-phénomènes — et le plus consensuel à travers les différentes théories linguistiques — est la coordination de constituants, présentée par les phrases de l'exemple (5.1) dans lesquelles deux propositions indépendantes, deux syntagmes nominaux et deux syntagmes adjectivaux sont coordonnés. Il apparaît que des constituants similaires doivent toujours pouvoir se coordonner. La similitude s'entend comme la compatibilité de nature et de fonction.

(5.1) (a) Il pleut et il neige.
(b) Jean et son frère discutent.
(c) un drapeau rouge et noir
(d) * Jean discute et son frère.
(e) * Jean a été introduit par Pierre et par la grande porte.

L'exemple (5.1c) pourrait également s'interpréter comme une coordination lexicale où les deux adjectifs coordonnés se comportent comme un mot unique [Abe06]. Nous ne considérons pas ce type de construction dans ce chapitre. Plus précisément, nous n'offrons pas de construction particulière à la coordination de deux mots par rapport à la coordination de deux groupes de mots.

Le contre-exemple (5.1d) illustre l'impossibilité générale de coordonner des constituants de natures différentes, en l'espèce une proposition et un syntagme nominal. Ceci conforte donc l'hypothèse de la similitude.

En revanche le contre-exemple (5.1e) montre que la nature ne suffit pas dans certains cas. Nous sommes ici à la limite du zeugme sémantique (ou attelage). Mais ici la fonction syntaxique nous permet d'interdire cette phrase. On peut imposer que l'agent *d'introduire* dans le sens de *faire entrer* requiert un agent animé et que donc le second groupe prépositionnel ne peut pas être agent. On voit ici que la fonction a également son importance. Mais on pourrait postuler que la notion d'animation (la distinction animé/inanimé) fait partie de la nature.

Plus généralement, une première approximation de ce fait linguistique est la loi de coordination des semblables [Cho57] qui établit que deux segments sont coordonnables si et seulement si ce sont des constituants de même nature. C'est même sur cette loi que se fonde une des règles de définition de la constituance. Si *un type de segment* peut être

utilisé dans une coordination alors il est un constituant. Cependant, on le verra dans la section 5.2.2, cette *loi* souffre de trop nombreuses exceptions pour être gardée : elle ne tient compte ni de la coordination de non-constituants ni de la coordination disparate. On peut toutefois saisir par cette loi le début de notre modélisation, en l'occurrence une certaine similarité entre les groupes conjoints.

5.2.2 Coordination de non-constituants

Ensuite viennent les coordinations de non-constituants, qui posent problème à la majorité des formalismes puisqu'il s'agit de donner un statut *officiel* à des fragments de constituants, ou à des constituants incomplets. Selon les approches linguistiques, ces coordinations peuvent être considérées comme des constructions elliptiques où une partie d'un des conjoints a été effacée et qu'il faut retrouver comme le font [BS04] et [SS06], ou, au contraire, comme la coordination de deux segments qui partagent une partie de la phrase, qui est donc factorisée, et que la conjonction distribue [Ste90]. Par exemple, dans la phrase *Jean boit du vin et mange* (5.2a), soit on considère que l'on coordonne deux propositions mais que le sujet de la seconde a été effacé ou alors que l'on coordonne deux groupes verbaux[26] ayant le même sujet, ici *Jean*.

(5.2) (a) [Jean boit du vin et ~~Jean~~ mange] *vs.* Jean [boit du vin et mange].
(b) [Jean aime ~~la compétition~~ mais Marie déteste la compétition] *vs.* [Jean aime mais Marie déteste] la compétition.

Pour tenir compte de ces exemples, on affine alors la loi de coordination des semblables par la loi de Wasow [PZ86] qui indique que l'on peut coordonner deux segments si chacun d'entre eux peut se trouver seul dans le contexte qu'offre le reste de la phrase (sans tenir compte des règles d'accord entre le segment et le contexte bien sûr). Les exemples (5.2) peuvent s'expliquer par cette règle. En revanche, cette règle est peut-être trop permissive, voir l'exemple (5.3a) repris de [PZ86] où l'on coordonne des constituants de natures différentes. L'exemple (5.3b) indique que même dans le cas où les natures sont similaires, il reste quelques problèmes à régler, notamment l'autonomie des clitiques.

(5.3) (a) ?* Jean veut [une bière et s'amuser].
(b) * Je [vais et viens].
(c) * [Jean aime et Marie va à] Paris.

Le problème vient de cette notion assez floue de contexte. Le contexte est-il simplement les suites de mots avant et après le groupe considéré, c'est-à-dire une chaîne de caractères avec un trou ? Ou bien est-ce un fragment de structure d'analyse dans lesquels peut s'insérer la structure associée au groupe conjoint ? Le contre-exemple (5.3c) indique plutôt que c'est au niveau des structures d'analyse qu'il faut chercher la notion de contexte.

[26] Nous faisons l'hypothèse, commune dans les grammaires catégorielles, que le groupe verbal n'est pas un constituant. Une façon de caractériser les constituants en GI est la présence de traits polarisés $\rightarrow$ ou $\leftarrow$ uniquement sur la racine. Ce n'est pas le cas des verbes qui ont également un nœud polarisé représentant l'attente d'un sujet.

Montées de nœuds

Nous allons distinguer un premier groupe de non constituants.

(5.4) (a) Jean chante des chansons et joue de la guitare.
(b) Jean adore mais Marie déteste vraiment les burritos.

Dans les exemples (5.4a) et (5.4b), la partie élidée ou factorisée (selon l'approche choisie) se trouve en périphérie. Dans le premier exemple, c'est *Jean* qui apparaît en début de syntagme et dans le second c'est *les burritos* qui apparaît en fin de syntagme.

Le phénomène en (5.4b) est appelé montée de nœud droit. C'est la partie droite du second conjoint qui *monte* et se retrouve partie droite des deux conjoints.

En revanche l'exemple (5.4a) peut prêter à débat. Si l'on considère que des groupes verbaux sans sujets sont des constituants alors cette phrase ne présente qu'une coordination de constituants. En revanche, si l'on considère qu'un groupe verbal sans sujet est défectif, c'est donc un non-constituant, et l'on est alors face au symétrique du phénomène évoqué ci-dessus : la montée de nœud gauche. *Jean* est promu de sujet du premier conjoint au rôle de sujet des deux conjoints.

Trou verbal (*Gapping*)

Malheureusement, d'autres non-constituants existent, qui diffèrent de ceux présentés plus haut. Plus exactement, on distingue les coordinations de non-constituants avec *montée de nœud* où la partie factorisée est en périphérie de syntagme – comme dans les exemples déjà vus (5.4a) et (5.4b) – de celles où elle se situe à l'intérieur (phénomène dit de *gapping* ou trou verbal, d'après [God05]), comme dans (5.5a) où le verbe *être* manque à la seconde proposition. Ce type de coordination est réservé aux propositions. Ce genre de construction est néanmoins assez peu naturel, voir l'exemple (5.5b).

(5.5) (a) Joseph est informaticien et Charles, mécanicien.
(b) ? Le président a demandé aux députés de travailler dans le calme et le premier ministre aux sénateurs de faire de même.
(c) Jean veut que l'on appelle la police et Marie, les pompiers.
(d) Mes sœurs préfèrent la moquette et mon frère, le parquet.

Ce dernier type de coordination permet clairement de coordonner des segments de valences différentes. Les trous verbaux constituent la première exception à notre loi de coordination et la modélisation elliptique paraît mieux se généraliser aux trous verbaux, comme l'exemple (5.5a), où la seconde proposition voit son verbe élidé puisque déjà présent dans la première.

L'exemple (5.5c) illustre le fait que le trou verbal peut être occupé par un groupe verbal complexe, ici *veut que l'on appelle* tandis que l'exemple (5.5d) montre que le groupe verbal élidé n'est pas qu'une simple copie. Dans ce dernier exemple, le verbe présent est au pluriel et le verbe élidé devrait être au singulier.

Il semble donc difficile d'avoir une modélisation à base de factorisation à moins de modifier la structure des conjoints pour faire apparaître l'élément commun en périphérie.

D'autre part, il serait peut-être judicieux de ne pas modéliser ce type de coordination au niveau syntaxique et de développer une vision plus large de l'ellipse dans laquelle

le *gapping* s'inscrirait. Ceci nous permettrait de ne considérer que des coordinations de segments de même valence. Nous proposerons tout de même une modélisation pour les cas de *gapping* courants.

Coordinations de séquences

Il existe d'autres non-constituants coordonnables, appelées séquences qui sont des suites de de constituants (deux, le plus souvent). On distingue en particulier les séquences de compléments dépendant d'une même tête (*argument clusters*) qui peuvent se coordonner, comme par exemple en (5.6a). Ici, les conjoints sont des groupes formés d'un objet direct (accusatif) et d'un complément d'attribution (datif). Ces séquences sont généralement constituées de plusieurs constituants de natures différentes mais ces séquences elles-mêmes ne forment pas de constituants à proprement parler. L'exemple (5.6b), repris de Mouret, montre que ces constructions ne sont pas réservées aux arguments verbaux mais également aux arguments de noms prédicatifs.

(5.6) a) Jean donne des fleurs à Marie et des bonbons à Pierre.
b) La destruction de la gare routière par les bombes et de la gare ferroviaire par les tanks rend l'accès à la ville difficile.

Les séquences coordonnées ne sont pas toujours constituées exclusivement d'arguments comme le montre l'exemple suivant, où l'on coordonne deux séquences composées d'un syntagme nominal objet direct et d'un marqueur temporel adjoint.

(5.7) Jean voit sa soeur lundi et son frère mardi.

Il n'y pas de rapport de dépendance ou de constituance entre les éléments des séquences. Les conjoints ne peuvent pas former une seule entité, c'est la différence avec la montée de nœud à gauche que nous avons déjà présentée. Le nœud montée est non seulement dupliqué pour chaque conjoint mais il doit aussi pouvoir se rapporter à chaque élément des séquences. Par exemple, dans (5.7), *Jean voit* doit avoir un rapport à *sa sœur* et *lundi* dans le premier conjoint ainsi qu'avec *son frère* et *mardi* dans le second.

Coordinations disparates

Jusqu'à présent les conjoints étaient de même nature, de même fonction et de même valence. Cependant, on peut dans certains cas coordonner des objets de natures différentes s'il peuvent remplir la même fonction comme dans (5.8c) où l'on coordonne un nom commun et un adjectif attribut du sujet. Par contre ce type de coordination est généralement interdit (5.8d). Il n'est autorisé que quand le groupe coordonné est attribut. Mais ce premier cas est discutable : on peut très bien admettre que *bûcheron* est considéré dans (5.8c) comme dans (5.8a) comme un adjectif et que l'on est tout simplement en présence de la coordination de deux adjectifs.

(5.8) a) Pierre est bûcheron.
b) Pierre est très costaud.
c) Pierre est bûcheron et très costaud.
d) * Un bûcheron et très costaud (dort/dorment) à l'orée du bois.

En revanche, si l'on pense que ce sont des constituants de natures différentes, on qualifie de disparate [Tes59] ce type de coordination. Il existe deux manières de le traiter : soit on ajoute des règles ad hoc et ce type de coordination constitue l'exception de la règle générale qui veut que les conjoints soient de même nature, soit l'on autorise ces coordinations comme naturelles et il faut donc un système d'unification plus fin pour donner une catégorie à ce type d'expressions et à celles avec lesquelles elles se combinent. Mais cette extension du système peut remettre en cause certains fondements d'un formalisme, comme le fait [Sag02] pour HPSG. Contrairement à l'exemple (5.8), ce type de coordination est indiscutable dans les langues où les cas sont marqués. Plus particulièrement, il s'agit de donner un statut syntaxique aux notions de syncrétisme (appelé sur-spécification dans [Lec96]) et d'ambiguïté. Pour illustrer ces aspects, nous reprenons quelques exemples célèbres.

(5.9) a) Er findet und hilft Frauen.
b) * Er findet und hilft Männer.
a) * Er findet und hilft Kindern.

Dans l'exemple (5.9a), *Frauen* est simultanément aux cas accusatif et datif, c'est ce que l'on appelle le syncrétisme de cas. Parallèlement, *findet* requiert un objet à l'accusatif et *hilft* un complément d'attribution au datif. Dans ce cas uniquement, la phrase est correcte. Dans (5.9b) *Männer* est à l'accusatif uniquement et dans (5.9c) *Kindern* est au datif. En aucun cas ils ne peuvent satisfaire la coordination des verbes.

Ce type de coordination existe également en français. Le système de cas étant limité aux pronoms clitiques dans notre langue, il faut donc développer des exemples comme 5.10 pour voir surgir le phénomène (exemple provenant de [Bay96] citant [Kay75]). Cet exemple est encore repris par [Ing90] dans le but de montrer les limites des formalismes à base d'unification qui ne peuvent pas modéliser ces exemples.[27].

(5.10) a) Paul l'a frappé et mis à la porte.
b) Paul l'a frappé.
c) Paul lui a donné un coup.
d) * Paul (l'/lui) a frappé et donné un coup.
e) Paul nous a frappés et donné un coup.

Encore une fois, pour que ce genre de coordination soit autorisé, il faut la présence de syncrétisme. Dans (5.10e) *nous* est à l'accusatif et au datif face au syncrétisme provoqué par la coordination de *frapper*, qui attend un objet à l'accusatif, et *donner*, qui attend un complément d'attribution au datif. La coordination des deux verbes crée un complexe verbal qui requiert un complément à la fois accusatif et datif. C'est pourquoi l'exemple (5.10d) est rejeté : ni *le* ni *lui* ne peuvent fournir simultanément les deux cas.

Nous avons parlé également d'ambiguïté qui est la notion duale du syncrétisme. Dans les exemples (5.8a) (5.8b) On voit que la copule accepte un attribut du sujet sous la forme d'un nom commun ou d'un adjectif. Mais elle accepte également la coordination d'un nom commun et d'un adjectif (5.8c). Dans ce cas on parle d'ambiguïté.

Ces aspects méritent d'être développés en détail, c'est pourquoi nous leur consacrons une section entière.

[27] Plus précisément, pour dénoncer l'emploi de l'unification comme mécanisme universel dans certains formalismes.

5.2.3 Circonscription des phénomènes traités

La coordination est un sujet vaste et même après s'être attardé sur ces quelques sous-phénomènes, nous sommes encore loin d'atteindre l'exhaustivité.

Nous ne parlerons pas dans ce chapitre de la modélisation des coordinations à redoublement (5.11a) et incidentes (5.11b), des énumérations (5.11c) et des coordinations comportant plus de deux conjoints[28] (5.11d), ni des spécificités de certaines conjonctions de coordination comme *ni, car, mais* (5.11e) par exemple. Ces spécificités ne sont pas traitées non pas parce qu'elles seraient difficiles mais parce que nous nous sommes plutôt concentrés dans cette thèse sur les éléments communs aux conjonctions de coordination. Enfin, certains aspects semi-grammaticaux comme la coordination d'affixes (5.11f) ou l'accord partiel (5.11g) (phénomène peu présent en français mais courant dans d'autres langues où le groupe coordonné a le genre ou le nombre de l'un des conjoints seulement) ne sont pas envisagés ici.

(5.11) (a) Jean vient et le lundi et le mardi.
(b) Jean, et c'est tout à son honneur, ne travaille que deux heures par jour.
(c) Je suis venu, j'ai vu, j'ai vaincu.
(d) Mon père, ma mère, mes frères et mes sœurs assisteront à ma soutenance.
(e) Il n'y avait ni eau ni chauffage mais nous étions heureux.
(f) Il est à la fois anti- et pro- nucléaire.
(g) Populi provinciaeque liberatae sunt.

5.3 Modélisation dans les grammaires d'interaction

Dans cette section, nous présentons de manière générale notre modélisation du phénomène de coordination dans les GI. Une fois le principe général expliqué, nous le déclinerons selon les sous-phénomènes présentés en section 5.2.

5.3.1 Les différentes approches du phénomène

Il existe plusieurs propositions de modélisation de la coordination, présentées par exemple dans [Abe03], que l'on peut diviser en deux groupes : les approches symétriques et les approches asymétriques, schématisées sur la figure 5.1.

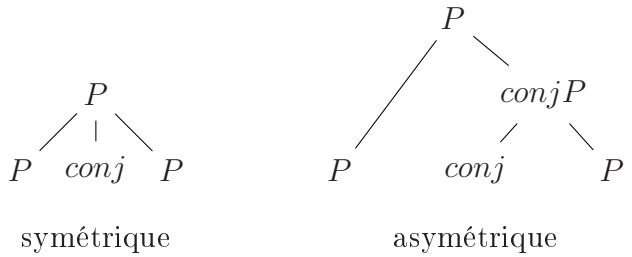


FIG. 5.1 – schémas des modélisations symétrique et asymétrique

[28] Nous donnons toutefois quelques pistes pour ces coordinations dans la section 5.3.7.

Dans l'approche asymétrique, la conjonction et l'un des conjoints forment un syntagme qui vient modifier l'autre conjoint, en d'autres termes l'un des conjoints est plus proche de la conjonction. Cette approche permet d'avoir une modélisation structurelle appelée hiérarchique par [Abe03] qui reflète cette différence entre les deux conjoints. De plus, cette approche autorise plus facilement la modélisation des coordinations/énumérations et celle des accords partiels.

Souvent l'un des conjoints est considéré comme le spécificateur et l'autre comme le complément. Cependant, cette modélisation implique que de nombreux traits sont partagés entre le spécificateur, le complément et le reste du syntagme, ce qui est tout à fait inhabituel et contraire aux hypothèses communément admises [Bor05]. Avec cette approche, la coordination de non-constituants doit s'interpréter comme la coordination d'un constituant complet et d'un constituant de même nature auquel il manque quelque chose, comme dans l'exemple suivant :

(5.12) Jean [[critique ~~son patron~~]$_{\text{SV}}$ [et [insulte son patron]$_{\text{SV}}$]$_{\text{ConjSV}}$]$_{\text{SV}}$.

On peut également voir dans l'exemple précédent, *et insulte son patron* comme un élément qui viendrait s'adjoindre à *critique* ~~*son patron*~~.

Dans l'approche symétrique, les deux conjoints sont à égale distance de la conjonction et sont habituellement de nature et de fonction équivalentes. Ils forment avec la conjonction un segment de même type. Cette approche laisse plus de liberté quant à la modélisation de la coordination de non-constituants. On peut comme précédemment y voir une coordination de constituants avec effacement de matériel ou alors admettre, comme c'est le cas dans les analyses utilisant les GC, que l'on peut coordonner des segments quelconques du moment qu'ils ont le même *comportement syntaxique*.

L'approche symétrique est celle qui est choisie pour modéliser la coordination dans les GC [Ste85; Ste90; Mor94]. C'est aussi celle que nous proposons de prendre comme point de départ pour les GI. Dans les GC, les conjonctions ont le type polymorphe $\forall X, (X\backslash X)/X$[29]. Ce type indique qu'une conjonction cherche à sa droite et à sa gauche des segments de même type X et attribue à leur réunion le type X. Malheureusement, ce type n'est pas suffisamment contraint. Premièrement, il indique que l'on peut tout coordonner, ce qui est une simplification très grossière de la réalité. Ensuite, ce type implique que l'on considère un calcul logique du second ordre, ce qui pose la question de la décidabilité. Il est nécessaire de trouver un fragment de ce calcul qui convienne mieux du point de vue calculatoire [Emm93].

En pratique, on liste plutôt tous les cas de conjonction pour toutes les catégories X que l'on souhaite coordonner. Cela pose un problème d'ambiguïté au moment de la sélection lexicale. Nous le traiterons au chapitre 6.

Cette modélisation s'étend naturellement aux coordinations disparates si l'on ajoute deux constructeurs analogues aux connecteurs additifs de la logique linéaire [Gir87] pour le syncrétisme et l'ambiguïté. De tels systèmes sont présentés dans [BJ95; Bay96].

[29] Par associativité, ce type est équivalent à $\forall X, X\backslash(X/X)$ et il n'y a donc pas de préférence pour un conjoint.

5.3.2 Le principe de superposition des interfaces

Nous décidons de nous placer dans le cadre symétrique présenté ci-dessus. D'après les observations que nous avons effectuées, on peut décider que deux groupes sont coordonnables s'ils ont le même comportement syntaxique. Dans les GI, cela revient à dire que si l'on superpose les DAP des éléments de chacun de ces groupes, on doit obtenir des structures similaires. Par similaires, nous voulons indiquer qu'elles peuvent agir avec le reste de la phrase de la même façon (cf. la loi de Wasow). Il nous faut donc définir une mesure de cette similitude.

Nous voulons donc coordonner des conjoints qui ont le même comportement syntaxique, ce qui revient à avoir le même type dans les GC et à avoir la même interface dans les GI. Nous avons également besoin de la notion de description duale.

L'interface représente la partie encore active d'une DAP :

Définition 11 (Interface)**.** *Nous appelons* interface *d'une DAP ce qui subsiste de cette DAP si l'on ne considère que la partie qui doit interagir avec d'autres DAP, c'est à dire les traits* $\rightarrow$, $\leftarrow$ *et* $\approx$ *ainsi que les nœuds qui les contiennent – les nœuds utiles – et les relations entre ces nœuds.*

Nous supposons que pour être coordonnés, les conjoints doivent avoir des DAP de même interface. C'est à dire que l'on impose que les conjoints ont les mêmes traits polarisés et que ces derniers sont placés de la même façon dans la DAP.

Il faut donc vérifier que les conjoints possèdent la même interface. La conjonction doit s'en assurer et le cas échéant proposer cette interface au reste de la phrase, en un seul exemplaire. Nous aurions pu pour cela définir une nouvelle opération sur les DAP qui aurait fusionné les DAP des conjoints pour donner une seule DAP avec la même interface. Cette opération aurait été déclenchée par la présence d'une conjonction de coordination. Mais nous ne voulions pas augmenter les GI pour modéliser le phénomène de coordination. Au contraire, nous montrons que ce formalisme est assez riche pour modéliser la majorité des cas de coordination. Nous allons donc émuler cette nouvelle règle avec les GI.

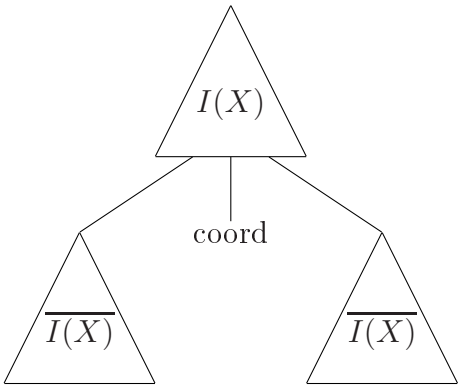


FIG. 5.2 – La forme générale des DAP associées aux conjonctions de coordination

La coordination doit saturer les interfaces des conjoints et également présenter cette même interface au reste de la phrase. C'est ici qu'intervient la notion de DAP duale. La duale d'une DAP est une DAP qui la sature complètement en cas de superposition :

Définition 12 (Duale)**.** *La* duale *d'une DAP est une DAP où les polarités sont opposées (→ devient ← et réciproquement ; ≈ devient =). Les traits neutres restent neutres mais comme nous appliquons cette notion aux interfaces ils ne seront pas considérés.*

À partir de ces hypothèses, nous pouvons établir un patron général pour une DAP associée aux conjonctions de coordination, présenté sur la figure 5.2. Cette DAP est constituée de trois parties :

- deux parties dites basses et destinées à saturer les conjoints. Elles sont construites à partir de $\overline{I(X)}$, la duale de l'interface commune des conjoints.
- une partie dite haute destinée à interagir avec le reste de la phrase. Elle est constituée de l'interface commune des deux conjoints, notée $I(X)$.

Les parties basses $\overline{I(X)}$ saturent complètement les conjoints. Ils ne peuvent pas interagir, c'est à dire dans les GI se superposer avec un autre élément de la phrase. Comme les deux parties basses sont identiques, elles forcent les deux conjoints à avoir la même interface.

La partie haute $I(X)$ agit exactement avec le reste de la phrase comme on voudrait que chacun des conjoints agisse. C'est la seule partie de la DAP qui peut se superposer avec d'autres DAP de la phrase.

Par exemple, on peut se poser la question de savoir si notre proposition permet d'analyser la phrase *Le salarié vient et signe le contrat.*. On doit d'abord vérifier que *vient* et *signe le contrat* ont le même comportement syntaxique, la même interface. La DAP associée à *vient* et son interface sont représentées en figure 5.3. Pour *signe le contrat*, la DAP et son interface sont en figure 5.4

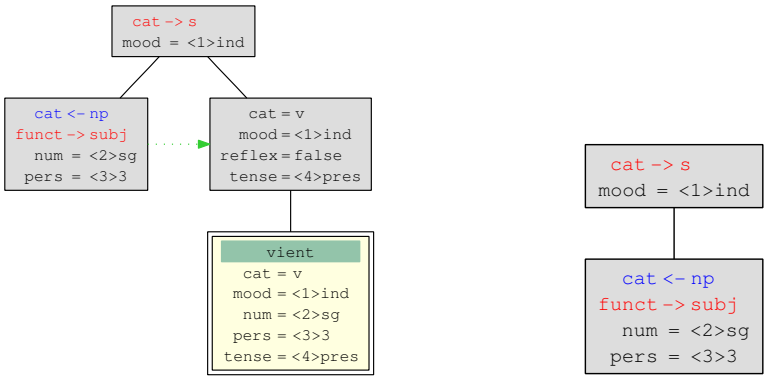


FIG. 5.3 – La DAP associée à *vient* et son interface

Les deux segments ont bien la même interface, notre modélisation autorise la coordination *vient et signe le contrat.* On sait également quelle est la forme de la DAP associée à la conjonction de coordination. Mais il faut nuancer ce schéma. Les trois parties contiennent souvent des nœuds qui ne figurent pas dans l'interface ni sa duale mais qui servent à *placer* correctement les nœuds de l'interface les uns par rapport aux autres. Ils permettent également d'indiquer la forme générale des conjoints et ce qu'ils attendent à gauche ou à droite.

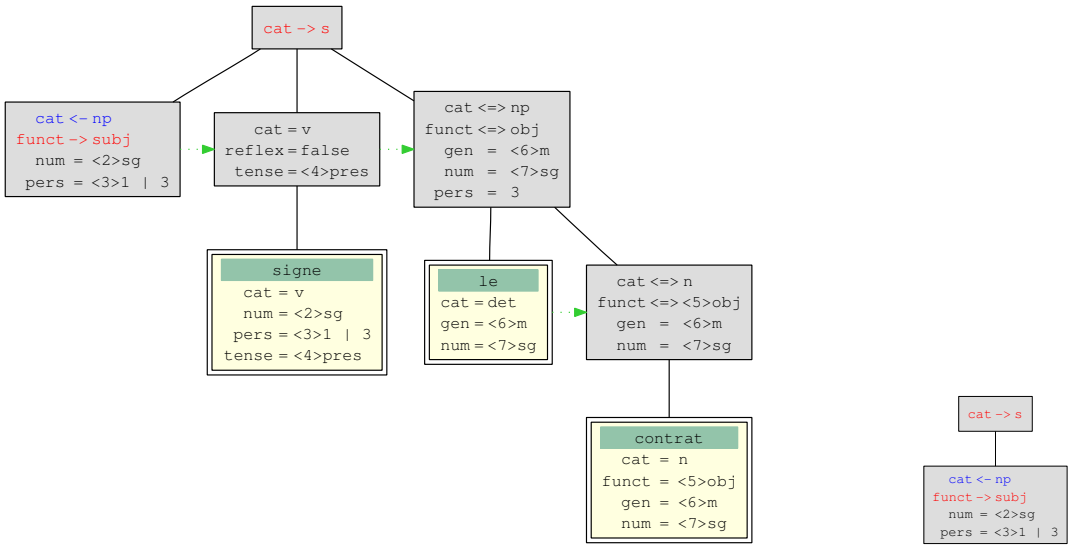


FIG. 5.4 – DAP et interface associées à *signe le contrat*

Ils servent à retrouver la structure de constituance des conjoints. Également, des traits neutres, c'est à dire absents de l'interface ou sa duale, sont ajoutés pour permettre de modéliser les règles d'accords entre les conjoints et le groupe coordonné qui sont propres à chaque langue et même à chaque type de syntagme. Nous verrons dans les exemples plus loin l'utilisation de ces nœuds et traits additionnels.

Nous appellerons *approche par factorisation* cette première modélisation puisqu'elle distribue le contexte symétriquement sur les conjoints.

Dans le cas de la coordination elliptique, il n'y a pas symétrie de la forme de surface autour de la conjonction, comme dans l'exemple suivant :

(5.13) Joseph est informaticien et Charles, mécanicien.

Il est nécessaire de faire apparaître la duplication de l'élément présent dans le premier conjoint et élidé du second dans la DAP associée à la conjonction de coordination, dans le but de retrouver une certaine symétrie. Pour ce phénomène, nous parlerons d'*approche par ellipse*.

Ces deux approches (factorisation et ellipse) peuvent sur des cas de coordination complexes s'utiliser conjointement.

(5.14) L'entreprise persuade le conseil et le directeur l'ingénieur que le spécialiste doit venir.

Pour résumer, nous transposons clairement la loi de Wasow dans notre formalisme en la limitant aux coordinations de segments ayant la même interface. En effet, nos DAP *empêchent* les segments coordonnés d'interagir directement avec la phrase et placent dans ce contexte le *dénominateur commun* de ces derniers. On vérifie donc que les segments peuvent être coordonnés si chacun d'eux peut l'être et on applique bien la loi de Wasow.

Dans les sections suivantes, nous allons voir pour chaque type de coordination, quelle est exactement la forme de la DAP associée à la conjonction de coordination. C'est à dire quels sont les nœuds et les traits ajoutés aux simples interfaces de la figure 5.1.

5.3.3 Les coordinations de constituants simples

Dans la situation la plus facile à comprendre, l'interface de chaque conjoint se résume à un seul nœud. C'est le cas des coordinations de constituants simples. Nous allons prendre l'exemple des syntagmes nominaux, voir figure 5.5. *Jean* et *l'homme* ont bien la même interface. Ils doivent être coordonnables. Leur interface ne prend en compte que leur racine qui est le seul nœud polarisé. Mais on va voir que la réalité de la modélisation est plus complexe que l'intuition que nous en avons donnée en section 5.3.2.

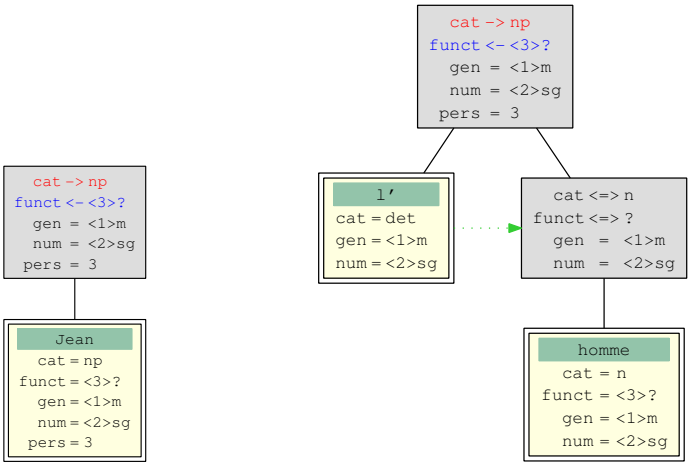


FIG. 5.5 – DAP associées à *Jean* et *l'homme*

Le problème vient du fait que les DAP des modificateurs (les épithètes et les pronoms relatifs notamment) se superposent sur la racine et sous le nœud fils de la racine de catégorie n ou np. On fait apparaître ainsi le syntagme simple et le syntagme modifié. Il faut donc que cette superposition puisse se faire sur la coordination des deux syntagmes nominaux, comme dans l'exemple suivant :

(5.15) J'ai vu [[Jean et Marie]$_{SN}$ qui partaient à la plage]$_{SN}$.

En conséquence la partie haute de la DAP de la conjonction de coordination doit avoir deux nœuds comme sur la figure 5.6. On peut voir que les traits de fonction syntaxique sont co-indexés entre la partie haute et les parties basses. La fonction est *propagée* du contexte aux conjoints par la conjonction de coordination.

Cette DAP contient également des traits neutres qui modélisent les règles d'accord. Ces traits ne dépendent pas de notre modélisation générale de la coordination, et donc nous ne leur avons pas donné de valeur spécifique dans l'exemple[30].

[30]Il est tout à fait possible d'en tenir compte lors de la conception de la grammaire avec XMG. Nous l'avons fait dans l'exemple de la section 4.3.

De plus, dans la version actuelle des grammaires d'interaction, les valeurs des traits ne sont pas des fonctions d'autres valeurs et il faut donc lister tous les cas d'accord[31]. Nous pensons ajouter dans une version ultérieure la notion de fonction de valeur de traits. Ici nous avons affaire à la coordination de deux syntagmes nominaux à la troisième personne où le premier est masculin. L'expression coordonnée est donc également au masculin et à la troisième personne, du pluriel quel que soit le nombre de chaque conjoint.

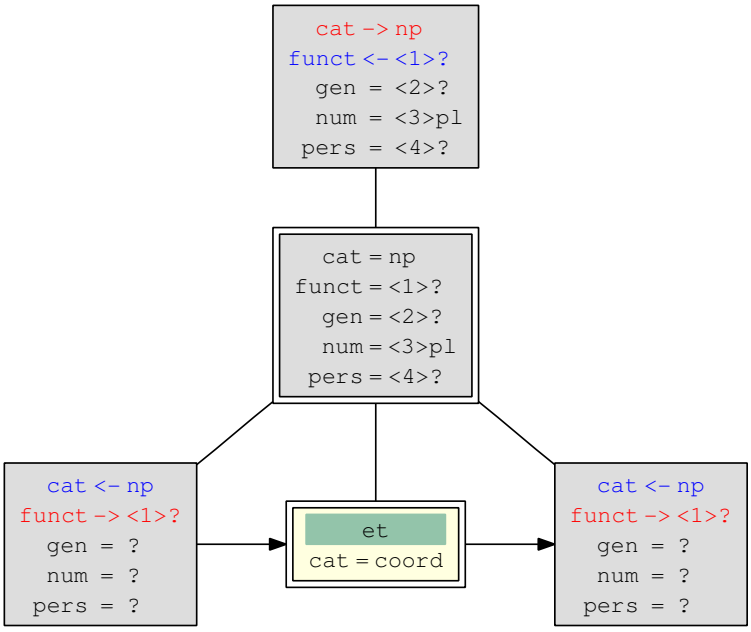


FIG. 5.6 – Une DAP pour *et* et la coordination de syntagmes nominaux

La figure 5.7 représente le groupe nominal *Jean et l'homme*. Les deux syntagmes nominaux et la conjonction forment un nouveau syntagme porteur d'une unique polarité $cat \rightarrow np$. Il est donc interprété comme un syntagme nominal complet.

Pour d'autres constituants comme les propositions par exemple, les modificateurs ne viennent que sur le nœud racine. Dans ce cas, on respecte le schéma de la figure 5.1.

5.3.4 La coordination de modificateurs

Dans cette partie, nous nous intéressons à la coordination de modificateurs en reprenant l'idée d'une coordination en trois parties, basée sur la notion d'interface. Nous allons prendre comme exemples les adjectifs épithètes.

La principale différence avec la coordination de constituants simples est que, dans le cas précédent seul le nœud racine était polarisé et donc l'interface se réduisait à un nœud. Dans le cas des adjectifs épithètes, le nœud racine et un de ses fils (le plus à droite ou le plus à gauche selon le modificateur) ont pour le trait *cat* la même valeur n mais des polarités différentes. La racine et son fils droit portent une catégorie virtuelle ($\approx$). Dans

[31] C'est pourquoi notre grammaire de la coordination est disponible en 2 versions : une version sans les traits d'accords et une version étendue avec les traits d'accord.

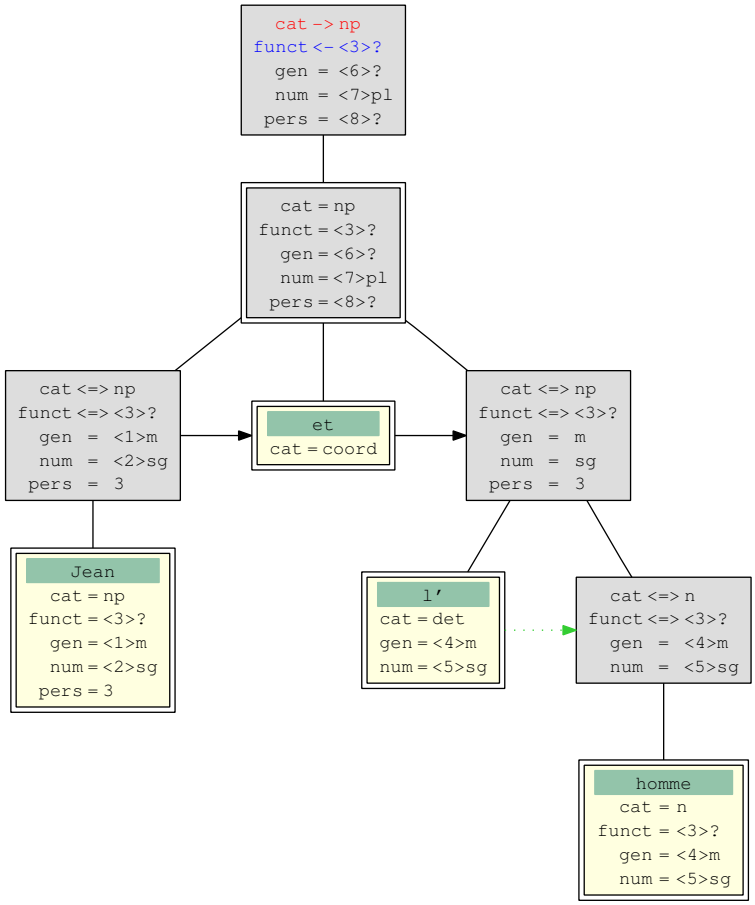


FIG. 5.7 – La description partielle de *Jean et l'homme*.

le cas des autres modificateurs, comme les pronoms relatifs ou les adverbes, le principe est le même. On peut voir une telle DAP sur la figure 5.8

Le nœud racine va se superposer sur la racine du nom commun (voir la DAP de *contrat* en figure 5.4). Le nœud fils va se superposer sur l'ancre du nom commun.

Nous voulons que la coordination de deux adjectifs ait le même comportement syntaxique. Il faut donc que la partie haute de la DAP associée à la conjonction de coordination ait ces nœuds.

Pour la coordination de modificateurs, la partie haute $I(X)$ de la DAP n'est donc pas atomique mais c'est un arbre unaire où nœud père et nœud fils ont la même valeur pour le trait *cat*, ainsi que la même polarité virtuelle. Quant aux parties basses, elles doivent requérir une DAP d'adjectif. Nous proposons qu'elle reprennent la forme de la DAP d'un adjectif mais pour le trait *cat* de valeur *adj* nous choisissons la polarité $\approx$, de manière à contraindre la superposition de l'adjectif.

Une telle description est présentée sur la figure 5.9. On peut observer que les coréférences des traits permettent d'accorder les adjectifs entre eux. Un exemple de coordination d'épithètes est représentée sur la figure 5.10. Le résultat de la superposition de la coor-

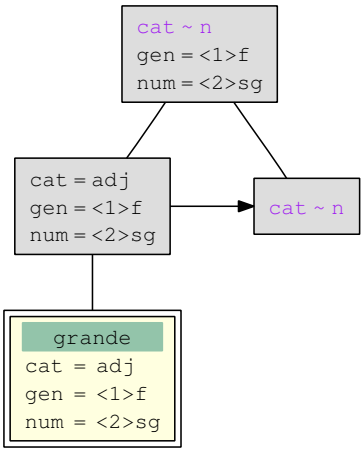


FIG. 5.8 – DAP de *grande* adjectif qualificatif épithète gauche

dination des adjectifs et d'un nom commun donne une DAP correspondant à un nom commun. Seule la racine est polarisée.

Nous avons modélisé la phrase de l'exemple 5.10 de la façon suivante :

(5.16) [belle e_i] et [grande e_i] femme$_i$

Cependant, nous préférons éviter de parler d'une théorie des catégories vides dans la cadre du travail présenté ici. Cette représentation est avant tout un choix pragmatique guidé par le fait que la structure finale de l'analyse est un arbre, ce que notre modélisation parvient à obtenir. C'est le choix qui a été fait dans la section 5.3.2. Ce que nous voulons exprimer ici, c'est le fait que les adjectifs conjoints partagent le nom qu'ils qualifient. Idéalement, il faudrait pour cela identifier des nœuds et donc la structure finale serait un graphe orienté acyclique (un arbre dans le quel on autorise l'identification de sous-arbres). Les nœuds vides et la coindexation des traits permettent de simuler cette identification.

5.3.5 La coordination de non-constituants

De nombreuses langues autorisent la coordination de segments qui ne sont pas des constituants, qui ne forment pas d'unité syntaxique. Nous allons montrer que la superposition partielle de descriptions d'arbres propre aux GI joue dans la modélisation de ce type de coordination un grand rôle.

Dans cette partie nous traitons d'abord du cas de montée de nœuds simples. Puis, nous étendrons la modélisation à la coordination de séquences de compléments.

La montée de nœuds

On peut distinguer les montées de nœuds gauches et les montées de nœuds droits, selon que la partie *factorisée* se trouve à gauche ou à droite du segment coordonné. Nous

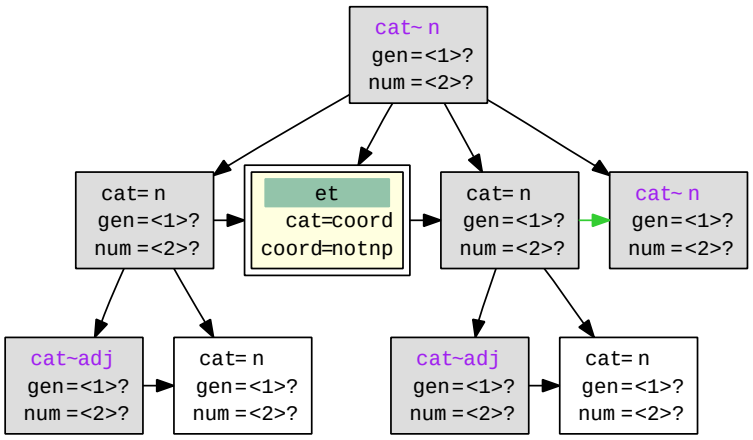


FIG. 5.9 – DAP pour la conjonction *et* dans le cas de la coordination d'adjectifs épithètes gauches

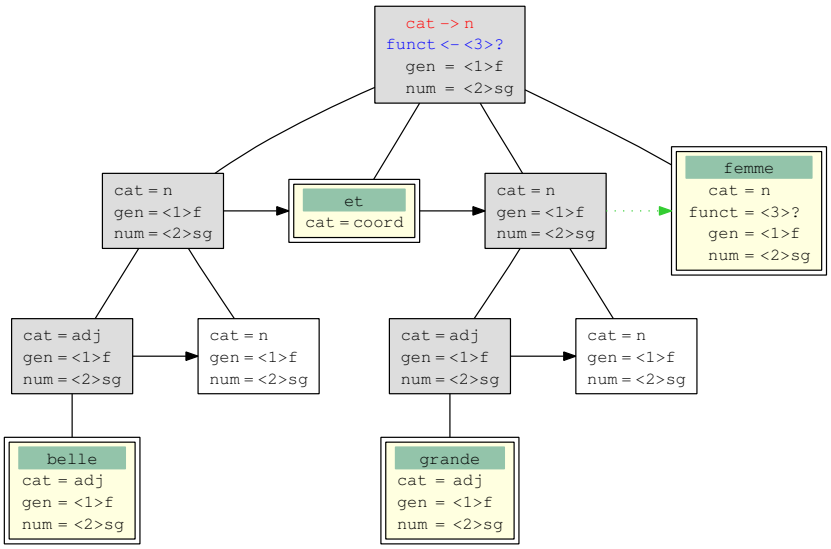


FIG. 5.10 – Coordination de modificateurs : DAP associée à *belle et grande femme*

rappelons qu'il n'y pas ici de syntagme verbal qui comprendrait le verbe et son cadre de sous-catégorisation moins le sujet. L'exemple (5.17a) est aussi une coordination de non-constituants.

(5.17) (a) Jean [boit du vin et mange]. (montée de nœud gauche)
(b) [Marie aime mais Pierre déteste] les glaces. (montée de nœud droit)

Nous nous plaçons toujours dans le cadre d'une modélisation par factorisation du phénomène. La seule différence avec la coordination de constituants complets est qu'ici les conjoints ont des dépendances non satisfaites. Leur interface a donc une structure plus complexe, que nous avons déjà évoquée, voir la figure 5.4. La figure 5.11 représente la

DAP associée à la conjonction dans l'exemple (5.17a). La partie haute comporte un nœud $cat \rightarrow s$, la coordination fournit une proposition au reste de la phrase et un autre nœud $cat \leftarrow np, funct \rightarrow subj$, la coordination requiert un syntagme nominal sujet. Dans les parties basses au contraire la conjonction demande une proposition à chaque conjoint et leur fournit un syntagme nominal sujet à chacun.

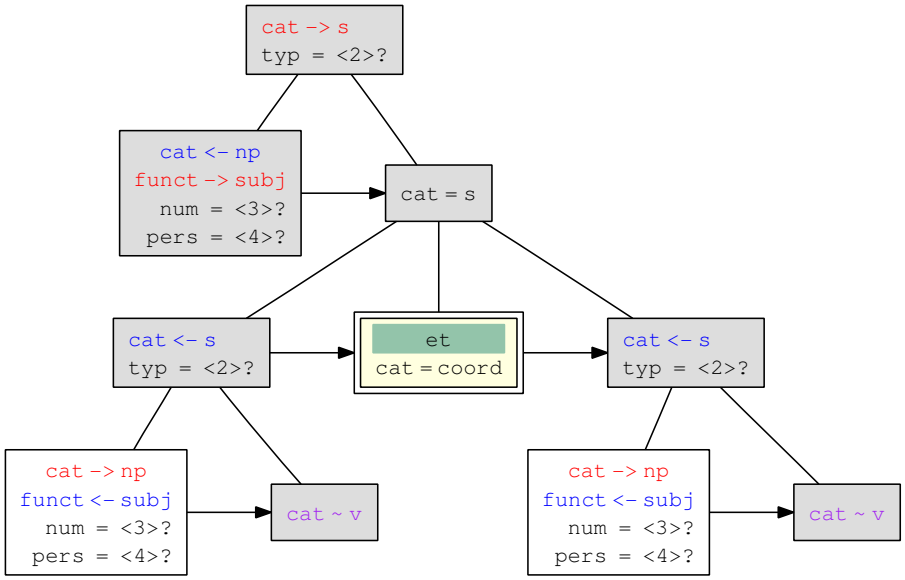


FIG. 5.11 – Montée de nœud : DAP de la conjonction pour une montée du sujet à gauche

On voit sur les parties basses de la DAP de la figure 5.11 des nœuds aux traits virtuels. Ils donnent une indication de contexte pour placer le nœud sujet. Ici, nous aurions pu ne pas les mettre et imposer à l'aide d'une marque que le nœud sujet est le plus à gauche des fils.

C'est la conjonction qui va leur fournir ces dépendances et présenter au reste de la phrase un seul bloc avec les mêmes demandes et les mêmes ressources que chacun des conjoints. On peut voir l'analyse de la phrase (5.17a) sur la figure 5.12.

La DAP associée à la conjonction ici n'autorise pas à aller chercher la dépendance profondément dans le conjoint[32] comme dans l'exemple suivant :

(5.18) Jean voudrait que l'on ratifie mais Marie refuse absolument que le président signe le traité.

C'est tout à fait envisageable en divisant le nœud associé en deux nœuds, le premier neutre dominant largement le second polarisé $\rightarrow$ ou $\leftarrow$, et en imposant que tous les nœuds intermédiaires aient la fonction objet

Les séquences de compléments

Dans la section 5.3.4, nous avons vu qu'il était envisageable de *décorer* les différentes parties d'une DAP associée à une conjonction de nœuds ne faisant pas partie de l'interface

[32] Ceci n'a pas lieu d'être pour le sujet mais les dépendances non-bornées sont autorisées pour l'objet.

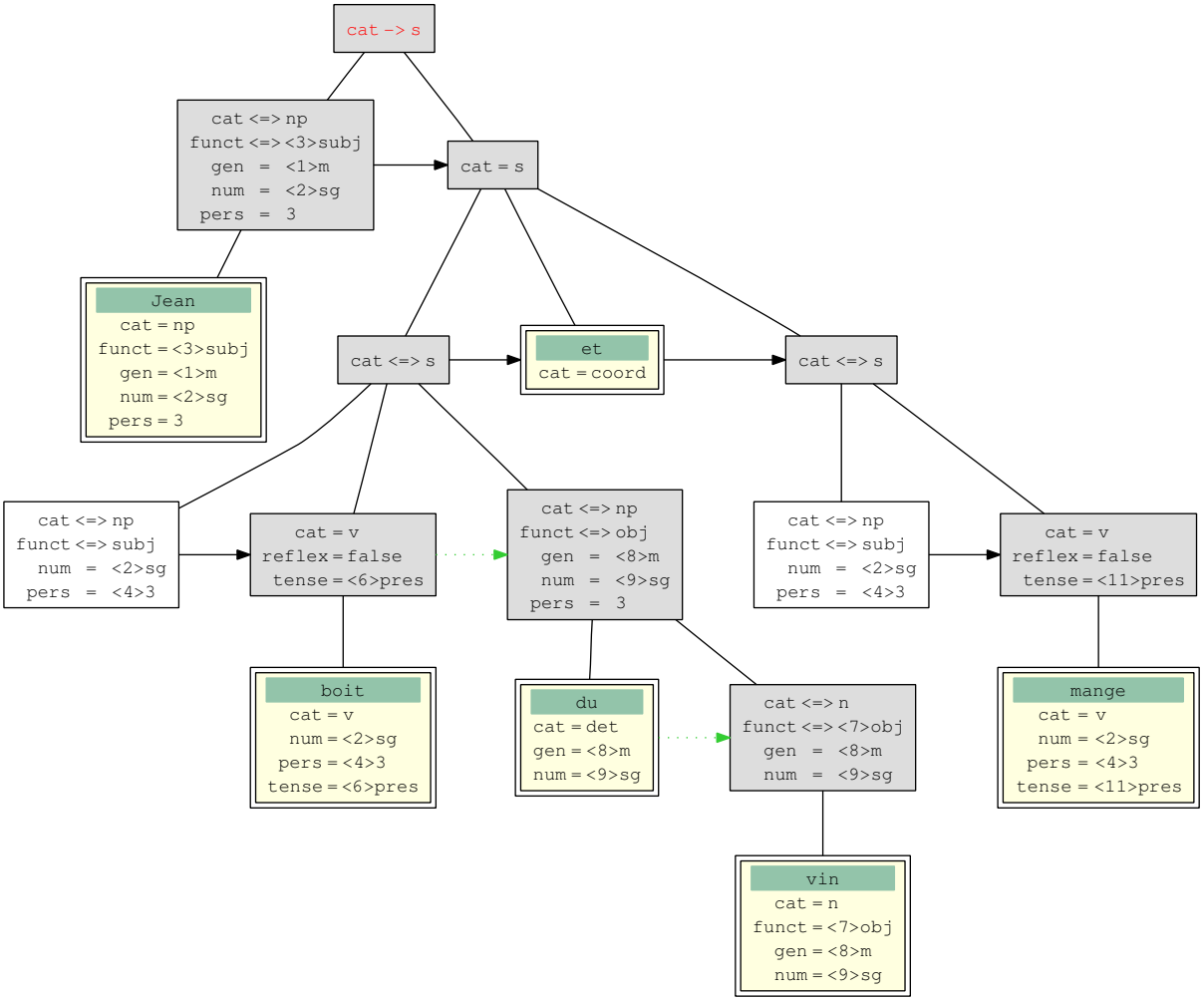


FIG. 5.12 – Montée de nœud : DAP correspondant à *Jean boit du vin et mange*

des conjoints dans le but de placer correctement ces conjoints. Ceci est une utilisation assez classique des traits virtuels dans les GI : ils fournissent un contexte pour la superposition des traits polarisés. Nous utilisons cette technique pour modéliser la coordination de séquences de compléments. Ils servent également à retrouver une structure syntagmatique de la phrase.

Les trois parties de la DAP vont contenir de nombreux nœuds neutres qui vont permettre de reconstituer une structure en constituants pour chaque conjoint. La DAP associée à la conjonction pour ce type de coordination possède toujours ces trois parties caractéristiques. Une DAP pour la conjonction de coordination en cas de coordination de séquences de compléments similaires à celui de l'exemple (5.19) est représentée sur la figure 5.13. La figure 5.14 représente l'arbre syntaxique de cette phrase.

(5.19) L'entreprise offre [un emploi à l'ingénieur et un stage à l'étudiant].

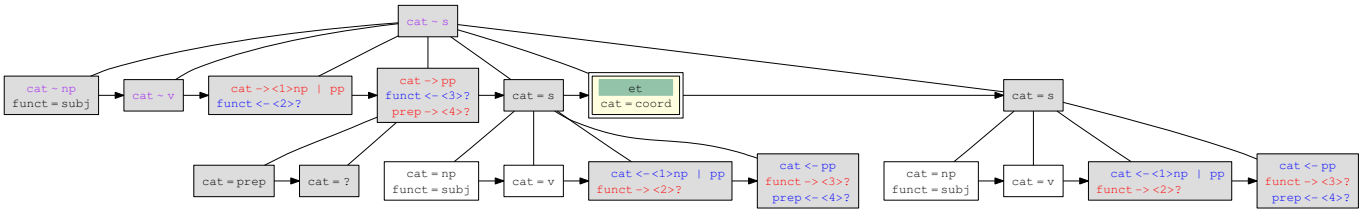


FIG. 5.13 – DAP de la conjonction pour une coordination de séquences de compléments

On veut ici pouvoir coordonner un syntagme nominal et un syntagme prépositionnel, qui sont tous les deux des constituants complets. L'interface de chacun étant son nœud racine, la partie haute et les parties basses sont au moins chacune composée de deux nœuds. Pour retrouver la structure syntagmatique et placer les conjoints, il faut donner plus de contexte.

La partie haute dans ce cas contient de nombreux nœuds virtuels qui servent donc à placer la DAP dans le bon contexte. Mais ils servent aussi dans une vision plus sémantique, à dupliquer l'information concernant le sujet et le verbe dans les parties basses. On peut même ici parler de DAP en quatre parties : une partie à dupliquer puis les trois parties habituelles. Nous verrons dans la partie traitant de l'implantation de la grammaire que nous traitons ce phénomène à part des précédents.

Mais les agrégats coordonnés ne sont pas exclusivement composés d'arguments verbaux, comme le montre les exemples 5.7 et 5.20. En adaptant la DAP pour les séquences d'arguments, on peut analyser la phrase :

(5.20) Jean voit Pierre ce soir et Marie demain.

Le résultat de l'analyse est présenté sur la figure 5.15.

Ces exemples posent problème aux analyses de type *montée de nœud* comme la nôtre. Seules les GC avec la montée de type disent n'avoir aucun problème avec ce type de coordination, voir par exemple [Dow88]. Dans ce formalismes on ne distingue pas particulièrement ce phénomène de celui de montée de nœud droit. Nous y reviendrons lors de la comparaison de notre approche avec les modélisations existantes.

Pour les phénomènes précédents, nous avons choisi une analyse par factorisation pour modéliser la coordination de constituants et les montées de nœud. Nous allons voir dans

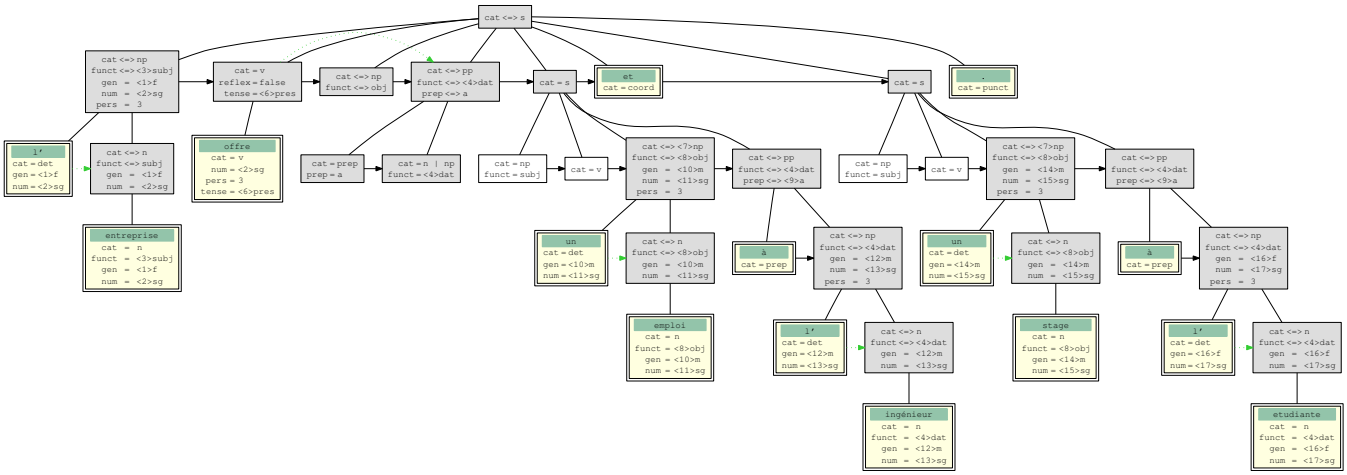


FIG. 5.14 – L'analyse de *L'entreprise offre un emploi à l'ingénieur et un stage à l'étudiant.*

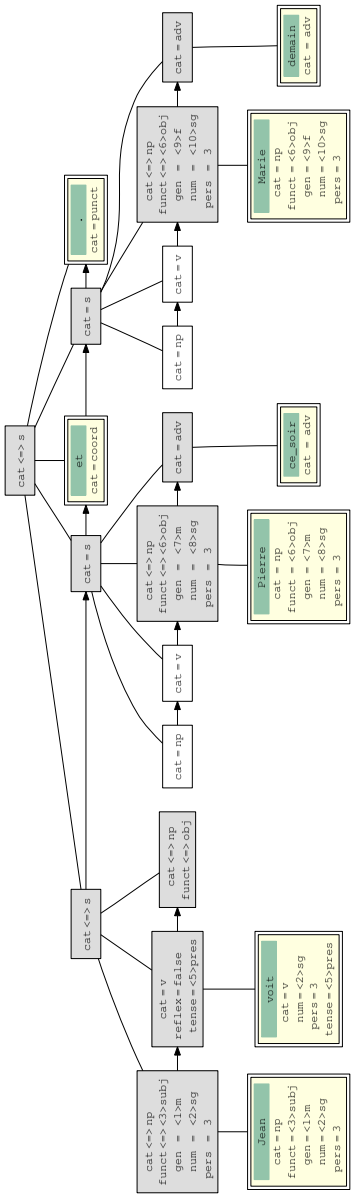


FIG. 5.15 – Analyse de *Jean voit Pierre ce soir et Marie demain.*

la section suivante que nous pouvons également exprimer l'idée d'ellipse dans les GI. Cela indique que nous aurions pu modéliser la coordination par l'approche elliptique également pour les montées de nœud.

5.3.6 La coordination avec ellipse ou gapping

Pour des phrases comme (5.5) que nous reproduisons ci-dessous, nous modélisons un effacement – du verbe dans cet exemple. C'est à première vue la seule entorse à l'analyse symétrique que nous développons. Pourtant, c'est justement pour retrouver une certaine symétrie entre les conjoints que nous avons besoin de cette analyse.

(5.21) (a) Joseph est informaticien et Charles, mécanicien.
(b) ? Le président a demandé aux députés de travailler dans le calme et le premier ministre aux sénateurs de faire de même.
(c) Jean veut que l'on appelle la police et Marie, les pompiers.
(d) Mes sœurs préfèrent la moquette et mon frère, le parquet.

Ici la DAP associée à la conjonction (cf. figure 5.16) vient saturer le premier conjoint, un proposition et fournir la partie manquante, le verbe, au second conjoint en la dupliquant depuis le premier. La partie haute agit comme une proposition complète. Le résultat est présenté sur la figure 5.17.

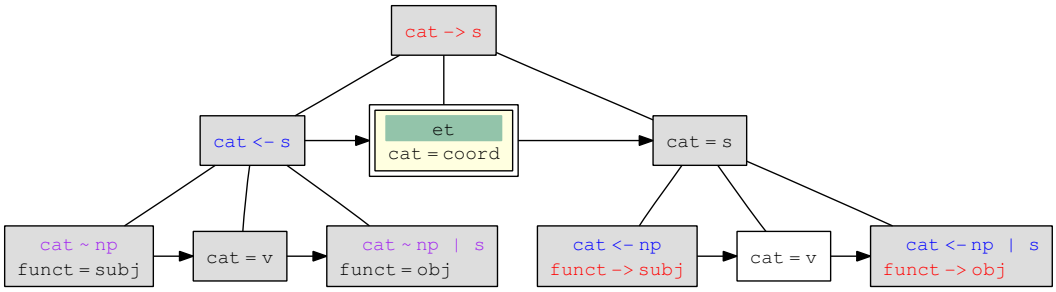


FIG. 5.16 – La DAP pour la coordination de propositions avec ellipse du second verbe

Cette possibilité de modélisation indique que les GI permettent tout à fait une analyse elliptique de tous les phénomènes de coordination de non-constituants comme la montée de nœud, et qu'elles offrent même le choix de la modélisation phénomène par phénomène.

Bien sûr, pour des cas plus complexes les deux approches peuvent être utilisées conjointement. L'exemple (5.22) fait intervenir ellipse et montée de nœud droit.

(5.22) L'entreprise persuade le conseil et le directeur l'ingénieur que le spécialiste doit venir.

Ceci nous semble être un point clé de notre approche. Les GI sont suffisamment souples pour exprimer à la fois les aspects de factorisation et les aspects d'ellipse de la coordination dans une même description. En revanche, le point faible de notre approche est le grand nombre de DAP associées aux conjonctions de coordinations.

5.3.7 Remarques sur les coordinations n-aires

Dans la section 5.2.3, nous avons indiqué que le traitement des coordinations de plus de deux conjoints ne serait pas abordé ici. Les difficultés liées à la coordination sont plutôt dues à la factorisation ou à l'ellipse – qui apparaissent déjà dans les coordinations binaires – qu'au nombre de conjoints mis en relation. Cependant, il demeure un problème qui doit être résolu pour modéliser les coordinations comprenant plus de deux conjoints : comment exprimer qu'une coordination fait intervenir un nombre arbitraire (supérieur ou égal à deux !) de conjoints dans un cadre lexicaliste ? Comme c'est un phénomène assez courant, nous développons dans cette section quelques pistes pour les modéliser.

(5.23) (a) John, Paul, George et Ringo formaient les Beatles.
(b) ? John, Paul, George, Ringo formaient les Beatles.
(c) ? Jean sonna, entra et Marie en fut surprise.
(d) Joseph est informaticien, Charles mécanicien et Marianne musicologue.

Une première solution

La solution qui semble la plus simple est de traiter les virgules comme des conjonctions de coordination. La phrase (5.23a) peut ainsi être modélisée si l'on considère que les virgules peuvent être associées à la DAP de la coordination de groupes nominaux. Cette première proposition surgénère : en particulier, on peut désormais coordonner sans réelle conjonction de coordination (5.23b). Une solution à ce problème est d'exiger dans le cas de la virgule que le conjoint droit soit lui-même un groupe coordonné. Dans (5.23b),

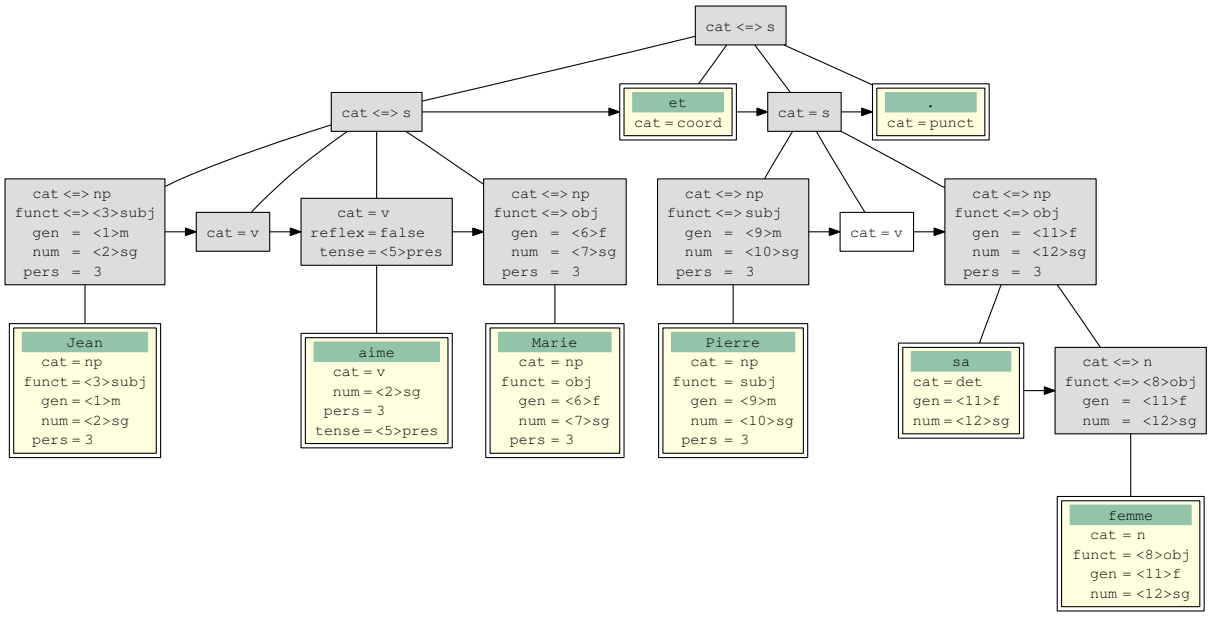


FIG. 5.17 – Coordination avec ellipse : analyse correspondant à *Jean aime Marie et Pierre sa femme.*

cela interdirait la coordination de *George* avec *Ringo* puisque ce dernier groupe nominal n'est pas une coordination. Comment modéliser cette exigence ? On peut imaginer que les nœuds sont équipées d'un trait à trois valeurs qui indique si ce nœud représente un groupe coordonné par une conjonction (valeur *conj*) ou par une virgule (valeur *vir*), ou si ce n'est pas un groupe coordonné (valeur *nil*). La virgule requiert que le conjoint droit porte les valeur *conj* ou *virg* pour ce trait tandis que les conjonctions de coordination requiert qu'il soit positionné à *conj* ou *nil*.

Un problème plus sérieux est illustré par l'exemple (5.23c). Dans cet exemple, la virgule et la conjonction ne sont pas des coordinations de segments similaires : la virgule coordonne deux verbes intransitifs tandis que la conjonction coordonne deux propositions[33]. Nous ne voyons pas de solution simple à ce problème des GI. Dans ce cas, il faudrait peut-être étendre les DAP en les équipant de branches réutilisables à loisir. La DAP d'une conjonction de coordination serait ainsi capable de saturer des listes de DAP de conjoints de même interface.

Séquences d'arguments et trou verbal : une nouvelle solution

Pour les coordinations de séquences ou les trous verbaux, les virgules doivent avoir une valence différente des conjonctions de coordination. En effet, la conjonction dans ces cas interagit beaucoup avec le contexte offert par le premier conjoint qui est dupliqué pour le second. La virgule doit juste permettre de superposer une proposition sans verbe dans le contexte offert par une phrase présentant un trou verbal comme nous l'avons vu précédemment. Dans notre exemple (5.23d), le groupe *Charles mécanicien* doit s'insérer dans le contexte *Joseph est informaticien et Marianne musicologue* que nous savons analyser. Le conjoint qui suit une virgule est donc vu comme s'adjoignant au groupe formé du premier conjoint et du dernier conjoint (celui qui est précédé de la conjonction de coordination). Il est modélisé comme un modificateur de groupe coordonné.

Cette dernière approche permet de traiter les cas jusque là inaccessibles. Mais elle peut très bien s'appliquer également aux coordinations de constituants et aux montées de nœuds. Elle offre deux avantages :

- uniformiser la modélisation de tous les types de coordination,
- offrir une structure d'analyse (un arbre) plus simple. En effet, avec cette approche le groupe coordonné formé de tous les conjoints est modélisé comme un étage de l'arbre d'analyse. En traitant la virgule comme une conjonction, un groupe de n conjoints est représenté par un sous-arbre de n étages.

5.4 Extension pour la coordination disparate

La coordination disparate est une coordination qui implique des conjoints de natures différentes. c'est un phénomène assez courant bien que jugé maladroit par les grammaires prescriptives du français.

[33]Il semble toutefois, d'après le commentaire d'un rapporteur, que cette construction soit tolérée en anglais : *John came in, started to smoke and drink beer.*

(5.24) L'individu vous a frappée au visage et ensuite donné des coups de pieds dans les côtes.

Dans l'exemple précédent, on coordonne un verbe qui attend un objet direct (*frapper*) et un verbe qui attend un complément d'attribution (*donner*).

Dans les langues à déclinaisons, certains cas peuvent être morphologiquement confondus : c'est le phénomène de *syncrétisme* de cas. Nous pouvons modéliser ce phénomène en établissant qu'un même mot peut fournir plusieurs cas. De la même manière certains arguments peuvent être réalisés par compléments de nature différente. Il y a donc *ambiguïté* sur la nature du complément. Comme on va le voir, multiplier les DAP associées à ces mots ne suffit pas pour traiter les cas complexes.

Les exemples suivants illustrent le syncrétisme et l'ambiguïté en polonais :

(5.25) (a)

*Kogo/*Co*	*[Janek*	*lubi]*	*a*	*[Jerzy*	*nienawidzi]?*
(acc∧gen)/(nom∧acc)		obj.acc			obj.gen ?
qui/*que	Jean	aime	mais	Georges	déteste

(b)

Dajcie	*[wina]*	*i*	*[cał̨a świnię]!*
obj.(acc∨gen)	gen		acc
Donne	du vin	et	un cochon entier !

(c)

**[Janek*	*lubi]*	*a*	*[Jerzy*	*nienawidzi]*	*[psy]*	*i*	*[kotów].*
					acc.		gen.
					les chiens	et	les chats.

[BJ95] et surtout [Ing90] montrent que les formalismes qui n'utilisent que l'unification ne peuvent pas accepter les phrases (a) et (b) tout en rejetant la phrase (c).

Pour prendre en compte ce type de coordination, nous allons reprendre et modifier les définitions des domaines de valeurs et de l'opération de superposition de traits données en 1.2.3. Nous nous inspirons en cela de l'approche de [BJ95; Bay96], également exposée dans [Lec96] pour les GC et reprise pour HPSG dans [Sag02]

5.4.1 Structure des domaines de valeurs

Puisque nous ne pouvons pas modéliser le syncrétisme et l'ambiguïté par la simple unification, nous allons ajouter des opérateurs explicites pour ces ces deux phénomènes.

Soient f un nom de trait appartenant à $\mathcal{F}$ et $\mathcal{V}_f$ l'ensemble des valeurs atomiques pour ce noms de trait, Nous pouvons définir $\mathcal{D}_f$ le domaine des valeurs de f construit inductivement sur $\mathcal{V}_f$. Nous définissons d'abord les ambiguïtés dont l'opérateur est $\vee$, les syncrétismes d'opérateur $\wedge$. Les éléments de $\mathcal{D}_f$ sont des valeurs atomiques ou des ambiguïtés et des syncrétismes de des valeurs atomiques.

ensembles d'alternatives, $\mathcal{D}_f = \{a | a \in Alt_f\}$. D'une certaine manière, nous munissons le domaine des valeurs d'une structure de treillis.

$$\begin{aligned} Amb_f &::= \mathcal{V}_f \vee \mathcal{V}_f \mid \mathcal{V}_f \vee Amb_f \\ Syn_f &::= \mathcal{V}_f \wedge \mathcal{V}_f \mid \mathcal{V}_f \vee Syn_f \\ \mathcal{D}_f &::= Amb_f \mid Syn_f \mid \mathcal{V}_f \end{aligned}$$

Par exemple, si nous reprenons l'exemple 5.25, *kogo* est à la fois accusatif et génitif : le trait polarisé le modélisant sera $(case, \rightarrow, \{acc \wedge gen\})$, que nous écrirons par la suite $(case, \rightarrow, acc \wedge gen)$. De la même façon, le fait que *dacjie* puisse accepter un objet à l'accusatif ou au génitif (en fait le partitif est réalisé au génitif) sera traduit par le trait $(case, \leftarrow, acc \vee gen)$. Bien que les exemples ci-dessus ne font intervenir que les polarités $\rightarrow$ et $\leftarrow$, ces nouvelles valeurs sont autorisées pour tous les traits quelles que soient les polarités.

Bien sûr, le comportement de ces valeurs de traits complexes doit être spécifié. Définissons la relation $\sqsubseteq$ telle que $x \sqsubseteq y$ (x est plus spécifique que y) est définie par les règles suivantes :

$$\frac{}{a \sqsubseteq a} ax. \qquad \frac{a \sqsubseteq c}{a \sqsubseteq b \vee c} \vee_{r1} \qquad \frac{a \sqsubseteq b}{a \sqsubseteq b \vee c} \vee_{r2}$$

$$\frac{a \sqsubseteq c \quad b \sqsubseteq c}{a \vee b \sqsubseteq c} \vee_l \qquad \frac{a \sqsubseteq c}{a \wedge b \sqsubseteq c} \wedge_{l1} \qquad \frac{b \sqsubseteq c}{a \wedge b \sqsubseteq c} \wedge_{l2}$$

$$\frac{a \sqsubseteq b \quad a \sqsubseteq c}{a \sqsubseteq b \wedge c} \wedge_r$$

Nous pouvons voir ces règles comme un système logique proche du fragment additif de la logique linéaire présenté dans [JY99]. Cependant, d'après la grammaire ci-dessus, nous nous limiterons aux formules *plates*, c'est à dire de formules étant soit des atomes soit un connecteur et une liste d'atomes : il n'y pas de parenthésage.

Dans la section suivante nous aurons besoin de la notion logique d'interpolant. Si $v_1 \sqsubseteq v_2$ alors un interpolant est une formule v telle que $v_1 \sqsubseteq v$, $v \sqsubseteq v_2$ et v est construit uniquement à partir d'atomes communs à v_1 et v_2. L'interpolant le plus spécifique v de $v_1 \sqsubseteq v_2$ est un interpolant tel que quel que soit un autre interpolant u de $v_1 \sqsubseteq v_2$, $v \sqsubseteq u$.

Nous profitons du fait que les formules sont plates pour indiquer comment calculer facilement cet interpolant le plus spécifique. On considère que les formules sont écrites modulo l'idempotence :

- Pour une preuve de $\bigwedge(a_1, \ldots, a_m) \sqsubseteq \bigwedge(b_1 \ldots, b_n)$, on applique $m - n$ fois $\wedge_l$ pour que les atomes soient les mêmes des deux côtés. C'est la partie de la preuve qui suffit à calculer notre interpolant. Le reste de la preuve se fait par application de $\vee_r$ et axiome. L'interpolant recherché est donc $\bigwedge(b_1 \ldots, b_n)$
- Pour une preuve de $\bigwedge(a_1, \ldots, a_m) \sqsubseteq \bigvee(b_1 \ldots, b_n)$, on applique $\wedge_l$ pour faire disparaître les atomes qui n'apparaissent pas à droite. On obtient notre interpolant. Le reste de la preuve se fait en appliquant $\vee_r$ pour faire disparaître tous les atomes sauf un qui est présent à gauche. On termine par l'axiome.
- Pour une preuve de $\bigvee(a_1, \ldots, a_m) \sqsubseteq \bigwedge(b_1 \ldots, b_n)$, $m = n = 1$ et l'interpolant est donc a_1. Dans tous les autres cas, il n'y pas de preuve du séquent.
- Pour $\bigvee(a_1, \ldots, a_m) \sqsubseteq \bigvee(b_1 \ldots, b_n)$, on applique $n - m$ fois $\vee_r$ pour que les atomes soient communs à gauche et à droite. On obtient alors notre interpolant, qui est $\bigvee(a_1, \ldots, a_m)$. Le reste de la preuve se fait par $\vee_l$ suivi de $\vee_r$ puis axiome.

5.4.2 Révision de la notion de modèle

Notre extension du domaine des valeurs nous oblige à revoir la notion de modèle pour nos DAP. Plus précisément, quelle est la valeur dans l'arbre syntaxique d'un trait d'un nœud qui interprète des nœuds de DAP aux traits complexes ? C'est loin d'être une question triviale. [Sag02] utilise un système de traits similaire et se pose également la question du domaine de valeurs autorisées dans les structures finales après analyse (structures de traits typées, puisqu'il travaille en HPSG). Une des hypothèses communément admises en HPSG est qu'à la fin d'une dérivation, tous les traits doivent avoir une valeur atomique (une valeur maximale sur le treillis). Ce n'est plus possible avec la coordination telle qu'elle est modélisée dans [BJ95].

Dans notre définition des modèles, nous remplaçons la règle suivante :

pour tout nœud n de D, si $(f, p, \langle x \rangle)$ est un trait polarisé de la structure de traits associée à n , la structure de traits associée à $I(n)$ contient un trait (f, v) tel que $v \in \mathcal{V}_f$ et $v \in \Gamma.\langle x \rangle$.

par :

Soit $N = \{n_i\}_{1 \leq i \leq n}$ l'ensemble des nœuds de D munis d'un trait polarisé de la forme $(f, p_i, \langle x_i \rangle)$ qui ont pour image par la fonction d'interprétation $I(N)$. $I(N)$ est dans ce cas muni d'un trait (f, v). Les conditions suivantes doivent être vérifiées :

- *Il existe une permutation π qui permet de réordonner les nœuds n_i de N telle que $\Gamma.\langle x_{\pi(1)} \rangle \sqsubseteq \cdots \sqsubseteq \Gamma.\langle x_{\pi(n)} \rangle$*
- *Si le nœud n_i appartenant à N est muni du trait polarisé $(f, \rightarrow, \langle x_i \rangle)$, alors $\pi(i) = 1$*
- *Si le nœud n_i appartenant à N est muni du trait polarisé $(f, \leftarrow, \langle x_i \rangle)$, alors $\pi(i) = n$*
- *la valeur v dans le modèle est l'interpolant le plus spécifique de $\Gamma.\langle x_{\pi(1)} \rangle \sqsubseteq \Gamma.\langle x_{\pi(2)} \rangle$.* Si $N = \{n\}$ est un singleton alors $v = \Gamma.\langle x \rangle$

Notre opération de superposition ne réalise plus l'unification des valeurs de traits (du moins pour les traits polarisés $\rightarrow$ et $\leftarrow$).

Le nouveau calcul indique que l'on souhaite que le trait positif, c'est à dire la ressource fournie, prouve le trait négatif, la ressource attendue. En d'autres termes, il faut qu'il y ait assez de ressources présentes pour satisfaire les besoins.

L'utilisation de l'interpolant indique quant à elle que l'on cherche dans le trait positif la partie réellement demandée par le trait négatif, c'est à dire ici la partie de la valeur positive réellement utilisée dans la preuve de la valeur négative. Cette quantité est exprimée par l'interpolant le plus spécifique de la preuve. Pour les valeurs atomiques, on retrouve le même résultat qu'avec la définition précédente. En effet, on a toujours $a \subseteq a$ et l'interpolant est encore a.

Maintenant, que nous avons l'outillage nécessaire, reprenons l'exemple 5.25, et plus particulièrement la phrase (b). Dans l'arbre syntaxique de cette phrase il y a un nœud N qui interprète à la fois l'objet attendu par le verbe *dacjie* muni du trait $(case, \leftarrow, acc \vee gen)$ et l'objet fourni par le segment coordonné *wina i całą świnię* muni du trait $(case, \rightarrow, acc \vee gen)$. D'après notre nouvelle définition des traits de l'arbre syntaxique N contient le trait $(case, acc \vee gen)$. Notre analyse indique bien que l'objet a un cas ambigu. La phrase (a) exhibe une situation duale où $\vee$ et $\wedge$ sont intervertis.

Notre nouveau mécanisme est toujours capable de traiter les cas plus classiques. Pour la phrase *Dacjie wina*, l'objet attendu par le verbe voit son cas modélisé par $(case, \leftarrow$

$, acc \vee gen$), tandis que *wina* est à l'accusatif d'où un trait $(case, \rightarrow, acc)$. On a bien $acc \sqsubseteq acc \vee gen$ et l'interpolant est *acc*. Le nœud du modèle interprétant l'objet aura donc le trait $(case, acc)$: seul demeure le cas pertinent.

Pour la phrase (c), comme $acc \vee gen \not\sqsubseteq acc \wedge gen$, la phrase sera rejetée.

5.4.3 Modélisation de la coordination disparate

Reprenons l'exemple (5.25b). Nous voulons coordonner deux groupes nominaux, le premier étant au génitif et le second à l'accusatif. Quel est le cas du groupe nominal issu de la coordination ? Il n'est ni un génitif pur ni un accusatif pur mais il présente indéniablement les deux aspects en partie. Nous attribuons donc au nœud racine de la partie haute le trait $(case, \rightarrow, acc \vee gen)$. Nous pouvons modéliser ce phénomène avec la DAP de la figure 5.18.

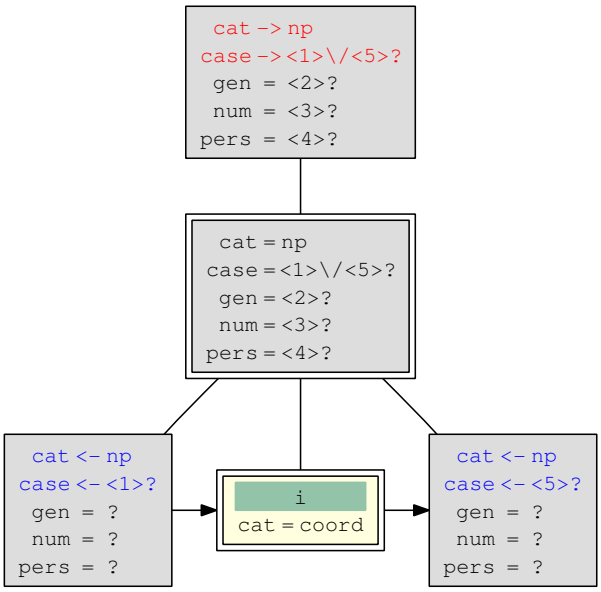


FIG. 5.18 – Une DAP associée à la conjonction de coordination *i* pour la coordination disparate de syntagmes nominaux.

La partie basse gauche est munie d'un trait $(case, \leftarrow, \langle x \rangle)$ et la partie basse droite $(case, \leftarrow, \langle y \rangle)$. Les deux conjoints sont donc autorisés à porter des cas différents. La racine est alors munie d'un trait $(case, \rightarrow, \langle x \rangle \vee \langle y \rangle)$. Si les deux conjoints ont des cas différents, la coordination des deux syntagmes nominaux ne sera ni au cas du conjoint gauche ni au cas du conjoint droit mais à un cas intermédiaire. S'ils sont au même cas, $a \vee a$ se simplifie en a et le cas sera le même que celui des conjoints.

De la même façon, dans l'exemple (5.25a), nous sommes en présence d'une coordination avec montée de nœud droit (en effet, l'objet canonique est à droite bien que nous soyons dans le cas d'une interrogative ou l'objet réel est à gauche). La coordination des deux segments se présente donc comme un complexe "sujet+verbe" attendant un groupe nominal complément qui doit satisfaire à la fois le cas réclamé par le premier verbe, accu-

satif, et par le second, génitif. Nous attribuons donc au nœud représentant l'objet de cette coordination le trait $(case, \leftarrow, acc \wedge gen)$. La figure 5.19 représente cette modélisation.

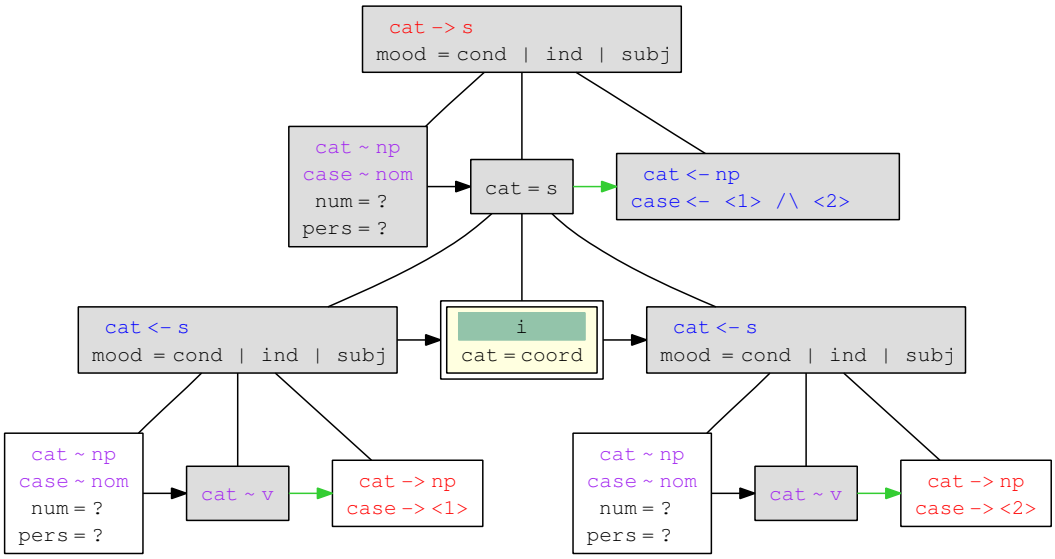


FIG. 5.19 – Une DAP associée à i pour la coordination de verbes avec montée de nœud droit.

Nous allons généraliser notre approche. Si le trait offert par la coordination est positif, c'est à dire que les traits offerts par chaque conjoints sont positifs, $(f, \rightarrow, a)$ et $(f, \rightarrow, b)$, alors ce trait a pour valeur $(f, \rightarrow, a \vee b)$. La coordination fournit un *compromis* entre les deux valeurs des conjoints. Bien sûr, l'opérateur $\vee$ étant idempotent, si a et b sont égaux on peut simplifier la valeur du résultat et nous retrouvons bien le cas des coordinations classiques.

Au contraire, si les conjoints offrent des traits négatifs $(f, \leftarrow, a)$ et $(f, \leftarrow, b)$ alors le trait résultat sera $(f, \leftarrow, a \wedge b)$. Intuitivement, on peut dire que les besoins s'additionnent. Par idempotence, on retrouve ici encore le cas classique.

Pour tenir compte de la coordination disparate, il faut donc pouvoir écrire des traits comme $(case, \rightarrow, \langle x \rangle \vee \langle y \rangle)$. Le système de traits et d'environnement que nous avons présenté au chapitre 1 ne le permet pas. L'extension du système de traits aux valeurs fonctionnelles devrait également résoudre ce problème.

5.4.4 Limites de la proposition

Cette extension des GI n'est pas encore implantée dans notre plate-forme `leopar`. Nous ne pouvons donc pas évaluer expérimentalement notre proposition. Cependant, d'un point de vue plus abstrait, on peut se demander si elle est intégrable à la grammaire actuelle.

Nous avons vu que nous pouvons coordonner des groupes nominaux de cas différents. Cette modélisation peut-elle s'étendre aux coordinations de segments de nature réellement différentes ? Nous devons faire face au problème suivant. Les traits de syntagmes de natures différentes ne diffèrent pas que sur les valeurs mais également sur les noms de traits. Pour fixer les idées, prenons les DAP associées au clitique *nous*, qui est objet direct sur la

figure 5.20 et complément d'attribution sur la figure 5.21. Les nœuds vides apportés dans les deux cas sont très différents. On voit alors mal comment *nous* pourrait exhiber le syncrétisme nécessaire à l'analyse d'une phrase comme *Jean nous a frappés et donné des coups*. On pourrait faire la même observation en regardant les DAP de *frappés* et *donné*.

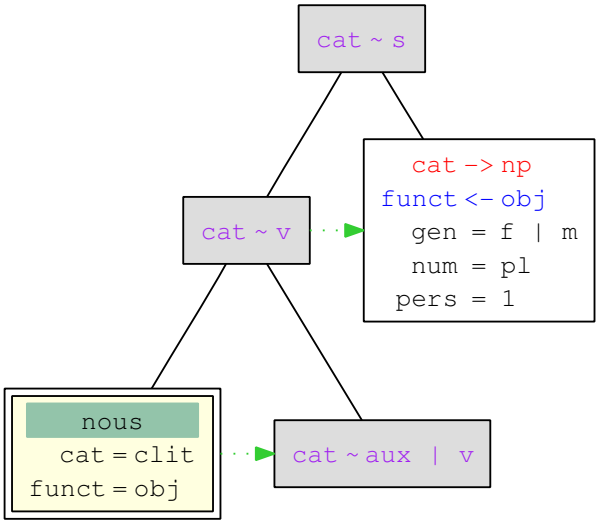


FIG. 5.20 – Une DAP pour *nous* objet direct.

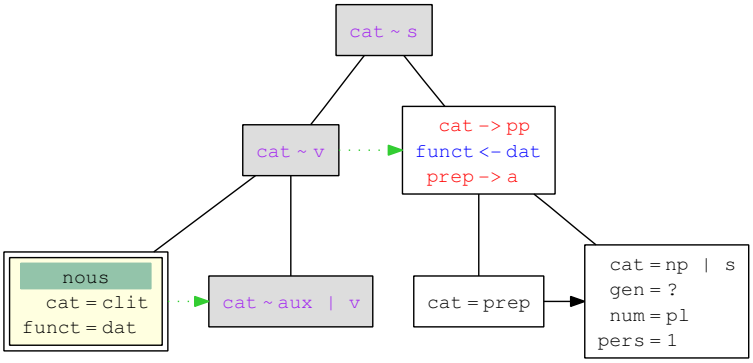


FIG. 5.21 – Une DAP pour *nous* complément d'attribution.

Notre modélisation est donc limitée par le fait que les deux DAP diffèrent plus que par la valeur d'un trait. Notre proposition ne peut rien dire sur le résultat. Pour que notre proposition s'intègre harmonieusement dans la GI du français de Guy Perrier, il faut que le nombre de catégories soit plus restreint et surtout que les noms de traits soient compatibles entre segments coordonnables. On peut par exemple imaginer que les groupes prépositionnels disparaissent au profit de groupes nominaux aux cas marqués, comme c'est le cas dans les propositions récentes des HPSG, par exemple les articles que nous évoquerons dans la section suivante.

5.5 Comparaison avec d'autres modélisations

Nous allons dans cette section comparer notre modélisation avec d'autres modélisations déjà proposées. Nous allons comparer notre approche avec celle développée par [Ste90] pour les GC combinatoires (GCC), par [BS04] pour les HPSG, par [Mou06] pour les HPSG également et finalement, par [MM96] pour les grammaires lexicales fonctionnelles (LFG).

5.5.1 Grammaires catégorielles combinatoires

La proposition généralement admise dans les GC est que la coordination s'analyse par factorisation [Ste90; Mor94]. Il y a deux raisons à cela :

- l'analyse des différents sous-phénomènes est uniforme, même pour le *gapping*,
- cette modélisation est relativement simple (sauf peut-être pour le *gapping*) et s'étend naturellement aux coordinations disparates.

Étant donné que notre modélisation des coordinations disparates est directement inspirée de la modélisation proposée pour les GC par [Bay96], nous ne revenons pas sur celle-ci dans cette section.

Définition

On rappelle que les GCC associent à chaque mot des catégories construites sur un ensemble de catégories atomiques $\{S, GN, GP, N\}$ et de symboles $\{\backslash, /\}$. Il existe aussi des combinateurs qui permettent de réécrire des suites de catégories. Si une suite de réécritures permet de passer de l'expression d'entrée à S, alors l'analyse réussit. En particulier, il existe deux combinateurs d'application, l'application avant (fa) et l'application arrière (ba). Nous suivons les auteurs des GCC dans leur notations aussi bien pour les symboles / et \ que pour les combinateurs que l'on note simplement au niveau des règles et jamais dans les expressions elles-mêmes. Nos deux premiers combinateurs se notent donc :

$$\begin{array}{llll} Y & X\backslash Y & \Rightarrow_{ba} & X \\ X/Y & Y & \Rightarrow_{fa} & X \end{array}$$

Ce sont les combinateurs de base. Nous allons maintenant voir d'autres combinateurs qui permettent de comprendre la modélisation de la coordination dans les GCC.

Le premier combinateur T permet de transformer un argument en fonction : c'est le combinateur de montée de types. Il en existe deux versions selon l'ordre d'application :

$$\begin{array}{lll} X & \Rightarrow_{<T} & Y\backslash(Y/X) \\ X & \Rightarrow_{>T} & Y/(Y\backslash X) \end{array}$$

Il nous faut également un combinateur pour composer les types. C'est l'équivalent du raisonnement hypothétique des GC de logique des types. Il s'appelle B pour des raisons historiques :

$$\begin{array}{llll} X/Y & Y/Z & \Rightarrow_{>B} & X/Z \\ Y\backslash Z & X\backslash Y & \Rightarrow_{<B} & X\backslash Z \end{array}$$

La coordination

On attribue à la conjonction de coordination un type *conj*. On se dote également d'un combinateur & :

$$X \quad conj \quad X \Rightarrow_{\&} X \tag{5.26}$$

Cette première présentation met les conjoints à la même distance de la conjonction. On retrouve ici l'origine de notre idée des DAP en trois parties. Les parties basses de notre modélisation sont les deux X en partie gauche de la règle & et la partie haute est représentée par le X en partie droite.

Une variante est également possible qui met le second conjoint plus proche de la conjonction. Il faut pour cela deux règles :

$$\begin{array}{llll} conj & X & \Rightarrow_{>\&} & [X]_{\&} \\ X & [X]_{\&} & \Rightarrow_{<\&} & X \end{array} \tag{5.27}$$

L'application de ces deux règles 5.27 permet de retrouver la règle 5.26. Mais ces deux règles offrent une granularité plus fine : en effet, entre l'application des deux règles la coordination est dans un état *non résolu* qui permettra de modéliser le gapping. Donc, pour les phénomènes simples, on peut appliquer indifféremment 5.26 ou les deux règles 5.27 consécutivement. Pour le gapping en revanche, on appliquera les deux règles 5.27 entrecoupées d'une transformation assez importante de la phrase.

Coordination de constituants Si la catégorie X est atomique, alors on a une coordination de constituants comme dans :

$$\dfrac{\dfrac{\text{Jean}}{GN} \quad \dfrac{\dfrac{\text{et}}{conj} \quad \dfrac{\text{Marie}}{GN}}{GN}\,\& \quad \dfrac{\text{dorme}}{S\backslash GN}}{S}\,ba$$

Coordination de non-constituants Si la catégorie X est complexe, on a une coordination de non-constituants. Il faut cependant utiliser les combinateurs T et B pour retrouver la même catégorie de part et d'autre de la conjonction de coordination :

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Jean}}{\text{GN}}}{S/(S\backslash GN)}\,{>T} \quad \dfrac{\text{aime}}{(S\backslash GN)/GN}}{S/GN}\,{>B} \quad \dfrac{\dfrac{\text{et}}{conj} \quad \dfrac{\dfrac{\dfrac{\text{Marie}}{\text{GN}}}{S/(S\backslash GN)}\,{>T} \quad \dfrac{\text{déteste}}{(S\backslash GN)/GN}}{S/GN}\,{>B}}{S/GN}\,\& \quad \dfrac{\text{Picasso}}{GN}}{S}\,{fa}$$

Coordination de séquences On ne distingue pas les coordinations de séquences d'arguments des autres types de coordinations, voir [Ste90; Dow88]. Qu'il y ait un argument ou plusieurs, l'analyse est semblable à celle que nous venons de présenter pour la montée de nœuds. Tout d'abord, on utilise la montée de type (T) et la composition (B) pour exhiber la symétrie des conjoints. Puis, on applique le combinateur de coordination. On peut tout de même constater que les types mis en jeu sont plus complexes. L'exemple de coordination de séquences est représenté sur la figure 5.5.1.0.0. On note $V = S\backslash GN$.

Avant de passer au *gapping*, il faut remarquer que les catégories que nous avons coordonnées dans cette dernière partie sont des catégories qui font apparaître la nature verbale des éléments mis en jeu dans la coordination. On peut donc dire qu'en creux, la modélisation par factorisation fait apparaître l'élément élidé.

Gapping Pour le *gapping*, il faut permettre de retrouver la symétrie qui autorise une modélisation par factorisation. [Ste90] donne une règle de décomposition lexicale qui permet d'aller chercher dans le premier conjoint la partie manquante au second conjoint.

Nous allons prendre comme exemple, la phrase *Joseph est informaticien et Charles mécanicien.* On décompose l'analyse en deux. Pour le premier conjoint l'analyse est simple :

$$\dfrac{\dfrac{\text{Joseph}}{\text{GN}} \quad \dfrac{\dfrac{\text{est}}{(S\backslash GN)/N} \quad \dfrac{\text{informaticien}}{N}}{S\backslash GN}\,{}_{fa}}{\text{S}}\,{}_{ba}$$

Ensuite, il nous faut pouvoir analyser le second conjoint comme un constituant auquel il manque quelque chose, en l'occurrence une phrase à laquelle il manque un verbe. Nous avons besoin d'un nouveau combinateur qui est une composition particulière :

$$[X/(S\backslash GN)]_{\&} \quad (S\backslash GN)\backslash Y \Rightarrow_{B_{\&}} [X\backslash Y]_{\&}$$

Ce combinateur indique que si l'on a un conjoint droit de type X qui réclame un verbe à sa droite et un verbe auquel il manque une dépendance de type Y à sa gauche, alors on peut réécrire le tout en un conjoint droit de type X à laquelle il manque un Y à sa gauche. Le verbe disparaît : il est élidé, c'est le trou verbal.

Analysons notre second conjoint. Il faut tout d'abord monter les types de *Charles* et *mécanicien* pour faire apparaître le verbe qui nous permettra d'appliquer notre nouveau combinateur.

$$\dfrac{\dfrac{\text{et}}{conj} \quad \dfrac{\dfrac{\text{Charles}}{GN}}{S/(S\backslash GN)}\,{}_{>T}}{[S/(S\backslash GN)]_{\&}}\,{}_{>\&} \quad \dfrac{\dfrac{\text{mécanicien}}{N}}{(S\backslash GN)\backslash((S\backslash GN)/N)}\,{}_{<T}}{[S\backslash((S\backslash GN)/N)]_{\&}}\,{}_{>B_{\&}}$$

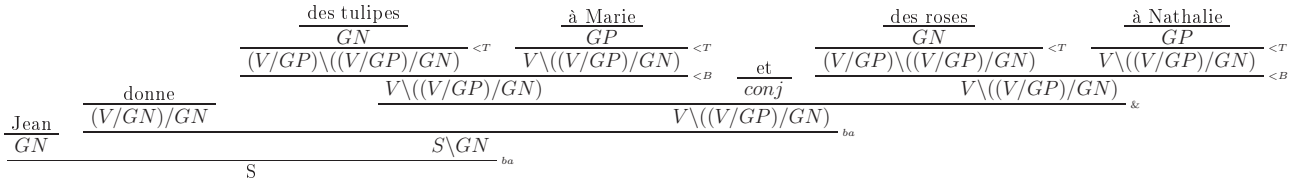


FIG. 5.22 – Coordination de séquences avec les grammaires catégorielles.

Le type de ce second conjoint indique qu'il a le même comportement syntaxique qu'une phrase à laquelle il manque la copule, qui est attendue à gauche.

Il est maintenant temps de joindre les deux analyses. Mais il faut tout d'abord exhiber la copule dans le premier conjoint. [Ste90] définit pour cela un nouveau combinateur qu'il appelle le combinateur de décomposition lexicale[34] :

$$S \Rightarrow_{dec} X \quad S\backslash X \text{ (où } X \text{ est une catégorie présente dans l'analyse)}$$

X est une catégorie qui apparaît dans l'analyse qui a donné S. C'est un combinateur atypique puisqu'il dépend du contexte dans lequel il est utilisé.

Ce combinateur est aussi très particulier puisque il y a plus de catégories dans la partie droite de la règle que dans la partie gauche. Il indique qu'une proposition peut être scindée en deux parties. La partie droite attend la partie gauche pour fournir une proposition par application arrière. Ce combinateur rend la règle d'application arrière inversible, à condition que la catégorie résultat de cette application soit S.

On peut maintenant analyser notre phrase :

$$\dfrac{(S\backslash GN)/N \qquad \dfrac{\text{Joseph est informaticien}}{S}\,{}_{dec} \qquad \dfrac{\text{et Charles mécanicien}}{[S\backslash((S\backslash GN)/N)]_{\&}}}{\dfrac{S\backslash((S\backslash GN)/N)}{S}\,{}_{<_{\&}}}\,{}_{ba}$$

L'analyse se termine par une application de la seconde règle du combinateur &, qui indique que les catégories des conjoints sont les mêmes. C'est à dire que l'on revient à un traitement de la coordination normal, après avoir fait en sorte de retrouver cette symétrie par l'utilisation des nouveaux combinateurs.

Le combinateur de décomposition est tout de même très particulier : il augmente la longueur des expressions et il doit s'aider du contexte de la phrase (sinon, il y a une infinité de possibilités pour le X). On imagine mal pouvoir généraliser ce type de combinateurs sans remettre en cause la complexité voire la décidabilité des GCC. D'ailleurs dans les articles récents, comme [SB07], cette analyse n'est pas présentée. En effet, on génère des phrases incorrectes, comme dans l'analyse suivante :

$$\dfrac{(S\backslash GN) \qquad \dfrac{\dfrac{\dfrac{\text{Jean}}{\text{GN}} \quad \dfrac{\text{dort}}{S\backslash GN}\,{}_{ba}}{S}\,{}_{dec} \qquad \dfrac{\dfrac{\dfrac{\text{et}}{conj} \quad \dfrac{\dfrac{\text{Marie}}{GN}}{S/(S\backslash GN)}\,{}_{>T}}{[S/(S\backslash GN)]_{\&}}\,{}_{>_{\&}} \qquad \dfrac{\text{tranquillement}}{(S\backslash GN)\backslash(S\backslash GN)}\,{}_{>B_{\&}}}{[S\backslash(S\backslash GN)]_{\&}}}{S\backslash(S\backslash GN)}\,{}_{<_{\&}}}{\text{S}}\,{}_{ba}$$

[34] il y a bien sûr un autre combinateur où \ est remplacé par /. Mais Steedman ne l'utilise pas.

Le problème peut sans doute se résoudre en ajoutant des modalités pour contrôler l'associativité des combinateurs. Il apparaît néanmoins que l'analyse du *gapping* n'est pas tout à fait satisfaisante. Elle est relativement complexe, elle met en jeu des combinateurs aux comportements inhabituels et elle provoque une surgénération.

Plus généralement, les GCC, comme les GC de logique des types, surgénèrent. On peut le constater sur les phénomènes comme la montée de nœud droit notamment. Par exemple, considérons l'exemple suivant :

(5.28) * [Jean aime] et [Marie va à] Paris.

Dans ce cas, les grammaires catégorielles attribuent le type S/GN aux deux *conjoints*. Les GC leur attribuent le même comportement syntaxique, c'est-à-dire celui d'un groupe auquel il manque un syntagme nominal à droite pour donner une proposition complète. Et cette phrase est donc acceptée.

Cela est dû au fait que pour être coordonnés, deux segments ont juste besoin d'être du même type. Par analogie avec les GI, on peut dire que deux segments sont coordonnables si et seulement si leurs interfaces sont les mêmes. Cette équivalence est trop permissive. Dans notre proposition, nous indiquons que cette condition est nécessaire mais elle n'est pas suffisante. Il faut également que les structures arborescentes (les DAP) se superposent. Les GC se munissent donc de modalités pour contrôler l'associativité et la commutativité des combinateurs.

Mais l'on peut s'interroger sur l'utilisation des modalités qui compliquent de plus en plus l'écriture et la compréhension de la grammaire. D'ailleurs les *nouvelles* GC, comme par exemple celles que présente [Mus03; de 01], s'opposent très nettement à l'emploi de modalités. Elles appliquent (simultanément ou en cascade) plusieurs niveau de composition : morphologique, syntagmatique, sémantique... de manière à ce que chaque partie soit simple à écrire et à maintenir bien sûr mais aussi pour distinguer les niveaux de composition clairement. Il est ainsi plus facile de distinguer le niveau des chaînes du niveau des arbres, qui est la principale cause de surgénération pour les GC.

5.5.2 HPSG et approche elliptique

Dans cette section, nous présentons l'approche de [BS04]. Les auteurs définissent un cadre général pour modéliser la coordination de constituants, de non constituants avec montée de nœuds, les séquence d'arguments et enfin les coordinations disparates. Ils s'inscrivent dans une modélisation elliptique du phénomène.

Les auteurs adressent deux critiques à l'approche présentée plus haut :

1. Les coordinations de non-constituants et les dépendances non-bornées sont modélisées de le même façon dans les GC. Notamment, les deux nécessitent les combinateurs pour la montée de type et la composition.
2. Les GC doivent supposer que les symboles de ponctuation sont traités comme des conjonctions de coordination.

Le premier point indique que l'on ne peut pas, dans les GCC, modéliser les coordinations de non-constituants sans modéliser les dépendances non-bornées, et réciproquement. Or les auteurs donnent un certain nombre de langues, comme le Hausa, dans lesquelles

les constructions faisant intervenir des dépendances non-bornées comme des extrapositions, des topicalisations, des constructions clivées existent, sans que les coordinations de non-constituants soient autorisées. Ces langues sont impossibles à modéliser dans les GC.

Le second point, fondé sur des exemples, indique que le fait de donner aux symboles de ponctuation, comme les virgules ou les points de suspension, le statut de conjonctions de coordination pour modéliser certaines coordinations de non-constituants entraîne une surgénération.

La thèse des auteurs est qu'il n'existe que des coordinations de constituants, qui peuvent parfois mettre en jeu des phénomènes d'ellipse. Dans la phrase suivante, on coordonne deux propositions. La première est défective : il y manque l'objet du verbe.

(5.29) [Jean aime ~~Picasso~~] mais [Marie déteste Picasso].

De la même façon, il n'existe pas dans l'approche elliptique de coordination disparate : ici encore, l'on coordonne deux propositions et non pas un nom et un adjectif.

(5.30) [Jean est âpre à la tâche] et [~~Jean est~~ bon bûcheron].

Un schéma général Une coordination de n conjoints peut se représenter schématiquement par une chaîne $AB_nB_{n-1}\dots B_2CB_1D$, où C est une conjonction et A et D sont des parties élidées (qui peuvent être vides). Cette chaîne doit se réécrire pour faire apparaître clairement les parties élidées pour chaque conjoint. Elle prend alors la forme suivante :

$$(AB_nD)C(AB_{n-1}D)C\dots(AB_2D)C(AB_1D)$$

Dans l'expression ci-dessus les AB_iD doivent tous être des constituants de même nature.

De plus, les auteurs peuvent caractériser les phénomènes de coordinations selon A et D :

- Si $A = \epsilon$ et $D = \epsilon$ alors, on a une coordination de constituants.

$$\begin{array}{rl} & \epsilon \text{ Jean Paul et Marie } \epsilon \\ \Rightarrow & (\epsilon \text{ Jean } \epsilon) \text{ et } (\epsilon \text{ Paul } \epsilon) \text{ et } (\epsilon \text{ Marie } \epsilon) \end{array}$$

- Si $D = \epsilon$ et $A \neq \epsilon$, alors on a affaire à une coordination de séquence :

$$\begin{array}{rl} & \text{(Jean donne) des tulipes à Marie et des roses à Nathalie } \epsilon \\ \Rightarrow & \text{(Jean donne des tulipes à Marie } \epsilon) \text{ et (Jean donne des roses à Nathalie } \epsilon) \end{array}$$

- Si $D \neq \epsilon$ et $A = \epsilon$, alors on est en présence d'une montée de nœud droit :

$$\begin{array}{rl} & \epsilon \text{ Jean aime mais Marie déteste (Picasso)} \\ \Rightarrow & (\epsilon \text{ Jean aime Picasso }) \text{ et } (\epsilon \text{ Marie déteste Picasso}) \end{array}$$

Modélisation Nous ne prétendons pas ici donner une présentation des HPSG. Nous renvoyons le lecteur au très pédagogique [SWB03]. Nous supposons que le lecteur possède quelques connaissances de ce formalisme.

Les conjonctions de coordination sont traitées comme des marqueurs qui introduisent une nouvelle marque CRD qui est positionnée à + si le syntagme commence par une conjonction de coordination et à − sinon. L'entrée lexicale représentée sur la figure 5.23 est celle d'une conjonction de coordination. Elle doit se combiner avec un élément marqué

CRD − pour donner une structure qui contient les mêmes marques mais CRD +. On distingue cette marque CRD de la marque souvent prépositionnelle MARKING puisque les deux peuvent marquer un constituant (par exemple *et à la plage*).

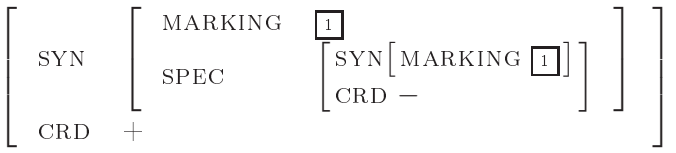


FIG. 5.23 – Entrée lexicale pour une conjonction de coordination

Il faut également considérer que les constructions à partir de entrées lexicales propagent notre nouvelle marque.

On peut maintenant donner la règle qui rend compte du schéma de réécriture des coordinations. Elle est présentée sur la figure 5.24.

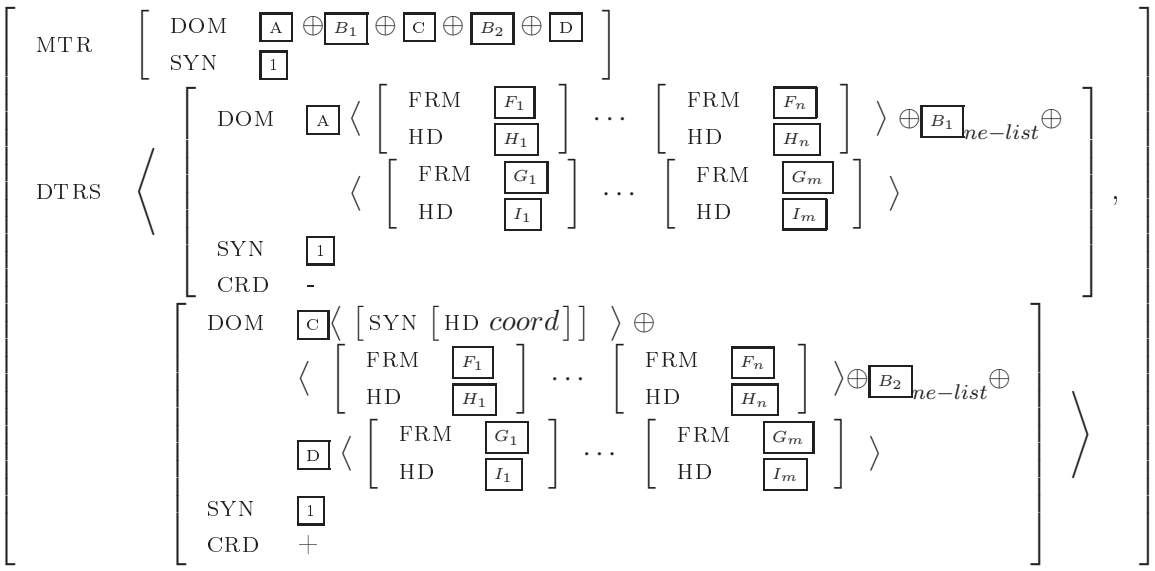


FIG. 5.24 – Règle de construction d'un segment coordonné

Le domaine de la structure mère est exactement la chaîne d'entrée de notre schéma. Le trait SYN coindexé entre la mère et les deux filles indique que l'on coordonne deux éléments de même nature que leur coordination. Ils ont notamment les marques en commun, par l'intermédiaire du trait MARKING. On comprend maintenant pourquoi le trait CRD est distinct des autres marques. Il est nécessaire que la première fille soit marquée CRD - et la seconde CRD +. La structure mère est libre : elle peut être marquée CRD - ou +. De plus, la coordination C est optionnelle (liste à 0 ou 1 élément). Ceci permet d'emboîter de telles structures binaires pour modéliser la coordination n-aire.

Les structures des deux structures filles nous donne la chaîne du schéma après réécriture.

Le premier conjoint est constitué de la liste d'éléments A, qui peut être vide, puis de la liste d'éléments B_1 qui elle ne doit pas être vide, suivie d'une liste potentiellement vide d'éléments qui n'appartiennent pas au domaine de la structure mère. Cette dernière

liste n'a pas de réalité dans la forme de surface. Elle représente donc la partie élidée qui correspond à D et qui doit être ajoutée au premier conjoint.

On remarque que chaque élément de la liste d'éléments élidée du premier conjoint doit s'unifier avec un élément de D pour le trait HD (la partie syntaxique) et FRM (la partie phonologique). Les autres traits sont libres. C'est ainsi que l'on duplique le matériel manquant. Et il en sera de même pour les éléments de la partie élidée du second conjoint qui doivent s'unifier pour certains traits avec des éléments de A.

Le second conjoint commence par la conjonction de coordination suivie d'une liste d'éléments élidés de la structure mère puis la liste B_2 des éléments du second conjoint tels qu'ils apparaissent dans la chaîne d'entrée. Le second conjoint se termine par la liste d'éléments D qui peut être vide.

Limites de l'approche Cette approche bien qu'elle donne satisfaction dans de nombreux cas possède quelques défauts. Elle peine à expliquer pourquoi certaines formes contractées (avant réécriture) sont acceptées tandis que les formes étendues (après réécriture du schéma) ne le sont pas.

(5.31) (a) J'ai donné un DVD à Jean et non une cassette à Pierre.
(b) * J'ai donné un DVD à Jean et j'ai donné non une cassette à Pierre.

Un autre problème vient de l'analyse des adverbes restrictifs :

(5.32) Jean donne seulement dix roses blanches à Pierre et vingt roses rouges à Paul.

Selon le schéma que nous avons vu, les deux réécritures possibles sont :

(5.33) (a) [Jean donne seulement dix roses blanches à Pierre] et [Jean donne seulement vingt roses rouges à Paul.]
(b) [Jean donne seulement dix roses blanches à Pierre] et [Jean donne vingt roses rouges à Paul.]

Aucune de ces deux réécritures ne donne le sens désiré, c'est-à-dire celui où l'adverbe *seulement* restreint les deux conjoints à la fois.

Il existe aussi un problème calculatoire. Il faut considérer toutes les possibilités de suites d'éléments A et D et donc énumérer tous les contextes à gauches et à droite des conjoints.

Enfin, l'argument que les auteurs donnent pour éliminer l'approche par factorisation des GC est le suivant : la phrase 5.34 requiert dans l'approche de donner à la virgule le statut de conjonction de coordination. On peut alors générer des phrases sans véritable conjonction de coordination qui exhibent le phénomènes de coordination. Les auteurs en concluent que cette modélisation amène à surgénérer. Pourtant, [Mou07] indique que de telles phrases existent en français, essentiellement dans le contexte du style journalistique.

(5.34) Jan travels to Paris on Monday, to London on Tuesday and will fly to Tokyo on Sunday.

Leur proposition est d'analyser la phrase de la façon suivante :

(5.35) Jan [[travels to Paris on Monday] , [[~~travels~~ to London on Tuesday] and [will fly to Tokyo on Sunday]]].

Malheureusement, cette analyse ne correspond pas à la structure obtenue par leur schéma de réécriture. Si *travels* est assimilé à A alors il doit être répété dans les trois conjoints et donc également dans le dernier.

En conclusion, cette approche grâce à sons schéma de réécriture simple permet de donner une représentation unifiée des coordinations de constituants et de certaines coordinations de non-constituants (mis à part le *gapping*). En revanche, elle souffre de quelques lacunes, au niveau linguistique et au niveau informatique. D'ailleurs, aucune implantation de cette modélisation n'existe pour la simple raison que ce type de règle est impossible à écrire avec les systèmes de type LKB [Cop99]. En particulier, c'est la coindexation des F_i, G_j H_k et I_l dans la règle de construction qui est difficile à exprimer.

5.5.3 HPSG et approche par factorisation

[Mou07] propose d'accommoder la vision par factorisation de la coordination et les HPSG. Comme précédemment, les conjonctions de coordination sont vues comme des marqueurs qui ajoutent une marque CRD. Le domaine de cette marque est étendue : à la place de $\{-,+\}$, on a maintenant $\{nil, et, ou, mais, ni, ainsique\}$ selon la conjonction utilisée. Les mots qui ne sont pas des conjonctions sont tous marqués CRD *nil*. L'entrée lexicale pour les conjonctions de coordinations est donc similaire à celle que nous avons présentée sur la figure **??**

Il utilise la règle suivante :

$$\left[\begin{array}{ll}
\text{SYN} & \left[\begin{array}{ll}
\text{CRD} & nil \\
\text{HD} & \boxed{H_0}, \boxed{H_0} \leq \boxed{H_1}, \ldots, \boxed{H_0} \leq \boxed{H_n} \\
\text{MARKING} & \boxed{M_0}, \boxed{M_0} \leq \boxed{M_1}, \ldots, \boxed{M_0} \leq \boxed{M_n} \\
\text{VALENCE} & \boxed{V} \\
\text{SLASH} & \boxed{S}
\end{array}\right] \\
\text{DTR} & \left\langle \left[\begin{array}{ll}
\text{HD} & \boxed{H_1} \\
\text{MARKING} & \boxed{M_1} \\
\text{VALENCE} & \boxed{V} \\
\text{SLASH} & \boxed{S}
\end{array}\right], \ldots, \left[\begin{array}{ll}
\text{CRD} & \neg nil \\
\text{HD} & \boxed{H_n} \\
\text{MARKING} & \boxed{M_n} \\
\text{VALENCE} & \boxed{V} \\
\text{SLASH} & \boxed{S}
\end{array}\right] \right\rangle
\end{array}\right]$$

Cette structure de traits indique que la structure mère à la même valence et les mêmes arguments manquants (trait SLASH) que les structures filles qui sont identiques pour ces traits. Pour les traits CRD et MARKING, les traits entre les structures filles peuvent varier et la structure mère *généralise* les valeurs de ces deux traits. Cette opération de généralisation est définie dans [Sag02]. Elle donne une structure de treillis aux valeurs que peut prendre un trait et transpose les opérateurs $\wedge$ et $\vee$ pour la coordination disparate en HPSG.

Plus précisément, la généralisation de a et de b donne $a \vee b$. En revanche, nous avons cacher une partie de la difficulté pour les traits VALENCE et SLASH. Ces traits sont coindexés entre la mère et les filles, ce qui revient à dire que l'on prend la valeur la plus spécifique qui puisse convenir à toutes les structures de traits. Cela revient à opérer $\wedge$ sur toutes les valeurs par défaut des filles. Ainsi, on peut établir :

(5.36) (a) âpre à la tâche $\left[\text{HD } adj\right]$
(b) bon bûcheron $\left[\text{HD } nom\right]$
(c) âpre à la tâche et bon bûcheron $\left[\text{HD } adj \vee nom\right]$

(d) a frappé $[\textsc{comps}\ [\textsc{case}\ \mathit{acc}]]$
(e) a donné des coups $[\textsc{comps}\ [\textsc{case}\ \mathit{dat}]]$
(f) a frappé et donné des coups $[\textsc{comps}\ [\textsc{case}\ \mathit{acc} \wedge \mathit{dat}]]$

Avec cette règle, [Mou07] modélise la coordination de constituants, la coordination disparate, et la montée de nœuds.

Pour les coordinations de séquences, l'auteur développe la notion de *cluster* de syntagmes. Il énonce un première règle qui prend une liste de syntagmes pour en faire un cluster. Un cluster est une liste de syntagmes qui ne peut pas être argument d'une autre construction de type nom ou verbe. Ces clusters peuvent tout de même se coordonner.

Puis un seconde règle va prendre un prédicat et une coordination de cluster. Elle va séquencer la coordination en cluster et pour chaque cluster va le convertir en une liste d'arguments. Elle va ensuite *distribuer* le prédicat sur chaque liste d'arguments.

Cette modélisation et celle que nous proposons sont très proches, bien qu'elles s'appliquent dans des formalismes assez éloignés et qu'elles diffèrent donc dans la machinerie mise en place. En revanche, le fait de pouvoir manipuler des listes et donc des coordinations n-aires directement permet d'aller plus loin que notre proposition pour les séquences de longueur arbitraires que nous ne pouvons pas modéliser dans les GI. On peut tout de même faire à cette approche le même reproche qu'à la précédente : les règles données sont pour modéliser la coordination sont complexes et ne sont pas modélisables dans les analyseurs HPSG actuels.

5.5.4 LFG

Pour les grammaires lexicales fonctionnelles, nous nous basons sur les travaux de [KM88; MM96]. Comme pour les HPSG, nous supposons que le lecteur est familier avec le formalisme, défini dans [Bre00] par exemple.

Coordination de constituants et coordination lexicale

Dans ce formalisme, on modélise généralement la coordination grâce à l'approche par factorisation. [KM88] modélisent la coordination de constituants[35] dans les LFG. Par exemple, pour coordonner deux propositions indépendantes, il suffit d'ajouter la règle :

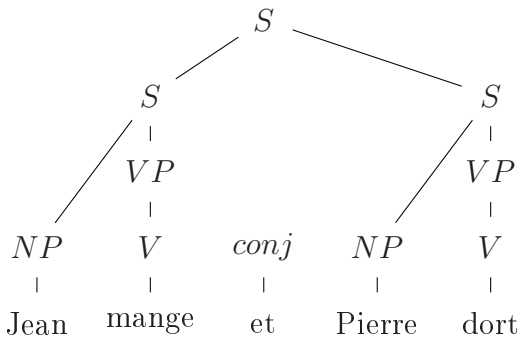
$$\begin{array}{lcccc} S & \rightarrow & S & \mathit{conj} & S \\ & & \downarrow \in \uparrow & & \downarrow \in \uparrow \end{array}$$

Cette règle indique qu'une proposition indépendante peut être construite à partir de deux propositions indépendantes reliées par une conjonction de coordination. La f-structure associée à cette nouvelle proposition est composée de deux éléments : les f-structures des propositions coordonnées.

(5.37) Jean mange et Pierre dort.

Par exemple, pour la phrase (5.37), on a la c-structure et la f-structure suivantes :

[35] et la coordination lexicale dans le cas où le constituant peut être construit à partir d'un mot, tel le constituant V.



$$\begin{bmatrix} \begin{bmatrix} \text{PRED} & \text{manger} \\ \text{SUBJ} & \text{jean} \end{bmatrix} \\ \\ \begin{bmatrix} \text{PRED} & \text{dormir} \\ \text{SUBJ} & \text{pierre} \end{bmatrix} \end{bmatrix}$$

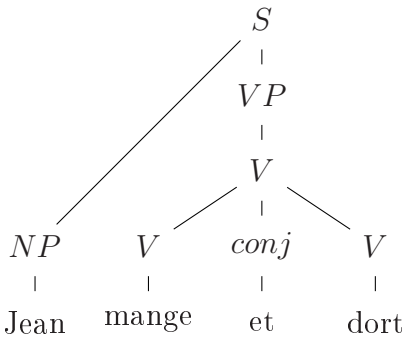
Pour des coordinations de verbes, le règle est similaire :

$$\begin{array}{lcccc} V & \rightarrow & V & conj & V \\ & & \downarrow \in \uparrow & & \downarrow \in \uparrow \end{array}$$

Mais il faut de cas que les deux verbes partagent leurs arguments. La f-structure de la coordinations doit *généraliser* les f-structures associées aux conjoints. Comme dans l'exemple suivant :

(5.38) Jean mange et dort.

On a les structures d'analyse suivante :



$$\begin{bmatrix} \begin{bmatrix} \text{PRED} & \text{manger} \\ \text{SUBJ} & \text{jean} \end{bmatrix} \\ \\ \begin{bmatrix} \text{PRED} & \text{dormir} \\ \text{SUBJ} & \end{bmatrix} \end{bmatrix}$$

Coordination de non-constituants

Pour modéliser les montées de nœuds droits et les coordinations de séquences au niveau de la c-structure, il ne serait pas raisonnable d'écrire toutes les règles de production. [MM96] suggèrent de dériver les règles de production pour ces phénomènes à partir des règles de la grammaire standard et de la symétrie qui existe entre les conjoints.

Pour modéliser les montées de nœuds, l'idée est de pouvoir décrire des constituants partiellement analysés, de donner un statut à un *début* ou à une *fin* de constituant. Cette définition se fait à partir des règles de formation des c-structures. Comme les parties droites des règles de formation des constituants sont représentables par des automates à états finis déterministes (nous en donnerons une définition exacte dans la section 6.2), les auteurs proposent de dire que l'on peut coordonner des conjoints s'ils correspondent aux mêmes points d'arrêts dans des constituants, c'est-à-dire aux mêmes états dans des chemins de ces automates.

Ces non-constituants sont écrits $s_1 X s_2$, où X est un symbole non terminal de la grammaire et les s_i sont des points d'arrêts, des états, dans les parties droites de règles de la grammaire dont la partie gauche est X. Si s_1 est l'état initial où si s_2 est l'état final, ils sont omis. Ils ajoutent donc la métarègle suivante :

$$XP \rightarrow XP s_1 \; [s_1 XP s_2 \; conj \; s_1 XP s_2] \; s_2 XP \tag{5.39}$$

Cette règle signifie qu'un constituant XP est formé d'un début de constituant XP, c'est-à-dire d'un fragment de partie droite d'une règle qui commence à l'état initial et finit à l'état s_1, d'une coordination de non-constituants correspondant des fragments de règles qui vont d'un état s_1 à un état s_2, et finalement d'un fragment de chemin qui commence à l'état s_2 et finit à l'état final.

Pour revenir à notre approche, la conjonction peut fournir aux conjoints incomplets les dépendances qui leur dont défaut si ces dépendances apparaissent à gauche ou à droite du groupe coordonné.

Par exemple, pour la phrase suivante :

(5.40) Jean donne toujours des roses à Marie et des tulipes à Nathalie.

et la grammaire :

$$\begin{array}{rcl} \text{S} & \rightarrow & \text{NP VP} \\ \text{VP} & \rightarrow & \text{V Adv}^* \text{ NP PP} \end{array}$$

La partie droite de la seconde règle peut se représenter sous la forme de l'automate de la figure 5.25.

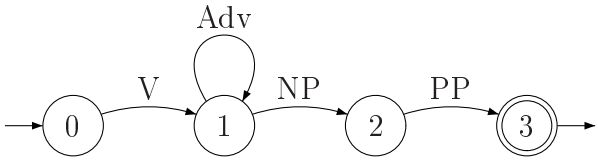


FIG. 5.25 – automate correspondant à la seconde règle de notre grammaire.

On obtient la c-structure de la figure 5.26, où x est l'état 1 de notre automate.

Au niveau de la f-structure, le principe de généralisation est maintenu.

Nous avons volontairement simplifier l'approche de [MM96]. En effet, dans la règle 5.39, au lieu d'avoir un état s_1, c'est une pile d'états que l'on peut avoir puisque des non-constituants peuvent être enchâssés les uns dans les autres. Mais le principe reste évidemment le même.

Cette modélisation se fonde elle aussi sur la symétrie qui peut exister entre les conjoints. En ce sens, elle partage de nombreux points communs avec notre proposition. La conjonction est chargée de factoriser les dépendances des conjoints qui doivent être fournies par le contexte.

La forme arborescente de la c-structure permet d'éviter la surgénération des GC. On a donc à la fois une notion de valence avec les états qui décorent les non-terminaux et qui indiquent des dépendances non-satisfaites mais sans préciser quel chemin exact a été suivi pour arriver à cette état. On a ici une information que l'on retrouve dans notre

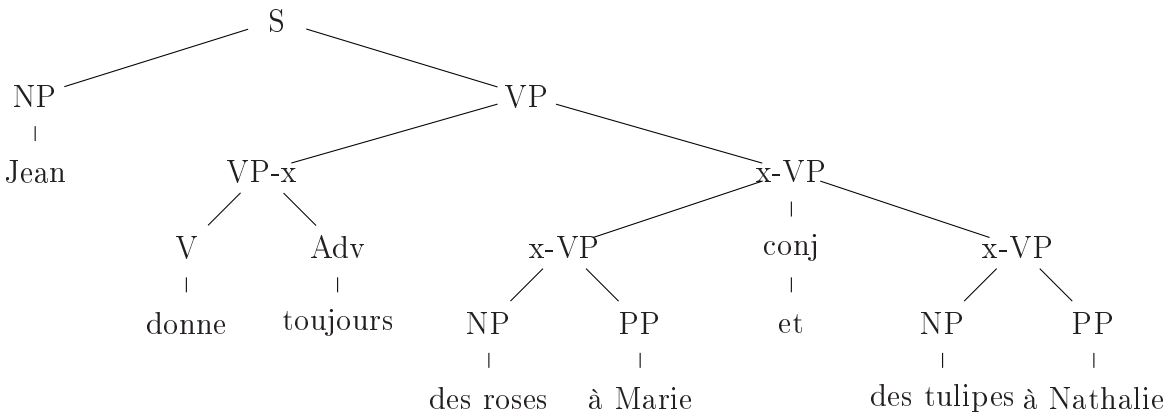


FIG. 5.26 – c-structure pour une coordination de séquences

modélisation avec la notion d'interface. Cependant dans notre modélisation, le système de polarité et des relations sous-spécifiées permet de donner des informations plus fines sur les dépendances et la structures des conjoints.

On peut tout de même formuler deux reproches à l'encontre de cette approche. Tout d'abord, il a fallu étendre le formalisme de départ. Dans la présentation classique des LFG il n'est fait aucune mention spécifique aux parties droites des règles de construction des c-structures. Ici, elles sont détournées de leur rôle originel pour construire de nouvelles règles dynamiquement. Deuxièmement, les f-structures ont maintenant une interprétation ensembliste, ce qui n'était pas défini par la présentation originale des LFG. Cette possibilité complique grandement l'implantation des analyseurs pour les LFG.

5.6 Implantation de la grammaire

Dans cette partie, nous décrivons l'organisation de la grammaire de la coordination que nous proposons. Cette grammaire est implantée et utilisée par notre analyseur `leopar`[36]. C'est aussi un exemple d'implantation d'une grammaire de la coordination construite manuellement visant une couverture large. Cette implantation s'appuie sur l'outil XMG que nous avons présenté au chapitre 3 et la méthodologie décrite dans [Cra05]. En particulier, on retrouve des classes qui produisent des fragments réutilisables qui correspondent à des fonctions particulières. C'est le cas des classes qui modélisent les dépendances pour les coordinations de non-constituants. Ensuite d'autres classes qui correspondent à des structures complètes agrègent ces premiers fragments. Enfin des alternances autorisent à grouper des constructions dans des familles.

[36] `http ://www.loria.fr/equipes/calligramme/leopar`

5.6.1 Organisation des classes

Selon la méthodologie définie par [Cra05] pour XMG, le grammairien commence par définir des fragments de DAP réutilisables qui se comportent comme des modules que d'autres fragments peuvent importer de manière à construire des fragments plus importants qui finalement forment des DAP complètes. La réutilisation des fragments s'apparente à la programmation orientée objet avec une hiérarchie d'héritage multiple.

On l'a vu dans la section 5.3, les DAP associées aux conjonctions de coordination proviennent toutes, sauf pour le phénomène de trou verbal et les coordinations de séquences qui ont besoin de plus de contextes, d'un même patron. Ces DAP sont constituées de 3 parties : deux parties basses qui saturent les segments conjoints et une partie haute qui interagit avec le reste de la phrase. De plus, partie haute et partie basse sont duales l'une de l'autre.

Ces parties sont
- soit composées d'un unique nœud pour les coordinations de constituants simples,
- soit deux nœuds pour les modificateurs,
- soit, dans le cas des non-constituants avec montée de nœud, il y a un nœud représentant le constituants défectif et d'autres nœuds représentants les dépendances non-satisfaites qui le rendent justement défectif.

On va donc organiser notre métagrammaire selon trois aspects. Le premier décrira la forme générale des DAP pour les conjonctions de coordination. La seconde donnera des informations sur les nœuds racines des 3 parties et sur les partages des traits dans chacune des parties pour chaque type de constituant. Enfin le troisième axe, définira les dépendances non-satisfaites et les agencera sur les DAP de constituants simples. Un fragment de l'organisation est représenté par la figure 5.27.

5.6.2 Forme générale des DAP

Les deux premières classes de la hiérarchie sont `Prototype_Coordination` et `Prototype_Coordination_Polarisee` qui décrivent la forme générale d'une DAP de conjonction de coordination. La figure 5.28 présente la DAP définie par la première de ces classes. On reconnaît la forme en 3 parties.

La taille des parties n'est pas encore définie. Leur contenu n'est pas précisé si ce n'est que les catégories des racines des parties sont coindicées. La différence entre la version de base et la version polarisée est que ces racines ont dans le premier cas des traits de polarité virtuelle et dans le second des traits polarisés $\rightarrow$ pour la racine de la partie haute et $\leftarrow$ pour les parties basses. Le fait d'avoir deux classes est principalement dû à une limitation de XMG qui ne permet pas de manipuler les polarités directement mais uniquement comme des décorations d'un couple trait/valeur.

Les classes qui héritent de ces deux premières classes spécifient la forme et instancient les structures de traits des nœuds. On peut distinguer les coordinations de modificateurs qui vont hériter de la première classe et les autres qui vont hériter de la deuxième.

En plus de la hiérarchie d'héritage commençant par les deux classes précédentes, on retrouve une seconde hiérarchie pour les montées de nœud. À chaque fois, d'après notre modélisation, chaque montée de nœud est représentée par trois nœuds : un nœud en partie

haute qui représente *stricto sensu* la dépendance non-satisfaite qui sera apportée par le contexte de la phrase et un nœud dans chaque partie basse qui satisfait les dépendances des conjoints. De plus les traits polarisés sont inversés entre partie haute et partie basse.

5.6.3 Coordination nominale

Nous allons ici détailler la partie de la métagrammaire portant sur la coordination de syntagmes nominaux avec éventuellement montée de nœud de manière à bien comprendre l'emploi de ces deux hiérarchies. La méthodologie est la même pour les autres types de coordination. La figure 5.27 rend compte de cette organisation.

Une hiérarchie pour un type de conjoint `T` commence toujours par un fragment `Prototype_Coordination_T` qui fixe les traits et les valeurs communs aux DAP de ce type. Pour les syntagmes nominaux, `Prototype_Coordination_SN` impose les traits $cat \rightarrow np$ et $funct \leftarrow ?$ signifiant ainsi que les coordinations sont des groupes nominaux qui doivent recevoir une fonction grammaticale (quelconque). Ce dernier trait est coindexé entre les conjoints et leur conjonction.

Après cette étape, les hiérarchies de coordination diffèrent sensiblement. Pour les syntagmes nominaux, l'étape suivante, décrite pas la classe `Accord_Coordination_SN`, consiste à ajouter les informations d'accord, principalement pour la personne, le genre et le nombre des conjoints et de la coordination. Cette étape est optionnelle et permet de générer deux grammaires. On veut pouvoir générer une grammaire *simplifiée* de la coordination à comparer avec la grammaire complète, notamment pour mesurer la couverture et les performances. On verra dans le chapitre 6 que le nombre important de DAP associées aux conjonctions est un handicap assez lourd pour l'étape d'analyse syntaxique, notamment dans le cas des coordinations nominales puisque les traits d'accords sont neutres et n'interviendront pas dans le processus de filtrage. Il est donc important d'avoir une *petite* grammaire pour pouvoir tester rapidement une extension à notre modélisation que nous pouvons ensuite étendre en tenant compte des traits d'accord dans un second temps.

Ensuite, soit on évalue la classe `Coordination_SN` qui génère les DAP de syntagmes nominaux complets simples soit on va croiser la classe `Accord_Coordination_SN` avec des classes modélisant des dépendances non-satisfaites (pour les cas de montée de nœud). Ces classes héritent toutes de `Montée_Noeud_Droit` qui indique l'existence de nœuds supplémentaires à droite dans les DAP associées et spécifient le type de dépendance (préposition, complémenteur...). Par exemple pour la coordination *«[la femme et la fille] de Jean»*, la DAP associée à la conjonction est issue de la classe `Coordination_SN_Prep` qui croise les classes `Accord_Coordination_SN` et `Montée_Prep`. Pour la coordination *«[l'acceptation ou le refus] de l'entrée aux mineurs»*, `Coordination_NP_X` est croisé avec deux instances de `Montée_Prep`.

La classe `Montée_Nœud_Droit` hérite quant à elle de `Montée_Nœud` la classe abstraite qui déclare l'existence des nœuds porteurs des dépendances supplémentaires non-satisfaites. Les montées de nœuds dans le cas des coordinations de propositions en héritent également.

5.6.4 Coordination verbale

Pour les coordinations de propositions, la démarche est la même que pour les groupes nominaux. On note cependant deux différences :

- la nature et la fonction des dépendances non-satisfaites,
- le fait que ces dépendances non-satisfaites peuvent être venir de la droite comme de la gauche. On autorise les deux types de montées de nœuds.

Concrètement, les dépendances peuvent être des syntagmes nominaux, des syntagmes prépositionnels ou des propositions (avec éventuellement complémenteurs). En plus des fonctions qui sont autorisées pour les coordinations nominales, compléments obliques (attribution...), il faut ici considérer les fonctions sujet et objet. Le sujet est une dépendance qui est attendue à gauche du verbe tandis que les autres dépendances sont attendues à droite.

5.6.5 Séquences et trou verbal

Les DAP pour les phénomènes de coordination de séquences et de coordination avec trou verbal sont assez différentes des autres, coordinations de simples constituants ou de non-constituants avec montée de nœud. Intuitivement, on duplique la partie commune, le noyau verbal aussi bien pour les séquences que pour le trou verbal. C'est pourquoi ces deux phénomènes ont leur propre hiérarchie, présentée sur la figure 5.29.

La classe `Coordination_Duplication` décrit la forme générale des coordinations avec duplication. C'est à dire, la partie dupliquée et les trois parties habituelles

On réutilise la hiérarchie des dépendances déjà définie que l'on croise à une nouvelle hiérarchie présentant les 3 parties. Dans ce cas, la plupart des nœuds de la partie haute est virtuelle puisque leur rôle est de donner des informations sur le contexte dans lequel peuvent advenir ces coordinations.

5.6.6 Bilan de l'implantation

XMG permet de créer rapidement une grammaire assez conséquente de la coordination en gardant une vue globale du phénomène. L'approche métagrammaticale s'inscrit bien dans notre modélisation puisque les DAP que nous voulons obtenir respectent toutes un patron général commun, défini dans les classes *hautes* telles que `Prototype_Coordination` ou `Prototype_Coordination_SN`, qui est ensuite spécifié et instancié vers des DAP concrètes.

Nous pouvons produire deux grammaires, selon que l'on tient compte des traits d'accord (450 DAP) ou pas (45 DAP). Cette différence de taille est un artefact du formalisme même, puisque nous ne pouvons pas exprimer dans la version actuelle des GI le fait que, par exemple, le genre du syntagme nominal coordonné est fonction du genre des conjoints. Nous ne pouvons qu'indiquer les corrélations entre les genres, c'est à dire lister les cas. Les GI doivent donc incorporer à l'avenir les fonctions de valeurs de trait, ce qui ne doit pas poser de problème majeur.

Ces deux grammaires sont intégrées à une grammaire plus générale du français et utilisées dans l'analyseur `leopar`.Nous avons évalué notre implantation sur la TSNLP[37] et

[37]La TSNLP est une suite de phrases tests pour les analyseurs syntaxiques. Elle n'est plus maintenue

d'ores et déjà nous modélisons 80% des phrases grammaticales contenant des coordinations. Les phrases agrammaticales sont rejetées intégralement.

5.7 Bilan

Nous avons proposé une modélisation de certains aspects de la coordination dans les GI, notamment la coordination de constituants et la majorité des coordinations de non-constituants. À partir de celle-ci nous avons implanté une grammaire pour la plate-forme `leopar` en nous appuyant sur l'outil XMG précédemment présenté. Nous avons évalué cette implantation sur la suite de tests TSNLP. Enfin, nous proposons une extension au système de superposition des GI pour prendre en compte les coordinations disparates.

Cette extension finale permet également de mettre en relief un problème souvent sous-estimé en linguistique, celui de la co-occurrence des phénomènes et des propositions de modélisation au sein d'un même formalisme et au sein d'une même grammaire. On a ici proposé une extension pour modéliser un phénomène particulier, qui n'est pas implantable immédiatement pour des raisons techniques d'une part – les règles de superposition doivent être ajoutées à `leopar` – mais surtout parce qu'elle est en contradiction avec la modélisation d'une partie des compléments verbaux implantée dans notre grammaire actuelle. C'est vraiment un aspect sous-estimé dans de nombreux articles, en particulier ceux qui traitent de la coordination, qui n'abordent jamais la question de l'insertion de la proposition dans des grammaires existantes.

Nous voulons ici insister sur certaines caractéristiques de notre modélisation, qui vont justifier le reste des travaux présentés dans ce document.

- Notre modélisation insiste sur la valence des conjoints comme condition nécessaire à leur coordination. Il faut également qu'ils aient la même *forme*. Ces deux notions sont regroupées sous le terme d'interface. Le fait que les conjoints doivent avoir la même valence est la base de notre méthode de désambiguïsation que nous verrons au chapitre suivant.
- Les conjoints n'interagissent ni entre eux ni avec le reste de la phrase. Il ne peuvent se combiner qu'avec la DAP associée à la conjonction de coordination[38]. C'est cette dernière qui interagit avec le reste de la phrase. Plus précisément c'est la partie haute de sa DAP qui interagit avec le reste de la phrase. En quelque sorte, la conjonction de coordination retire les conjoints de la phrase et prend leur place dans le contexte. Nous pouvons donc avancer l'idée suivante : analyser d'une part la coordination des deux conjoints — que nous pourrons délimiter grâce à notre méthode de désambiguïsation — et d'autre part le reste de la phrase avec la partie haute de la DAP de la conjonction de coordination. Nous procéderons donc pour les phrases contenant des coordinations à plusieurs analyses, une pour chaque groupe coordonné et une pour le *squelette* de la phrase restante. Pour ce faire nous avons développé un nouvel algorithme déductif d'analyse descendante que nous présentons

mais reste accessible à `http ://cl-www.dfki.uni-sb.de/tsnlp/`

[38] C'est pourquoi une DAP particulière de conjonction de coordination est associée à chaque type de conjoints coordonnés. Notre grammaire de la coordination est don vouée à évoluer (à grossir) au fur et à mesure que de nouveaux phénomènes sont pris en compte dans la grammaire principale.

dans la dernière partie de cette thèse.

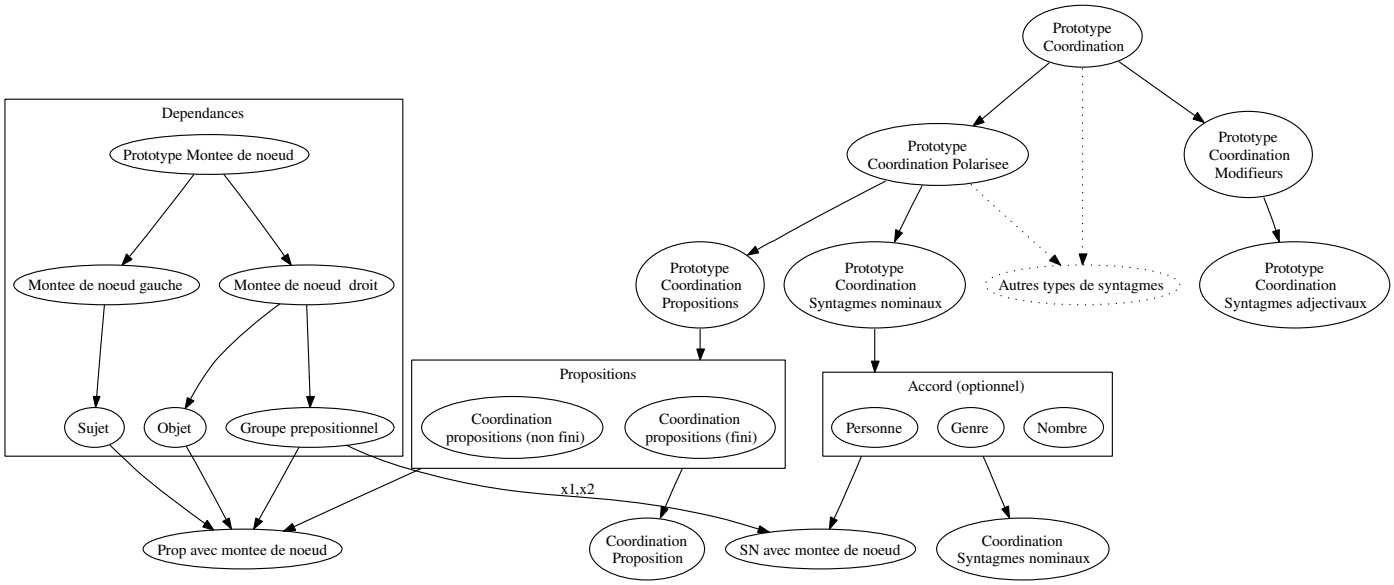


FIG. 5.27 – Organisation d'un fragment de la grammaire de la coordination

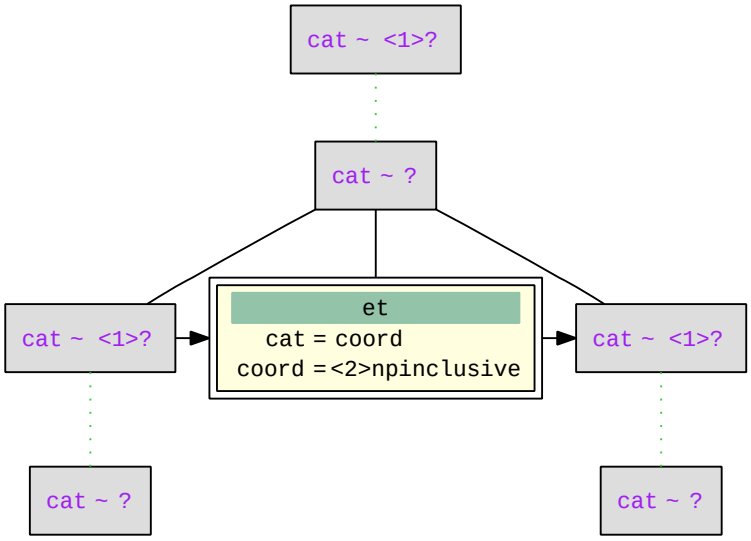


FIG. 5.28 – Prototype des DAP pour la coordination

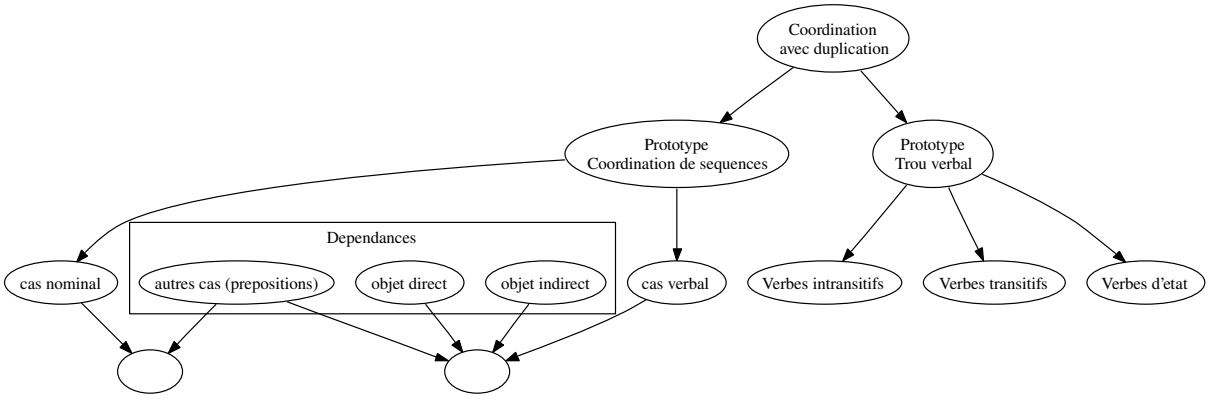


FIG. 5.29 – Seconde hiérarchie pour les phénomènes à duplication

Troisième partie

Filtrage lexical et coordination

Chapitre 6

Automates de filtrage lexical

Sommaire

6.1 Introduction

Dans ce chapitre, nous allons traiter de la désambiguïsation lexicale, aussi appelée filtrage, préalable aux analyses grammaticales dans les GI.

Nous l'avons déjà fait remarquer, les GI manipulés par notre plate-forme `leopar` sont fortement lexicalisées c'est à dire qu'elles sont complètement définies par le lexique qui associe à chaque mot de la langue un ensemble fini de DAP, décrivant les différents usages de ce mot. En fait, dès cette étape d'association de DAP aux mots d'une phrase à analyser surgit un problème de combinatoire.

Nous appelons *sélection lexicale* pour une phrase donnée, un choix pour chaque mot de cette phrase d'une DAP. Or le nombre de façons d'associer une suite de DAP à une phrase, c'est à dire le nombre de sélections lexicales, est le produit du nombre d'associations possibles pour chaque mot de cette phrase. Dans nos grammaires, par exemple, un mot est associé à 5 DAP en moyenne. En conséquence, pour une phrase qui compte 30 mots comme celle-ci, il y a en moyenne $5^{30} = 9,3 \cdot 10^{20}$ possibilités de sélections lexicales, et toutes ces sélections peuvent peut-être donner une analyse.

La simple énumération de l'ensemble des sélections est un processus exponentiel si elle est réalisée par l'analyse de manière explicite. Par exemple, pour notre phrase précédente de 30 mots, en supposant qu'il faille 10^{-9} secondes pour afficher une sélection, l'énumération complète prend $9,3 \cdot 10^{20} \cdot 10^{-9} = 9,3 \cdot 10^{11}$ secondes, soit environ 10^7 jours. On comprend pourquoi cette énumération explicite n'est pas raisonnable.

La coordination nous pousse à développer des techniques de filtrage pour deux raisons :

1. la longueur des phrases avec coordination est plus grande et le nombre de sélections augmente donc en présence de coordinations (exponentiellement)
2. le nombre de DAP associées aux conjonctions de coordination est de l'ordre de 40-50 (cf. chapitre précédent). Le nombre de sélections dans les cas de coordinations sans filtrage peut donc être gigantesque.

L'idée consistant à modéliser les sélections possibles en utilisant des automates à états finis n'est pas nouvelle. Elle est même assez commune, la principale raison étant que tous les choix lexicaux pour une phrase peuvent être représentés dans un espace linéaire en fonction de la taille de la phrase. En particulier, les méthodes fondées sur les modèles de Markov cachés (HMM) utilisent de telles techniques pour l'étiquetage des parties du discours [Kup92; Mer94; WMS+93]. Il est même possible avec cette représentation de concevoir des techniques de programmation dynamique et donc de diminuer la complexité temporelle comme cela a déjà été fait pour la complexité spatiale.

Or, nous voulons ici désambiguïser pour ensuite analyser. Plus exactement, nous nous inscrivons dans le cadre de l'analyse syntaxique exacte et profonde. C'est ce qui nous

empêche d'utiliser des méthodes probabilistes. Voyons les problèmes que soulève l'analyse exacte et profonde :

- [JS94] utilise les probabilités pour filtrer les sélections lexicales avant l'analyse avec les TAG et GCC. Ils s'en remettent aux n-grammes dans le but de garder les sélections les plus probables. Le problème, en contradiction avec les objectifs de l'analyse exacte, vient du fait que cette méthode peut *rater* des sélections menant à une solution si ces sélections sont considérées comme très peu probables. Ensuite, l'analyseur ne peut pas réparer cette erreur faite durant l'étape précédente. Notre priorité n'est pas d'avoir un petit nombre de sélections mais plutôt d'être sûr de garder les bonnes sélections, aussi peu probables soient-elles.
- Pour que les méthodes à base de HMM soient efficaces, il faut travailler sur des ensembles d'étiquettes relativement petits. Or ici, l'ensemble des étiquettes à utiliser serait l'ensemble des constructions syntaxiques, c'est-à-dire les DAP, et pas seulement des étiquettes de partie du discours. Ceci entraînera une explosion du nombre de probabilités à calculer. Il faudrait dans ces conditions un corpus d'apprentissage gigantesque. En conséquence nous n'utiliserons pas les HMM.

Pour ces deux raisons, nous reconsidérons le problème de la désambiguisation lexicale dans le cadre de l'analyse exacte et nous allons exploiter les spécificités des GI.

C'est la polarisation des traits qui est la caractéristique qui va nous intéresser dans ce chapitre, notamment le fait que dans une analyse valide chaque polarité $\rightarrow$ rencontre une polarité $\leftarrow$ pour se neutraliser. Les polarités sont également utilisées pour guider la composition syntaxique, comme nous le verrons au chapitre suivant.

Les DAP, si nous feignons d'oublier leur structure, deviennent des collections de traits polarisés. Nous pouvons considérer cette simplification comme une abstraction du formalisme. Dans cette grammaire abstraite, l'« analyse » est triviale puisqu'elle revient à compter les polarités pour chaque nom de trait et chaque valeur, $\rightarrow$ donnant $+1$ et $\leftarrow$ -1. L'analyse réussit si la somme pour une sélection donne zéro. Comme cette analyse est très simple, elle peut être utilisée comme moyen de filtrer les entrées lexicales de la grammaire initiale. Les fondements de cette technique déjà présentée dans [BGP04] viennent de l'homomorphisme de la grammaire initiale vers la grammaire abstraite : chaque analyse dans la première est transposée dans une analyse dans la seconde. Et ce qui nous intéresse plus pour le filtrage, c'est que si une analyse échoue dans la seconde alors elle échoue dans la première.

Dans ce chapitre nous ne considérons que les GI mais la méthode présentée peut être adaptée à tout formalisme polarisé ou à tout formalisme pouvant se polariser, comme le montre [Kah04]. C'est le cas des CFG et des TAG notamment. Ces techniques sont également appliquées à la génération par [Kow05].

Dans un premier temps nous présentons la sélection lexicale telle que notre plate-forme d'analyse `leopar` la réalise. Les techniques sont sont assez originales et sont à rapprocher de l'analyse dans les grammaires d'intersection [Tap97], où l'analyse syntaxique revient à effectuer l'intersection de l'automate de sélection avec les automates qui donnent les règles de la grammaire.

Nous nous intéresserons ensuite à la question de la complexité de l'opération décrite dans [BGP04], en abordant particulièrement la complexité en états de l'intersection sur les automates particuliers que nous manipulons.

Enfin, nous présentons dans ce chapitre un raffinement de la méthode proposée par [BGP04] qui tirent profit de la modélisation syntaxique de la coordination vue au chapitre précédent pour filtrer plus efficacement.

6.2 Automates à états finis déterministes acycliques

Notre méthode est techniquement fondé sur des intersections d'automates déterministes. Dans cette section nous présentons ces structures de données.

6.2.1 Définition

Un automate à états finis déterministe (DFA) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où Q est l'ensemble fini des états de l'automate, Σ est un alphabet fini, $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition, $q_0 \in Q$ est l'état initial et $F \subseteq Q$ l'ensemble des états finaux (ou acceptants).

Soit ϵ le mot vide. Nous appelons $\hat{\delta}$ l'extension de la fonction de transition aux mots de Σ^* : $\hat{\delta}(q, \epsilon) = q$, $\hat{\delta}(q, ax) = \hat{\delta}(\delta(q, a), x)$ pour $a \in \Sigma, q \in Q$ et $x \in \Sigma^*$.

Un DFA est acylique si pour toute suite de lettres ax et pour tout état $q \in Q$ $\hat{\delta}(q, ax) \neq q$, c'est-à-dire qu'il n'a pas de boucle. Un automate est non-déterministe (NFA) si δ n'est pas une fonction mais une relation dans $Q \times \Sigma \times Q$. Le lecteur est reporté vers [HU79] pour tout complément en théorie des automates.

Nous allons tout de même définir ici l'opération d'intersection qui sera l'ossature de notre méthode de désambiguïsation.

6.2.2 Opération d'intersection

Les automates à états finis permettent de décrire les langages rationnels d'un point de vue calulatoire et opérationnel [Kle56]. Le langage décrit par un automate A est noté $L(A)$. Les opérations sur les langages rationnels sont donc directement traduisibles en opérations sur ces structures de données. Les automates déterministes donnent une nouvelle notion de complexité pour les opérations sur les langages rationnels, le nombre d'états de l'automate.

Soient L_1 et L_2 deux langages (rationnels). L'intersection de L_1 et L_2 est un langage (rationnel) L_3 tel que

$$L_3 = L_1 \cap L_2 = \{u | u \in L_1 \text{ et } u \in L_2\}$$

Il existe donc une opération qui à partir de deux automates A_1 et A_2 représentant respectivement L_1 et L_2 permet de construire l'automate A_3 représentant L_3 intersection de L_1 et L_2. Par abus de langage, cette opération est également appelée intersection.

Définition 13. *Soient $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ et $A_1 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ deux DFA. L'automate intersection $A_1 \cap A_2$ qui reconnaît le langage $L(A_1 \cap A_2) = L(A_1) \cap L(A_2)$ est :*

$$A_1 \cap A_2 = \{Q_1 \times Q_2, \Sigma, \delta_{12}, (q_1, q_2), F_1 \times F_2\}$$

où la fonction de transition δ_{12} est :

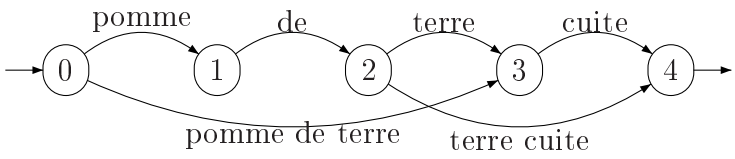


FIG. 6.1 – Automate de segmentation

$$\delta_{12}((p,q),a) = \begin{cases} (p',q') & si\ \delta_1(p,a) = p'\ et\ \delta_2(q,a) = q' \\ n'est\ pas\ définie & sinon \end{cases}$$

Il est important de noter que l'automate ainsi construit n'est pas forcément minimal dans le sens où il a potentiellement des états inutiles, l'ensemble des états $Q_1 \times Q_2$ est un sur-ensemble des états accessibles de l'automate.

C'est de cette observation que provient la notion de *complexité en états* des opérations sur les automates. La complexité en états d'une opération est le nombre d'états de l'automate minimisé équivalent à l'automate résultat de cette opération, en fonction des automates d'entrées. La complexité en états des opérations communes sur les automates (concaténation, union, intersection, étoile) est décrite dans [YZS94]. C'est cette complexité en états qui va nous intéresser plus loin dans ce chapitre.

6.2.3 Automates de segmentation

En général, il existe plusieurs lectures d'une même phrase, que l'on appelle des segmentations. Un automate de segmentation représente alors toutes les segmentations possibles d'une phrase à analyser. Chaque chemin de l'état initial à l'état final représente une segmentation possible. L'exemple de la figure 6.1 illustre le fait qu'il peut exister plusieurs chemins pour une même phrase, selon la lecture que l'on en fait. Donc, pour une phrase de texte brut, il sera peut-être nécessaire d'analyser plusieurs phrases constituées de mots du lexique de notre grammaire. Nous allons représenter ces différentes phrases à analyser à l'aide d'un automate.

Le point de départ des opérations de filtrage est *l'automate de segmentation*. Mais dans cette thèse, nous ne nous préoccupons pas de la construction de cet automate à partir d'une chaîne de caractères (ou d'un flot de phonèmes). C'est un sujet de recherche toujours actif, voir par exemple [BK03] et [Hue05]. Nous nous contenterons de supposer qu'il existe une fonction de segmentation s qui renvoie toutes les segmentations possibles d'une phrase de texte brut. Pour une phrase d'entrée u, on notera l'ensemble des segmentations $s(u) = \{s_i | s_i = w_{i_1} \dots w_{i_n}\}$ où tous les w_i appartiennent au vocabulaire de la grammaire courante. On peut retrouver la phrase d'origine à partir d'une segmentation : $s_i \in s(u) \iff u = s^{-1}(s_i)$

Pour une GI $G = (V, D, L)$ et une phrase u qui admet les segmentations $S = \{s_i | s_i = w_{i_0} \dots w_{i_n}\}$ où pour toute segmentation s_i on impose $i_0 = 0$ et $i_n = n$, l'automate de segmentation est un DFA acylique $A_S = (Q, \Sigma, q_0, \delta, F)$ où $Q \subseteq [0; n]$ et $\Sigma \subseteq V$. La fonction de transition est définie par $\delta(j, w_{i_k}) = i_{k+1}$. Chaque chemin est une segmentation de u. De plus, un automate de segmentation a un unique état final $\{n\}$.

6.3 Automates de polarités

6.3.1 Sélections lexicales

Nous rappelons qu'une phrase $w_1 \cdots w_n$ appartient au langage engendré par une GI $G = (V, D, L)$ s'il existe $d_1, \dots, d_n \in L(w_1) \times \cdots \times L(w_n)$ admettant un modèle saturé et minimal.

Nous généralisons cette définition aux automates de segmentation.

Définition 14 (sélection lexicale)**.** *Une sélection lexicale pour un automate de segmentation A_S et une grammaire $G = (V, D, L)$ est une suite de descriptions $s = d_1, \dots, d_n \in L(w_1) \times \cdots \times L(w_n)$ telle que $w_1, \dots w_n$ est un chemin de l'état initial à l'état final de A_S, autrement dit un chemin maximal.*

Si la sélection S admet un modèle saturé et minimal, alors on dira par abus de langage que c'est une sélection valide.

Par exemple, pour l'exemple que nous avons vu dans la section précédente, une sélection pour *pomme de terre cuite* est un élément de

- L(pomme de terre) $\times$ L(cuite) ou de
- L(pomme) $\times$ L(de) $\times$ L(terre cuite) ou de
- L(pomme) $\times$ L(de) $\times$ L(terre) $\times$ L(cuite).

Il apparaît que le nombre de sélections lexicales n'est pas lisible directement sur l'automate de segmentation. Nous allons donc définir un nouveau type d'automate. Il s'agit de remplacer chaque arc étiqueté w_i de l'automate de segmentation par des arcs étiquetés par d où $d \in L(w_i)$

Définition 15. *Un automate de sélection pour une* GI $G = (V, D, L)$ *et un automate de segmentation $A_S = (Q, \Sigma, q_0, \delta, F)$ est un DFA $A = (Q, \Sigma', q_0, \delta', F)$ où $\Sigma' = L(\Sigma)$, les arcs de A sont étiquetés par les DAP associées aux arcs de A_S par le lexique de G de la façon suivante :*

$$\delta'(q_1, d) = \begin{cases} q_2 & si\ \delta(q_1, w) = q_2\ et\ d \in L(w) \\ indéfini & sinon \end{cases}$$

L'automate de sélection pour notre exemple est donné en figure 6.2.

Une sélection lexicale est alors un chemin acceptant de l'automate de sélection. Le nombre de sélections lexicales est directement donné par le nombre de chemins de l'automate de sélection. L'ensemble des sélections valides est un sous-ensemble des chemins de cet automate.

Le but idéal du filtrage lexical est de ne donner à l'analyseur que des sélections valides. Plus simplement, nous allons donner un critère qui permet d'exclure un grand nombre de sélections tout en gardant toutes les sélections valides.

6.3.2 Critère de correction

Dans cette section nous allons montrer qu'il existe un critère simple pour filtrer les sélections valides. Tout d'abord nous devons compter les polarités. Nous allons donner une valeur $+1$ aux traits $\rightarrow$ et une valeur -1 aux traits $\leftarrow$. Et on va compter ces traits pour chaque chemin de l'automate de sélection. Les chemins maximaux qui n'aboutissent pas à la valeur 0 n'ont donc pas autant de traits $\rightarrow$ que de traits $\leftarrow$. Ces chemins peuvent être retirés car ils ne donneront pas une sélection valide. En effet, il n'y aura aucune analyse valide pour laquelle le nombre de traits $\rightarrow$ n'est pas égal au nombre de traits $\leftarrow$. De manière à être plus exacts nous allons compter les polarités pour chaque nom de trait et chaque valeur de trait de ce nom. On aura alors un décompte beaucoup plus précis des polarités. Pour réaliser ce décompte sur un chemin complet de l'automate, il faut tout d'abord pouvoir le réaliser sur chaque arc qui compose ce chemin, c'est-à-dire qu'il faut être capable de préciser ce que chaque DAP apporte en terme de polarités pour un nom et une valeur de traits donnés : c'est ce que nous appelons son bilan de polarités qui est en fait la somme des bilans de polarités des nœuds qui la composent.

Si l'on veut le bilan de polarité d'un nœud par rapport à un nom f et une valeur v qui est un ensemble de valeurs atomiques, il faut regarder si la structure de traits associée à ce nœud contient un trait polarisé de la forme (f, p, q). S'il n'en a pas, le bilan est nul. Dans le cas contraire, il faut examiner la polarité p et la valeur q.

- Si $v \cap q = \emptyset$, le bilan est également nul.
- Si $v = q$ et $p = \rightarrow$, le bilan est $+1$.
- Si $v = q$ et $p = \leftarrow$, le bilan est -1.
- Si $v \cap q \neq \emptyset$ et $p = \rightarrow$, il est possible que le bilan soit $+1$ ou 0, selon la valeur qui sera choisie lors de l'analyse syntaxique.
- Si $v \cap q \neq \emptyset$ et $p = \leftarrow$, il est possible que le bilan soit -1 ou 0, pour les mêmes raisons.

Pour encoder l'indéterminisme qui peut apparaître, un bilan de polarités sera un ensemble inclus dans $\{-1, 0, +1\}$. La définition suivante précise la discussion ci-dessus :

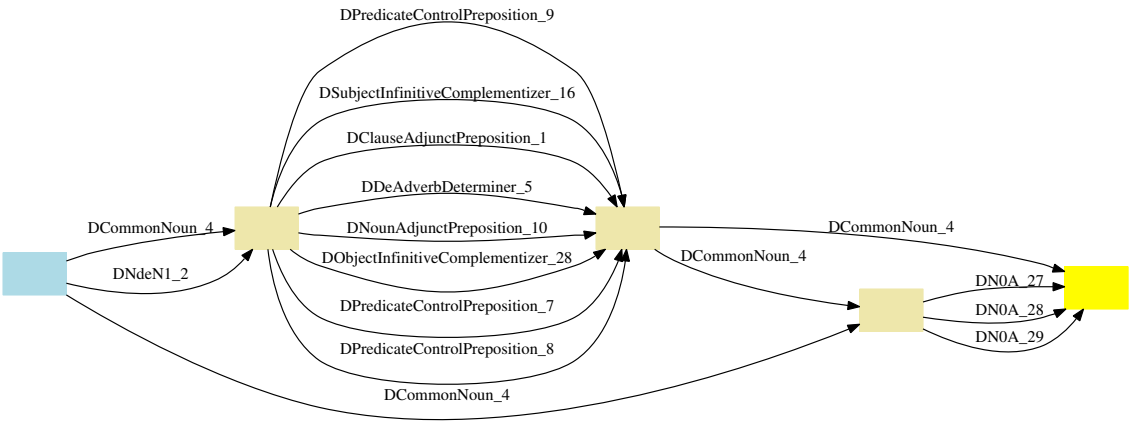


FIG. 6.2 – Automate de sélection pour *pomme de terre cuite*

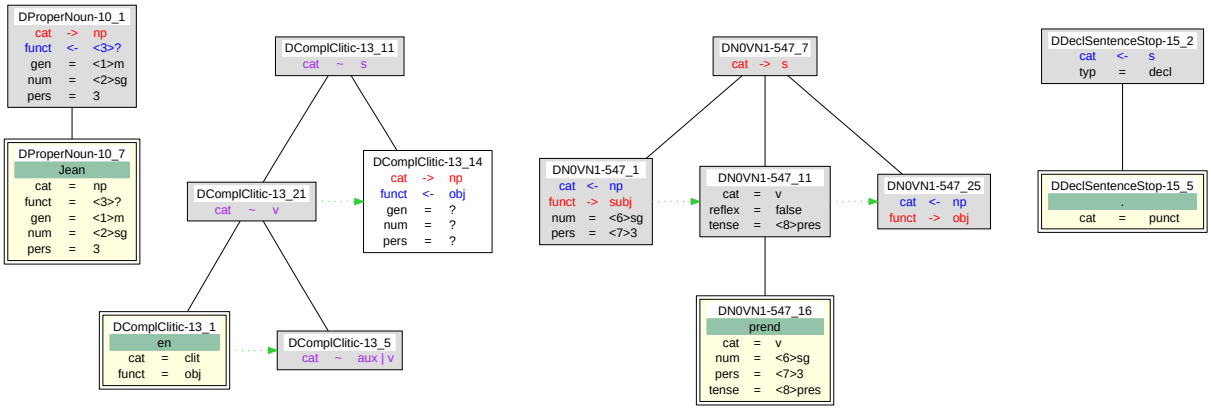


FIG. 6.3 – Une sélection (valide) de «Jean en prend.»

Définition 16. *Étant donnés un nœud n d'une DAP d, un nom de trait f et une valeur $v_1 \subseteq \mathcal{V}_f$,* le bilan de polarités p_n *est une fonction des valeurs de traits vers les ensembles d'entiers, définie de la manière suivante :*

1. *Si la structure de traits effective de n contient un trait $(f, \rightarrow, v_2)$ et $v_2 \subseteq v_1$ alors* $p_n(v_1) = \{+1\}$.
2. *Si la structure de traits effective de n contient un trait $(f, \rightarrow, v_2)$, $v_1 \cap v_2 \neq \emptyset$ et $v_1 \cap v_2 \neq v_2$ alors* $p_n(v_1) = \{0, +1\}$.
3. *Si la structure de traits effective de n contient un trait $(f, \leftarrow, v_2)$ et $v_2 \subseteq v_1$ alors* $p_n(v_1) = \{-1\}$.
4. *Si la structure de traits effective de n contient un trait $(f, \leftarrow, v_2)$, $v_1 \cap v_2 \neq \emptyset$ et $v_1 \cap v_2 \neq v_2$ alors* $p_n(v_1) = \{0, -1\}$.
5. *Dans les autres cas,* $p_n(v_1) = \{0\}$.

Nous étendons la fonction p à une DAP d dont les nœuds sont $n_1, \ldots, n_k$. Étant donnés un nom de trait f et une valeur v (incluse dans $\mathcal{V}_f$), le bilan de polarités *de d est* $p_d(v) = \{n \in \mathbb{Z} | n = p_1 + \cdots + p_k$ *avec* $p_1 \in p_{n_1}(v), \ldots, p_k \in p_{n_k}(v)\}$.

De la même manière le bilan d'une suite de DAP $s = d_1, \ldots, d_p$ est $p_s(v) = \{n \in \mathbb{Z} | n = p_1 + \cdots + p_k$ *avec* $p_1 \in p_{d_1}(v), \ldots, p_k \in p_{d_k}(v)\}$.

La figure 6.3 rappelle l'exemple vu en section 1.2.5 dans lequel on voit les quatre descriptions d_1, d_2, d_3 et d_4 d'une sélection pour la phrase « *Jean en prend.*» et la grammaire d'interaction de l'analyseur `leopar`.

Pour le nom de trait *funct* et la valeur $\{subj, obj\}$, nous avons les bilans de polarités suivants :

- $p_{d_1}(\{subj, obj\}) = \{0, -1\}$ (le symbole `?` est utilisé pour indiquer l'ensemble des valeurs atomiques correspondant au nom de trait)
- $p_{d_2}(\{subj, obj\}) = \{-1\}$
- $p_{d_3}(\{subj, obj\}) = \{+2\}$

– $p_{d_4}(\{subj, obj\}) = \{0\}$
– $p_{d_1,d_2,d_3,d_4}(\{subj, obj\}) = \{0, +1\}$

0 appartient au bilan de polarités de la sélection. Cette sélection ne sera pas rejetée par notre méthode de filtrage.

Proposition 1. *L'ensemble retourné par la fonction p est toujours un intervalle de* $\mathbb{Z}$.

Les bilans de polarités sont des intervalles pour les nœuds. D'après la définition du bilan de polarités d'une description et d'une sélection (des sommes d'intervalles) il est clair que ce sont également des intervalles.

6.3.3 Arithmétique d'intervalles

Dans la suite, nous allons manipuler des intervalles d'entiers.

Définition 17 (Intervalle)**.** *On appelle l'intervalle d'entiers* $[a; b]$ *l'ensemble de tous les entiers compris entre a et b :* $[a; b] = \{n | a \leq n \leq b\}$.

On définit l'opération de sommes d'intervalles

Définition 18 (Somme d'intervalles)**.** *Soient deux intervalles* $I_1 = [a; b]$ *et* $I_2 = [c; d]$. *On définit la somme de ces deux intervalles par* $I_1 + I_2 = [a + c; b + d]$.

La somme d'un entier n et d'un intervalle $I = [a; b]$ *est défini par* $n + I = [n + a; n + b]$.

Proposition 2. *Soient* I_1 *et* I_2 *deux intervalles. L'égalité suivante est toujours vraie :*

$$I_1 + I_2 = \{n | n = n_1 + n_2 \text{ avec } n_1 \in I_1 \text{ et } n_2 \in I_2\}$$

On peut donc redéfinir les bilans de polarité à l'aide de nos sommes d'intervalles de la façon suivante. Pour les DAP :

$$p_d(v) = \sum_{n \in N(d)} p_n(v), \text{où } n \text{ est un nœud de } d$$

Pour les suites de DAP :

$$p_{d_1,\ldots,d_n}(v) = \sum_{1 \leq i \leq n} p_{d_i}(v)$$

6.3.4 Un critère sur les sélections

Nous donnons maintenant un critère qui distingue un sur-ensemble des sélections valides que nous appelons *sélections saturées*. Ce critère est nécessaire pour que la sélection soit valide.

Définition 19. *Une sélection S est saturée si pour chaque nom de trait f et chaque valeur atomique v de* $\mathcal{V}_f$, $0 \in p_S(v)$.

Proposition 3. *Toute sélection S valide est saturée.*

En effet, si une sélection est valide, c'est qu'il existe un modèle saturé et minimal de cette sélection. La saturation du modèle implique que chaque trait $(f, \rightarrow, v_1)$ est associé à son dual $(f, \leftarrow, v_2)$ avec $v_1 \cap v_2 \neq \emptyset$. Ceci revient dans notre calcul à dire que chaque $+1$ est compensé par un -1 et donc que le bilan est nul.

Plus précisément, d'après les définitions le bilan de polarité d'une sélection est la somme des bilans de polarité des nœuds des DAP qui la composent : $p_{d_1 \ldots, d_m}(v) = \sum_{1 \leq i \leq m} \sum_{n \in N(d_i)} p_n(v)$. En tenant compte de l'indéterminisme causé par les intervalles, il existe un choix de valeurs dans ces intervalles qui donne un bilan de polarité nul.

En conséquence, toutes les sélections qui ne sont pas saturées peuvent être écartées puisqu'elles n'amèneront jamais à une analyse car il ne peut exister de modèle saturé pour ces sélections.

Ce critère que nous appellerons par la suite *critère de saturation* est global à la sélection et donc à la phrase à analyser. Il n'est pas sensible à l'ordre des mots. Pour certaines applications comme la génération où l'ordre des mots n'est pas connu, c'est un avantage. En revanche, c'est une limitation pour l'analyse puisque dans ce cas les phénomènes locaux sont pertinents. La permutation d'une sélection saturée est également une sélection saturée. Ce n'est évidemment pas le cas pour une sélection valide. Or un automate permet de représenter l'ordre des mots. Nous allons donc développer plus loin dans ce chapitre des méthodes de filtrage basées sur l'ordre des mots, l'une spécifique à le coordination et l'autre plus générale. Le filtrage complet sera le résultat de l'application successive des différentes méthodes de filtrage. Dans l'immédiat nous allons revenir à notre critère de saturation globale.

6.3.5 Modélisation à l'aide d'automates

Plutôt que de compter les polarités chemin par chemin, nous pouvons profiter du partage de ces chemins induit par l'automate de sélection. À chaque état de l'automate, on compte le bilan de polarités.

Soit $A = (Q, \Sigma, q_0, q_f, \delta)$ un automate de sélection. A partir de cet automate et pour un nom de trait et une valeur de trait donnés, on va créer un nouvel automate représentant le bilan de polarités des sélections pour ce trait et cette valeur. On décore chaque état avec le bilan de polarités à cette position dans la phrase. Les états finaux seront donc décorées du bilan de polarité d'une sélection pour la phrase d'entrée.

De l'automate de segmentation à l'automate de sélection nous avions changé les étiquettes des arcs mais nous avions gardé les mêmes états. Ici pour passer des automates de sélection aux automates de filtrage, nous allons gardé les mêmes arcs mais nous allons multiplier les états. Si on arrive à un état de l'automate de sélection par différents chemins ayant des bilans de polarités différents, cet état sera divisé en plusieurs.

Définition 20 (Automate de filtrage naïf)**.** *Un automate de filtrage naïf pour un trait v et pour un automate de sélection $A_S(v) = (Q, \Sigma, \delta, q_0, q_f)$ est un automate non déterministe acyclique $A(v) = (Q', \Sigma, \delta_v, q_0, F)$ tel que les états sont des paires (i, j) constituées d'un entier naturel et d'un entier relatif, $Q' \subseteq Q \times \mathbb{Z}$. i est un entier naturel indiquant une position dans la phrase d'entrée et j est un élément de la somme des bilans de polarité*

des DAP à cette position. j représente une suite de choix dans les bilans de polarité des nœuds rencontrés de la position 0 à la position i.

La relation de transition est :

$$\delta_v((i,j),d) = \begin{cases} (i',j') & si\ \delta(i,d) = i'\ et\ j' \in j + p_d(v) \\ indéfinie & sinon \end{cases}$$

L'état initial est $(0,0)$ et l'état final est $(n,0)$ où n est la longueur de la phrase d'entrée (le nombre de mots qui la composent).

Nous avons donc des automates non-déterministes pour chaque trait. Nous verrons dans la suite que nous aurons besoin de calculer l'intersection de tels automates. Une première étape pour faciliter cette intersection est de ne travailler que sur des automates déterministes. Plutôt que d'appliquer une méthode classique de déterminisation, nous tirons profit de la proposition 1 : plutôt que de décorer les états avec un élément du bilan de polarité, nous allons les décorer avec l'intervalle entier. En réalité, nous construisons directement l'automate déterministe correspondant :

- les états sont des paires (i,p) où i est toujours la position dans la phrase mais p est un intervalle de $\mathbb{Z}$. Au lieu d'avoir un état par élément de $p_{d_1,\ldots,d_i}$(v), nous construisons un seul état contenant l'intervalle $p_{d_1,\ldots,d_i}(v)$.
- l'état initial est $(0,[0;0])$
- Les états finaux sont de la forme (n,q) où n est la longueur de la phrase et q un intervalle contenant 0.
- $\delta_v((i,p),d)) = (i',q)$ où $i' = \delta(i,d)$ et q est l'intervalle $p + p_d(v)$.

Chaque chemin maximal de l'état initial $(0,[0;0])$ vers un état (n,q) représente une sélection complète et son bilan de polarité. Si ce chemin n'aboutit pas à un état final, $0 \notin q$, alors la sélection n'est pas saturée. Nous pouvons donc simplifier l'automate (c'est-à-dire retirer tous les états et les arcs qui n'apparaissent sur un chemin acceptant) pour ne garder que les sélection *potentiellement* saturées. Elles ne seront *vraiment* saturées qu'à la condition que ce chemin existe dans tous les automates de filtrages pour toutes les valeurs de trait v.

Le critère de saturation doit donc être réalisé pour toute valeur de trait. Par conséquent une sélection s est saturée si et seulement si s est un chemin maximal de l'automate intersection, c'est-à-dire qu'elle appartient au langage défini par cet automate :

$$s \text{ est une sélection saturée } \iff s \in L(\bigcap_{v \in \bigcup_f D_f} A(v))$$

Notre méthode de filtrage consiste donc à calculer cet automate intersection.

Remarque Notre déterminisation semble très originale mais nous ne faisons que reprendre la méthode classique qui consiste à établir des classes d'équivalences d'états sur l'automate non-déterministe qui seront les états de l'automate déterministe. Nous profitons du fait que les états sont des entiers relatifs et que les classes d'équivalence sont représentables sous la forme d'intervalles d'entiers (deux entiers quelle que soit la taille de l'intervalle).

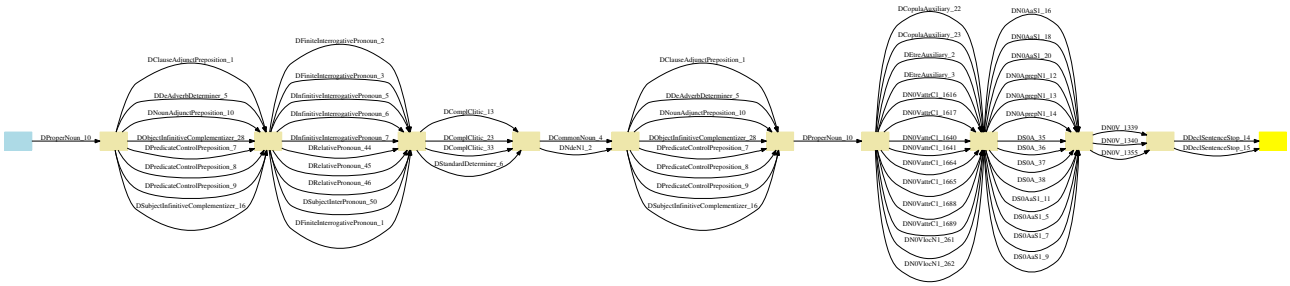


FIG. 6.4 – Automate initial pour «Jean de qui la femme de Pierre est amoureuse dort.»

6.4 Intersection d'automates

6.4.1 Algorithme de filtrage

Notre algorithme de filtrage procède comme suit :

1. Construire l'automate A_S de sélection ;
2. Collecter toutes les valeurs de traits présentes dans les DAP de A_S ;
3. Pour chaque valeur de trait v_i construire l'automate de filtrage $A(v_i)$;
4. Calculer l'intersection des $A(v_i)$.

Dans cette section nous allons prendre pour exemple la phrase « Jean de qui la femme de Pierre est amoureuse dort.» L'automate de sélection est représenté sur la figure 6.4. Cet automate possède 12 états et 6021120 chemins acceptants, c'est-à-dire que la phrase correspondante admet 6021120 sélections lexicales.

Après cette étape, on collecte les noms de traits et les valeurs présents sur les différentes DAP. En effet, si une valeur v n'est présente dans aucune DAP d'une sélection S, $p_S(v) = \{0\}$ et tous les chemins de $A(v)$ aboutissent à un état final. L'automate est isomorphe à l'automate de sélections, donc l'automate ne filtre pas. Pour les mêmes raisons, on ne considère que les valeurs de traits associées au moins une fois à une polarité positive $\rightarrow$ ou négative $\leftarrow$. Dans notre exemple, si on collecte les couples noms, valeurs effectives (f, v) de ce type, alors on obtient $C = \{(prep, a), (prep, comme), (prep, de), (prep, loc), (prep, pour), (mood, pastp), (funct, ?), (funct, adj), (funct, adj|aobj|dat|deobj|obl), (funct, aobj), (funct, attr), (funct, deobj), (funct, loc), (funct, obj), (funct, obl), (funct, subj), (funct, void), (cpl, ?), (cpl, ceque), (cpl, de), (cat, ap), (cat, aux), (cat, n), (cat, n|np), (cat, np), (cat, pp), (cat, s), (break, close), (aux, cop), (aux, etre)\}$.

On se rend compte que le couple (cat, s) est présent. On construit donc l'automate $A(s)$, que l'on peut voir sur la figure 6.5. Cet automate possède 29 états et 278880 chemins acceptants.

Finalement, on construit l'automate intersection de tous ces automates de filtrage. Pour notre exemple, cela donne l'automate représenté sur la figure 6.6. Cet automate a 65 états et 95 chemins, c'est-à-dire que nous avons éliminé 99, 998% des sélections lexicales. Nous donnons certaines tailles d'automates intermédiaires dans la table 6.1.

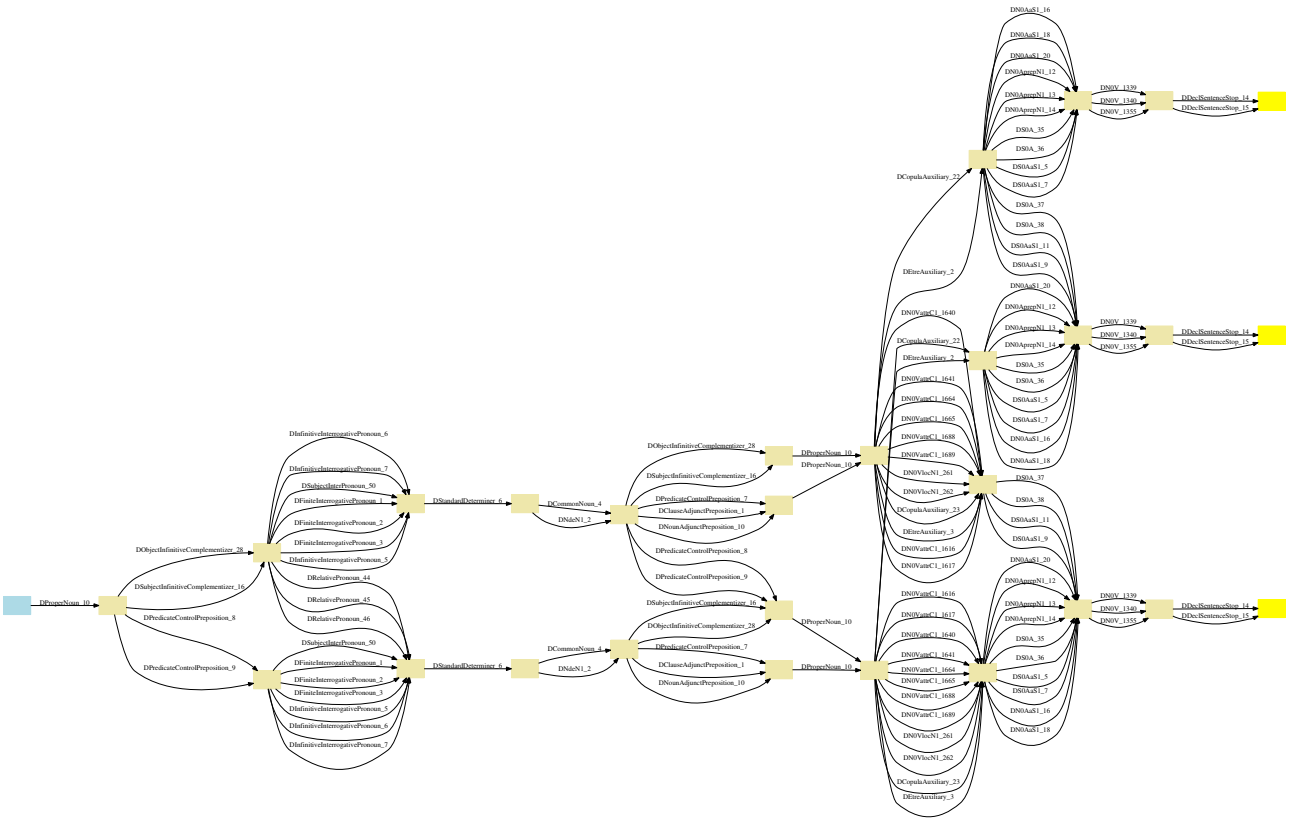


FIG. 6.5 – Automate de filtrage $A(s)$

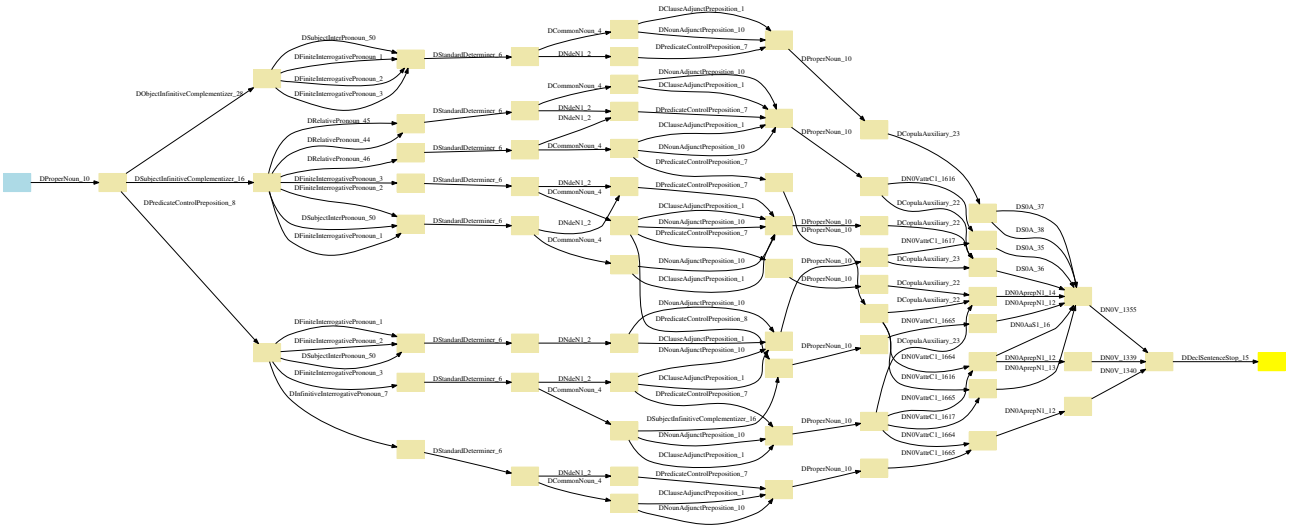


FIG. 6.6 – Automate des sélections saturées

f	*v*	*états*	*chemins*	*arcs*
		12	6021120	67
aux	*etre*	13	564480	58
break	*close*	13	329280	59
mood	*pastp*	13	564480	58
aux	*cop*	14	295680	60
cat	*aux*	14	309120	60
cat	*ap*	14	430080	60
cat	*ceque*	16	423360	63
cat	*n|np*	20	69440	79
cat	*np*	22	76160	82
cat	*n*	22	658560	112
cat	*pp*	25	188640	101
cpl	*?*	25	248640	108
prep	*pour*	26	559776	107
prep	*a*	26	428064	103
cpl	*de*	27	423360	107
prep	*loc*	27	592704	122
prep	*comme*	27	526848	120
cat	*s*	29	278880	125
funct	*aobj*	29	658560	115
funct	*loc*	31	658560	143
funct	*attr*	37	548560	153
funct	*void*	43	658560	195
funct	*subj*	45	658560	192
funct	*adj*	45	658560	218
prep	*de*	45	357504	201
funct	*obj*	47	627200	170
funct	*obl*	76	658560	305
funct	*deobj*	104	658560	513
funct	*obj|aobj|dat|deobj|obl*	122	658560	568
total		995		
automate final		65	95	109

TAB. 6.1 – Tailles des automates de filtrage pour *«Jean de qui la femme de Pierre est amoureuse dort.»*

Nous avons constaté que notre analyseur `leopar` effectue en moyenne, en appliquant la méthode présentée, 30 intersections pour une phrase courte de 12 mots. Cette opération peut prendre quelques secondes et nous constatons parallèlement que certains automates sont redondants. Par exemple nous avons observé qu'un automate $A(\{np, s\})$ ne sert

à rien si les automates $A(np)$ et $A(s)$ ont déjà été utilisés. En revanche, en utilisant uniquement le premier des trois automates, le filtrage est à peine moins efficace qu'en utilisant l'intersection des deux derniers, mais l'automate produit par intersection est beaucoup plus petit et donc le processus de filtrage se fait plus rapidement. Le choix des valeurs de traits sur lesquels filtrer est donc important et nous y reviendrons.

Nous allons maintenant nous intéresser à la complexité de notre méthode filtrage.

6.5 Complexité de l'intersection des automates de filtrage

La dernière étape de l'algorithme de sélection lexicale présenté plus haut consiste à réaliser l'intersection des automates de filtrage. En moyenne, on constate que l'on réalise l'intersection d'une dizaine d'automates. Il est alors naturel de s'intéresser à la complexité temporelle et spatiale de cette opération. Nous réalisons également la minimisation de l'automate obtenu après intersection. Quel est la taille, c'est-à-dire le nombre d'états de l'automate final ?

Si on note $|A|$ la taille de A, c'est-à-dire le nombre d'états de l'automate, on sait que la taille de l'automate minimal intersection de n automates quelconques est $|A_1 \cap \ldots \cap A_n| \leq |A_1| \times \cdots \times |A_n|$. D'autre part, [YZS94] montre que l'intersection de n automates de taille n peut atteindre $n^n - n + 1$ états.

6.5.1 Rôle de la structure des automates

Cependant, dans le cas qui nous intéresse particulièrement les automates A_i manipulés sont particuliers.

1. ils sont acycliques
2. Ils sont tous structurés en *étages* d'états à une distance de l'état final, ils ont le même nombre d'étages et un arc entre deux étage d'un automate de filtrage se trouve entre les mêmes étages s'il existe dans un autre automate de filtrage.

Pour réaliser l'intersection de deux automates, il faut généralement considérer tous les couples d'états formés à partir d'un état de B et d'un état de C et vérifier qu'ils ont une transition sortante commune et recommencer pour explorer le futur de ces deux états.

Or ici, grâce à la structure particulière de nos automates nous savons que deux états (p_B, q_B) et (p_C, q_C) ne peuvent avoir une transition sortante commune que si les positions des deux états sont les mêmes : $p_B = p_C$. Nous n'avons donc pas à considérer tous les couples d'états mais uniquement les couples d'états de premières composantes équivalentes. La taille de l'intersection est donc inférieure dans le pire des cas au résultat théorique.

Le problème est que dans les algorithmes classiques d'intersection, on ne regarde pas le contenu d'un état pour vérifier sa distance par rapport à l'état initial (ou final). C'est pourquoi, dans le but de profiter de la structure de nos automates tout en appliquant un algorithme classique d'intersection, nous allons marquer les transitions avec l'étage de départ et l'étage d'arrivée de ces transitions. Nous reportons la connaissance des étages

sur les arcs pour que l'intersection tiennent compte de cette information. Par exemple, une transition d_i entre (p_1, q_1) et (p_2, q_2) dans l'automate de sélections sera plutôt marquée (p_1, d_i, p_2).

Ainsi, lors des intersections on force les transitions communes à partir des mêmes étages et à arriver aux mêmes étages. On peut donc prendre en compte la structure particulière de nos automates de filtrage pour raisonner sur la taille de l'intersection.

Proposition 4. *Notons A^i l'ensemble des états de A de la forme (i, q). Soient A_1 et A_2 deux automates de filtrage. Nous avons pour chaque niveau i, l'inégalité*

$$|(A_1 \cap A_2)^i| \leq |A_1^i| \times |A_2^i|$$

Nous pouvons donc borner la taille de l'automate intersection

$$|A_1 \cap A_2| \leq \sum_{i \geq 0} |A_1^i| \times |A_2^i| \leq |A_1| \times |A_2|$$

En particulier, pour certaines valeurs de trait v, les automates de filtrage sont *plats*, c'est-à-dire que chaque état à un seul successeur (sauf l'état final, bien sûr, qui n'en a pas) et donc $|A^i| = 1$. Dans ce cas, si A_1 est plat $|A \cap A_1| \leq |A|$, le filtrage par A_1 ne fait qu'enlever des transitions mais n'ajoute pas d'états à A. Plus précisément, si A_1 et A_2 sont plats

$$|A_1 \cap A_2| = \begin{cases} |A_1| & \text{si } L(A_1) \cap L(A_2) \neq \emptyset \\ 0 & \text{sinon} \end{cases}$$

Cette dernière observation nous pousse à distinguer les automates de filtrage plats des autres automates de filtrage. L'intersection des premiers se fait en temps linéaire et en espace constant. Nous effectuons donc nos intersections en deux passes. la première réalise l'intersection de tous les automates plats en parallèle. Nous traitons les autres automates dans une deuxième phase en effectuant les intersections des automates deux à deux.

6.5.2 Importance de l'ambiguïté lexicale

Nous pouvons également tenir compte de l'ambiguïté de chaque mot de l'automate de selection pour borner la taille de l'automate intersection. Cette borne est très pessimiste puisqu'elle suppose que les automates de filtrages ne permettent pas de réduire significativement le nombre de chemins par rapport à l'automate de sélections mais qu'en plus ils augmentent le nombres de d'états.

Soit a_i le nombre d'arcs sortant de l'état $i-1$ de l'automate de sélections. Dans le cas d'une segmentation unique de la phrase, ceci correspond à l'ambiguïté lexicale, c'est-à-dire au nombre de DAP associées au mot à la position i par le lexique de la GI. L'automate résultat A étant déterministe, on a

$$|A^i| \leq t(i) = \begin{cases} 1 & \text{si } i = 1 \\ a_i \times t(i-1) & \text{sinon} \end{cases} \tag{6.1}$$

Or l'automate est déterministe et minimal. En particulier il a un seul état final, ce qui contraint la taille de l'étage terminal à 1 état et de proche en proche de tous les étages précédents. On par exemple pour l'étage $n-1$ le résultat $|A^{n-1}| \leq 2^{a_{n-1}} - 1$. Comme l'automate est minimal, deux états à l'étage $n-1$ se différencient par les transitions sortantes vers l'état final. Il y a $2^{a_{n-1}}$ sous-ensembles différents de transitions sortantes mais l'ensemble vide est interdit. Pour les étages précédents, il faut prendre en compte le fait qu'il y a plusieurs états destinations et que les états d'un étage se distinguent les uns des autres par les combinaisons d'arcs sortants vers les états de l'étage suivant. [Tap97] donne le résultat suivant :

$$|A^i| \leq t'(i) = \begin{cases} 1 & \text{si } i = n \\ \sum_{k=0}^{k=a_i} \binom{a_i}{k} \times t(i+1)^k & \text{sinon} \end{cases} \tag{6.2}$$

On a donc par la proposition 4 et les équations 6.1 et 6.2 ci-dessus le résultat suivant pour la taille de chaque niveau l'automate final A construit à partir des automates de filtrage $A_1, \ldots, A_n$:

$$|A^i| \leq \min(t(i), t'(i), |A_1^i| \times \cdots \times |A_n^i|)$$

En conséquence nous pouvons borner la taille de l'automate de la manière suivante qui fait apparaître l'étage k à partir duquel $t'(k) \leq t(k)$:

$$\begin{array}{rcl} |A| & \leq & \sum_{i=0}^{i=n} \min(t(i), t'(i), |A_1^i| \times \cdots \times |A_n^i|) \\ & \leq & \sum_{i=0}^{i=k} \min(t(i), |A_1^i| \times \cdots \times |A_n^i|) + \sum_{i=k+1}^{i=n} \min(t'(i), |A_1^i| \times \cdots \times |A_n^i|) \end{array}$$

où k est une position de la phrase à partir de laquelle la taille des niveaux diminue, $t(k) \leq t'(k)$ et $t'(k+1) \leq t(k+1)$.

Nous avons donc montré que dans le cas des automates de filtrages, la complexité en états de l'intersection est inférieure à la complexité en états de l'intersection d'automates déterministes quelconques. Pour le montrer, nous avons fait appel d'une part au fait que les automates de filtrages sont acycliques et déterministes et d'autres part qu'ils ont une structure en niveaux. Intuitivement, ils ont une forme *en diamant* puisqu'ils sont minimaux et acycliques. De l'état initial à l'état k les étages sont de plus ne plus grand et à partir de $k+1$ la taille des niveaux diminue jusqu'au dernier niveau n qui ne contient qu'un état final.

Cette borne est assez importante mais elle néglige certains aspects que nos grammaires exhibent en pratique. En particulier, les bilans de polarités sont toujours compris dans des intervalles assez petits (disons $[-3, 3]$, voire moins pour de nombreux traits) et que les variations de polarités pour certains traits correspondent toujours à des schémas assez stricts (par exemple pour certaines valeurs comme n, la catégorie des noms communs, on assiste toujours à une demande, -1, formulée par un déterminant ou la copule, suivie d'un plateau, 0, qui correspond à une suite d'adjectifs, puis de la ressource,+1 qui correspond au nom commun proprement dit). Mais ces aspects sont très liés à la grammaire et non au formalisme proprement dit. C'est pourquoi nous n'en parlons pas ici.

Nous connaissons maintenant une borne à la taille de l'automate résultat du filtrage. Cette intersection se fait en construisant de nombreux automates intermédiaires. Une

question se pose alors : existe-t'il un ordre plus intéressant qu'un autre dans lequel effectuer ces intersections ?

6.6 NP-Complétude de l'optimisation d'intersection

Nous allons nous intéresser maintenant au problème du choix de l'ordre dans lequel effectuer les intersections des automates de filtrage. Nous allons donner un exemple, issu de la désambiguïsation lexicale telle qu'elle a été montrée au chapitre précédent, pour montrer l'importance de l'ordre des intersection même sur une phrase très courte.

Ensuite, nous nous intéresserons non pas au problème qui consiste à trouver l'ordre optimal mais au problème de décision sous-jacent. c'est-à-dire que l'on fixe une borne et on se demande si l'on peut atteindre cette borne. Nous montrons que ce dernier est NP-complet, ce qui implique que le problème d'optimisation est NP-difficile. La démonstration est assez technique mais elle consiste en une réduction polynomiale du *problème du voyageur de commerce* (cf. [GJ79]) dans notre problème.

6.6.1 Exemple

Au chapitre précédent, nous avons appliqué notre méthode de désambiguïsation à la phrase *Jean de qui la femme de Pierre est amoureuse dort.*, dont toutes les sélections avant filtrage sont représentées sur la figure 6.4. Le résultat du filtrage est visible sur la figure 6.6.

La table 6.2 illustre l'importance du choix de l'ordre dans lequel sont faites les intersections. Les deux premières colonnes correspondent au choix du nom et de la valeur de trait. Les 3 colonnes suivantes indiquent la taille de l'automate selon le nombre d'états (qui est la seule notion qui nous intéressera par la suite), le nombre de chemins acceptants et le nombre d'arcs. Les deux dernières colonnes donnent le nombre d'états de l'automate intersection, pour la cinquième colonne depuis la première ligne et pour la sixième colonne depuis la dernière ligne[39].

Les deux dernières lignes indiquent les tailles cumulées des automates intermédiaires. Le choix de l'ordre permet de multiplier/diviser le nombre d'états intermédiaires créés par $2,6$. Dans des phrases plus longues, il n'est pas rare de voir des écarts de 1 à 10.

Il est donc important d'essayer d'effectuer les intersections dans le meilleur ordre possible.

6.6.2 Problème du voyageur de commerce

Nous allons prouver la NP-difficulté de notre problème par réduction polynomiale du problème du voyageur de commerce (*traveling salesman problem* dans [GJ79], ou TSP).

Une instance du TSP est un triplet (V, d, K) où $V = \{1, \ldots, n\}$ est un ensemble de villes, d la fonction de distance définie pour toute paire de villes différentes $d(i, j) \in \mathbb{N}^+$ et une borne $K \in \mathbb{N}^+$. Le problème consiste à décider s'il existe un circuit passant par

[39]Nous n'avons pas minimisé les automates intermédiaires. Mais on retrouverait des proportions semblables entre les deux séquences d'intersections si nous avions effectuer les minimisations.

f	v	*états*	*chemins*	*arcs*	*intersection* ↓	*intersection* ↑
		12	6021120	67	12	337
aux	*etre*	13	564480	58	13	337
break	*close*	13	329280	59	13	337
mood	*pastp*	13	564480	58	14	337
aux	*cop*	14	295680	60	30	337
cat	*aux*	14	309120	60	47	361
cat	*ap*	14	430080	60	55	481
cat	*ceque*	16	423360	63	53	481
cat	*n|np*	20	69440	79	53	2196
cat	*np*	22	76160	82	53	1487
cat	*n*	22	658560	112	51	2456
cat	*pp*	25	188640	101	72	3021
cpl	*?*	25	248640	108	72	3021
prep	*pour*	26	559776	107	65	2552
prep	*a*	26	428064	103	239	2448
cpl	*de*	27	423360	107	351	2448
prep	*loc*	27	592704	122	713	1729
prep	*comme*	27	526848	120	1037	1729
cat	*s*	29	278880	125	1089	1143
funct	*aobj*	29	658560	115	1128	1143
funct	*loc*	31	658560	143	1167	1143
funct	*attr*	37	548560	153	1167	666
funct	*void*	43	658560	195	1337	666
funct	*subj*	45	658560	192	1337	468
funct	*adj*	45	658560	218	567	253
prep	*de*	45	357504	201	394	123
funct	*obj*	47	627200	170	394	96
funct	*obl*	76	658560	305	337	75
funct	*deobj*	104	658560	513	337	33
funct	*obj|aobj|dat|deobj|obl*	122	658560	568	337	122
total		995			12535	31930
automate final (min)		65	95	109		

TAB. 6.2 – Tailles des intersections pour *«Jean de qui la femme de Pierre est amoureuse dort.»*

toutes les villes et revenant à la ville de départ de longueur plus petite que K ou, plus formellement, s'il existe une permutation π des villes telle que $(\sum_{i=1}^{i=n-1} d(\pi(i), \pi(i+1))) + d(\pi(n), \pi(1)) \leq K$.

Nous distinguons du TSP traditionnel tel qu'il vient d'être exposé, une variante dans

laquelle le tour doit être de longueur exactement K. Nous appelons cette variante E-TSP.

6.6.3 Énoncé des problèmes

Nous présentons un premier problème d'optimisation d'intersection que nous enrichirons pour obtenir les second et troisième problèmes. Ce dernier correspond à notre problème de filtrage.

Problème 1. ***(IO1)*** *Soient $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ un ensemble de n automates , $B \in \mathbb{N}^+$ une borne K la taille ciblée. Existe-t'il une injection $\pi : [1..j] \to [1..n]$ telle que*
- $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| = K$
- *pour tout $k < j$, $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(k)}| \leq B$.*

En d'autres termes, peut-on trouver un sous-ensemble $\mathcal{A} \subseteq \mathcal{A}_n$ tel que $|\bigcap_{A\in\mathcal{A}} A| = K$ et toutes les étapes intermédiaires plus petites que B ?

Problème 2. ***(IO2)*** *Soient $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ un ensemble de n automates et $B \in \mathbb{N}^+$ une borne. Peut-on trouver une bijection π $[1..n] \to [1..n]$ telle que pour tout $j \leq n$ on ait $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?*

Problème 3. ***(IO3)*** *Soient $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ un ensemble de n automates et $B \in \mathbb{N}^+$ une borne. Existe-t'il une bijection π $[1..n] \to [1..n]$ telle que $\sum_{1\leq j\leq n} |(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?*

Si nous nous ramenons à notre problème de désambiguïsation, des problèmes peuvent se reformuler de la manière suivante :
- (IO1) Étant donné un entier (K), est-ce que l'automate final résultat de la désambiguïsation peut être de cette taille ? Dans ce problème, on se donne aussi une borne sur la taille des automates intermédiaires, qui peut ici correspondre à la borne que nous avons donnée dans la section précédente (B).
- (IO2) Étant donné un entier, est-ce que les automates intermédiaires produits par intersections successives peuvent ne jamais dépasser cet entier ?
- (IO3) Étant donné un entier, est-ce que le nombre total d'états créés durant les n intersections successives peut être inférieur à cet entier ?

Le premier problème illustre la technique employée dans les réductions suivantes pour compter la longueur d'un tour. La deuxième reprend les idées précédentes et ajoute une contrainte sur l'ordre d'utilisation des automates. Toute la difficulté réside dans cette deuxième réduction. Finalement, la troisième réduction n'est qu'une adaptation de la précédente où une partie de l'automate qui compte la longueur du tour est effacée à chaque intersection.

Le dernier problème est le problème de décision sous-jacent au problème d'optimisation qui nous intéresse pour le filtrage. En effet, nous voulons connaître le meilleur ordre dans lequel réaliser l'intersection des automates de filtrage. Notre problème est donc au moins aussi difficile que **IO3**.

6.6.4 NP-Complétude

Nous nous bornons ici à donner les résultats. Les preuves sont données complètement en annexe A.

Proposition 5. ***(IO1)*** *est NP-complet.*

Proposition 6. ***(IO2)*** *est NP-complet.*

Proposition 7. ***(IO3)*** *est NP-complet.*

Ce dernier problème correspond à notre méthode de sélection lexicale. Si le problème de décision (avec une borne B donnée) est NP-complet alors trouver la meilleure borne (la plus petite) est encore plus difficile.

6.6.5 Conclusion

Dans cette section ainsi que dans la précédente, nous avons montré des résultats plutôt négatifs :

- d'une part que les automates que nous manipulons sont particuliers. Nous pouvons tirer profit de cette observation et développer des techniques plus efficaces pour effectuer l'intersection. Cependant, la taille de l'automate résultat reste élevée dans le pire des cas
- que la technique globale qui consiste à enchaîner les intersections est intrinsèquement difficile.

Nous avons donc intérêt à développer des heuristiques qui permettent de réduire la taille des automates de filtrages et de réduire le nombre d'automates de filtrage. C'est ce que nous allons développé dans la suite.

6.7 Choix des valeurs de traits pour le filtrage

Deux phénomènes rendent difficiles l'intersection de nos automates : la taille de ces automates (section 6.5) et le nombre de ces automates (section 6.6). Nous allons montrer dans cette section qu'en choisissant soigneusement les valeurs de traits pour construire les automates de filtrage, on peut à la fois diminuer la taille et le nombre des automates. Il y a évidemment un prix à payer. Le critère de correction sera plus lâche et des sélections non saturées seront tout de même retenues.

Pour savoir quels automates de filtrage construire, nous avons proposé de prendre les valeurs de traits présentes avec les polarités $\rightarrow$ et $\leftarrow$ dans les DAP de l'automate de sélections. Beaucoup de valeurs de traits différentes sont présentes dans cet automate. Nous créons donc de nombreux automates de filtrages plus ou moins redondants et effectuons l'intersection de ces automates. Nous avons donc intérêt à choisir notre ensemble de valeurs qui servira de support au filtrage pour qu'il y ait peu d'automates et que ces automates soient petits. Dans [BLRP06] est proposé le choix suivant de valeurs de trait. Ces valeurs sont construites à partir des valeurs présentes dans l'automate de sélections.

Nous partons de l'observation suivante : d'après la définition des bilan de polarités, pour deux valeurs de traits v_1 et v_2, si $v_1 \subseteq v_2$ alors pour toute DAP D on a $p_D(v_2) \subseteq p_D(v_1)$. En effet plus v, l'ensemble de valeurs atomiques que l'on passe en argument de la fonction p_D, est grand, plus il y a de chance que D contiennent des traits dont toutes les valeurs appartiennent à v, ce qui implique que la valeur de $p_D(v)$ est un singleton. À la limite, $p_D(?)$ est un singleton pour toute DAP D. Mais il existe d'autres valeurs intermédiaires pour lesquelles on obtient aussi des singletons.

Quel est l'intérêt d'avoir des singletons ? Supposons que nos intervalles soient tous bornés, qu'il sont tous inclus dans $[-K; K]$ (ce qui est le cas en pratique, 3 semble être une borne naturelle). Or sur cette partie des entiers $[-K, K]$ il existe évidemment moins de singletons (il y en a $2K + 1$) que d'intervalles en général (il y en a $(K + 1)(2K + 1)$). Nous avons donc intérêt à choisir des valeurs de traits pour lesquelles les états de l'automates sont des singletons. Les automates seront donc petits, et l'automate résultat de l'intersection sera également petit.

Pour illustrer ce point, considérons la figure 6.7. Comparons le bilan de polarité pour les valeurs np, n et $\{np, n\}$.

- $p(np) = [-2; -1]$ n'est pas un singleton
- $p(n) = [-1; 0]$ n'est pas un singleton
- $p(\{np, n\}) = [-2; -2]$ est un singleton

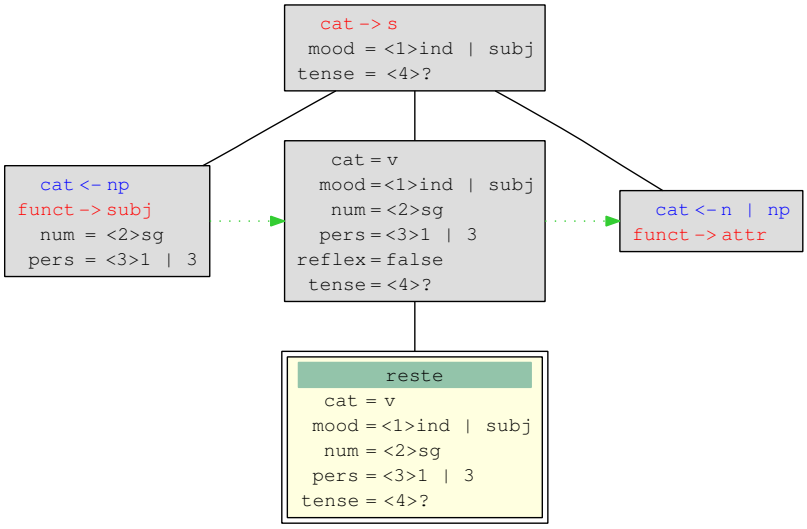


FIG. 6.7 – Une DAP associée à *reste*

On retrouve ces différences au niveau des automates de filtrage des valeurs np, n et $\{np, n\}$ pour la phrase *Jean de qui la femme de Pierre est amoureuse dort.* représentés sur les figures 6.8, 6.9 et 6.10.

L'automate obtenu à la fin de notre méthode de filtrage est identique, que l'on prenne les deux automates $A(n)$ et $A(np)$ ou uniquement l'automate $A(\{n, np\})$. En revanche, utiliser uniquement le dernier automate permet d'arriver plus vite au résultat.

Comment choisir un ensemble de valeurs pour lesquelles bilans de polarités seront le plus souvent des singletons et qui permettent de filtrer efficacement néanmoins ?

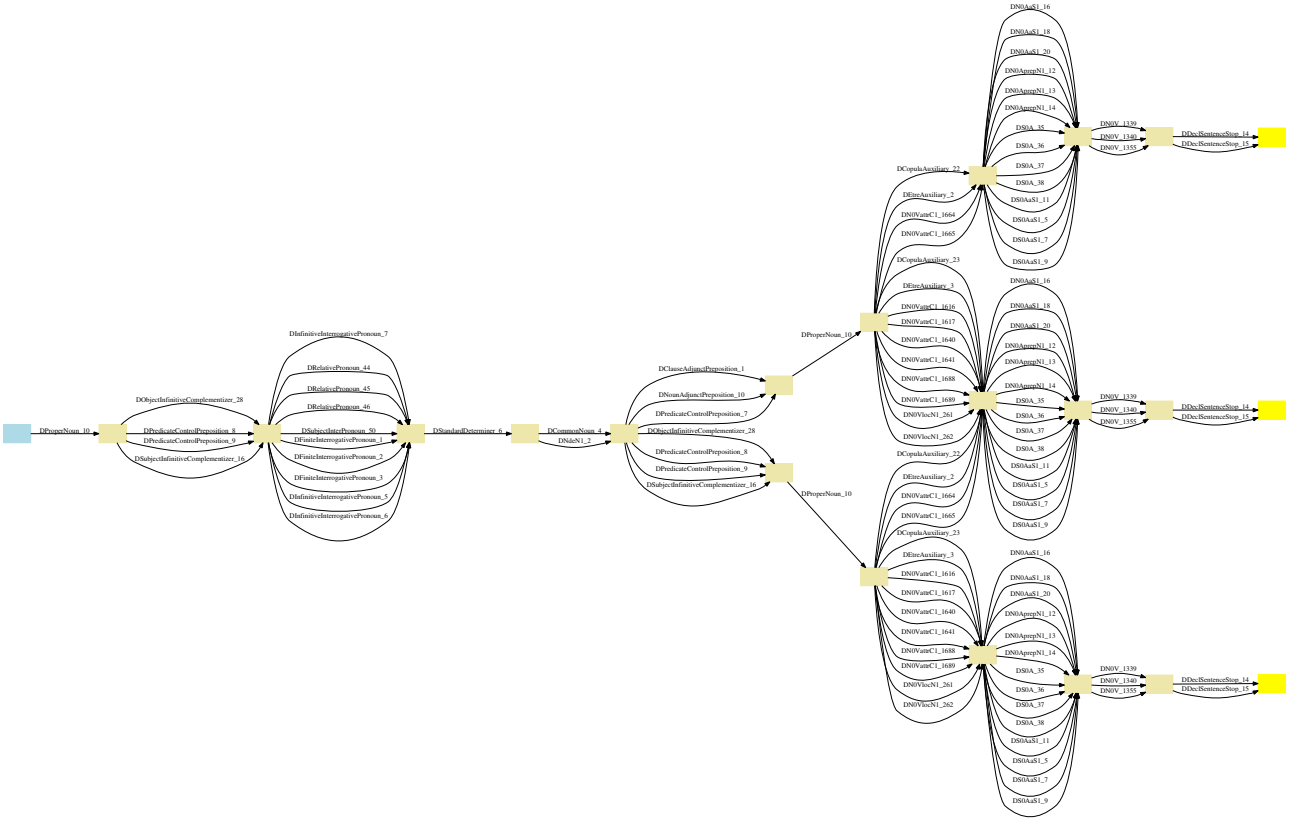


FIG. 6.8 – Automate de filtrage pour n : 22 états, 658560 chemins

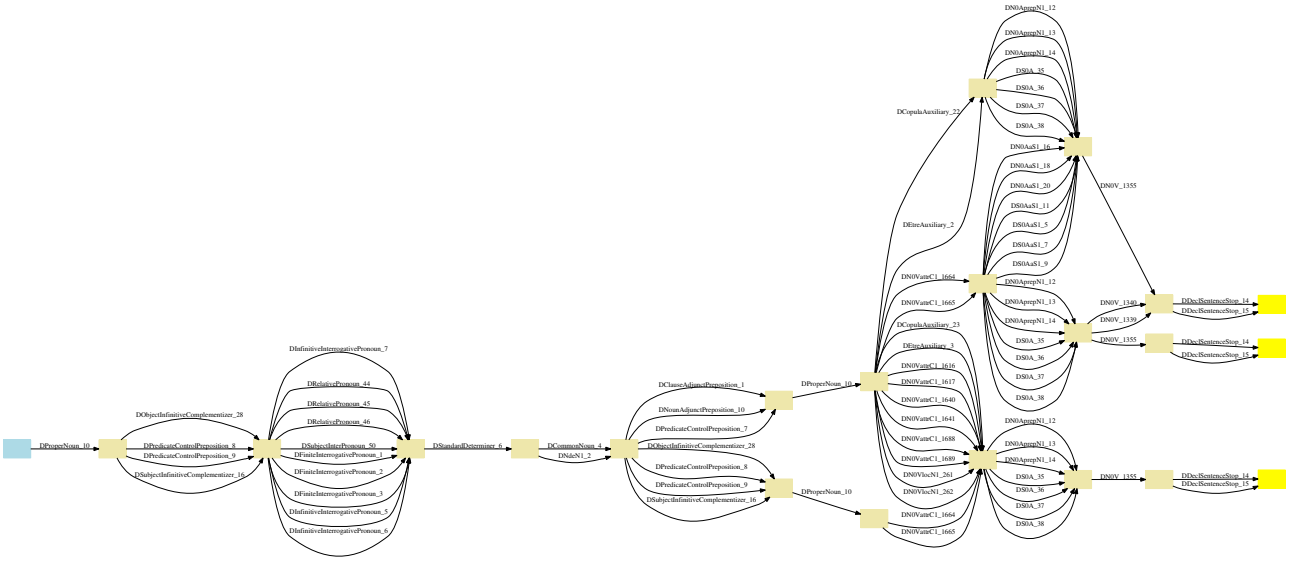


FIG. 6.9 – Automate de filtrage pour np : 22 états, 76160 chemins

Premièrement, notre première observation ci-dessus indique que si des valeurs de traits présentes dans l'automate de sélections sont ordonnées par la relation $\subseteq$ alors on peut se contenter de ne garder que l'élément maximal. On réduira la taille des intervalles et on aura donc plus de chance d'avoir des singletons. Appelons S_{pol} le sous-ensemble des valeurs de traits présentes dans l'automate initial avec les polarités $\rightarrow$ et $\leftarrow$ dont on ne garde que les valeurs maximales par $\subseteq$. Pour réduire encore le nombre de valeurs, considérons $v_1, v_2 \in S_{pol}$. Deux cas se présentent :

1. $v_1 \cap v_2 = \emptyset$. Dans ce cas $p_D(v_1 \cup v_2)$ est un singleton si et seulement si $p_D(v_1)$ et $p_D(v_2)$ sont des singletons. Cette union n'est pas intéressante, car elle ne permet pas de réduire la taille des intervalles.
2. $v_1 \cap v_2 \neq \emptyset$. Dans ce cas, $p_D(v_1 \cup v_2)$ peut être un singleton même si $p_D(v_1)$ ou $p_D(v_2)$ ne l'est pas. Cette union est intéressante car elle permet de réduire la taille des intervalles, donc le nombre d'états de l'automate.

On ajoute donc à S_{pol} les valeurs de traits $v = v_1 \cup v_2$ si $v_1 \cap v_2 \neq \emptyset$ avec $v_1, v_2 \in S_{pol}$. On itère cette procédure jusqu'à stabilisation et on ne garde alors que les valeurs maximales qui sont finalement les bornes supérieures par $\subseteq$ des valeurs présentes dans l'automate de sélections. On peut montrer que dans ce cas les bilans de polarités sont des singletons pour toutes les sélections.

Cette méthode permet de maximiser le nombre d'états singletons, en ne filtrant que sur des traits présents dans l'automate initial. C'est la méthode utilisée dans `leopar`.

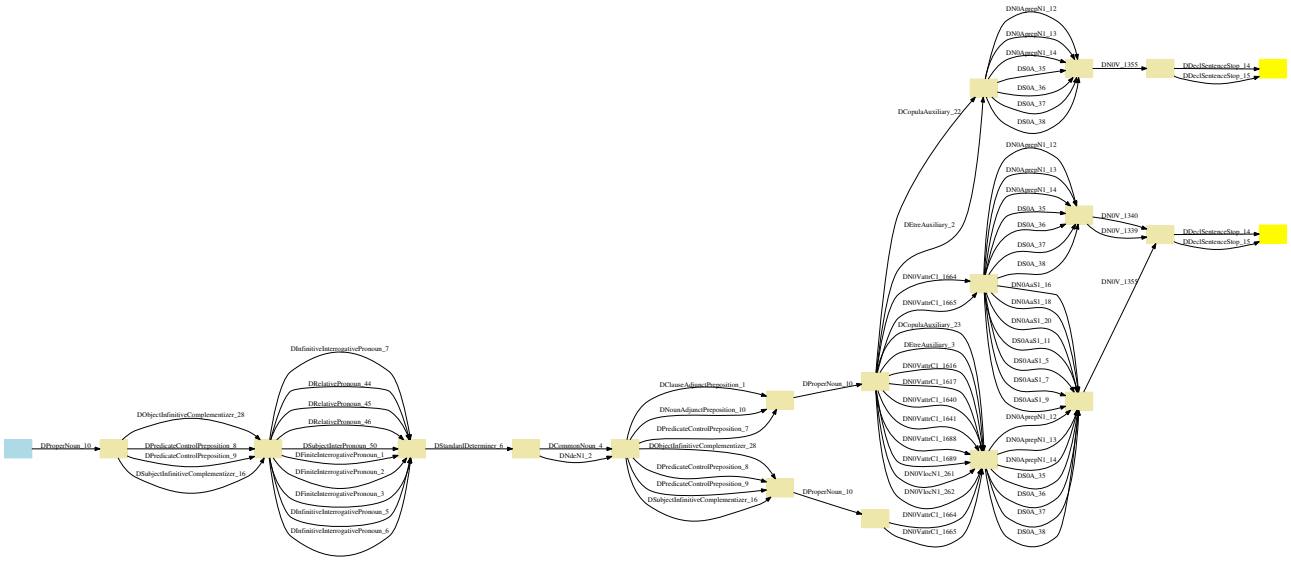


FIG. 6.10 – Automate de filtrage pour $\{np, n\}$: 20 états, 69940 chemins

6.8 Informations syntaxiques et filtrage : le cas de la coordination

Cette méthode de désambiguisation est efficace sur des phrases simples mais dès que le nombre de propositions et de constituants augmente, le nombre de sélections saturées augmente également. A cela s'ajoute le fait que l'ordre des mots n'est pas du tout pris en compte par notre algorithme. En particulier, dans des phrases avec coordination, on assiste à une dégradation des résultats de notre méthode. Nous allons tirer profit de la modélisation syntaxique de la coordination que nous proposons au chapitre précédent pour améliorer le filtrage des phrases qui contiennent une ou des coordinations.

Nous avons insisté sur le fait que les DAP associées aux conjonctions de coordination exhibaient la symétrie entre les deux conjoints, notamment la symétrie des interfaces des DAP associées aux conjoints. Nous allons donc repérer les transitions étiquetées par une DAP associée aux conjonctions de coordination dans les automates de filtrage et vérifier qu'immédiatement à gauche et immédiatement à droite on trouve bien des segments de même bilan de polarités. Si de tels segments n'existent pas, nous pouvons retirer la transition de l'automate. Nous ne considérons pas les coordinations avec *gapping* puisque dans ce cas il n'y a pas symétrie. Nous allons prendre comme exemple la phrase *Je persuade Jean et la femme de Pierre de venir.* On coordonne dans ce cas *Jean* et *la femme de Pierre*.

6.8.1 Un critère sur les sélections

Soit $d_1, \ldots, d_i, \ldots, d_p$ une sélection pour $w_1, \ldots, w_i, \ldots, w_p$ où w_i est une conjonction de coordination et d_i une DAP pour la coordination. Dans ce cas, nous l'avons vu au chapitre précédent, les deux conjoints doivent être immédiatement à gauche et à droite et avoir le même interface. C'est bien le cas dans notre exemple : *Jean* et *la femme de Pierre* sont immédiatement à gauche et à droite de la conjonction de coordination *et*.

Or une condition nécessaire pour que deux suites de DAP aient la même interface est que leurs bilans de polarité aient une intersection non vide. C'est le cas ici. *Jean* et *la femme de Pierre* sont équivalents à des syntagmes nominaux. *la femme de Pierre* peut former une seule entité qui est considérée comme un syntagme nominal.

Dans le cas présent, il doit exister deux positions dans la phrases c_1 et c_2 telles que la première est avant la coordination $1 \leq c_1 \leq i-1$, la seconde est après la coordination $i+1 \leq c_2 \leq p$ et pour tout trait v les bilans de polarités sont compatibles $p_{d_{c_1},\ldots,d_{i-1}}(v) \cap p_{d_{i+1},\ldots,d_{c_2}}(v) = C \neq \emptyset$. Intuitivement, les bilans de polarités devraient être égaux pour les deux conjoints. Mais comme les valeurs de traits peuvent être des ensembles, il se peut que seules des parties de ces ensembles correspondent. C'est pourquoi, la condition d'égalité est relâchée. On se contente alors d'une intersection non vide. C'est encore le cas ici dans notre exemple. Pour le trait *cat* et la valeur *np*, les deux groupes *Jean* et *la femme de Pierre* ont exactement le même bilan de polarité $[+1; +1]$.

Cela pourrait nous donner un premier critère. En effet, si on ne peut pas trouver c_1 et c_2 telles que la l'égalité ci-dessus puisse être vérifiée alors il n'existe pas de coordination possible dans la phrase. Le problème de ce critère est qu'il est trop grossier. On ne distingue

pas les DAP pour la conjonction de coordination. Or ce qui nous intéresse ici, c'est bien de savoir lesquelles de ces DAP ne peuvent pas être utilisées. Dans notre exemple, on sait que *Jean* et *la femme de Pierre* peuvent être coordonner mais on aimerait également indiquer que c'est une coordination de syntagmes nominaux.

Nous pouvons aller plus loin en remarquant qu'il existe une relation entre l'intervalle C (l'intervalle commun des bilans de polarités des conjoints) et le bilan de polarité de la DAP à utiliser pour modéliser la conjonction de coordination. On pourra ainsi vérifier la validité de la DAP d_i au cas par cas.

La DAP d_i associée à la conjonction de coordination est composée de deux parties basses et d'une partie haute. Les deux premières doivent neutraliser les conjoints et ont pour duale la partie haute. On peut décomposer le bilan de polarité de d_i selon ces trois parties où pb est le bilan de polarité d'une partie basse, ph ce lui de la partie haute :

$$\begin{aligned} p_{d_i}(v) &= pb_{d_i}(v) + pb_{d_i}(v) + ph_{d_i}(v) \\ &= pb_{d_i}(v) + pb_{d_i}(v) - pb_{d_i}(v) \\ &= pb_{d_i}(v) \end{aligned}$$

La partie basse doit neutraliser chaque conjoint, en d'autres termes $0 \in p_{d_{c_1},\ldots,d_{i-1},d_i}(v)$ pour le premier conjoint et $0 \in p_{d_i,d_{i+1},\ldots,d_{c_2}}(v)$ pour le second. On peut transformer ces égalités pour obtenir l'allure générale des bilans de polarités des conjoints, qui sont des intervalles :

$$\begin{aligned} 0 \in p_{d_{c_1},\ldots,d_{i-1},d_i}(v) &\iff p_{d_{c_1},\ldots,d_{i-1},d_i}(v) = [a_1; b_1] \text{ où } a_1 \leq 0 \leq b_1 \\ 0 \in p_{d_i,d_{i+1},\ldots,d_{c_2}}(v) &\iff p_{d_i,d_{i+1},\ldots,d_{c_2}}(v) = [a_2; b_2] \text{ où } a_2 \leq 0 \leq b_2 \end{aligned}$$

Ce qui est important de remarquer, c'est que ces deux intervalles ont des bornes autour de zéro. Si on les additionne avec un intervalle quelconque $[c; d]$, on obtient un intervalle plus grand : $[c; d] \subseteq [a_i; b_i] + [c; d]$. Cette observation nous permet alors d'écrire :

$$\begin{aligned} 0 \in p_{d_{c_1},\ldots,d_{i-1},d_i}(v) &\implies p_{d_1,\ldots,d_{c_1}}(v) \cap p_{d_1,\ldots,d_i}(v) \neq \emptyset \\ 0 \in p_{d_i,d_{i+1},\ldots,d_{c_2}}(v) &\implies p_{d_1,\ldots,d_{c_2}}(v) \cap p_{d_1,\ldots,d_{i-1}}(v) \neq \emptyset \end{aligned}$$

Dans notre exemple, le bilan de polarité des DAP associées à *Jean* et *et* dans le cadre d'une coordination de syntagmes nominaux (que nous avons déjà vue sur sur la figure 5.6) pour le trait *cat* et la valeur *np* est $[0; 0]$. De la même façon pour *et la femme de Pierre*, le bilan de polarité est également nul. Pour d'autres DAP de conjonctions de coordination comme celles d'une montée de nœud par exemple, 0 n'appartiendrait pas au bilan de polarité.

On a donc maintenant un critère qui permet de filtrer les DAP associées aux conjonctions de coordination. Si on ne trouve pas dans une sélection lexicale les positions c_1 et c_2 pour lesquelles l'inégalité ci-dessus est vraie, on peut rejeter la sélection, en particulier rejeter le choix de la DAP pour la conjonction de coordination. De plus c_1 et c_2 permettent de délimiter groupe coordonné. Si nous reprenons notre critère, on voit que pour une coordination de syntagmes nominaux, cette méthode donnerait pour le premier conjoint une unique solution *Jean* et pour le second conjoint deux solutions *la femme* et *la femme de Pierre*.

6.8.2 Un critère sur les automates

Ce critère sur les sélections se transpose facilement en un critère sur les automates si on se rappelle que $p_{d_1,\ldots,d_k}(v)$ est l'intervalle d'un état de l'automate de filtrage $A(v)$. Donc, dans chaque automate de filtrage $A(v)$, s'il existe une chemin de l'état initial à l'état final $(s_0, d_0), \ldots, (s_{i-1}, d_{i-1})(s_i, d_i)(s_{i+1}, d_{i+1}) \ldots, (s_n, d_n)$ et que d_i est une DAP associée à une conjonction de coordination alors il doit exister deux états $c_1 \in \{s_0 \ldots s_{i-1}\}$ et $c_2 \in \{s_i \ldots s_{n+1}\}$ où c_1 est le début du premier conjoint et c_2 est la fin du second conjoint (c_1 et c_2 délimitent le groupe coordonné) tels que $c_1 = (p_1, I_1)$ $c_2 = (p_2, I_2)$, $s_i = (p_i, I_i)$ et $s_{i+1} = (p_{i+1}, I_{i+1})$. Si d_i est une DAP de conjonction de coordination valide alors

$$I_1 \cap I_{i+1} \neq \emptyset$$
$$I_2 \cap I_i \neq \emptyset$$

Ces deux conditions doivent être vérifiées par les automates de filtrage $A(v)$. Si elles ne le sont pas c'est que d_i ne pourra pas être utilisée dans une analyse ultérieure. On peut donc l'enlever de l'automate $A(v)$. Par intersection avec les autres automates de filtrage, elle disparaîtra de l'automate des sélections saturées.

De plus, cette méthode permet de donner les triplets (d_i, c_1, c_2) de positions pouvant délimiter les positions du groupe coordonné par la DAP d_i. Cette possibilité nous permettra d'envisager des améliorations des algorithmes d'analyse syntaxique au chapitre 7.

6.9 Patrons interdits

Dans les deux sections précédentes nous avons vu que

- le filtrage par polarité est très efficace pour des phrases simples (une seule proposition) mais que l'ordre des mots n'est pas pris en compte, ce qui peut poser des problèmes lors du filtrage de phrases complexes (qui contiennent plusieurs propositions)
- certaines connaissances syntaxiques peuvent être exploitées au moment du filtrage pour filtrer plus efficacement, notamment la modélisation d'un phénomène comme la coordination. Nous avons exploité l'ordre des mots et les polarités mises en jeu.

En particulier, dans la section précédente, nous avons tenu compte de l'ordre des mots qui est contenu dans l'automate. C'est l'ordre des transitions d'un chemin.

Nous pouvons généraliser cette approche par la définition de ce que nous appelons des patrons interdits. Un patron interdit est une expression rationnelle qui dénote des suites de DAP interdites dans une phrase bien formée.

Prenons comme exemple la phrase *«Le chat la mange.»* Le filtrage par polarité ne peut pas distinguer l'article de l'objet clitique, pour *le* comme pour *la*. On peut voir l'automate de filtrage sur la figure 6.11.

Or nous pourrions postuler qu'un pronom clitique n'est jamais suivi par un nom commun. Ainsi, *le* ne pourrait pas être étiqueté comme clitique puisqu'il est suivi de *chat* dont la seule catégorie possible est celle de nom commun. Si on procède alors au filtrage par polarités, *la* est donc obligatoirement un pronom clitique pour satisfaire le critère de saturation. L'automate de filtrage serait alors celui de la figure 6.12. Cette technique n'est

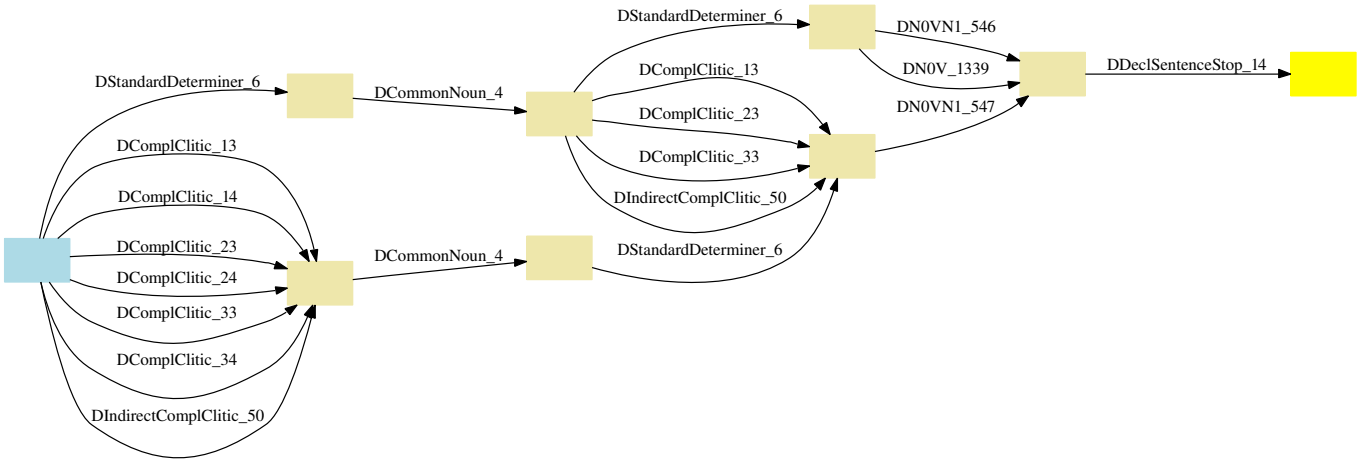


FIG. 6.11 – Automate de filtrage par polarités de *Le chat la mange.*

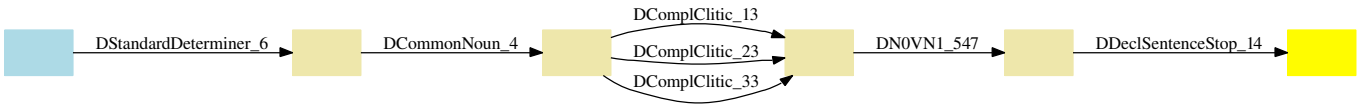


FIG. 6.12 – Automate de filtrage par polarités et patrons interdits de *Le chat la mange.*

efficace qu'en conjonction avec le critère de saturation. Elle *nettoie* l'automate initial de certaines aberrations qui pénaliseraient l'application du critère de saturation globale.

Les patrons interdits que nous utilisons sont cependant différents de ce premier exemple. Dans la plupart des cas, ils stipulent des propriétés sur les DAP et non pas sur les mots proprement dits comme dans l'exemple que nous avons donné. Par exemple, on peut imposer que la DAP d'un nom prédicatif avec complément de la forme «préposition *à* suivie d'un syntagme nominal» soit suivi obligatoirement par une DAP de la préposition «à» sélectionnant un complément syntagme nominal.

Cette technique est proche de l'analyse dans les grammaires d'intersection [Tap97] où chaque règle de la grammaire est une expression rationnelle. Une phrase doit respecter toutes les règles modélisées sous la forme de l'intersection des expressions rationnelles.

6.10 Résultats expérimentaux

Comme pour la modélisation, nous avons effectué des mesures de l'efficacité notre méthode de filtrage en prenant comme référence la TSNLP.

Nous avons effectué les tests sur les phrases de la suite par longueur de phrase croissante, sur des phrases sans coordination dans un premier temps. Les résultats sont visibles sur la table 6.3. Dans la première colonne, nous donnons la longueur des phrases et le nombre de phrases de cette longueur dans la TSNLP. Dans la deuxième colonne se trouve le

nombre de sélections moyen pour une phrase de la longueur correspondante. On constate que le nombre de sélections par phrase augmente de manière exponentielle dans la longueur des phrases.

Grâce à ces deux premières colonnes, nous pouvons calculer l'ambiguïté moyenne par mot. On trouve entre 5 et 6 DAP en moyenne par mot, quelle que soit la longueur des phrases. Nous rappelons que pour une conjonction de coordination, on associe 50 DAP. C'est pourquoi nous avons dans un premier temps illustré les résultats sur des phrases sans coordination. Les temps donnés sont les temps moyens de filtrage par phrase en millisecondes.

6.10.1 Importance du choix des valeurs

Ensuite nous donnons le nombre de sélections retenues selon divers moyens. Nous séparons les cas où nous avons utilisé les patrons interdits, qui sont placés dans les dernières colonnes. Nous avons également divisé les résultats selon la méthode de choix des valeurs de traits pour construire les automates de filtrage.

- Le choix (1) consiste à prendre toutes les valeurs de traits présentes sur l'automate de sélections.
- Le choix (2) consiste à prendre les valeurs de traits qui ne sont pas déjà incluses dans une autre valeur de traits. Si nous reprenons la terminologie de la section 6.7, on se contente de prendre S_{pol}
- Le choix (3) consiste à utiliser l'heuristique que nous avons présentée pour maximiser le nombre de singletons dans les bilans de polarités.

Il est clair que les deux premières méthodes filtrent beaucoup plus que la troisième. Il n'y a quasiment pas de différence entre les deux premières méthodes. Mais il faut ramener ce nombres de sélections retenues au temps écoulé pour filtrer. Le nombre de solutions saturées retenues par (3) passe de 1,4 à 7,27 fois le nombre de solutions retenues par (1). Mais dans le rapport des temps varie de 0,5 à 0,14. On peut conclure que la méthode proposée pour choisir les valeurs de traits de manière à maximiser le nombre de singletons pour les bilans de polarité atteint son but : améliorer la vitesse du filtrage sans trop dégrader les résultats de filtrage.

6.10.2 Rôle des patrons

Ensuite on peut essayer d'estimer l'impact qu'a l'utilisation des patrons interdits pour enlever des chemins de l'automate sans tenir compte des polarités. Nous avons utilisé comme seul patron interdit, celui que nous avons présenté dans la section 6.9 qui stipule qu'il ne peut y avoir de pronom clitique objet devant un nom commun. Sur les phrases courtes l'utilisation des patrons a très peu d'impact. Le filtrage par polarité suffit. Mais à mesure que les phrases s'allongent, l'ordre des mots amène trop d'ambiguïté dans notre critère de saturation. Et on voit très vite (dès les phrases de longueur 4) que même en utilisant la méthode (3) avec les patrons, on obtient de bien meilleurs résultats qu'avec la méthode (1) sans les patrons, et en un temps bien moins grand.

		nb. de sél. retenues (sans patrons)						nb. de sél. retenues (avec patrons)					
longueur	sélections	(1)	tps	(2)	tps	(3)	tps	(1)	tps	(2)	tps	(3)	tps
2 (122)	136,3	2,7	1,98	2,7	1,94	3,7	1,15	2,6	2,7	2,6	2,04	3,7	1,1
3 (225)	1111	9,5	6,3	9,5	6,3	15,7	3,4	8,4	5,9	8,4	5,9	12,5	3,3
4 (153)	3848,8	9,7	11,5	9,7	11,5	21,4	5,6	7,05	9,0	7,05	9,1	10,4	5,0
5 (223)	27909,4	26,2	18,3	26,2	18,3	55,4	6,4	14,0	10,6	14,0	10,8	18,0	4,9
6 (157)	124626,6	35,1	21.8	35,2	21,4	316,0	6,7	11,9	9,6	11,9	9,4	18,6	4,6
7 (142)	1758409,0	143,5	66,4	143,8	65,7	1253,8	15,4	39,2	31,4	39,2	31,2	70,6	12,2
8 (100)	1650811,9	259,5	70,8	259,7	70,5	1968,0	15,7	30,0	24,7	29,9	25,7	50,1	11,2
9 (54)	35780692,9	1519,0	572,9	1526,1	586,8	11631,8	29,0	80,7	65,7	80,7	65,3	322,2	19,8
10 (52)	57040592,6	1013,2	229,9	1016,7	225,4	5535,5	31,7	70,3	60,9	70,9	56,8	149,6	23,3
11 (15)	248453939,2	16101,4	777,4	16101,4	763,5	117190,1	37,8	270,5	74,0	270,5	70,32	354,9	26,7

TAB. 6.3 – Résultats expérimentaux du filtrage

6.10.3 Coordination

	Saturation		Saturation + Coordination	
	nb. sélections	tps	nb. sélections	tps
(1)	134101,3	43080,5	115020,8	36656,7
(3)	1869150,2	73,5	1831861,1	92,2
(1)+Patrons	2444,2	2555,5	1444,0	1266,7
(3)+Patrons	10004,4	41,4	7566,5	54,1
nb. sélections	3128262884001,0			

TAB. 6.4 – Filtrage de la coordination

La table 6.4 présente les résultats expérimentaux de notre méthode de filtrage de la coordination. Nous avons pris les 235 phrases de la TSNLP contenant des coordinations et nous avons filtré selon différentes méthodes. Nous avons réutilisé les patrons et les différentes de méthodes de choix de valeurs que nous avons présentées plus haut. Nous avons retiré la méthode (2) puisqu'elle n'apporte aucun avantage par rapport à la méthode (1). On peut constater que l'heuristique que nous avons proposée permet d'éliminer un certain nombre de sélections qui ne vérifient pas la symétrie qui doit exister entre les conjoints.

Attention, les chiffres donnés ne semblent pas à la hauteur de nos prédictions. Mais il ne faut pas oublier que les phrases de ce corpus sont plus longues que les phrases du corpus sans coordination. Ici la longueur moyenne des phrases est de 11 mots. Dans le corpus précédent c'était la longueur maximale. La longueur maximale pour notre corpus est de 24 mots. De plus notre lexique associe aux conjonctions de coordination une cinquantaine de DAP. C'est dix fois plus que pour un mot quelconque.

Aussi, nos expérimentations montrent l'apport du critère de symétrie de la coordination au critère de saturation globale. Il est clair que des sélections sont éliminées par les deux critères, ce qui pourrait laisser entendre que notre critère de symétrie n'est pas efficace. En réalité, ce sont nos deux critères qui ne sont pas orthogonaux.

De plus, nous n'avons implanté qu'une approximation du critère que nous avons présenté. En reprenant les notations de la section 6.8, nous vérifions bien l'existence de deux états c_1 et c_2 sur l'automate, le premier à une position p_1 et le second à une position p_2, et d'une conjonction de coordination en position i avec $p_1 < i < p_2$ qui respectent les deux contraintes : $I_1 \cap I_{i+1} \neq \emptyset$ et $I_2 \cap I_i \neq \emptyset$. En revanche nous ne vérifions pas que c_1 et c_2 sont sur un même chemin de l'automate, c'est-à-dire qu'ils appartiennent à la même sélection. Nous gardons donc beaucoup de solutions qui ne vérifient pas la symétrie entre les conjoints.

Néanmoins, notre critère de symétrie élimine bien des solutions, et le temps pour calculer ce critère est négligeable. Si l'on utilise la méthode de choix de valeurs (1), le temps de calcul de ce critère est compensé par le fait de travailler sur des automates plus petits. Pour la méthode (3), les temps augmentent de moins de 20%.

6.11 Conclusion

Nous avons montré les techniques originales de désambiguïsation utilisées dans notre analyseur `leopar`. En effet les différents points abordés dans ce chapitre ont tous fait l'objet d'implantation. Nous profitons de la notion de polarité pour tirer un critère de correction sur les sélections lexicales ; nous tirons parti des connaissances linguistiques issues de la modélisation de certains phénomènes (ici la coordination) et des DAP de la grammaire (patrons interdits).

On peut noter que grâce aux heuristiques développées pour traiter plus efficacement la coordination, nous pouvons délimiter les segments coordonnés et donner les positions initiale et finale d'un groupe coordonné. Il nous restera à définir un algorithme d'analyse qui puisse analyser un fragment de phrase et *brancher* cette sous-analyse sur l'analyse de la phrase complète sans ce fragment.

Finalement, les techniques que nous avons présenté ne sont pas spécifiques aux GI. Elles sont adaptables à tous les formalismes fondés sur les polarités, comme les GC ou les grammaires d'unifications polarisés [Kah04]. On peut même imaginer de polariser certains formalismes comme les CFG et les TAG.

Quatrième partie

L'analyse syntaxique dans les grammaires d'interaction

Chapitre 7

L'analyse syntaxique

Sommaire

7.1 Introduction

Au chapitre 1, nous avons vu les GI comme un système de contraintes. Le langage engendré par une GI était l'ensemble des modèles (arbre et fonction d'interprétation) qui satisfont certains critères. Cependant, pour l'analyse syntaxique, cette vision n'est pas appropriée. Dans ce cadre précis, on cherche en effet à disposer d'algorithmes permettant de construire pas à pas les modèles depuis une suite de DAP correspondant à une sélection lexicale.

Ce chapitre est donc dédié à la présentation de tels algorithmes. D'une part, ils nous permettront d'aborder les GI sous un nouvel angle, celui de la syntaxe générative/énumérative. D'autre part, en ramenant l'analyse dans les GI à des algorithmes connus, ou à des extensions d'algorithmes connus, on pourra plus finement étudier les différences existant entre les GI et d'autres formalismes tels les CFG et les TAG par exemple.

Enfin, dans le cadre de cette thèse, le phénomène de coordination nous intéresse particulièrement et nous allons développer certaines méthodes propres aux phrases comportant des conjonctions de coordination.

7.2 Complexité du problème de l'analyse

Le problème de l'analyse, c'est à dire, étant données une phrase et une grammaire, décider si cette phrase appartient au langage engendré par la grammaire est un problème difficile.

Un algorithme non-déterministe consiste à constituer des ensembles de nœuds qui auront la même interprétation et de vérifier ensuite que les relations de préséance et de dominance sont respectées, que la linéarisation des feuilles est la phrase à analyser, que chaque polarité $\rightarrow$ rencontre une polarité $\leftarrow$ et que chaque polarité $\approx$ a rencontré une autre polarité.

Cet algorithme indique que le problème d'analyse est dans la classe de complexité NP, puisqu'il consiste en une suite de choix (arbitraires) suivie d'une vérification de la correction de ces choix.

Établir qu'une phrase appartient au langage engendré par une GI est équivalent à dire qu'il existe une fonction d'interprétation valide d'une liste de DAP de la grammaire vers les nœuds d'un modèle tel que la linéarisation de ses feuilles est la phrase donnée en entrée. La complexité du problème d'analyse peut donc se ramener à la complexité du problème consistant à trouver les modèles de suites de descriptions d'arbres. Vérifier que la linéarisation des feuilles est la phrase d'entrée se fait ensuite en un temps polynomial dans la taille de la phrase. Or [KNT01] ont prouvé que la complexité de la recherche de modèles de descriptions d'arbres est un problème NP-complet. Certes les polarités permettent de réduire quelque peu la complexité mais dans le pire des cas (où tous les traits portent des polarités = et la même valeur) elles ne sont d'aucune utilité. L'analyse dans les GI est donc un problème difficile.

Nous présentons dans ce document deux algorithmes d'analyse très différents pour les GI. Le premier est basé sur la notion de raffinement de DAP. Il simplifie pas à pas les DAP par hypothèses successives sur la fonction d'interprétation pour obtenir la DAP la plus proche possible d'un arbre fini complètement ordonné et n'effectue la recherche de modèle qu'à partir de cette dernière DAP. En théorie, ce second problème est toujours difficile mais en pratique, c'est-à-dire en utilisant les grammaires que nous avons développées, la recherche de modèle est ensuite très rapide. Une heuristique d'ordre psycho-linguistique permet même d'atteindre une complexité polynomiale au prix de la perte de la propriété de complétude de l'algorithme.

Le second algorithme est largement inspiré de l'algorithme d'Earley pour les CFG. Il construit l'arbre syntaxique en partant de la racine et en explorant tous les fils de gauche à droite (parcours infixe). C'est sur ce dernier algorithme que nous explorerons les pistes entrevues par la modélisation de la coordination et les techniques de filtrage. Cet algorithme nous permettra d'analyser séparément les conjoints, le groupe coordonné et le reste de la phrase.

7.3 L'algorithme *shift-reduce*

Nous allons voir un premier algorithme d'analyse, présenté dans [BGP03]. C'est le premier algorithme d'analyse implanté dans la plate-forme `leopar`. C'est aussi l'algorithme

le plus proche de l'intuition des GI.

7.3.1 DAP étendues

Cet algorithme manipule des structures qui diffèrent quelque peu des DAP que nous avons présentées au chapitre 1. Nous parlerons de DAP étendues pour nommer ces structures. Il existe deux différences entre les DAP et les DAP étendues :

1. les graphes sous-jacents ne sont plus des arbres mais des graphes acycliques avec éventuellement plusieurs composantes connexes.
2. toutes les polarités présentées au chapitre 1 sont autorisées, c'est-à-dire $\{\rightarrow, \leftarrow, =, \approx, \leftrightarrow\}$.

Les DAP que nous avons présentées sont donc des cas particuliers de DAP étendues (DAPE). L'avantage d'utiliser ces dernières est qu'une suite de DAP définies sur un même environnement peut également se représenter sous la forme d'une DAPE. Dans ce cas, pour évoquer le modèle saturé et minimal de la suite de DAP, on parlera du modèle saturé et minimal de la DAPE qui représente cette suite. Cet algorithme interprète une sélection lexicale comme une DAPE et essaye de trouver le modèle saturé et minimal de cette DAPE.

Cet algorithme est fondé sur la notion de raffinement de DAPE.

Définition 21 (Raffinement)**.** *Une DAPE D_r est un raffinement d'une DAPE D (noté $D_r \preceq D$) si tous les modèles saturés et minimaux de D_r sont également modèles de D. Si elles sont un raffinement l'une de l'autre alors elles sont équivalentes.*

Qu'entendons nous par *est également un modèle de* ? Un modèle (T_r, I_r) de D_r raffinement de D est aussi un modèle de D s'il existe un modèle (T, I) de D tel que T et T_r sont isomorphes et qu'il existe une fonction r des nœuds de D, notés N, vers les nœuds de D_r telle que $I(N) = I_r \circ r(N)$.

Le but ici est de calculer les raffinements de la DAPE qui représente une sélection lexicale pour la phrase d'entrée. Nous voulons obtenir des raffinements maximaux par la relation de neutralisation que nous donnerons plus bas. Ce sont des DAPE qui ont une forme d'arbre et dont tous les traits de leur nœud ont une polarité dans $\{=, \leftrightarrow, \bot\}$. Si une polarité est $\bot$ c'est que la DAPE d'entrée n'admet pas de modèle saturés.

L'algorithme procède comme suit :

1. choisir deux nœuds de la DAPE qui peuvent avoir la même interprétation et les superposer, (en s'appuyant sur les polarités comme on va le voir)
2. simplifier la DAPE obtenue, c'est à dire donner une DAPE équivalente contenant moins d'arcs et/ou moins de nœuds. Reprendre la première étape.

La première étape de cet algorithme d'analyse est effectuée par *fusion de nœuds*. On cherche un couple de nœuds qui ont des traits munis de polarités opposées et de valeurs compatibles et qui peuvent donc interpréter le même nœud du modèle. Ce couple de nœuds est identifié, c'est à dire que ces deux nœuds interprètent le même nœud de l'arbre syntaxique, qu'ils ont la même image par la fonction d'interprétation.

Définition 22 (Fusion de nœuds)**.** *Soient D une DAPE, n et m deux nœuds de D munis respectivement des structures de traits S et T.*

Si S et T ne sont pas superposables, alors la fusion de de n et m donne la DAPE incohérente $\bot$.

Sinon, la fusion de n et m dans D donne une nouvelle DAPE D' qui est égale à D sauf qu'elle ne contient plus les nœuds n et m mais les remplace par un nœud $[nm]$ muni de la structure de traits $S + T$. Tous les arcs qui avaient pour extrémité n ou m remplacent cette extrémité par $[nm]$. On notera $D \overset{n=m}{\rightarrow} D'$

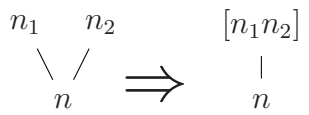
Nous verrons dans la suite comment l'algorithme choisit les nœuds à fusionner. Une fois la fusion effectuée, on va chercher à raffiner la DAPE obtenue par l'intermédiaire des règles présentées ci-dessous.

7.3.2 Règles de simplification

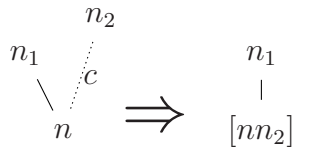
Nous allons maintenant donner quelques règles qui permettent de raffiner une DAPE. Ce n'est pas un ensemble exhaustif mais il constitue l'ensemble des règles implantées dans notre analyseur.

On peut constater que ces règles rapprochent la DAPE d'un arbre, en fait de l'arbre syntaxique. Dans tous les cas, elles s'appliquent à des nœuds qui ont deux arcs entrants et font disparaître un des arcs.

Unicité du père (R1) : Si une DAPE D contient deux relations de la forme $n_1 \overset{d}{\rightarrow} n$ et $n_2 \overset{d}{\rightarrow} n$, alors D est équivalente à D' telle que $D \overset{n_1=n_2}{\rightarrow} D'$, c'est à dire que l'on peut fusionner n_1 et n_2. Le schéma suivant illustre cette règle :



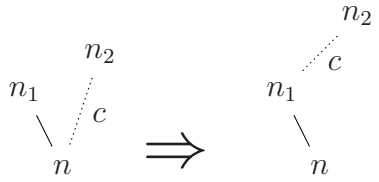
Dominance vers égalité (R2) : Si une DAPE D contient deux relations $n_1 \overset{d}{\rightarrow} n$ et $n_2 \overset{s}{\rightarrow}_c n$ et si la structure de traits associée à n_1 n'est pas superposable avec S alors D est équivalente à la DAPE obtenue en supprimant la seconde relation $n_2 \overset{s}{\rightarrow}_c n$ et en fusionnant n_2 et n.



Dominance vers parenté (R3) : Si une DAPE contient deux relations $n_1 \overset{d}{\rightarrow} n$ et $n_2 \overset{s}{\rightarrow}_c n$ et si les structures de traits associées à n et n_2, respectivement S et S_2, ne sont pas superposables alors deux cas peuvent se présenter :

1. Si S_1, la structure de traits associée à n_1, et c ne sont pas superposables alors on remplace D par la DAPE incohérente $\bot$.

2. Si S_1 et c sont superposables, on remplace D par D' obtenue à partir de D en superposant S_1 et c en remplaçant la relation $n_2 \xrightarrow{s}_c n$ par $n_2 \xrightarrow{s}_c n_1$



D'autres règles existent mais elles sont liées aux marques utilisées dans la grammaire. Par exemple si un nœud est marqué par une information contraignant l'arité du nœud dans le modèle, lors de la fusion de ce nœud avec un autre nœud ayant lui aussi des fils il est nécessaire de fusionner certains fils pour respecter la contrainte d'arité.

Il existe également des règles pour les marques contraignant l'existence de frère à gauche ou à droite.

Nous avons vu que chaque fusion de nœud ou chaque règle de simplification précise la fonction d'interprétation. Nous allons voir maintenant que ce sont les polarités qui vont indiquer quelles fusions effectuer.

7.3.3 Rôle des polarités

Cet algorithme est *guidé* par les polarités dans le sens où c'est l'existence de nœuds munis de polarités $\rightarrow$ ou $\leftarrow$ qui entraîne de nouvelles fusions selon l'opération de *neutralisation*.

Définition 23 (Neutralisation)**.** *Soient une DAPE D contenant deux nœuds n et m munis respectivement des traits $(f, \rightarrow, x)$ et $(f, \leftarrow, y)$. La neutralisation de ces traits consiste à fusionner n et m. Cette opération est notée $D \overset{n \overset{f}{=} m}{\rightarrow} D'$*

Cette opération permet de raffiner la description courante :

Proposition 8. *Si $D \overset{n \overset{f}{=} m}{\rightarrow} D'$ alors D' est un raffinement de D.*

La proposition précédente est en fait valide pour tout choix de nœuds n et m sans que l'on tienne compte des traits qui les composent. Nous avons aussi la *réciproque* de la proposition précédente qui justifie cet algorithme.

Proposition 9. *Si une description D_1 admet un modèle saturé minimal (T, I) et contient au moins deux nœuds munis respectivement des traits $(f, \rightarrow, x)$ et $(f, \leftarrow, y)$ alors il existe une neutralisation $D_1 \overset{n \overset{f}{=} m}{\rightarrow} D_2$ pour laquelle D_2 admet (T, I) comme modèle saturé et minimal.*

Cet algorithme non déterministe peut donc se formuler de la manière suivante. Soit $s = w_1, \ldots, w_n$ la phrase d'entrée et $G = (V, L, D)$ une GI. On peut distinguer deux phases dans cet algorithme. La première phase (phase A) s'effectue selon les cinq étapes suivantes :

1. Soit D une DAPE créée à partir d'une sélection lexicale pour s.
2. Soient deux nœuds n et m munis respectivement des traits $(f, \rightarrow, x)$ et $(f, \leftarrow, y)$. Calculer D' telle que $D \overset{n \overset{f}{=} m}{\rightarrow} D'$. Si n ou m n'existent pas, aller à l'étape 5.
3. Appliquer les règles de simplifications sur D' jusqu'à stabilisation. On obtient D''
4. Revenir à l'étape 2 en considérant cette fois D''.
5. Arrêt de la phase A ; retourner la DAPE courante.

La phase A retourne une DAPE maximale pour la relation de raffinement par neutralisation. Un propriété intéressante des DAPE est qu'elle ne contiennent que des polarités incluses dans $\{neutr, virt, satu\}$. Il n'y a plus de polarité $\rightarrow$ ni $\leftarrow$.

Il faut ensuite à partir de cette DAPE trouver le modèle saturé et minimal et cette recherche peut être coûteuse. Cependant, dans les cas standard que nous rencontrons en pratique, la DAPE obtenue après la première phase est tellement contrainte que la recherche du modèle se termine immédiatement. Dans la plupart des cas, il s'agit uniquement de superposer les traits virtuels.

L'algorithme que nous avons présenté est non déterministe : le choix des nœuds à fusionner, bien que guidé par les polarités restantes, est arbitraire. De nombreux couples de nœuds sont considérés lors de neutralisations et un grand nombre aboutissent à des raffinements qui n'admettent pas de modèle. La section suivante explore un critère psycho-linguistique qui permet de diminuer drastiquement l'espace de recherche.

7.3.4 Une heuristique psycho-linguistique

On s'inspire ici de la façon dont les humains lisent un texte pour développer une approximation efficace (polynomiale) de l'algorithme présenté ci-dessus. Ils procèdent de manière incrémentale, c'est à dire de gauche à droite en effectuant l'analyse au fur et à mesure. Surtout ils utilisent une mémoire finie, assez petite, pour ceci.

On va s'intéresser aux nœuds actifs, c'est à dire les nœuds porteurs de traits $\rightarrow$ ou $\leftarrow$. Ces nœuds actifs représentent les dépendances non résolues de la phrase à un instant particulier de l'analyse. On va donc borner l'espace que peuvent occuper ces nœuds actifs en partant du début de la phrase.

On s'équipe donc d'une mémoire, c'est à dire un ensemble de nœuds porteurs de polarités $\rightarrow$ ou $\leftarrow$. Cet ensemble est vide à l'initialisation de l'algorithme. La première phase de notre algorithme est modifiée de la façon suivante.

empilement Notre mémoire étant limitée, nous lui fixons une taille arbitraire b. Tant que le nombre de nœuds actifs n'a pas atteint la borne b, on ajoute à la DAPE les DAP de la sélection une à une de gauche à droite[40] et on ajoute les nœuds actifs de ces DAP à notre mémoire.

réduction Si on atteint la borne, on essaie d'effectuer une neutralisation à partir des nœuds de notre mémoire. Supposons que l'on ait choisi n et m. On retire ces nœuds de notre mémoire. On effectue la fusion correspondant à la neutralisation puis on

[40] C'est à dire les nœuds des DAP dans l'ordre des mots de la phrase.

applique les règles de simplification et on ajoute le nœud résultat de la fusion nm à la mémoire s'il est un nœud actif.

Si jamais la borne est atteinte et qu'aucune neutralisation ne donne un résultat, l'algorithme s'arrête et retourne $\perp$.

Cette modification de l'algorithme permet d'en diminuer la complexité mais dans ce cas l'algorithme n'est plus complet, tout l'espace de recherche n'est plus exploré.

Dans l'implantation actuelle, la taille de cette mémoire est donnée en fonction non pas du nombre de nœuds actifs mais en fonction du nombre de traits actifs, ce qui donne une meilleure mesure de la complexité des phrases.

C'est cette utilisation de la borne qui donne son nom à l'algorithme. D'abord, on accumule les nœuds des DAP de gauche à droite et une fois la mémoire pleine, on considère uniquement les nœuds de cette mémoire sur lesquels on tente d'effectuer les fusions, ou réductions, de nœuds.

7.3.5 Exemple

Pour illustrer l'utilisation de cet algorithme, nous allons prendre la phrase «*Jean que Marie semble aimer dort.*» de manière à manipuler quelques règles de simplification.

La première étape est de construire la DAPE correspondant à cette phrase. Les DAP sont obtenues par filtrage lexical en appliquant les méthodes que nous avons définies. Le résultat est présenté sur la figure 7.1.

Nous allons considérer une mémoire de taille 8. Cette borne va porter comme dans l'implantation sur les traits actifs, c'est à dire les traits de polarité $\rightarrow$ ou $\leftarrow$.

On va donc accumuler tous les nœuds actifs des 4 premières DAP pour atteindre cette borne. On dispose alors de 12 traits actifs, ce qui est supérieur à la borne fixée. Il faut donc procéder à une fusion de nœuds parmi les nœuds accumulés. On choisit (arbitrairement) de fusionner la racine de la DAP associée à *Marie* (`D11_1`) avec la feuille gauche de la DAP associée à *semble* (`D1664_5`). On obtient la DAPE de la figure 7.2.

Nous sommes maintenant revenus à 8 traits actifs en mémoire. Nous allons ajouter les traits de la DAP suivante qui correspond à *aimer* et passons donc à 12 traits actifs parmi lesquels il faut procéder à une fusion. On choisit alors de fusionner les feuilles les plus à droite de la DAP associée à *que* (`D44_6`) et de la DAP associée à *aimer* (`D549_23`). Nous avons donc la DAPE présentée en figure 7.3.

Le nœud obtenu par fusion a deux nœuds pères. On peut appliquer la règle de simplification **R1** qui consiste à fusionner les deux pères (`D549_5` et `D44_2`). On obtient la DAPE de la figure 7.4.

A nouveau, nous sommes redescendus sous la borne. Les traits actifs de la DAP associée à *dort* sont donc ajoutés à notre mémoire et on passe à 12 traits actifs. On fusionne le nœud obtenu par la simplification précédente (`D549_5-D44_2`) avec la feuille la plus à droite de *semble* (`D1664_2`). On obtient la structure de la figure 7.5

Le nouveau nœud a deux pères par deux relations différentes. On applique **R2** qui va reporter le père par $\xrightarrow{s}$ (`D44_22`) au dessus du père par $\xrightarrow{d}$ (`D1664_11`) (malheureusement, `leopar` ne permet pas de visualiser cette étape). Finalement, les deux pères seront neutralisés après l'ajout des traits actifs de la dernière DAP pour à nouveau atteindre notre

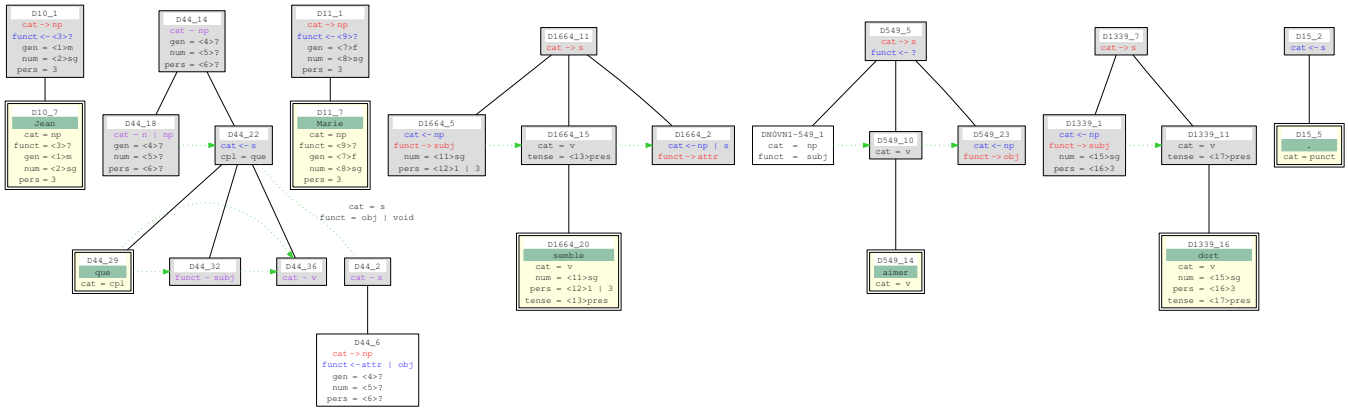


FIG. 7.1 – DAPE de la phrase *Jean que Marie semble aimer dort.*

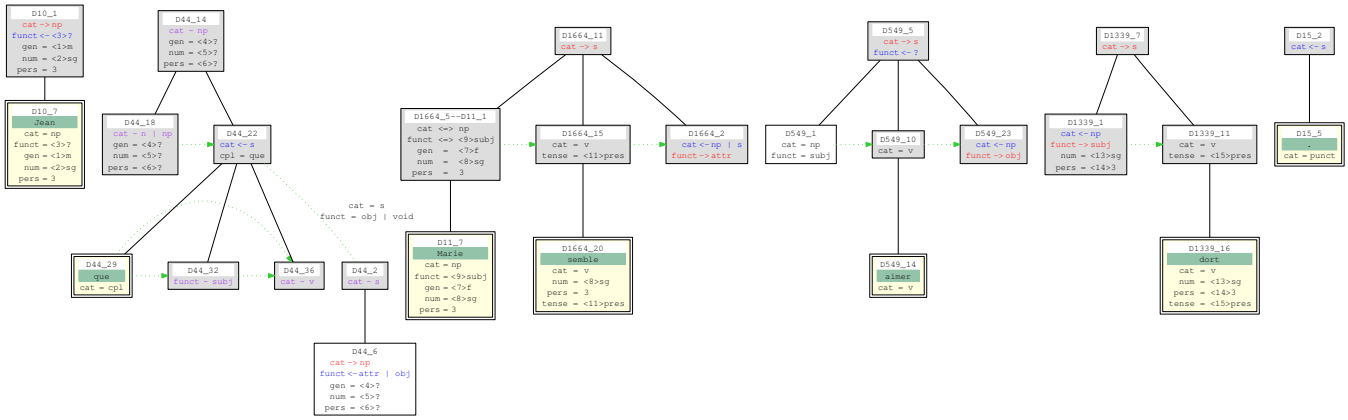


FIG. 7.2 – La DAPE après une première neutralisation

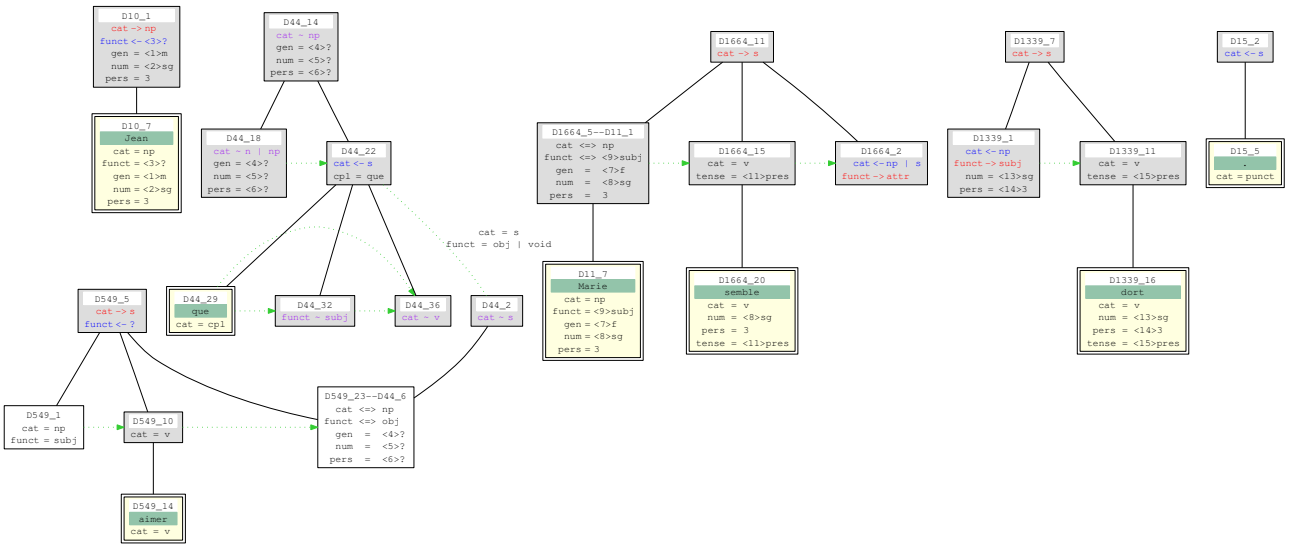


FIG. 7.3 – La DAPE après une deuxième neutralisation.

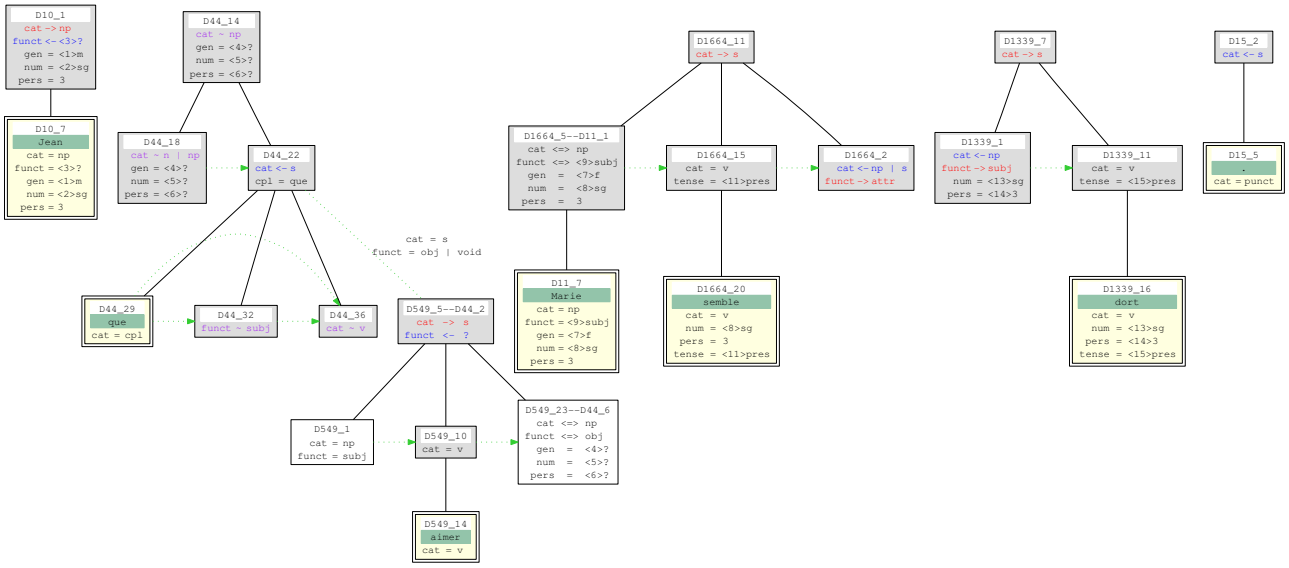


FIG. 7.4 – La DAPE après simplification

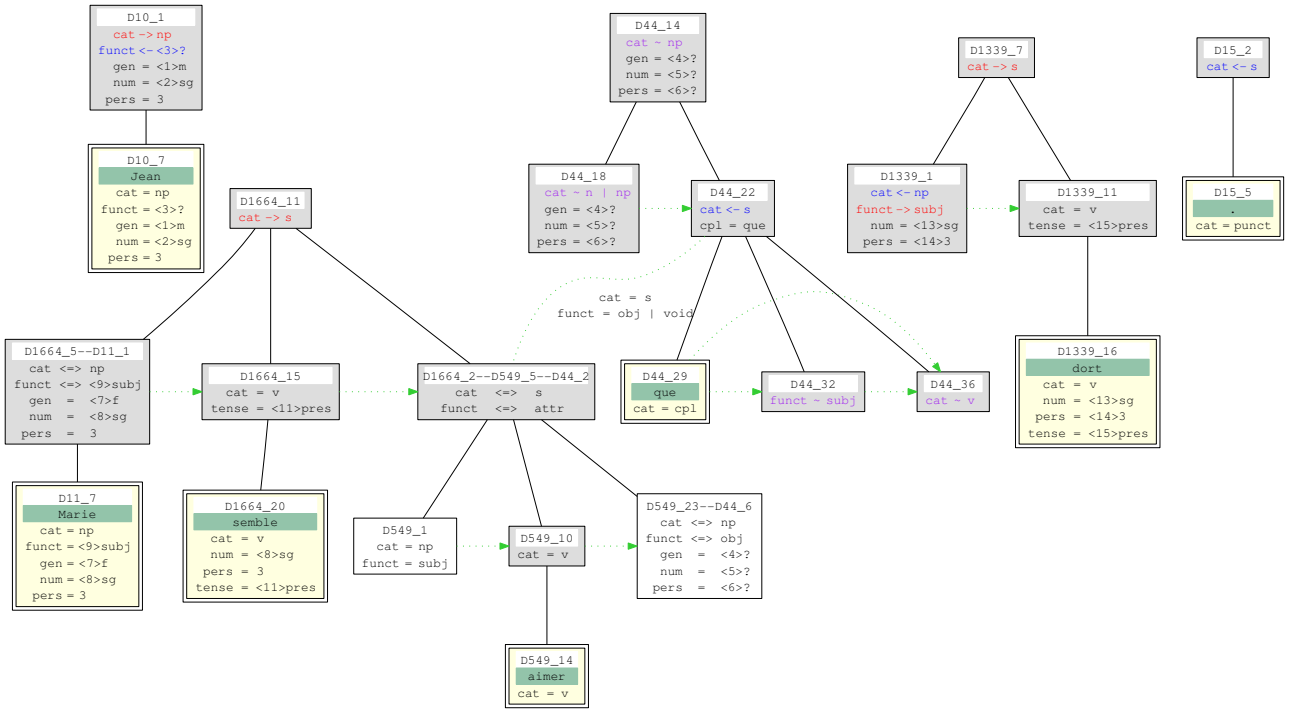


FIG. 7.5 – Après la troisième neutralisation

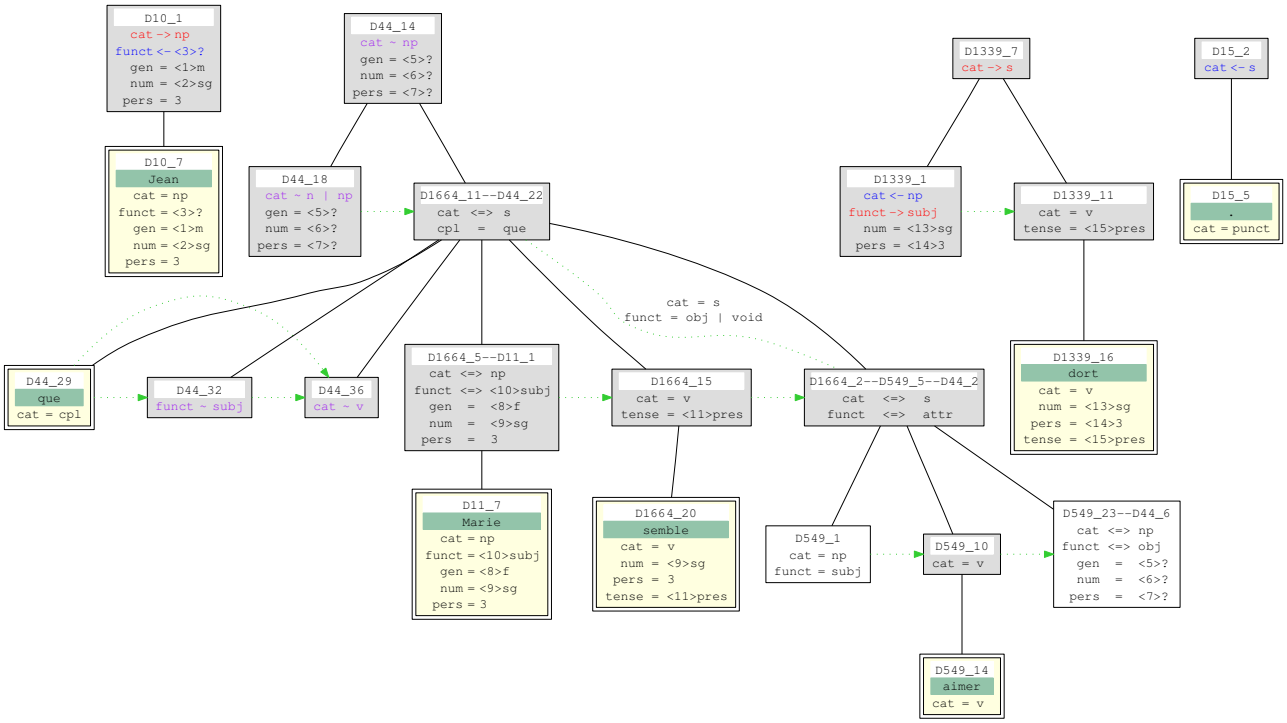


FIG. 7.6 – Après la quatrième neutralisation

borne. On obtient la DAP de la figure 7.6.

Désormais, la borne ne peut plus être atteinte, puisque nous avons considéré tous les traits actifs. On procède alors à toutes les neutralisations restant à effectuer pour qu'il n'y ait plus de trait actif. On obtient la figure 7.7. Sur cette figure, la DAPE est formée de deux composantes connexes : l'une correspond au squelette de la phrase sans modificateur et l'autre à la proposition relative.

Il reste maintenant à trouver un modèle de cette DAPE. Cette recherche se fait en plusieurs étapes. Premièrement, on va tenter de superposer les nœuds porteurs de traits virtuels sur d'autres nœuds. Ici, on va obtenir la DAPE de la figure 7.8.

On procède ensuite à la recherche de modèle proprement dite. Il s'agit essentiellement d'ordonner les nœuds frères. On obtient l'arbre syntaxique de la figure 7.9.

7.3.6 Conclusion

L'algorithme que nous avons présenté est le plus efficace des algorithmes implantés dans la plate-forme `leopar`. L'utilisation de la borne permet d'obtenir une complexité polynomiale dans la taille de la phrase.

En revanche, deux points nous semblent poser problème pour analyser plus efficacement les coordinations avec cet algorithme.

– Les DAP associées aux conjonctions de coordination sont, on l'a vu, assez complexes. Elle contiennent de nombreux nœuds actifs. Elles représentent donc des barrières à

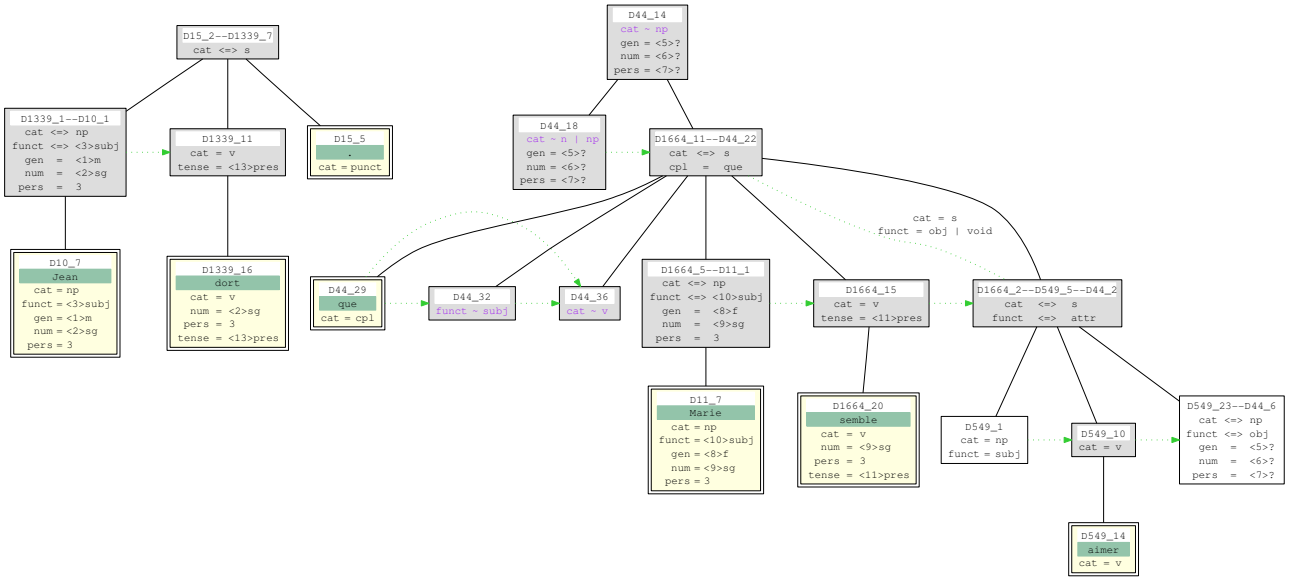


FIG. 7.7 – La DAPE raffinement sans trait actif.

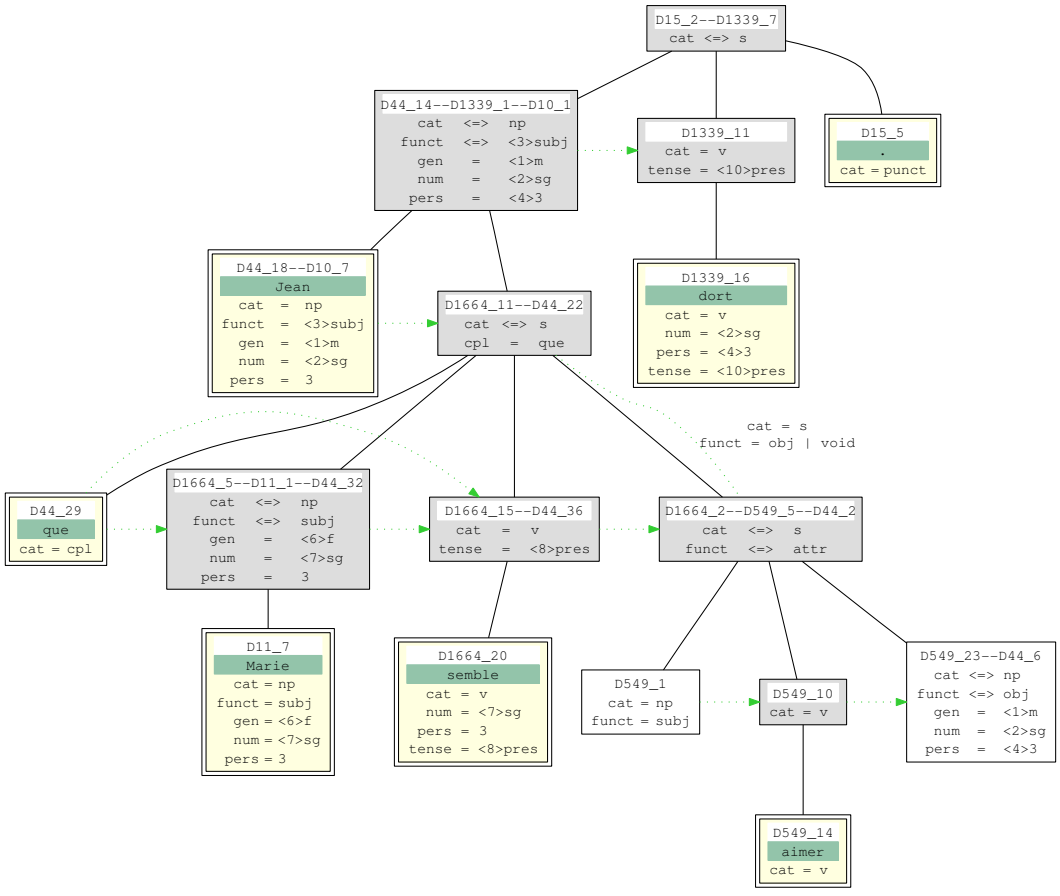


FIG. 7.8 – La DAPE après réalisation polarités virtuelles

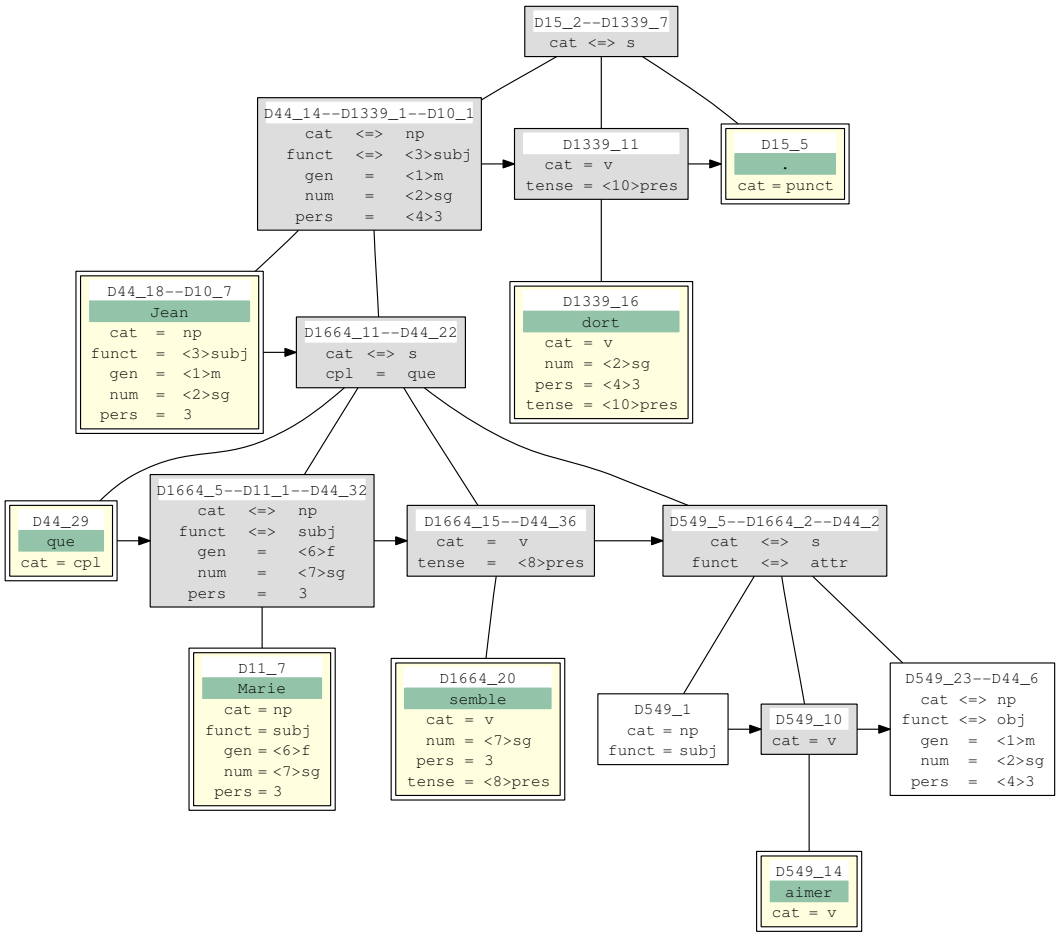


FIG. 7.9 – L'arbre syntaxique de la phrase *Jean que Marie semble aimer dort.*

l'opération d'empilement. Des phénomènes comme les montées ne sont pas analysables avec une borne inférieure à 14 quand la majorité des phrases sans coordination sont analysables avec une borne de 8. Les coordinations pénalisent donc lourdement l'analyse.

– Il n'y a pas de notion claire de sous-analyse qui nous permettrait de décomposer l'analyse d'une phrase contenant une coordination en plusieurs analyses que l'on pourrait combiner. C'est pourquoi nous avons développé l'algorithme que nous présentons au chapitre suivant.

Chapitre 8

Un algorithme d'analyse déductif à la Earley

Sommaire

Nous avons présenté un premier algorithme pour l'analyse dans les GI, ainsi qu'une heuristique d'inspiration psycho-linguistique qui permettent de rendre polynomiale sa complexité tout en gardant les analyses les plus probables de ce point de vue.

Nous avons également vu que grâce à notre modélisation de la coordination et aux méthodes de désambiguation lexicale utilisée, nous pouvons délimiter les positions des segments coordonnés. Une idée pour casser la complexité des analyses présentant des

coordinations vient à l'esprit. On peut imaginer analyser d'une part le groupe coordonné, c'est à dire les conjoints et la coordination, et d'autre part le reste de la phrase où ce groupe est remplacé par la partie haute de la DAP associée à la conjonction puisque c'est la seule partie du groupe coordonné qui peut interagir avec le reste de la phrase. Il convient ensuite de *brancher* la sous-analyse sur l'analyse principale.

Cette idée nous paraît en contradiction avec la notion de borne évoquée plus haut et plus généralement difficile à implanter avec l'algorithme précédent puisque ce dernier ne donne pas de statut spécifique aux structures de l'analyse. Nous avons donc développé un algorithme d'analyse différent qui définit clairement les objets manipulés par l'analyse.

Nous proposons donc un nouvel algorithme d'analyse qui est guidé cette fois par la forme arborescente du modèle autant que par les polarités. Cet algorithme essaie de construire le modèle en commençant par la racine de l'arbre syntaxique, en se positionnant avant le premier mot de la phrase. Il étend vers le bas à chaque itération de la boucle principale l'arbre syntaxique du modèle en lui ajoutant des feuilles et en fixant les nœuds de la sélection lexicale qu'elles interprètent en respectant les contraintes de domination, préséance, ainsi que la saturation et la minimalité.

Une analyse en construction est donc à tout moment un préfixe de l'arbre syntaxique que l'on étend au fur et à mesure de l'analyse jusqu'à arriver à l'arbre syntaxique final.

Cet algorithme s'inspire largement de l'algorithme d'analyse dit d'Earley pour les CFG. Avant de présenter notre algorithme, nous présentons son lointain ancêtre sous une forme déductive que nous utiliserons aussi pour présenter notre algorithme. Nous précisons que cet algorithme a fait l'objet d'un rapport de fin de master par Jonathan Marchand [Mar06].

8.1 L'algorithme d'Earley

L'algorithme d'Earley [Ear70] est un algorithme descendant prédictif pour les CFG. La présentation que nous allons faire de cet algorithme est largement influencée par l'approche des algorithmes d'analyse déductifs, formalisée par [SSP95].

8.1.1 Grammaires hors-contexte

Une grammaire hors-contexte (CFG) est un quadruplet (N, T, S, P) où :
- N est un ensemble fini de symboles dits non-terminaux
- T est un ensemble fini de symboles dits terminaux, et $N \cap T = \emptyset$
- S est un élément distingué de T, appelé *symbole initial*
- P est un ensemble fini de *règles de production* de la forme $A \rightarrow \alpha$ où A est un non terminal et α une suite fini de terminaux et non terminaux ($\alpha \in (N \cup T)^*$)

Dans la suite nous notons les terminaux par des minuscules, les non-terminaux par des majuscules et les suites de symboles par des lettre grecques.

Soit G_{ex} la grammaire $(\{le, chat, aime, Marie\}, \{S, N, GV, GN, D\}, S, R)$ avec R l'ensemble de règles suivantes :
- $S \rightarrow GN \quad GV$
- $GN \rightarrow Marie$
- $GN \rightarrow D \quad N$

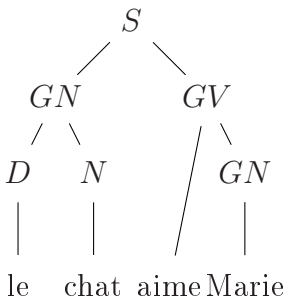


FIG. 8.1 – Dérivation de *Le chat aime Marie.*

- $D \rightarrow le$
- $N \rightarrow chat$
- $GV \rightarrow aime \quad GN$

Les CFG définissent une opération, la dérivation, sur les suites de symboles.

Définition 24 (Dérivation)**.** *Une* CFG $G = (N, T, S, P)$ *permet de réécrire une suite de symboles* $\alpha A \beta$ *par dérivation s'il existe une règle de production* $A \rightarrow \gamma$ *dans* P*.*

Le résultat de la dérivation est la chaîne $\alpha\gamma\beta$*. On note* $\alpha A\beta \Rightarrow \alpha\gamma\beta$*.*

On notera la fermeture transitive réflexive de la dérivation $\overset{*}{\Rightarrow}$*.*

On peut maintenant définir le langage engendré par une CFG

Définition 25 (Langage)**.** *Le langage défini par une grammare hors-contexte* $G = (N, T, S, P)$ *est* $L(G) = \{w_1 \dots w_n | w_i \in N \text{ et } S \overset{*}{\Rightarrow} w_1 \dots w_n\}$

Reprenons notre grammaire exemple. La phrase *le chat aime Marie* appartient au langage de G_{ex}. On a par exemple la dérivation suivante :

$$\begin{aligned} S &\Rightarrow GN \quad GV \\ &\Rightarrow D \quad N \quad GV \\ &\Rightarrow le \quad N \quad GV \\ &\Rightarrow le \quad chat \quad GV \\ &\Rightarrow le \quad chat \quad aime \quad GN \\ &\Rightarrow le \quad chat \quad aime \quad Marie \end{aligned}$$

D'autres dérivations permettent d'arriver à ce même résultat. Ici elles ne diffèrent que par l'ordre d'applications des règles de production. C'est pourquoi on représente généralement les dérivations sous forme d'arbre, ce qui permet de manipuler les dérivations modulo l'ordre d'application des règles. Ici, on obtient l'arbre de la figure 8.1.

On remarque que les règles de production peuvent être vues comme des *niveaux* de l'arbre de dérivation que l'on branche les uns dans les autres.

8.1.2 Analyse déductive

Nous présenterons une version déductive de l'algorithme. C'est à dire que l'on s'équipe d'une *table* d'items syntaxiques et d'une liste d'items syntaxiques appelée l'*agenda*[41]. La structure exacte des items dépend du formalisme et de l'algorithme.

Au début de l'algorithme (étape d'initialisation), la table est vide et l'agenda ne contient qu'un nombre limités d'items, les axiomes. Le reste de l'algorithme est simple : tant que l'agenda n'est pas vide (qu'il reste des items jamais utilisés) on retire un item de l'agenda, on le place dans la table et on essaye d'appliquer une *règle de déduction* à partir de cet item et éventuellement d'autres items de la table. Si une telle règle existe alors elle permet de produire de nouveaux items que l'on ajoute à l'agenda.

Dans le cadre déductif, un algorithme d'analyse se définit donc par un ensemble de règles de déduction de la forme :

$$\frac{I_1, \ldots, I_n}{I} R$$

où R est le nom de la règle, et $I, I_1, \ldots, I_n$ sont des schémas d'items syntaxiques. La règle R s'interprète de la manière suivante : s'il existe des items correspondant aux schémas $I_1, \ldots, I_n$ dans la table ou l'agenda alors on peut ajouter un item correspondant à I dans la table.

8.1.3 Items

Pour savoir si une phrase appartient au langage engendré par une grammaire hors-contexte $G = (N, T, S, P)$ grâce à l'algorithme d'Earley, il faut définir les items correspondants.

L'algorithme d'Earley ne travaille pas directement avec les règles de production de P mais avec des règles pointées. Si une règle de production est de la forme $X \rightarrow \alpha\beta$ (où α ou β peuvent être vides), alors une règle pointée est de la forme $X \rightarrow \alpha \bullet \beta$. Pour chaque règle de production non vide, il existe donc plusieurs règles pointées correspondant chacune à un découpage en deux de la partie droite de la règle.

Un item est défini par le triplet $[R, i, j]$ où R est une règle pointée obtenue à partir d'une règle de P et i, j deux entier naturels tels que $i \leq j$. L'item $[A \rightarrow \alpha \bullet \beta, i, j]$ est produit si l'on a établi que :

- $S \stackrel{*}{\Rightarrow} w_1 \ldots w_i A \gamma$
- $\alpha \stackrel{*}{\Rightarrow} w_{i+1} \ldots w_j$

La premier point indique le contexte dans lequel est produit l'item. C'est à dire que que l'on a déjà une analyse partielle de la phrase d'entrée : le préfixe de la phrase d'entrée $w_1 \ldots w_i$ est le préfixe d'une phrase du langage engendré par la grammaire G. Et il faut analyser A pour étendre ce préfixe. Le second point indique ce que l'item appporte réellement. Il signifie que l'on a réussi à analyser partiellement A. Plus précisément, $w_{i+1} \ldots w_j$ est un préfixe de A placé après la position i. Cette analyse partielle peut donc étendre le préfixe de la phrase d'entrée déjà analysé à condition qu'elle puisse être achevée.

[41] En réalité, l'agenda est facultatif mais il est utile pour distinguer les items que l'on n'a pas encore utilisés

Nous nous restreignons à présenter un reconnaisseur, c'est à dire un algorithme qui décide si la phrase d'entrée appartient au langage engendré par la grammaire. Si nous voulions que l'algorithme donne l'arbre d'analyse en cas de succès, il faudrait augmenter nos items de manière à pouvoir remonter l'historique des items dits terminaux, c'est à dire ceux qui sont présents si la phrase est reconnue, jusqu'aux items initiaux, donnés par la règle d'axiome (cf. infra). Cet historique donnerait la construction de l'arbre pas à pas.

8.1.4 Règles de déduction

En fonction de la grammaire d'entrée $G = (N, T, S, P)$, cet algorithme a 4 types de règles de la forme :

Axiome

$$\frac{}{[S \to \bullet\alpha, 0, 0]} Ax$$

pour toute règle de P de la forme $S \to \alpha$ où S est le symbole initial de la grammaire. Cette règle sert à initialiser les premières prédictions qui sont données par la règle suivante.

Prédiction

$$\frac{[A \to \alpha \bullet B\beta, i, j]}{[B \to \bullet\gamma, j, j]} Pred$$

On a réussi à partiellement dériver un A depuis la position i jusqu'à la position j. Pour continuer, il faut analyser un B (non-terminal). On prédit donc qu'il faut analyser un B à partir de la position j pour continuer.

Attention, ici A et B sont des *méta non-terminaux*. Ils représentent n'importe quel vrai non-terminal de la grammaire G. De plus, cette règle bien qu'elle n'ait qu'une seule conclusion est à appliquer autant de fois qu'il y a de règles de la forme $B \to \gamma$ dans P.

Balayage

$$\frac{[A \to \alpha \bullet w_j\beta, i, j]}{[A \to \alpha w_j \bullet \beta, i, j+1]} Scan$$

On a réussi à partiellement dériver un A depuis la position i jusqu'à la position j. Pour continuer, il faut lire le j$^{\text{e}}$ mot de la phrase. Si c'est le cas, on peut continuer notre analyse partielle de A.

Complétion

$$\frac{[A \to \alpha \bullet B\beta, i, j] \quad [B \to \gamma\bullet, j, k]}{[A \to \alpha B \bullet \beta, i, k]} Comp$$

Cette règle permet de *brancher* une sous-analyse dans l'analyse principale. Si d'une part, on a réussi à partiellement dériver un A depuis la position i jusqu'à la position j et qu'il faut pour continuer analyser un B et d'autre part on a réussi à analyser complètement un B à partir de la position j jusqu'à la position k (le point est à droite de la règle), alors on peut continuer à analyser le A en avançant jusqu'à la position k.

Terminaison L'algorithme présenté s'arrête quand l'agenda est vide. Pour savoir si la phrase appartient au langage engendré par la grammaire, il faut vérifier que la table contient un item correpondant à l'analyse complète du symbole initial de même longueur que la phrase d'entrée, c'est à dire l'item :

$$[S \rightarrow \gamma\bullet, 0, n]$$

où $S \rightarrow \gamma$ est une règle de P et n est la longueur de la phrase.

Remarques

Nous aurions pu donner la règle d'axiome de la manière suivante :

$$\frac{}{[\top \rightarrow \bullet S, 0, 0]} Ax_2$$

où $\top$ est un symbole non terminal qui n'appartient pas à la grammaire G. Dans ce cas, on obtient les items produits par la règle Ax en appliquant Ax_2 suivie de $Pred$. L'avantage de cette présentation est qu'il n'y a qu'un seul item généré par l'axiome et surtout qu'il n'existe dans ce cas qu'un item indiquant le succès de la reconnaissance : c'est $[\top \rightarrow S\bullet, 0, n]$.

8.2 Exemple d'analyse

En appliquant l'algorithme présenté à la grammaire donnée plus haut, on obtient les items suivants (parmi d'autres) qui correspondent à la dérivation déjà donnée.

0	$[S \rightarrow \bullet GN \quad GV, 0, 0]$	axiome
1	$[GN \rightarrow \bullet D \quad N, 0, 0]$	prédiction 0
2	$[D \rightarrow \bullet le, 0, 0]$	prédiction 1
3	$[D \rightarrow le\bullet, 0, 1]$	balayage 2
4	$[GN \rightarrow D \bullet N, 0, 1]$	complétion 1 3
5	$[N \rightarrow \bullet chat, 1, 1]$	prédiction 4
6	$[N \rightarrow chat\bullet, 1, 2]$	balayage 5
7	$[GN \rightarrow D \quad N\bullet, 0, 2]$	complétion 4 6
8	$[S \rightarrow GN \bullet GV, 0, 2]$	complétion 0 7
9	$[GV \rightarrow \bullet aime \quad GN, 2, 2]$	prédiction 8
10	$[GV \rightarrow aime \bullet GN, 2, 3]$	balayage 9
11	$[GN \rightarrow \bullet Marie, 3, 3]$	prédiction 10
12	$[GN \rightarrow Marie\bullet, 3, 4]$	balayage 11
13	$[GV \rightarrow aime \quad GN\bullet, 2, 4]$	complétion 10 12
14	$[S \rightarrow GN \quad GV\bullet, 0, 4]$	complétion 8 13

La présence de l'item 14 $[S \rightarrow GN \quad GV\bullet, 0, 4]$ dans la table indique que la phrase appartient au langage $L(G)$. On peut remarquer que tous les items de la forme $A \rightarrow \alpha\bullet$ donnent précisément les règles utilisées dans la dérivation.

8.3 Complexité et extension

La règle de complétion est la règle la plus compliquée à appliquer. Chaque item produit par cette règle provient d'un nombre de couples d'items quadratique dans la taille de la phrase d'entrée qu'il faut tous examiner.

La présentation impérative de [Ear70] permet mieux que la présentation déductive donnée ici de le voir clairement mais l'algorithme *avance* de gauche à droite. Les items générés couvrent de plus en plus de positions en commençant de 0 jusqu'à la longueur de la phrase. Le nombre d'itérations de la boucle principale de l'algorithme est donc linéaire dans la taille de la phrase.

Ces deux observations permettent de conclure que l'algorithme est d'une complexité temporelle cubique dans la longueur de la phrase notée $o(n^3)$, sans que la taille des règles n'intervienne, à la différence de l'algorithme CYK [Kas65; You67; CS70] qui requiert des règles de la forme $A \rightarrow B\ C$ pour atteindre cette borne. Si la grammaire est non-ambigüe alors l'analyse peut même se faire en temps quadratique.

Enfin, le principe de cet algorithme qui est de commencer à construire l'arbre d'analyse par la racine en effectuant des prédictions fondées sur les règles de la grammaire et vérifiées par l'ordre des mots de la phrase a été adapté dans de nombreux formalismes, notamment pour le traitement automatique des langues puisqu'il traite indifféremment les grammaires ambigües. Pour les TAG, [SJ88], et surtout [Ned99] pour une présentation déductive, ont montré que cet algorithme était adaptable avec la complexité standard de l'analyse dans ce formalisme, $O(n^6)$.

8.4 Un algorithme de type Earley pour les GI

Dans cette section, nous allons présenter un algorithme descendant prédictif pour les grammaires d'interaction selon le paradigme de l'analyse déductive. Nous allons commencer par donner l'intuition de notre algorithme à travers un exemple.

8.4.1 Intuition

Nous reprenons la phrase *Jean que Marie semble aimer dort* dont une sélection valide est représentée en figure 8.2. Nous voulons cette fois-ci construire l'arbre syntaxique par la racine. Cet arbre syntaxique est donné en figure 8.3.

La racine de l'arbre syntaxique est forcément l'image par la fonction d'interprétation de nœuds racines de DAP de la sélection[42]. On doit donc chercher un ensemble de nœuds racines tel que la superposition des structures de traits de ces nœuds soit saturée. Il y a bien sûr plusieurs possibilités. On va considérer que l'on a choisi la racine de la DAP associée à *dort* (`D1339_7`)et la racine de la DAP associée au point (`D15_2`). On suppose à partir de maintenant que ces deux nœuds ont pour image par la fonction d'interprétation la racine de l'arbre syntaxique. Nous prédisons pour l'instant un fragment de l'arbre syntaxique, représenté sur la figure 8.4 Essayons d'étendre l'arbre syntaxique.

[42]Pour être exhaustif, il faut aussi considérer les fils par $\xrightarrow{s}$ de ces racines et les fils par $\xrightarrow{s}$ de ces fils ...et ainsi de suite.

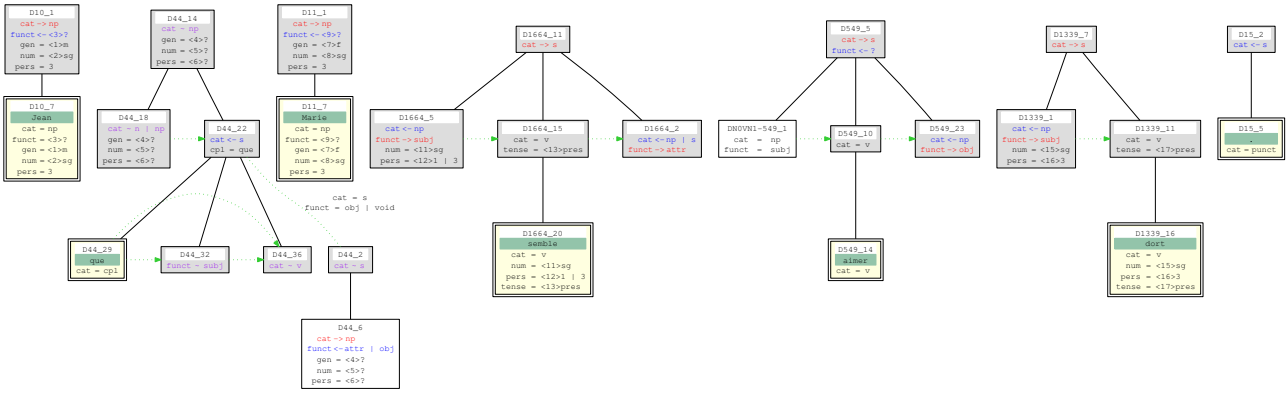


FIG. 8.2 – La sélection lexicale de *Jean que Marie semble aimer dort.*

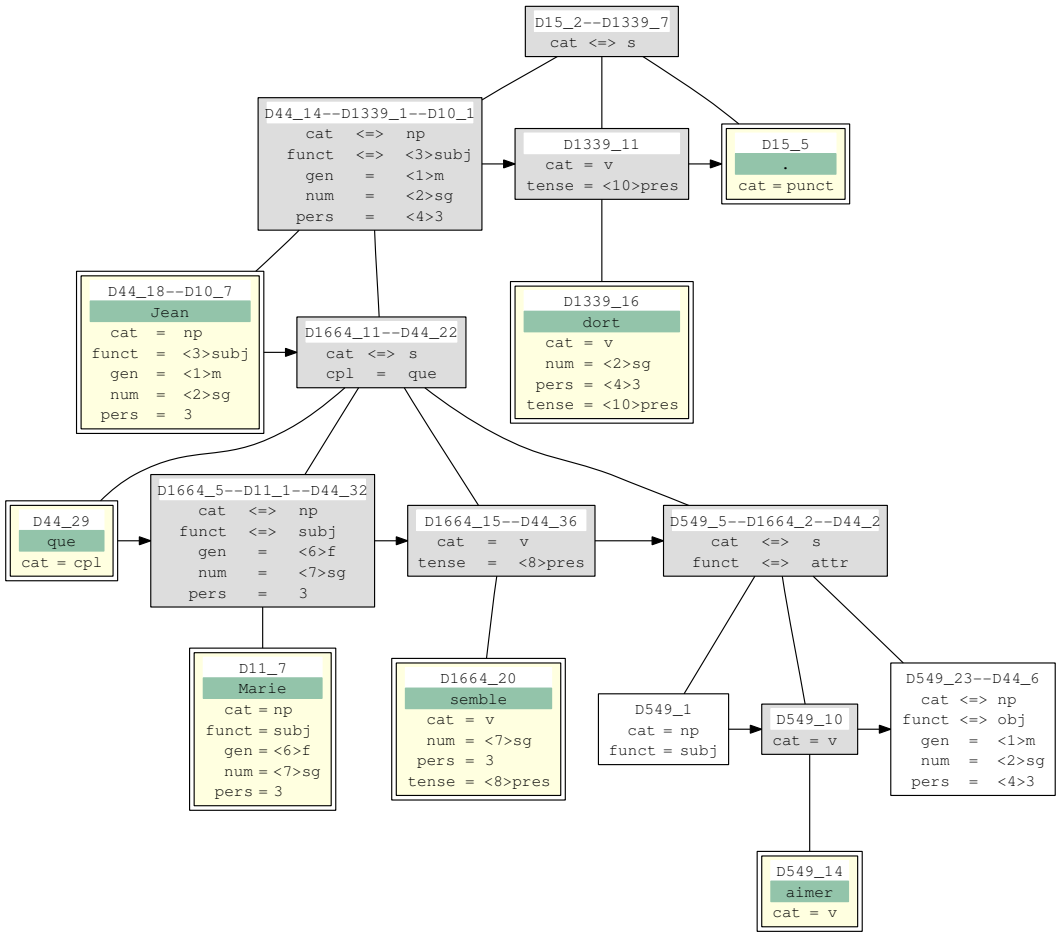


FIG. 8.3 – L'arbre syntaxique de la phrase *Jean que Marie semble aimer dort.*

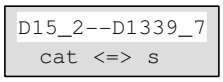


FIG. 8.4 – Première étape de l'algorithme : la racine

Ces deux nœuds que nous avons choisi pour être antécédents de la racine ont des fils par $\xrightarrow{d}$. Ces fils et l'ordre qui existe entre eux vont nous guider pour étendre l'arbre syntaxique. On va sélectionner un sous-ensemble de ces fils qui pourraient être les premiers de la fratrie et qui sont superposables. On va ensuite compléter ce sous-ensemble avec des racines de DAP de la sélection de manière à ce que le nœud résultat de la fusion de l'ensemble soit saturé pour tous les traits. On va choisir ici le premier fils de la racine de la DAP associée à *dort* (`D1339_1`) et on va compléter avec la racine de la DAP associée à *Jean* (`D10_1`) et la racine de la DAP associée à *que* (`D44_14`). On va supposer que cet ensemble de trois nœuds, que l'on notera E_1 par la suite, est l'antécedent du fils le plus à gauche de la racine de l'arbre syntaxique. Pour l'instant, notre prédiction s'étend et devient l'arbre de la figure 8.5. On va maintenant considérer les fils par $\xrightarrow{d}$ de E_1 pour étendre l'arbre syntaxique.

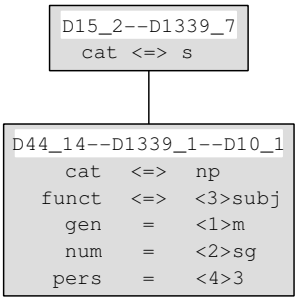


FIG. 8.5 – Deuxième étape de l'algorithme

On va choisir la feuille de la DAP associée à *Jean* (`D10_7`) et le premier fils de la racine de la DAP associée à *que* (`D44_18`). L'ensemble est saturé et contient une ancre. Cette ancre est la première ancre que l'on rencontre et elle correspond au premier mot de la phrase. Nos prédictions sont vérifiées. On peut donc remonter d'un niveau dans la branche que l'on a construit et continuer à étendre le modèle. On confirme notre dernière prédiction. On obtient l'arbre de la figure 8.6

On remonte donc d'un nœud et on regarde les fils de E_1 par $\xrightarrow{d}$ qui n'ont pas encore été visités. Il ne reste que le second fils de la racine de la DAP associée à *que* (`D44_22`). Il doit être complété par d'autres nœuds pour être saturé. On a ici le choix d'utiliser ou non son fils par $\xrightarrow{s}$. On décide de ne pas le faire. On va considérer ce fils comme la racine d'une nouvelle DAP de la sélection que l'on pourra utiliser pour complémenter un ensemble comme les autres racines de DAP de la sélection. Mais cette relation $\xrightarrow{s}$ donne une contrainte supplémentaire. Elle oblige à utiliser ce nœud dans une branche du

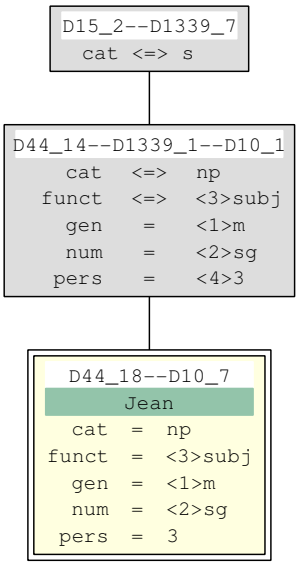


FIG. 8.6 – Troisième étape de l'algorithme

sous-arbre syntaxique courant. C'est ici que réside la difficulté de l'algorithme que nous présentons. Il faut garantir que les fils par $\xrightarrow{s}$ sont utilisés dans une branche d'un sous-arbre dont la racine est leur père. Comme on ne sélectionne pas tout de suite le fils par $\xrightarrow{s}$, on doit utiliser une racine de DAP qui n'a pas encore été visitée : on utilise la racine de la DAP associée à *semble* (`D1664_11`). On prédit alors l'arbre de la figure 8.7. Sur cette figure on fait apparaître la branche qui doit être utilisé dans la sous analyse du nœud que l'on vient d'ajouter.

Le reste de l'analyse est similaire à ce que nous avons présenté. On essaie toujours d'étendre une branche de l'arbre syntaxique jusqu'à arriver à une feuille. Dans ce cas on remonte d'un niveau et on étend la branche suivante. L'algorithme se termine quand on remonte à la racine de l'arbre syntaxique et que l'on a visité tous les nœuds des DAP de la sélection. Nous donnerons l'analyse complète de cette phrase par notre algorithme plus loin sous la forme de la suite d'items produite pour valider l'analyse.

L'algorithme construit l'arbre syntaxique à partir de la racine. Il est guidé par la relation $\xrightarrow{d}$ et les polarités. La difficulté réside dans la gestion des fils par $\xrightarrow{s}$. Nous allons maintenant formaliser cette intuition.

8.4.2 Ensembles saturés nœuds

Contrairement à l'algorithme *shift-reduce*, notre algorithme n'essaie pas de simplement neutraliser les nœuds polarisés deux à deux. Il va tenter à chaque nouvelle étape d'ajouter un nœud au modèle syntaxique en construction et de trouver tous les antécédents de ce nœud, en fait tous les ensembles d'antécédents possibles, par la fonction d'interprétation, avant de passer à l'étape suivante. Bien sûr, le choix des ensembles d'antécédents est guidé

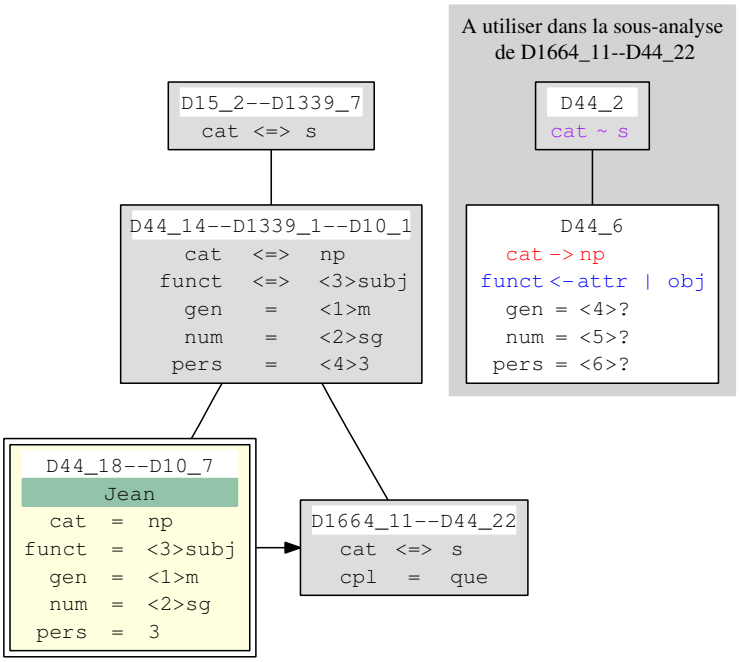


FIG. 8.7 – Quatrième étape de l'algorithme

par les polarités. Notre algorithme essaie donc de neutraliser des ensembles de nœuds.

Pour décider si un ensemble de nœuds peuvent être antécédents d'un nœud du modèle, une condition nécessaire est qu'ils respectent la contrainte sur les antécédents, donnée au chapitre 1 :

> Pour tout nœud N de D, si $(f, p, \langle x \rangle)$ est un trait polarisé de la structure de traits associée à N , la structure de traits associée à $I(N)$ contient un trait (f, v) tel que $v \in \mathcal{V}_f$ et $v \in \Gamma.\langle x \rangle$. En particulier, $\Gamma.\langle x \rangle = \emptyset$ est interdit.

Cette première contrainte impose que les valeurs de traits pour chaque trait des antécédents doivent être compatibles.

Mais, en plus des valeurs de traits, les polarités interviennent. Ces dernières doivent permettre d'arriver à un modèle saturé, dont nous rappelons la définition ici :

> Un modèle (T,I) d'une description syntaxique D est saturé si pour tout trait de T, les traits de D qu'il interprète sont :
> - soit tous neutres ou virtuels avec au moins un neutre
> - soit il y a exactement un trait positif, exactement un trait négatif et un nombre quelconque de traits neutres ou virtuels.

Cette seconde contrainte va limiter l'espace de la recherche des ensembles d'antécédents possibles. Elle donne une *condition d'arrêt* à la recherche d'antécédents.

Enfin nous rappelons une remarque faite au chapitre 1 :

Si un nœud n de l'arbre syntaxique de structure de traits S interprète des nœuds $n_1, \ldots, n_m$ de structures de traits respectives $S_1, \ldots, S_m$ (sur un environnement Γ) alors on peut construire la structure de traits $S = S_1 + \cdots + S_m$ qui est saturée et si S contient le trait $(f, p, \langle x \rangle)$ alors n contient un trait (f, v) tel que $v \in \Gamma.\langle x \rangle$.

Nous réunissons ces éléments dans la définition suivante :

Définition 26 (Ensemble saturé)**.** *Un ensemble de structures de traits $\{S_1, \ldots, S_n\}$ est* saturé *si la somme $S = S_1 + \cdots + S_n$ n'est pas incohérente et qu'aucun de ses traits n'est porteur d'une polarité $\{\rightarrow, \leftarrow, \perp, \approx\}$. C'est à dire que tous ces traits ont la polarité $\leftrightarrow$ ou $=$.*

Un ensemble de nœuds est saturé *si l'ensemble constitué de leurs structures de traits est saturé.*

La proposition suivante est triviale mais sert d'assise à notre algorithme :

Proposition 10. *Si (T, I) est un modèle pour la sélection S et la phrase s et si le nœud n de T est l'image de l'ensemble de nœuds $N = \{N_1, \ldots, N_p\}$ de DAP de la sélection S par la fonction d'interprétation I alors N est saturé.*

Bien sûr, cette condition n'est pas suffisante, puisqu'il faut aussi que les relation de dominance et de préséance sur ces nœuds soient compatibles entre elles. Nous allons donc faire en sorte que notre algorithme garantisse que le choix de nœuds saturé ne puisse se faire que parmi des nœuds compatibles. En particulier, la relation $\xrightarrow{d}$ s'interprète comme la relation de dominance dans l'arbre syntaxique et la relation $\xrightarrow{s}$ s'interprètent comme la cloture transitive et réflexive de la relation de dominance dans l'arbre syntaxique. La difficulté sera la gestion de ces dominances sous-spécifiées.

8.4.3 Forme des items et invariant

Niveaux

Les items que nous manipulons s'inspirent des items de l'algorithme de Earley. En effet, chaque item doit entre autres choses représenter une partie de l'arbre syntaxique et de la fonction d'interprétation. C'est notre équivalent des règles pointées que nous appelons un *niveau*. Nous noterons un niveau sous la forme $A \rightarrow \alpha \bullet \beta$ où A et β est un ensemble de nœuds de DAP d'une sélection lexicale pour la phrase à analyser et α est une suite de tels ensembles pour indiquer que :

- A est saturé
- tous les ensembles de $\alpha = A_1, \ldots, A_m$ sont saturés
- les nœuds de β sont des fils de nœuds de A par $\xrightarrow{d}$ qui ne sont dans aucun A_i
- Dans chaque ensemble A_i, il y a un ou plusieurs fils des nœuds de A par la relation $\xrightarrow{d}$. Les autres nœuds sont soit des racines de DAP de la sélection soit des fils par la relation $\xrightarrow{s}$ de nœuds de la sélection.

Items

Pour une GI $G = (V, L, D)$, une sélection lexicale P associée à la phrase d'entrée $p = w_1, \ldots, w_n$, nos items sont de la forme $[A \rightarrow \alpha \bullet \beta, i, j, (S, U, D)]$ où $A \rightarrow \alpha \bullet \beta$ est un niveau construit à partir de nœuds de DAP de P , i, j sont des indices de position, $0 \leq i \leq j \leq n$ et S, U, D des ensembles de nœuds de P. Ces trois derniers ensembles vont permettre de contrôler les contraintes sur les nœuds fils par $\xrightarrow{s}$. Un tel item est représenté graphiquement sur la figure 8.8.

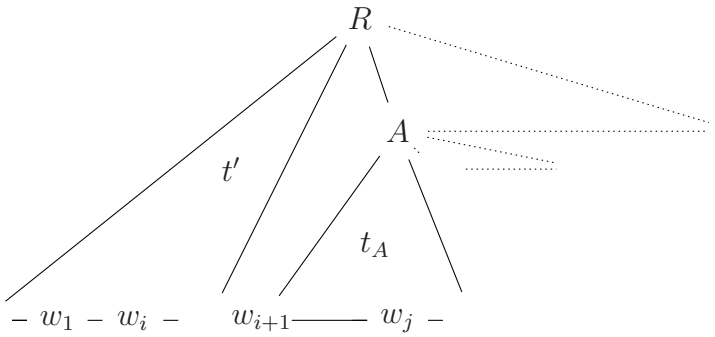


FIG. 8.8 – L'arbre t défini par l'item $[A \rightarrow \alpha \bullet \beta, i, j, (S, U, D)]$

Un item représente un sous-arbre syntaxique. C'est le fragment de la sous-analyse qui commence au mot de la phrase d'entrée à la position $i+1$. et qui est dominé par un nœud dont les antécédents sont les nœuds qui constituent l'ensemble A. Cet item suppose qu'il existe déjà une sous-analyse de de la phrase d'entrée, de la position 0 à la position i. Les items de notre algorithme ont donc la même signification que les items de l'algorithme d'Earley : il existe une analyse du préfixe de la phrase jusqu'à la position i à partir de laquelle pourrait s'insérer une analyse d'un préfixe de A. Les ensembles S, U, D n'ont pas de contenu graphique et c'est pour cela qu'il n'apparaissent pas sur la figure.

Invariants

Dans ce contexte, la production d'un item $[A \rightarrow \alpha \bullet \beta, i, j, (S, U, D)]$ indique qu'il existe un sous-ensemble N des nœuds de la sélection lexicale, un arbre t et une surjection I de N vers les nœuds de t tels que :

- pour tout nœud n de t, $I^{-1}(n)$ est un ensemble de nœuds sans incohérence. De plus, aucune relation entre des nœuds de t n'est en contradiction avec les relations de P (la sélection lexicale).
- $I(A)$ est un nœud de t
- t a comme feuilles à gauche de $I(A)$ $F_1 \ldots F_k$ telles que la restriction de $F_1 \ldots, F_k$ aux ancres donne $w_1, \ldots, w_i$ (dans les GI, les feuilles ne sont pas forcément des ancres, ce qui correspond aux productions vides des CFG)
- Soit t' la restriction de t aux nœuds compris entre la racine de t et $F_1, \ldots, F_k$. Pour tout nœud n' de t', $I^{-1}(n')$ est saturé. Intuitivement t' est la partie déjà analysée de la phrase au moment de la production de l'item.
- On appelle t_A le sous-arbre de t de racine $I(A)$. t_A a pour feuilles $G_1, \ldots, G_m, \beta$ telles que la restriction de $G_1 \ldots, G_m$ aux nœuds ancres donne la chaîne $w_{i+1} \ldots w_j$.

Les fils β n'ont pas encore été visités.
- tous les nœuds de t_A (dont les α font partie) ont par I^{-1} une image qui est un ensemble de nœuds cohérents et pour les nœuds de la restriction de t_A qui ne considère pas les nœuds β, cette image est saturée.
- les nœuds de S sont des fils de nœuds de A par la relation $\xrightarrow{s}$
- $A \rightarrow \alpha \bullet \beta$ est un niveau bien formé, c'est à dire :
  - si $\alpha = A_1 \dots A_m$ alors pour $1 \leq i \leq m$ on a $I(A) \rightarrow_t I(A_i)$
  - la partie droite du niveau est ordonnée[43] : $I(A_1) \prec_t \cdots \prec_t I(A_m)$

Ces invariants sont similaires à ceux maintenus par l'algorithme d'Earley : nous avons l'existence d'une racine (l'interprétation d'un ensemble de nœuds saturés) qui domine le début de la phrase (les i premiers mots) et l'interprétation de l'ensemble de nœuds saturés A domine au moins les mots de la positions $i+1$ à la position j. De plus, l'interprétation des ensembles $A_1, \dots, A_m, B_1, \dots, B_n$ sont des nœuds frères complètement ordonnés (de gauche à droite).

Un item représente donc une analyse partielle de la phrase d'entrée. On peut également considérer qu'un item décrit une classe d'équivalence d'arbres produits à partir de nœuds de la sélection lexicale qui vérifient les contraintes énoncées plus haut, comme dans [Sik93] pour les CFG.

Un item précise donc un niveau de l'arbre syntaxique, c'est à dire un nœud de cet arbre et ses fils complètement ordonnés. Si la règle est de la forme $A \rightarrow \gamma\bullet$ alors tous les antécédents du niveau sont connus.

Relations sous-spécifiées

Un élément reste à approfondir : la gestion des relations sous-spécifiées $\xrightarrow{s}$. Le triplet d'ensembles (S, U, D) permet la gestion des relations sous-spécifiées et des DAP de la sélection utilisées ou à utiliser. Intuitivement les nœuds dans S sont les nœuds *décrochés*, les nœuds dans D sont les nœuds qui ont été *décrochés* précédemment dans l'analyse et qui n'ont pas été *raccrochés* et les nœuds de U sont les les nœuds qui ont été *décrochés* précédemment dans l'analyse et qui ont été *raccrochés*.

- S permet d'indiquer le niveau maximal (le plus haut, le plus proche de la racine) auquel peuvent être utilisés les nœuds qui le composent. Un nœud est dans S si son père par $\xrightarrow{s}$ est dans A, la partie gauche du niveau, sans que lui même y soit.
- Les éléments de D appartenaient au S d'un item déjà produit et n'ont pas été utilisés dans un niveau d'un item.
  Un nœud est dans D si son père par $\xrightarrow{s}$ s'est trouvé dans la partie gauche du niveau d'un item ayant amené à la production de l'item courant (sans que lui même n'ait été utilisé dans la partie gauche d'un item).
- Les éléments de U appartenaient au S d'un item déjà produit et ont depuis été utilisés dans un niveau d'un item produit
  Un nœud est dans U si son père par $\xrightarrow{s}$ s'est trouvé dans la partie gauche du niveau d'un item ayant amené à la production de l'item courant et que lui-même a été

[43] Dans notre implantation les nœuds de β sont tout de même ordonnées de manière à réaliser les prédictions plus rapidement.

utilisé dans la partie gauche d'un item.

8.4.4 Règles

Les règles permettent d'étendre la l'arbre d'analyse partiel donné par un item pas à pas, en garantissant les invariants maintenus par les items produits. Nous donnons nos quatre règles de déduction analogues de celles déjà présentées pour l'algorithme d'Earley :

Axiome

$$\frac{}{[A \rightarrow \bullet\gamma, 0, 0, (S, \emptyset, \emptyset)]} Ax$$

où

- A est un ensemble saturé de nœuds que l'on a sélectionnés parmi les racines de la sélection lexicale et leurs fils larges (par $\xrightarrow{s}$). La racine de l'arbre syntaxique est en effet créée à partir de certaines racines de la sélection auxquelles peuvent s'ajouter des fils sous-spécifiés de ces racines, et par transitivité, des fils sous-spécifiés de ces fils sous-spécifiés, etc.
  Formellement : $A = \biguplus_i A_i$, A_0 est un sous-ensemble des racines de la sélection lexicale et $n \in A_j$ si la sélection lexicale contient une DAP D telle que $m \xrightarrow{s} n$ et $m \in A_i$, $i < j$.
- γ est l'ensemble des fils des nœuds de A par $\xrightarrow{d}$
- S est constitué des nœuds racines de DAP de la sélection lexicale qui ne sont pas dans A et des nœuds fils de nœuds de A par la relation $\xrightarrow{s}$ et qui eux-mêmes ne sont pas dans A.

Comme U et D ne peuvent contenir que des nœuds ayant appartenu au S d'un item déjà produit, ils sont nécessairement vides pour les axiomes. S est composé de nœuds qui doivent obligatoirement être utilisés dans la sous analyse de A. Il y a donc les nœuds fils par $\xrightarrow{s}$ de A. Dans ce cas précis des axiomes, comme A doit être la racine de l'arbre syntaxique, il y a également les racines des DAP de la sélection absentes de A.

Cette première règle peut créer un grand nombre d'items. En effet, le choix d'un ensemble A, sans autre contrainte que sa saturation, n'est pas assez limitant. Dans l'implantation, on vérifie également que la structure de traits de A présente certaines propriétés (catégorie s par exemple). Le nombre d'axiomes reste tout de même très important sur les corpus de test.

Prédiction

$$\frac{[A \rightarrow \alpha \bullet C \uplus \beta, i, j, (S_1, U_1, D_1)]}{[C \uplus O \rightarrow \bullet\gamma, j, j, (S_2, U_2, D_2)]} Pred$$

Cette règle permet de faire une prédiction sur un niveau inférieur, c'est à dire commencer une sous-analyse. La première chose est de trouver un ensemble C de fils de A par $\xrightarrow{d}$ et un ensemble O de nœuds qui n'ont pas encore été visités de tel sorte que $C \uplus O$ soit saturé. Comme pour la règle précédente, O est constitué de racines de DAP et de fils larges de ces racines mais il peut aussi y avoir comme éléments des nœuds fils par $\xrightarrow{s}$ tels que leur père a déjà été visité.

Cette règle ne peut s'appliquer que si $C \uplus O$ ne contient pas d'ancre. Dans le cas contraire, c'est la règle suivante qui s'appliquera. Examinons les propriétés de cet item :

- C est choisi de telle sorte qu'aucun nœud de β ne précède un nœud dans C. Évidemment, C est non-vide.
- L'ensemble $C \uplus O$ est saturé. Comme pour l'ensemble de A de la règle d'axiome, il faut considérer des chaînes de nœuds par la realtion $\xrightarrow{s}$. $O = \biguplus_i O_i$ où :
  - $O_0 \subseteq S_1 \uplus D_1 \uplus C_1$ où C_1 est l'ensemble des fils de C par la relation $\xrightarrow{s}$,
  - $n \in O_j$ s'il existe une DAP de la sélection lexicale telle que $m \xrightarrow{s} n$ et $m \in O_i$, $i < j$.
- D'après le point précédent, on note $D = S_1 \backslash O_0$ et $D' = D_1 \backslash O_0$. On a $D_2 = D \uplus D'$. D_2 contient les racines non utilisées.
- S_2 est l'ensemble des fils de O par la relation $\xrightarrow{s}$ qui ne sont pas eux-mêmes éléments de O. On *décroche* les fils sous-spécifiés non utilisés. Ils deviennent en quelque sorte des racines de DAP que l'on ajoute à la sélection.
- $U_2 = U_1 \uplus (S_1 \backslash D) \uplus (D_1 \backslash D')$. Les nœuds *raccrochés* sont les les nœuds déjà raccrochés auxquels on ajoute les nœuds décrochés utilisés dans le niveau.
- γ est l'ensemble des fils de $C \uplus O$ par la relation $\xrightarrow{d}$.
- les nœuds de S_2 sont des fils par $\xrightarrow{s}$ de $C \uplus O$, qui ne peuvent pas être utilisés ailleurs que dans la sous-analyse de $C \uplus O$.
- les nœuds de D_2 sont les nœuds de D_1 et de S_1 qui n'ont pas encore été utilisés.
- les nœuds de U_2 sont bien des nœuds ayant appartenus au S d'un item précédent et qui ont été utilisés dans un niveau.

Balayage

$$\frac{[A \to \alpha \bullet C \uplus \beta, i, j, (S_1, U_1, D_1)]}{[C \uplus O \to \bullet, j, j+1, (S_2, U_2, D_2)]} Scan$$

Cette règle est similaire à la précédente[44] et est réservée au cas où $C \uplus O$ contient un (et un seul) nœud ancre et que cette ancre est le j^{e} mot de la phrase d'entrée. De plus, elle ne peut s'appliquer que si les nœuds de $C \uplus O$ n'ont pas de fils par la relation $\xrightarrow{d}$. Dans l'implantation, cette règle est programmée comme un cas spécial de la prédiction.

Les ensembles de nœuds $S_1, S_2, \ldots$ sont calculés comme précédemment.

Complétion

$$\frac{[A \to \alpha \bullet C \uplus \beta, i, j, (S_1, U_1, D_1)] \; [C \uplus O \to \gamma \bullet, j, k, (S_2, U_2, D_2)]}{[A \to \alpha (C \uplus O) \bullet \beta, i, k, (S_3, U_3, D_3)]} Comp$$

Comme pour l'algorithme d'Earley, cette règle permet de brancher des sous-analyses complétées.

Il faut tout d'abord vérifier certaines conditions sur les items hypothèses :

- O aurait pu être construit par prédiction depuis le premier item hypothèse. C'est à dire que O est construit à partir de nœuds de S_1, D_1 et de fils par $\xrightarrow{s}$ de ces nœuds.

[44] Nous aurions pu la présenter de manière à effectuer une complétion en même temps. Mais la présentation en aurait été plus compliquée.

- $S_2 = \emptyset$, tous les fils par $\xrightarrow{s}$ de $C \uplus O$ ont été utilisés dans la sous-analyse.
- $U_1 \subseteq U_2$, les DAP considérées comme utilisées dans l'analyse principale ont aussi été considérées utilisées par la sous-analyse.
- $D_2 \subseteq D_1 \cup S_1$. Après la sous-analyse les DAP qui restent à utiliser forment un sous-ensemble des DAP qui restaient à utiliser avant.

Les ensembles S_3, U_3, D_3 sont donnés par les égalités suivantes :

- $D_3 = D_2 \backslash S_1$ les DAP qui restent à utiliser sont les DAP qui restaient à utiliser après la sous-analyse moins les DAP décrochées.
- $U_3 = U_2 \backslash S_1$ les DAP considérées comme utilisées sont les DAP considérées comme utilisées dans la sous-analyse moins les nœuds décrochés.
- $S_3 = S_1 \backslash U_2$, les nouvelles DAP décrochées sont les anciennes DAP décrochées desquelles ont été omises les DAP utilisées dans la sous-analyse.

Terminaison Comme précédemment, il faut regarder l'existence d'items particuliers après le déroulement de l'algorithme pour savoir si la phrase d'entrée a été reconnue. On cherche alors des items de la forme :

$$[A \rightarrow \gamma\bullet, 0, n, (\emptyset, \emptyset, \emptyset)]$$

où n est la longueur de la phrase d'entrée. En pratique on vérifie également la catégorie des nœuds de A.

Remarques

1. Certaines contraintes imposées au modèle par l'intermédiaire des marques de nœuds de DAP peuvent également être vérifiées au moment de la production d'items. Nous n'en parlerons pas dans cette présentation.
2. La règle d'axiome peut également se donner sous la forme suivante :

$$\frac{}{[\top \rightarrow \bullet\emptyset, 0, 0, (S, \emptyset, \emptyset)]} Ax_2$$

où $\top$ est un symbole spécial, représentant le contexte vide au dessus du nœud racine de l'arbre syntaxique et S est l'ensemble des racines des DAP de la sélection lexicale. Dans ce cas, l'item indiquant le succès de la reconnaissance est de la forme $[\top \rightarrow A\bullet, 0, n, (\emptyset, \emptyset, \emptyset)]$

8.4.5 Correction et complétude

L'algorithme présenté ci-dessus maintient l'invariant donné en section 8.4.3. La preuve se fait par induction sur les règles.

L'algorithme se termine avec succès si l'item $[\top \rightarrow A\bullet, 0, n, (\emptyset, \emptyset, \emptyset)]$ est généré. Il indique par l'invariant maintenu, que l'on a trouvé une fonction surjective I des nœuds de la sélection lexicale vers les nœuds d'un arbre fini complètement ordonné telle que l'image par I^{-1} des nœuds de cet arbre renvoie un ensemble de nœuds saturés et qu'aucune des relations de dominance ou de préséance dans l'arbre n'est en contradiction avec les

relations de la sélection lexicale. Enfin la linéarisation des feuilles de cet arbres nous donne la phrase d'entrée. La fonction I est donc une fonction d'interprétation et l'arbre image de I est l'arbre syntaxique.

La complétude de l'algorithme est également vérifiée. Le modèle de la sélection lexicale est bien une fonction surjective des nœuds de la sélection vers les nœuds de l'arbre syntaxique qui est décrite par l'item de terminaison. On peut montrer que cet item de terminaison est produit par une chaîne d'items qui décrivent l'arbre modèle niveau par niveau.

8.4.6 Exemple

Nous reprenons comme exemple la phrase *Jean que Marie semble aimer dort.* Nous reproduisons la sélection lexicale d'entrée pour cette phrase en figure 8.9. Nous donnons la convention suivante pour désigner les nœuds de cette sélection : la racine de la i^{e} DAP sera notée i et les autres nœuds seront appelées par leur adresse de Gorn. Par exemple la feuille la plus à droite de la DAP associée à *que* sera notée 2241. Les items de cette reconnaissance sont donnés dans la table 8.1.

Nous ne donnons pas tous les items engendrés par l'algorithme présenté (il y en aurait trop !) mais uniquement ceux qui vont amener à la génération de l'item qui permet de décider de l'appartenance de la phrase : c'est l'item 34 qui indique que la phrase est reconnue. Nous avons choisi d'utiliser l'axiome 2.

L'item 0 est donc le produit de cet axiome. Les 7 DAP de la sélection sont placées dans l'ensemble S. La partie droite du niveau de cet item est composée d'une suite d'un seul ensemble de nœuds. Cet ensemble vide pour l'instant sera occupé par la racine de l'arbre syntaxique.

L'item 1 est donc produit par prédiction à partir de l'axiome. On choisit donc un ensemble de nœuds saturé dans l'ensemble S de l'axiome (puisque D est vide). On choisit ici 6 et 7 (*dort* et le point) et on crée l'item correspondant : la l'ensemble des fils de 6 et 7 est $\{61, 62, 71\}$. 6 et 7 sont placés dans l'ensemble U, les autres racines dans l'ensemble D.

Ensuite, on effectue une nouvelle prédiction en ayant comme contrainte de trouver un ensemble saturé contenant le nœud 61. En effet c'est le seul nœud qui peut être choisi compte tenu des relations de préséance.

Nous avons le choix ici parmi 1 (*Pierre*) ou 3 (*Marie*) auxquelles peut s'ajouter 2 (la relative). L'item que nous choisissons de produire aura comme partie gauche l'ensemble $\{61, 1, 2\}$. Intuitivement, sur le sujet de *dort* on branche *Jean* et la relative. Il s'agit ensuite d'ordonner les fils de ces nœuds. 61 n'a pas de fils, 1 a un fils (une ancre) et 2 en a deux, partiellement ordonnés. L'ensemble des fils par $\xrightarrow{d}$ de 1 et 2 est $\{11, 21, 22\}$.

Pour la prédiction suivante on sélectionne 11 et 21. Ces deux nœuds forment un ensemble saturé. La prochaine règle à appliquer est alors un balayage plutôt qu'une prédiction car l'ensemble $\{11, 21\}$ contient une ancre (*Jean*) qui correspond à la position à laquelle se trouve l'analyse.

Comme on atteint la bonne ancre, on peut attester la dernière prédiction par la règle de complétion (item 4). Les contraintes d'égalités et d'inclusion sur les ensembles S, U, D sont

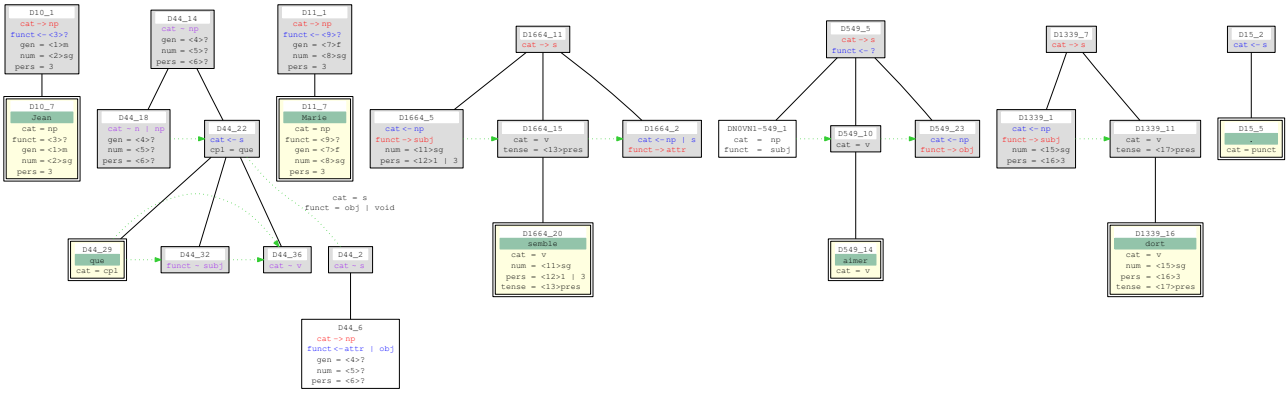


FIG. 8.9 – La sélection lexicale de *Jean que Marie semble aimer dort.*

respectées. On se place donc à la position suivante et on effectue une nouvelle prédiction, qui génère l'item 5. Il faut trouver de quoi saturer le nœud 22. Ici on a le choix entre 4 ou 5 auxquels on peut ajouter 224. On choisit 4, que l'on place dans U et on ne sélectionne pas 224 qui va donc se retrouver dans S. Cela va forcer à l'utiliser dans une sous-analyse ou sinon on ne pourra pas utiliser le niveau dans une règle de complétion.

Le reste de l'analyse ne présente pas de difficultés supplémentaires et on passe directement à l'item 25 qui est justement créé par complétion depuis un item utilisant 224 et l'item 15 dont le S contient 224. Ce nœud disparaît donc puisque il a bien été utilisé dans la sous-analyse du nœud 22.

La fin de l'analyse est sans problème particulier. On peut remarquer dans cette analyse que chaque nœud est visité, c'est à dire apparaît en partie gauche d'un niveau, une fois plus une fois par fils de l'interprétation de ce nœud dans l'arbre syntaxique.

8.4.7 Complexité

La complexité de cet algorithme est élevée. Ceci est dû à :

- la construction des ensembles O, lors des prédictions qui est dans le pire des cas de complexité exponentielle dans le nombre de racines et le nombre de relations $\xrightarrow{s}$ de la sélection lexicale d'entrée.
- le choix des ensembles C et O, lors des prédictions. Comme nous l'avons déjà dit, dans la pratique nous ordonnons γ pour retrouver facilement les éléments minimaux par $\prec$ de cet ensemble. Le choix reste tout de même un facteur limitant. [45] Mais de toute façon, dans le pire des cas, toutes les parties de γ seront énumérées.

Contrairement à l'algorithme d'Earley, c'est ici la prédiction qui est l'étape la plus complexe. Après le déroulement de l'exemple précédent, on se rend compte qu'un item apporte beaucoup d'informations de contexte. Ceci est dû aux ensembles (S, U, D) qui indiquent quelles ont été les DAP utilisées et quelles sont les parties de DAP *décrochées* pour être utilisées dans une sous-analyse, et qui vont devoir être *raccrochées*.

Pour abstraire davantage nos items, nous pouvons dans la règle de complétion éviter d'ajouter l'ensemble O dans l'item produit. C'est ce qui est fait dans l'implantation. La règle devient :

$$\frac{[A \to \alpha \bullet C\beta, i, j, (S_1, U_1, D_1)]\ [C \uplus O \to \gamma\bullet, j, k, (S_2, U_2, D_2)]}{[A \to \alpha C \bullet \beta, i, k, (S_3, U_3, D_3)]} Comp$$

On génère ainsi beaucoup moins d'items. Nous n'avons pas parlé des contraintes portant sur les relations $\xrightarrow{s}$. Elles sont vérifiées à la remontée, c'est à dire à chaque complétion entre le descendant et son ancêtre. Pour les traiter nous ajoutons un dernier ensemble à nos items qui contient des structures de traits qui doivent être vérifiées par le nœud en partie gauche du niveau à chaque complétion jusqu'à rencontrer l'ancêtre de la relation $\xrightarrow{s}$ associée. Nous n'avons pas voulu compliquer davantage l'exemple et c'est pourquoi nous avons omis d'en parler dans les règles.

[45] Une autre possibilité serait de d'inspirer de [NSS03] qui manipule directement des CFG où l'ordre dans la partie droite des règles de production est sous-spécifié.

0	$[\top \rightarrow \bullet\emptyset, 0, 0, (\{1,2,3,4,5,6,7\}, \emptyset, \emptyset)]$	Axiome(2)
1	$[\{6,7\} \rightarrow \bullet\{61,62,71\}, 0, 0, (\emptyset, \{6,7\}, \{1,2,3,4,5\})]$	Prédiction 0
2	$[\{61,1,2\} \rightarrow \bullet\{11,21,22\}, 0, 0, (\emptyset, \{1,2,6,7\}, \{3,4,5\})]$	Prédiction 1
3	$[\{11,21\} \rightarrow \bullet, 0, 1, (\emptyset, \{1,2,6,7\}, \{3,4,5\})]$	Balayage 2
4	$[\{61,1,2\} \rightarrow \{11,21\} \bullet \{22\}, 0, 1, (\emptyset, \{1,2,6,7\}, \{3,4,5\})]$	Complétion 2 3
5	$[\{22,4\} \rightarrow \bullet\{221,222,41,223,42,43\}, 1, 1, (\{224\}, \{1,2,4,6,7\}, \{3,5\})]$	Prédiction 4
6	$[\{221\} \rightarrow \bullet, 1, 2, (\emptyset, \{1,2,4,6,7\}, \{3,5,224\})]$	Balayage 5
7	$[\{22,4\} \rightarrow \{221\} \bullet \{222,41,223,42,43\}, 1, 2, (\{224\}, \{1,2,4,6,7\}, \{3,5\})]$	Complétion 5 6
8	$[\{222,41,3\} \rightarrow \bullet\{31\}, 2, 2, (\emptyset, \{1,2,3,4,6,7\}, \{5,224\})]$	Prédiction 7
9	$[\{31\} \rightarrow \bullet, 2, 3, (\emptyset, \{1,2,3,4,6,7\}, \{5,224\})]$	Balayage 8
10	$[\{222,41,3\} \rightarrow \{31\}\bullet, 2, 3, (\emptyset, \{1,2,3,4,6,7\}, \{5,224\})]$	Complétion 8 9
11	$[\{22,4\} \rightarrow \{221\}\{222,41,3\} \bullet \{223,42,43\}, 1, 3, (\{224\}, \{1,2,3,4,6,7\}, \{5\})]$	Complétion 7 10
12	$[\{223,42\} \rightarrow \bullet\{421\}, 3, 3, (\emptyset, \{1,2,3,4,6,7\}, \{5\})]$	Prédiction 11
13	$[\{421 \rightarrow \bullet, 3, 4, (\emptyset, \{1,2,3,4,6,7\}, \{5\})\}]$	Balayage 12
14	$[\{223,42\} \rightarrow \{421\}\bullet, 3, 4, (\emptyset, \{1,2,3,4,6,7\}, \{5\})]$	Complétion 12 13
15	$[\{22,4\} \rightarrow \{221\}\{222,41,3\}\{223,42\} \bullet \{43\}, 1, 4, (\{224\}, \{1,2,3,4,6,7\}, \{5\})]$	Complétion 11 14
16	$[\{43,5,224\} \rightarrow \bullet\{51,52,53,2241\}, 4, 4, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Prédiction 15
17	$[\{51\} \rightarrow \bullet, 4, 4, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Prédiction 16
18	$[\{43,5,224\} \rightarrow \{51\} \bullet \{52,53,2241\}, 4, 4, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Complétion 16 17
19	$[\{52\} \rightarrow \bullet\{521\}, 4, 4, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Prédiction 18
20	$[\{521\} \rightarrow \bullet, 4, 5, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Balayage 19
21	$[\{52\} \rightarrow \{521\}\bullet, 4, 5, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Complétion 19 20
22	$[\{43,5,224\} \rightarrow \{51\}\{52\} \bullet \{53,2241\}, 4, 5, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Complétion 18 21
23	$[53,2241 \rightarrow \bullet, 5, 5, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Prédiction 22
24	$[\{43,5,224\} \rightarrow \{51\}\{52\}\{53,2241\}\bullet, 4, 5, (\emptyset, \{1,2,3,4,5,6,7,224\}, \emptyset)]$	Complétion 22 23
25	$[\{22,4\} \rightarrow \{221\}\{222,41,3\}\{223,42\}\{43,5,224\}\bullet, 1, 5, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Complétion 15 24
26	$[\{61,1,2\} \rightarrow \{11,21\}\{22,4\}\bullet, 0, 5, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Complétion 4 25
27	$[\{6,7\} \rightarrow \{61,1,2\} \bullet \{62\}\{71\}, 0, 5, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Complétion 1 26
28	$[\{62\} \rightarrow \bullet\{621\}, 5, 5, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Prédiction 27
29	$[\{621\} \rightarrow \bullet, 5, 6, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Balayage 28
30	$[\{62\} \rightarrow \{621\}\bullet, 5, 6, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Prédiction 28 29
31	$[\{6,7\} \rightarrow \{61,1,2\}\{62\} \bullet \{71\}, 0, 6, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Complétion 27 30
32	$[\{71\} \rightarrow \bullet, 6, 7, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Balayage 31
33	$[\{6,7\} \rightarrow \{61,1,2\}\{62\}\{71\}\bullet, 0, 7, (\emptyset, \{1,2,3,4,5,6,7\}, \emptyset)]$	Complétion 31 32
34	$[\top \rightarrow \{6,7\}\bullet, 0, 7, (\emptyset, \emptyset, \emptyset)]$	Complétion 0 33

TAB. 8.1 – Les items produits lors de la reconnaissance de *Jean que Marie semble aimer dort.*

On peut résumer la situation de la manière suivante : les items contiennent trop d'information sur le contexte d'utilisation qu'ils ne permettent pas d'abstraire suffisamment. On peut donc pas factoriser l'ensemble des solutions comme le fait l'algorithme d'Earley où à ses adaptations aux formalismes faiblement contextuels (comme les TAG). C'est le défaut de cet algorithme (et celui des formalismes contextuels).

8.5 Application à la coordination

8.5.1 Présentation

Dans cette section nous développons une extension de notre algorithme pour analyser les phrases présentant des coordinations. Elle tire profit de la modélisation et de l'opération de filtrage que nous avons présentées. Le but ici est de diviser l'analyse d'une phrase en plusieurs sous-analyses indépendantes.

Nous savons d'après notre modélisation que les conjoints n'interagissent pas avec le reste de la phrase. Ils interagissent uniquement avec la conjonction de coordination. Chaque conjoint est complètement saturé par une partie basse de la DAP de la conjonction de coordination. On peut donc analyser indépendamment les segments constitués d'un conjoint de la partie basse qui le sature.

D'autre part, nous savons délimiter les segments conjoints grâce à notre opération de filtrage. C'est à dire que nous connaissons les positions dans la phrase qui contiennent exactement les conjoints.

Nous proposons donc :

1. d'analyser dans un premier temps chaque conjoint (avec la partie basse correspondante) séparément et indépendamment entre les positions qui lui seraient attribuées dans la phrase d'entrée ;
2. puis d'analyser la phrase en considérant que le groupe coordonné est déjà analysé. C'est à dire que les items produits à l'étape précédente son toujours dans la table mais que les DAP des mots qui composent les conjoints ne sont plus accessibles pour l'étape de prédiction.

L'intuition derrière cette heuristique est la suivante : nous construisons à la première étape les structures associées aux conjoints, indépendantes du reste de la phrase. Puis dans la seconde étape nous analysons la phrase en considérant que le groupe coordonné est un bloc indivisible.

8.5.2 Modification de l'algorithme

L'algorithme que nous proposons prend en entrée
- une sélection $d_1, \ldots, d_n$ correspondant à une phrase d'entrée $w_1 \ldots, w_n$, comme l'algorithme précédent et
- des triplets de la forme (p_{c_1}, d_i, p_{c_2}) où p_{c_1} et p_{c_2} sont les positions de début et de fin du groupe coordonné, et d_i est la DAP associée à la conjonction de coordination pour la sélection courante. Il faut en plus être capable de retrouver les racines des

parties basses de cette dernière DAP, soit en les marquant, soit en supposant que ce sont les nœuds frères (à gauche et à droite) du nœud ancre.

Les DAP qui composent le premier conjoint sont $d_{c_1}, \ldots, d_{i-1}$ et celles qui composent le second conjoint sont $d_{i+1}, \ldots, d_{c_2}$.

L'étape se déroule en plusieurs étapes. D'abord on analyse chaque conjoint. Puis on analyse la phrase entière. Dans le cas où il y a plusieurs coordinations, il faut donner un ordre sur les conjoints. S'il existe un conjoint c_1 dans l'intervalle de position $[a; b]$ et un autre conjoint c_2 dans $[c; d]$ trois cas peuvent se produire.

1. $b < c$, il n'y a pas d'ordre imposé entre les conjoints,
2. $a \leq c < d \leq b$, c_2 doit être analysé avant c_1.
3. sinon, les intervalles se chevauchent. L'analyse peut être arrêtée car on a fait un mauvais choix de position pour les conjoints.

Soit $c_1, \ldots, c_m$ un ordre des conjoints de la phrase d'entrée. On va analyser les c_i séparément. Si un conjoint n'est pas reconnu alors l'analyse s'arrête tout de suite et la phrase d'entrée n'est pas reconnue par la sélection courante.

Pour analyser les conjoints, nous allons ajouter la règle suivante à notre algorithme déductif qui va remplacer la règle d'axiome. L'item qui indique la réussite de la reconnaissance est lui aussi modifié.

Axiome de la reconnaissance d'un conjoint

Pour un conjoint c compris entre les positions i_c et f_c avec d_b, la partie basse correspondant à la DAP de la conjonction de coordination.

$$\frac{}{[\top \rightarrow \bullet A, i_c, i_c, (S, \emptyset, \emptyset)]} Ax_c$$

A est un singleton $\{r\}$ où r est la racine de d_b et l'ensemble S est l'ensemble des racines des DAP $d_{i_c} \ldots, d_{f_c}$.

Terminaison de la reconnaissance conjoint

Le conjoint c est reconnu si l'item suivant est produit à partir de l'axiome conjoint de c :

$$\frac{}{[\top \rightarrow A' \bullet, i_c, f_c, (\emptyset, \emptyset, \emptyset)]}$$

$r \in A'$ où r est la racine de la partie basse de la DAP de la conjonction de coordination de la sélection courante.

Si l'item précédent est produit, alors les DAP du conjoint ainsi que la partie basse de la DAP de la conjonction de coordination sont considérées comme utilisées. Elles ne seront pas disponibles pour l'analyse de la phrase. De plus, A' sera donné comme axiome entre les positions i_c et f_c.

Analyse de la phrase sans les groupes coordonnés

Pour chaque item de reconnaissance $[\top \rightarrow A'\bullet, i_c, f_c, (\emptyset, \emptyset, \emptyset)]$ généré par une analyse de conjoint, nous allons donner l'axiome suivant à notre analyse.

$$\frac{}{[A' \rightarrow \bullet, i_c, f_c, (\square, \square, \square)]} Ax_h$$

Cet item indique que l'on a A', qui contient r la racine de la partie basse de la conjonction de coordination, entre les positions i_c et f_c. Nous utilisons le symbole $\square$ pour indiquer que les ensembles S, U, D n'ont pas de valeur réelle ici. Cet item ne peut être utilisé que dans une règle de complétion s'il est sélectionné par un item de la forme $[A \rightarrow \alpha \bullet \{r\}\beta, i, i_c, (S, U, D)]$ dans lequel r est une racine de partie basse. Notre nouvel axiome doit permettre de passer par r sans modifier S, U, D.

Pour préciser les choses, nous donnons la variante de la règle de complétion pour nos nouveaux axiomes :

$$\frac{[A \rightarrow \alpha \bullet \{r\}\beta, i, i_c, (S, U, D)][A' \rightarrow \bullet, i_c, f_c, (\square, \square, \square)]}{[A \rightarrow \alpha A' \bullet \beta, i, f_c, (S, U, D)]} Comp_h$$

Pour forcer l'utilisation de nos nouveaux axiomes, on peut aussi interdire qu'un item de la forme $[A \rightarrow \alpha\bullet\{r\}\beta, i, i_c, (S, U, D)]$ où r est une racine de partie basse de conjonction de coordination soit utilisé dans une règle de prédiction. Il ne pourra donc être utilisé que dans une complétion comme celle donnée plus haut.

Il ne reste plus qu'à donner la nouvelle règle d'axiome. La seule différence entre notre nouvel axiome et l'ancien et que l'ensemble S ne contient pas toutes les racines de DAP de la sélection mais seulement les racines des DAP qui n'appartiennent pas à un groupes conjoint.

$$\frac{}{[\top \rightarrow \bullet\emptyset, 0, 0, (S, \emptyset, \emptyset)]} Ax_3$$

où S est l'ensemble des racines qui n'ont pas été utilisées dans une analyse de conjoint.

8.5.3 Exemple

Nous allons dérouler un exemple pour fixer les idées. Nous prenons une phrase simple qui comporte deux coordinations, l'une de syntagmes nominaux, l'autre de syntagmes prépositionnels : $_0$ *Jean* $_1$ *et* $_2$ *Marie* $_3$ *mangent* $_4$ *au* $_5$ *restaurant* $_6$*ou* $_7$ *à* $_8$ *la* $_9$ *plage* $_{10}$. $_{11}$ Supposons que le filtrage lexical nous donne les triplets suivants $(0, d_2, 3)$ et $(4, d_7, 10)$ qui délimitent les groupes coordonnés *Jean et Marie* ainsi que *au restaurant ou à la plage*.

Comme aucun des groupes coordonnés n'inclut l'autre, on peut commencer par celui de son choix. Commençons par *Jean et Marie*. Il faut d'abord déterminer les parties basses gauche et droite de la DAP associée à *et* dans ce cas (coordination de syntagmes nominaux). On peut ensuite analyser *Jean* puis *Marie*.

Ensuite, on effectue les mêmes opérations pour *au restaurant ou à la plage*. On détermine la partie basse gauche de *ou* pour analyser *au restaurant* et la partie droite basse pour analyser *sur la plage*.

Comme ces quatre analyses réussissent, on procède à l'analyse de la phrase. Mais la phrase contient maintenant des trous : *[] et [] mangent [] ou []* . Mais l'analyse des conjoints nous a permis de remplir la table d'items qui témoignent de l'analyse de ces trous.

8.5.4 Conclusion

L'algorithme que nous venons de présenter permet diminuer l'ambiguïté de la prédiction en donnant plus d'information sur le contexte d'utilisation d'une DAP. C'est à dire que la règle de prédiction va générer moins d'items. Comme c'est la règle la plus coûteuse, elle permet de diminuer la complexité en moyenne de l'algorithme.

Notre heuristique tente de diviser l'analyse de la phrase d'entrée en plusieurs sous-analyses de conjoints qui permettent de reconstruire l'analyse de phrase complète. On pourrait dire que l'on utilise une technique de programmation dynamique qui donne la solution d'un problème difficile à partir de solutions de sous-problèmes.

Nous pensons également que cette technique peut s'adapter plus généralement. En adaptant l'utilisation des patrons interdits que nous avons présentée dans la section 6.9 pour en faire analyseur de constituants (*chunking*) on peut imaginer que l'on puisse découper l'analyse d'une phrase en sous-analyses de constituants que l'on pourrait brancher dans l'analyse de la phrase complète. C'est vraisemblablement une bonne piste pour augmenter les performance de l'analyse qui reste un problème NP-difficile tout en gardant des algorithmes complets, à la différence de l'utilisation de la borne présentée dans le section 7.3. Nous n'avons pas eu le temps de mener plus loin ces investigations ni d'entreprendre l'implantation de cette heuristique dans la plate-forme `leopar`. C'est pourquoi nous ne pouvons pas donner de résultats expérimentaux de cette partie de la thèse.

Enfin, pour indiquer qu'un sous-analyse ne modifie pas les ensembles S, U, D, nous avons utilisés le symbole $\square$. Il nous semble intéressant de généraliser cette utilisation pour indiquer des sous-analyses indépendantes du contexte. On pourrait ainsi beaucoup plus tabuler des sous-analyses réutilisables que dans l'algorithme actuel.

Conclusion et perspectives

Dans ce document, nous avons présenté les travaux que nous avons effectués en vue de la modélisation de la coordination dans les grammaires d'interaction. Nous nous sommes attachés à avoir une démarche fondée sur la validation expérimentale de chaque proposition. C'est pourquoi cette modélisation s'est accompagnée de travaux qui permettent de bénéficier d'une architecture assez complète pour que notre système de traitement automatique de la langue puisse couvrir les principaux cas de coordinations.

Nous pouvons d'ores et déjà distinguer plusieurs résultats et proposer quelques pistes pour de futures recherches.

Modélisation de la coordination. Nous avons présenté la modélisation proprement dite. Cette proposition est complètement lexicalisée, dans laquelle les deux conjoints sont au même niveau par rapport à la conjonction de coordination. Dans ce sens, notre proposition se rapproche de celle de Steedman [Ste85] ou Morrill [Mor94] pour les grammaires catégorielles mais la lexicalisation y est encore plus importante, puisqu'il n'existe pas d'équivalent aux combinateurs ni au raisonnement hypothétique dans les grammaires d'interaction.

Cette modélisation permet de traiter les principaux cas de coordinations : coordinations de constituants et de leur modificateurs, coordinations de non-constituants avec montée de nœuds, coordinations de séquences et coordinations avec ellipse, aussi appelé trou verbal. Cette proposition a été implantée dans notre système de traitement automatique des langues construit autour de l'analyseur `leopar`. Elle a ensuite été testée sur la TSNLP et nous avons pu constater que le taux de couverture est tout a fait satisfaisant, aussi bien au niveau de l'acceptation des phrases positives que du rejet des phrases négatives.

Nous avons également proposé, au-delà de notre première proposition, une extension de l'opération de superposition dans les grammaires d'interaction qui permettrait de prendre en compte les cas de coordinations disparates. Toutefois, cette extension suggère d'apporter des modifications à la grammaire principale pour être complètement exploitable.

Pour aller plus loin dans notre modélisation, nous envisageons trois directions :

- Premièrement, malgré notre hypothèse de départ, il est sûrement nécessaire d'étendre les GI pour mieux traiter la coordination. Tout d'abord, nous l'avons déjà évoqué, il faudrait pouvoir indiquer que la valeur d'un trait dépend d'autres valeurs. Par exemples pour indiquer le genre et la personne de la coordination de syntagmes nominaux, nous avons pu exprimer cette dépendance au niveau métagrammatical grâce à XMG, mais dans la grammaire cible cette dépendance n'apparaît plus. C'est

un premier point négatif. Nous n'arrivons pas à exprimer un fait linguistique. D'un point de vue pratique, pour justement transcrire cette dépendance dans la grammaire cible, nous devons multiplier les structures. La taille de notre grammaire est donc très importante. Il serait donc bon d'ajouter cette notion de dépendances dans les GI. Cela passe sûrement par l'ajout de fonctions (voire de relations) comme valeur de traits. Cette nouveauté ne serait pas utilisée que pour la modélisation de la coordination mais plus généralement pour modéliser les dépendances entre traits.
Une autre modification plus lourde de conséquence, consisterait à donner plus de souplesse aux structures employées par les GI. Plus particulièrement, nous voulons aborder deux points :

- Les DAP ne permettent pas d'exprimer en même temps plusieurs régimes (ou cadres de sous-catégorisation). Cela multiplie les structures associées aux mots. Nous pourrions peut-être nous inspirer de la grammaire produite par le compilateur de métagrammaires MGCOMP qui produit des TAG factorisées, où les structures du lexique associées aux mots ne sont plus des descriptions d'arbres simples mais des descriptions d'arbres dont certaines branches sont optionnelles, qui représentent toutes les descriptions d'arbres qui seraient associées par le lexique à ce mot, avec des gardes à chaque nœud qui indiquent quelles sont les fils effectifs dans le contexte de la phrase à analyser. Il faudrait évidemment dans ce cas modifier nos algorithmes de filtrage et d'analyse.
- Les structures d'analyses pourraient être elles aussi enrichies et devenir des graphes orientés acycliques. On peut se demander en effet pourquoi, à part pour des raisons de complexité calculatoire, la plupart des théories linguistiques se contentent d'arbres pour représenter les structures d'analyses. Pour la coordination, ces structures nous permettraient de refléter le partage de certaines parties de DAP. En particulier, les interfaces des conjoints pourraient être fusionnées pour représenter le rôle de la conjonction de coordination.

- Deuxièmement, tester sur corpus notre modélisation de la coordination disparate. Il faudra donc dans un premier temps mettre à our les outils XMG et `leopar` pour qu'ils prennent en compte les nouveaux connecteurs que nous avons introduits. Dans un second temps, il faudra mettre à jour notre grammaire. Comme nous l'avons déjà evoqué, il est fort probable que cette nouvelle possibilité remette en cause, certaines choix dans la grammaire principale de Guy Perrier. Nous pensons notamment à la modélisation des groupes prépositionnels. À plus long terme, c'est le bien de statut de notre grammaire de la coordination dans la grammaire principale qu'il faudra rendre plus clair.
- Il reste quelques points à améliorer dans la modélisation. En particulier, la coordination de séquence et la coordination avec ellipse. Quand la coordination ne met en jeu que deux conjoints, notre modélisation est satisfaisante. Au contraire, quand le nombre de conjoints est supérieur, notre modélisation ne permet pas de prédire la grammaticalité de la construction. Deux solutions sont envisageables :
  - soit on multiplie les structures pour ces coordinations en considérant le cas où seuls deux conjoints sont présents et le cas où les conjoints sont plus nombreux ;
  - Soit on utilise des DAP non ancrées, qui modéliseraient l'application d'une règle. On sortirait alors du cadre lexicaliste dans lequel nous nous sommes inscrits.

– Enfin, il est possible d'envisager de créer automatiquement une partie de la grammaire. En effet, notre modélisation est fondée sur la notion d'interface de DAP et de sa duale. Pour l'instant notre grammaire est écrite à la main. Prenons l'exemple des constituants. On commence par déterminer quelles sont les constituants coordonnables. De cette observation des DAP de la grammaire principale, on crée les DAP pour les conjonctions de coordination. À la place, on pourrait automatiquement considérer les DAP des constituants de la grammaire principale, calculer leur interface, sa duale, et créer la DAP correspondant à la coordination de ce type de constituant. Pour les non-constituants, cette approche semble cependant moins pertinente puisque l'on ne peut pas connaître l'interface de ces non-constituants coordonnables d'après la grammaire source.
  Pour aller encore plus loin, on peut penser à générer les DAP des conjonctions de coordinations *en ligne*. Notre méthode de filtrage permet d'identifier les conjoints potentiels de part et d'autres des conjonctions de coordination. Il suffirait alors d'en déterminer les interfaces pour construire la DAP associée à la conjonction faite d'une partie haute et des deux parties basses. Cette solution semble assez intéressante. Elle permettrait d'éviter d'écrire une grammaire de la coordination. Cependant, on accorderait alors un statut très spécifique au phénomène de coordination, ce que nous avons essayé de ne pas faire dans notre proposition.

Développement de grammaires. Nous nous sommes intéressés au problème de l'écriture des grammaires fortement lexicalisées. Dans ce contexte, nous avons participé à la mise au point de XMG, en collaboration avec Benoît Crabbé, Denys Duchier et Yannick Parmentier. Ce formalisme et l'outil qui en résulte nous ont permis de développer notre grammaire de la coordination. Plus généralement, ce système permet de s'affranchir des problèmes de redondance de l'information linguistique inhérents aux grammaires lexicalisées. Il permet également de s'accorder sur une méthodologie rigoureuse de développement de grammaires. Enfin, le fait que l'on puisse rapidement réviser et étendre une grammaire de très grande taille autorise les linguistes à avoir une approche plus expérimentale de la conception de grammaires, dans laquelle nous nous inscrivons complètement.

Ce logiciel rencontre un certain succès dans la communauté. C'est sûrement le signe qu'il est d'une certaine utilité pour les linguistes en répondant à un problème important.

Ici aussi, on peut imaginer quelques pistes d'amélioration :

– la possibilité d'utiliser de plus nombreuses dimensions, pour donner plus d'informations linguistiques. Pour l'instant, notre logiciel autorise uniquement l'emploi de trois dimensions : la première pour exprimer des descriptions d'arbres, la seconde pour manipuler des prédicats et la troisième des structures des traits. Pour pouvoir générer d'autre grammaires que des TAG et des GI, il faut disposer d'autres dimensions. On peut par exemple evisager que XMG propose une bibliothèque de dimensions paramétrables. Pour sa métagrammaire, l'utilisateur choisirait un certains nombres d'entre elles en les paramétrant. Il s'agit maintenant de savoir si l'on veut pousser cette possibilité jusqu'à autoriser la définition de nouvelles dimensions à l'intérieur même de XMG. Pour chaque nouvelle dimension, le travail comporte quatre parties :

- donner une sémantique à la conjonction pour la nouvelle dimension, c'est-à-dire donner une sémantique à la réutilisation,
- écrire un analyseur pour les nouvelles instructions,
- définir le comportement de la machine virtuelle pour chaque nouvelle instruction,
- éventuellement, donner un résolveur pour cette dimension.

Il faut donc imaginer un moyen de donner ces trois informations dans la métagrammaire elle-même. D'autres améliorations du logiciel sont demandées par les utilisateurs, notamment la possibilité de donner un autre système de couleurs que le système par défaut. Nous aimerions également pouvoir finir l'implantation du système de typage. On pourrait ainsi mieux vérifier la correction des métagrammaires. Grâce au typage, on pourrait aussi imaginer des systèmes de compositions de métagrammaire, à la manière de ce que propose [CSW06] pour modulariser des HPSG.

- Une dernière possibilité est d'aller vers plus d'abstraction. Au lieu de décrire des arbres, on pourrait imaginer un système où l'on décrit uniquement des contraintes linguistiques (relations prédicat/arguments, tête...) qui serait traduites vers des descriptions d'arbres par la suite, ou vers des structures de traits si l'on veut générer une HPSG.

Filtrage lexical. Nous avons repris les techniques originales de filtrage lexical pour en améliorer l'efficacité. Nous avons étudié la complexité théorique de la proposition qui existait dans l'analyseur `leopar`. Nous avons également enrichi cette proposition de deux façons. D'une part nous avons tiré profit de notre modélisation de la coordination pour proposer une heuristique qui permet à la fois de filtrer plus efficacement et de délimiter les segments coordonnés. Nous avons également proposé la possibilité de mieux tenir compte de l'ordre des mots dans cette étape de filtrage lexical par l'utilisation des patrons interdits.

Cet axe de recherche pourrait se poursuivre de la manière suivante :

- nous avons évoqué dans la section consacrée à la complexité de l'opération de filtrage, le fait que nous n'avons pas considéré certaines spécificités de nos grammaires. En effet, pour certaines valeurs de traits polarisés, le bilan de polarité dans une phrase évolue selon des schémas connus, et dans des intervalles relativement petits. Nous n'avons pas encore pris en compte ces informations. Elles pourraient nous permettre d'améliorer significativement notre méthode de filtrage. On pourrait alors retirer de nos automates de filtrage les chemins qui présentent des bilans de polarités trop extrêmes ou qui correspondent à des variations de polarités interdites.
- l'implantation peut être améliorée. Pour ce qui concerne le critère de saturation globale, nous sommes encore en train de tester de nouvelles heuristiques pour choisir l'ordre le plus pertinent possible pour réaliser nos intersections. Pour notre méthode de filtrage des conjonctions de coordination, il nous reste à implanter l'algorithme que nous avons défini dans ce document et non une grossière approximation comme c'est le cas à l'heure actuelle. Enfin, les patrons interdits méritent eux aussi une plus grande attention. Ils permettent de réellement augmenter les performances du filtrage lexicale, mais ils sont pour l'instant difficile à écrire. Enfin, nous n'avons pas

eu le temps d'implanter la reconnaissance des limites des groupes coordonnés.
- Nous voulons essayer de définir une notion de blocs (ou *chunks*) à partir des variations de polarités entre les états d'un automate de filtrage. Les expressions rationnelles qui décrivent habituellement ces blocs peuvent se traduire dans les GI par des variations de bilan polarités. Ce sera un premier pas vers l'analyse des phrases en surface (*shallow parsing*) qui pourra nous servir à améliorer les performances de l'analyseur.

Analyse dans les GI. Enfin, nous avons travaillé sur l'analyse dans les grammaires d'interaction. Nous avons développé un nouvel algorithme d'analyse prédictif, très différent des algorithmes existants. En effet, notre algorithme produit à chaque itération un arbre d'analyse de plus en plus complet, tandis que l'algorithme *shift-reduce* manipule les descriptions initiales pour obtenir une description simplifiée dont il faut trouver les modèles dans une deuxième étape.
- Nous avons là un premier pas vers la notion de sémantique opérationnelle pour les GI. On voit le modèle émerger petit à petit de l'analyse. Toutefois, il reste beaucoup de travail dans ce domaine. Notamment, il nous faut trouver un moyen d'exprimer de manière factorisée des analyses qui sont proches, c'est-à-dire des analyses qui auraient la même description simplifiée par l'algorithme *shift-reduce*.
- En ce qui concerne l'implantation, il nous reste beaucoup de travail également. Les performances de cet algorithme sont très en deçà de ce que nous espérions. C'est surtout dû à l'explosion combinatoires lors des premières prédictions. Une fois une prédiction suffisamment avancée, notre algorithme est plutôt efficace.
- Comme nous n'avons pas implanté dans la partie précédente la reconnaissance des groupes coordonnés, nous avons pas non plus eu le temps d'implanter l'extension de notre algorithme que nous proposons pour combiner des sous-analyses. Cet approche permettra sûrement d'améliorer les performances de l'analyses en pratique.

Bibliographie

[Abe02] A. Abeillé. Une grammaire électronique du français, 2002. CNRS Editions, Paris.

[Abe03] A. Abeillé. A lexicon- and construction-based approach to coordinations. In *Proceedings of the 10th International Conference on Head-Driven Phrase Structure Grammar*, pages 5–25, 2003.

[Abe06] A. Abeillé. In defense of lexical coordination. In Olivier Bonami and Patricia Cabredo Hofherr, editors, *Empirical Issues in Syntax and Semantics 6*, pages 7–36. CNRS, 2006.

[Ajd35] K. Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1 :1–27, 1935.

[AK91] H. Ait-Kaci. Warren's abstract machine : A tutorial reconstruction. In K. Furukawa, editor, *Logic Programming : Proc. of the Eighth International Conference*, page 939. MIT Press, Cambridge, MA, 1991.

[AL06] H. Anoun and A. Lecomte. Logical grammars with emptyness. In *Proceedings of the 11th conference Formal Grammar 06*, 2006.

[Bay96] S. Bayer. The coordination of unlike categories. *Language*, 72(3) :579–616, 1996.

[Bec00] T. Becker. Patterns in metarules. In A. Abeille and O. Rambow, editors, *Tree Adjoining Grammars : formal, computational and linguistic aspects*. CLSI Publications, 2000.

[BGP03] G. Bonfante, B. Guillaume, and G. Perrier. Analyse syntaxique électrostatique. *Traitement Automatique des Langues*, 2003.

[BGP04] G. Bonfante, B. Guillaume, and G. Perrier. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *20th Conference on Computational Linguistics, CoLing'2004, Genève, Switzerland*, pages 303–309, 2004.

[BH53] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29 :47–58, 1953.

[BJ95] S. Bayer and M. Johnson. Features and agreement. In *33rd Meeting of the Association for Computational Linguistics, San Francisco*, pages 70–76, 1995.

[BK03] K. R. Beesley and L. Karttunen. *Finite-State Morphology*. CLSI Publications, 2003.

[BL00] P. Bonhomme and P. Lopez. Resources for lexcalized tree adjoining grammars and xml encoding : Tagml. In *LREC2000*, 2000.

[BLRP06] G. Bonfante, J. Le Roux, and G. Perrier. Lexical disambiguation with polarities and automata. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *The 11th International Conference on Implementation and Application of Automata (CIAA 2006)*, 2006.

[BM72] R. Boyer and J. Moore. The sharing of structures in theorem proving programs. *Machine Intelligence*, 1972.

[Bor05] R. Borsley. Against conjp. *Lingua*, 115(4) :461–482, 2005.

[Bos96] J. Bos. Predicate logic unplugged. In *Proceedings of the Tenth Amsterdam Colloquium*, pages 133–142, 1996.

[Bre82] J. Bresnan. The passive in lexical theory. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 3–86. MIT Press, Cambridge, MA, 1982.

[Bre00] J. Bresnan. *Lexical-Functional Syntax*. Blackwell, 2000.

[BS04] J. Beavers and I. Sag. Coordinate ellipsis and apparent non-constituent coordination. In *Proceedings of the 11th International Conference on Head-Driven Phrase Structure Grammar*, pages 48–69, 2004.

[Can96] M.H. Candito. A principle-based hierarchical representation of LTAGs. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'96), Kopenhagen*, 1996.

[Can99] M.H. Candito. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au français et à l'italien*. PhD thesis, Université Paris 7, 1999.

[Car92] B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[CD04] B. Crabbé and D. Duchier. Metagrammar redux. In *International Workshop on Constraint Solving and Language Processing - CSLP 2004, Copenhagen*, 2004.

[CF00] A. Copestake and D. Flickinger. An open-source grammar development environment and broad-coverage english grammar using HPSG. In *LREC2000*, 2000.

[CFPS06] A. Copestake, D. Flickinger, C. Pollard, and I. Sag. Minimal recursion semantics : an introduction. *Research on Language and Computation*, 3(4) :281–332, 2006.

[CGR04] B. Crabbé, B. Gaiffe, and A. Roussanaly. Représentation et gestion du lexique d'une grammaire d'arbres adjoints, 2004. Traitement Automatique des Langues, 43,3.

[Cho57] N. Chomsky. *Syntactic Structures*. The Hague : Mouton, 1957.

[Cho95] N. Chomsky. *The Minimalist Program*. MIT Press, 1995.

[CK03] L. Clément and A. Kinyon. Generating LFGs with a metagrammar. In *Proceedings of the 8th International Lexical Functional Grammar Conference, Saratoga Springs, NY*, 2003.

[CKC83] A. Colmerauer, H. Kanoui, and M. Caneghemn. Prolog, theoretical principles and current trends. *Technology and Science of Informatics*, 2(4), 1983.

[Cop99] A. Copestake. *The (new) LKB system*. CSLI, Stanford University, 1999.

[CR98] T. Cornell and J. Rogers. Model theoretic syntax, 1998.

[Cra05] B. Crabbé. *Représentation informatique de grammaires fortement lexicalisées*. PhD thesis, Université Nancy 2, 2005.

[CS70] J. Cocke and J. Schwartz. Programming languages and their compilers : Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

[CSW06] Y. Cohen-Sygal and S. Wintner. Partially specified signatures : A vehicle for grammar modularity. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 145–152, 2006.

[de 01] P. de Groote. Towards abstract categorial grammars. In *ACL*, pages 148–155, 2001.

[DLP05] D. Duchier, J. Le Roux, and Y. Parmentier. Xmg : Un compilateur de métagrammaires extensible. In *Actes de TALN05*, 2005.

[DN00] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2000.

[Dow88] D. Dowty. Type raising, functional composition and non constituent conjunction. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 153–197. D. Reidel Publishing Company, 1988.

[DT99] D. Duchier and S. Thater. Parsing with tree descriptions : a constraint based approach. In *Natural Language Understanding and Logic Programming NLULP'99, Dec 1999, Las Cruces, New Mexico*, 1999.

[Ear70] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2) :94–102, 1970.

[Emm93] M. Emms. Parsing with polymorphism. In *EACL*, pages 120–129, 1993.

[Fli87] D. Flickinger. *Lexical rules in the hierarchical lexicon*. PhD thesis, Stanford University, 1987.

[FN92] D. Flickinger and J. Nerbonne. Inheritance and complementation : A case study of easy adjectives and related nouns. *Computational Linguistics*, 18(3) :269–309, September 1992.

[FPT85] D. Flickinger, C. Pollard, and Wasow T. Structure-sharing in lexical representation. In *ACL*, pages 262–267, 1985.

[GCR02] B. Gaiffe, B. Crabbé, and A. Roussanaly. A new metagrammar compiler. In *TAG+6*, 2002.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[GK03] C. Gardent and L. Kallmeyer. Semantic construction in FTAG. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest*, 2003.

[God05] D. Godard. Problèmes syntaxiques de la coordination et propositions récentes dans les grammaires syntagmatiques. *Langages*, 160 :3–24, 2005.

[GP07] C. Gardent and Y. Parmentier. SemTAG, une architecture pour le développement et l'utilisation de grammaires d'arbres adjoints à portée sémantique. In *Conference sur le Traitement Automatique des Langues Naturelles (TALN'2007)*, Toulouse, 2007.

[Hoa65] C.A.R. Hoare. Record handling. *Algol Bulletin*, 1965.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Hue05] G. Huet. A functional toolkit for morphological and phonological processing, application to a sanskrit tagger. *Journal of Functional Programming*, 15(4), 2005.

[Ing90] R. Ingria. The limits of unification. In *ACL*, pages 194–204, 1990.

[JS94] A. Joshi and B. Srinivas. Disambiguation of super parts of speech (or supertags) : Almost parsing. In *COLING'94, Kyoto*, pages 154–160, 1994.

[JY99] Marion J.-Y. From multiple sequent for additive linear logic to decision procedures for free lattices. *Theoretical Computer Science*, 224(1-2) :157–172, 1999.

[Kah04] S. Kahane. Grammaires d'unification polarisées. In *TALN2004, Fès, Maroc*, pages 233–242, 2004.

[Kas65] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, 1965.

[Kay75] R. Kayne. *French Syntax : The Transformational Syntax.* MIT Press, 1975.

[Kin00] A. Kinyon. Even better than supertags : Introducing hypertags, 2000.

[Kle56] S. Kleene. *Automata Studies*, chapter Representation of events in nerve nets and finite automata. Princeton University Press, 1956.

[KM88] R. Kaplan and J. Maxwell. Constituent coordination in lexical-functional grammar. *COLING-88*, pages 303–305, 1988.

[KNT01] A. Koller, J. Niehren, and R. Treinen. Dominance constraints : Algorithms and complexity. In M. Moortgat, editor, *Third International Conference on Logical Aspects of Computational Linguistics (Dec. 1998, Grenoble, France)*, volume 2014 of *Lecture Note in Artificial Intelligence.* Springer-Verlag, 2001.

[Kor96] L. Kornstaedt. Definition und Implementierung eines Front-End-Generators für Oz. Master's thesis, Universität Kaiserslautern und Programming Systems Lab, Universität des Saarlandes, September 1996.

[Kow05] E. Kow. Adapting polarised disambiguation to surface realisation. In *17th European Summer School in Logic, Language and Information - ESSLLI'05, Edinburgh, UK*, Aug 2005.

[KRS+06] A. Kinyon, O. Rambow, T. Scheffler, S. Yoon, and A. Joshi. The metagrammar goes multilingual : A cross-linguistic look at the v2-phenomenon. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8)*, 2006.

[Kup92] J. Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6 :225–242, 1992.

[Lam58] L. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3) :154–170, 1958.

[Lan64] P. J. Landin. The mechaincal evaluation of expressions. *Computer Journal*, 6(4) :308–328, 1964.

[LCP06] J. Le Roux, B. Crabbé, and Y. Parmentier. A constraint driven metagrammar. In *Proceedings of TAG+8*, 2006.

[Lec96] A. Lecomte. Grammaire et théorie de la preuve : une introduction. *Traitement automatique des langues*, 37(2), 1996.

[Mar06] J. Marchand. Algorithme de earley pour les grammaires d'interaction. Travaux universitaires, Nancy 1 – LORIA, 2006.

[Mer94] B. Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2) :155–172, 1994.

[MHF83] M. Marcus, D. Hindle, and M. Fleck. D-theory : talking about talking about trees. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 129–136, Morristown, NJ, USA, 1983. Association for Computational Linguistics.

[MM96] J. Maxwell and C. Manning. A theory of non-constituent coordination based on finite-state rules. In Proceedings of Workshop on Machine Learning and Robust Parsing, European Summer School in Logic Language and Information, 1996.

[MMVR95] M. Müller, T. Müller, and P. Van Roy. Multi-paradigm programming in Oz. In Donald Smith, Olivier Ridoux, and Peter Van Roy, editors, *Visions for the Future of Logic Programming : Laying the Foundations for a Modern successor of Prolog*, Portland, Oregon, 7 December 1995. A Workshop in Association with ILPS'95.

[Moo96] M. Moortgat. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1996.

[Mor94] G. Morrill. *Type Logical Grammar : Categorial Logic of Signs*. Kluwer, Dordrecht, 1994.

[Mou06] F. Mouret. A phrase structure approach to argument cluster coordination in french. In *13th International Conference on Head-Driven Phrase Structure Grammar*, 2006.

[Mou07] F. Mouret. *Grammaire des constructions coordonnées. Coordinations simples et coordinations à redoublement en français contemporain.* PhD thesis, Université Paris 7, 2007.

[Mus03] R. Muskens. *Resource Sensitivity in Binding and Anaphora*, chapter Language, Lambdas, and Logic. Kluwer, 2003.

[Ned99] M.J. Nederhof. The computational complexity of the correct-prefix property for TAGs. *Computational Linguistics*, 25(3) :345–360, 1999.

[NSS03] M.J. Nederhof, G. Satta, and S. Shieber. Partially ordered multiset context-free grammars and ID/LP parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies*, pages 171–182, Nancy, France, April 2003.

[Per00a] G. Perrier. From intuitionistic proof nets to interaction grammars. In *Proceedings of TAG+5*, 2000.

[Per00b] G. Perrier. Interaction grammars. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'2000), Saarbrucken, pp. 600 - 606*, 2000.

[Per01] G. Perrier. Intuitionistic multiplicative proof nets as models of directed acyclic graph descriptions. In *Proceedings of 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, pages 101–113, 2001.

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

[Per04] G. Perrier. La sémantique dans les grammaires d'interaction. *Traitement Automatique des Langues*, 45(3) :123–144, 2004.

[Per06] G. Perrier. A french interaction grammar. In *Workshop on Large-scale Grammar Development and Grammar Engineering*, 2006.

[Pro02] C. A. Prolo. Generating the XTAG english grammar using metarules. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'2002)*, 2002.

[PS01] G. Pullum and B. Scholz. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In *Proccedings of the 4th conference on Logical Aspects of Computational Linguistics*, 2001.

[PZ86] G. Pullum and A. Zwiky. Phonological resolution of syntactic feature conflict. *Language*, 62(4) :751–773, 1986.

[Ran07] A. Ranta. Modular grammar engineering in gf. *Research on Language and Computation*, 2007.

[Rob65] A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1) :23–41, 1965.

[RVS92] J. Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *30th Annual Meeting of the Association for Computational Linguistics*, pages 72–80, 1992.

[RWVS01] O. Rambow, D. Weir, and K. Vijay-Shanker. D-tree substitution grammars. *Computational Linguistics*, 27(1) :89–121, 2001.

[Sag02] I. Sag. Coordination and underspecification. In *The Proceedings of the 9th International Conference on Head-Driven Phrase Structure Grammar*, pages 267–291, 2002.

[SB07] M. Steedman and J. Baldridge. *Non-Transformational Syntax*, chapter Combinatory Categorial Grammar. Blackwell, 2007.

[Sch02] C. Schulte. *Programming Constraint Services*, volume 2302 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany, 2002.

[Sik93] K. Sikkel. *Parsing Schemata*. PhD thesis, Dept. of Computer Science, University of Twente, Enschede, NL, 1993.

[SJ88] Y. Schabes and A. Joshi. An earley-type parsing algorithm for tree adjoining grammars. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pages 258–269, Morristown, NJ, USA, 1988. Association for Computational Linguistics.

[SS06] D. Seddah and B. Sagot. Modélisation et analyse des coordinations elliptiques par l'exploitation dynamique des forêts d'analyse. In *Actes de TALN 2006*, pages 609–618, 2006.

[SSP95] S. Shieber, Y. Schabes, and F. Pereira Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2) :3–36, July/August 1995.

[Sta96] E. Stabler. Derivational minimalism. In *Logical Aspects of Computational Linguistics*, 1996.

[Ste85] M. Steedman. Dependency and coordination in the grammar of dutch and english. *Language*, 61(3) :525–568, Sept. 1985.

[Ste90] M. Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13(2) :207–263, 1990.

[SUP+83] S. Shieber, H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, techreport 4, pages 39–79. SRI International, Menlo Park, CA, November 1983. Final report for SRI Project 1894.

[SWB03] I. Sag, T. Wasow, and E. Bender. *Syntactic Theory : A Formal Introduction*. CSLI Publications, Stanford, 2 edition, 2003.

[Tap97] P. Tapanainen. *Finite-State Language Processing*, chapter Applying a Finite-State Intersection Grammar. MIT, 1997.

[Tes59] L. Tesnière. *Eléments de syntaxe structurale*. Librairie C. Klincksieck, Paris, 1959.

[TV05] F. Thomasset and É. Villemonte de La Clergerie. Comment obtenir plus des méta-grammaires. In *Proceedings of TALN'05*, Dourdan, France, June 2005. ATALA.

[Van90] P. Van Roy. Extended dcg notation : A tool for applicative programming in prolog. Technical report, Technical Report UCB/CSD 90/583, Computer Science Division, UC Berkeley, 1990.

[VH04] P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, March 2004.

[Vil04] E. Villemonte de la Clergerie. Designing efficient parsers with DyALog, 06 2004. Slides presented at GLINT, Universidade Nova de Lisboa.

[VS92] K. Vijay-Shankar. Using description of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4) :481–517, 1992.

[VSS92] K. Vijay-Shanker and Y. Schabes. Structure sharing in lexicalized tree adjoining grammars. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'92), Nantes, pp. 205 - 212*, 1992.

[War83] D. Warren. An abstract prolog instruction set. Technical report, SIR International, 1983.

[WMS+93] R. Weischedel, M. Meteer, R. Schwarz, L. A. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2) :155–172, 1993.

[XPVS98] F. Xia, M. Palmer, and J. Vijay-Shanker, K.and Rosenzweig. Consistent grammar development using partial-tree descriptions for lexicalized tree adjoining grammar. In *TAG+4*, 1998.

[XPVS99] F. Xia, M. Palmer, and K. Vijay-Shanker. Toward semi-automating grammar development. In *Proc. of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99), Beijing, China*, 1999.

[You67] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2) :189–208, 1967.

[YZS94] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2) :315–328, 1994.

Annexe A

NP-Complétude de l'optimisation d'intersections

A.1 Problème du voyageur de commerce

Nous allons prouver la NP-difficulté de notre problème par réduction polynomiale du problème du voyageur de commerce (*traveling salesman problem* dans [GJ79], ou TSP).

Une instance du TSP est un triplet (V, d, K) où $V = \{1, \ldots, n\}$ est un ensemble de villes, d la fonction de distance définie pour toute paire de villes différentes $d(i, j) \in \mathbb{N}^+$ et une borne $K \in \mathbb{N}^+$. Le problème consiste à décider s'il existe un circuit passant par toutes les villes et revenant à la ville de départ de longueur plus petite que K ou, plus formellement, s'il existe une permutation π des villes telle que $(\sum_{i=1}^{i=n-1} d(\pi(i), \pi(i+1))) + d(\pi(n), \pi(1)) \leq K$.

Pour faciliter l'écriture, quand π est une fonction $[1..n] \to [1..n]$ et s'il n'y a pas d'ambiguïté nous écrivons $\pi(n+1)$ pour $\pi(1)$ et $\pi(0)$ pour $\pi(n)$. La somme précédente peut donc se réécrire $\sum_{i=1}^{i=n} d(\pi(i), \pi(i+1)) \leq K$.

De plus, nous nous restreindrons aux cas où $d(i, j) \leq 2$. Le problème correspond à la réduction du problème du circuit hamiltonien (cf. [GJ79]) et reste NP-complet

Nous distinguons du TSP traditionnel tel qu'il vient d'être exposé, une variante dans laquelle le tour doit être de longueur exactement K. Nous appelons cette variante E-TSP.

A.2 Énoncé des problèmes

Nous présentons un premier problème d'optimisation d'intersection que nous enrichirons pour obtenir les second et troisième problèmes. Ce dernier correspond à notre problème de filtrage.

Problème 4. ***(IO1)*** *Soient $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ un ensemble de n automates , $B \in \mathbb{N}^+$ une borne K la taille ciblée. Existe-t'il une injection $\pi : [1..j] \to [1..n]$ telle que*

- $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| = K$
- *pour tout $k < j$, $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(k)}| \leq B$.*

En d'autres termes, peut-on trouver un sous-ensemble $\mathcal{A} \subseteq \mathcal{A}_n$ tel que $|\bigcap_{A \in \mathcal{A}} A| = K$ et toutes les étapes intermédiaires plus petites que B ?

Problème 5. ***(IO2)*** *Soient $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ un ensemble de n automates et $B \in \mathbb{N}^+$ une borne. Peut-on trouver une bijection π $[1..n] \rightarrow [1..n]$ telle que pour tout $j \leq n$ on ait $|(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?*

Problème 6. ***(IO3)*** *Soient $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ un ensemble de n automates et $B \in \mathbb{N}^+$ une borne. Existe-t'il une bijection π $[1..n] \rightarrow [1..n]$ telle que $\sum_{1 \leq j \leq n} |(\ldots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?*

Si nous nous ramenons à notre problème de désambiguation, des problèmes peuvent se reformuler de la manière suivante :

- (IO1) Étant donné un entier, est-ce que l'automate final résultat de la désambiguation peut être de cette taille.
- (IO2) Etant donné un entier, est ce que les automates intermédiaires produits par intersections successives peuvent ne jamais dépasser cet entier.
- (IO3) Etant donné un entier, est ce que le nombre total d'états créés durant les n intersections successives peut être inférieur à cet entier.

Le permier problème illustre la technique employée dans les réductions suivantes pour compter la longueur d'un tour. La deuxième reprend les idées précédentes et ajoute une contrainte sur l'ordre d'utilisation des automates. Toute la difficulté réside dans cette deuxième réduction. Finalement, la troisième réduction n'est qu'une adapatation de la de la précédente où une partie de l'automate qui compte la longueur du tour est effacçée à chaque intersection.

Dans les preuves suivantes, nous manipulons des automates sans cycle. Les problèmes ci-dessus peuvent donc être posées avec ou sans cette hypothèse supplémentaire.

Le dernier problème est le problème de décision sous-jacent au problème d'optimisation qui nous intéresse pour le filtrage. Notre problème est au moins aussi difficile que **IO3**.

A.3 Algorithmes non déterministes

Ces trois problèmes sont dans la classe de complexité NP. Ils peuvent être traiter en temps polynomial par une machine de Turing non-déterministe.

La partie non-déterministe consiste dans les trois cas à choisir uin permutation π puis

- pour (IO1), si une intersection intermédiaire est vide alors la réponse est «non» (sauf si $K = 0$ où la réponse est «oui»). De même, si la taille de l'automate intermédiaire est supérieure à B, la réponse est aussi négative. Sinon, on effectue l'intersection suivante. Quand j intersections ont été effectuées, on compare la taille de l'automate résultat avec K et on conclue. Remarquons que ces intersections sont effectuées dans le pire des cas dans un temps proportionnel à B^2 puisque tous les automates intermédiaires sont plus petit que B. Le problème est donc polynomial en B
- pour (IO2), si un automate intermédiaire est vide alors la réponse est «oui» sinon, si sa taille est supérieure à B (ou B^2 avant minimisation) la réponse est «non». Autrement, on passe à l'intersection suivante.
- pour (IO3), nous devons additionner les tailles des automates intermédiaires et vérifier que aucune somme n'est plus grande que B. Si une intersection est vide ou si une somme partielle dépasse B alors on peut conclure immédiatement.

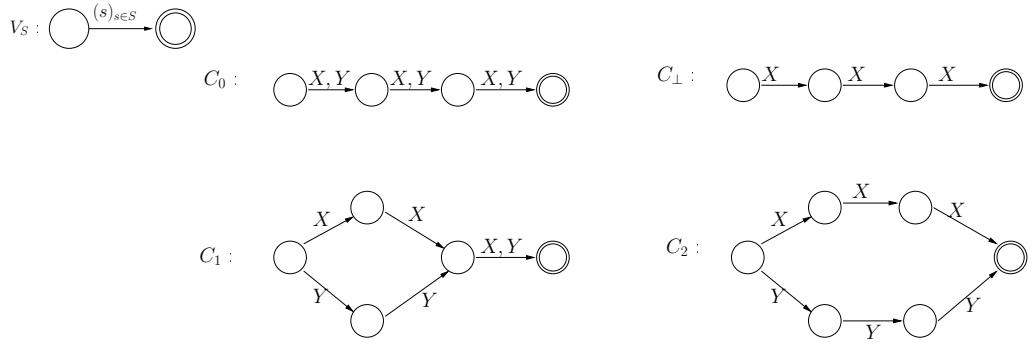


FIG. A.1 – Quelques automates utiles pour nos réductions

A.4 NP-Complétude

Pour conclure à la NP-complétude des trois problèmes évoqués, nous prouvons dans cette section leur NP-difficulté. Comme nous avons montré qu'ils sont dans NP précédemment, nous pouvons conclure qu'ils ont NP-complets.

Proposition 11. ***(IO1)*** *est NP-complet.*

Démonstration. Nous considérons quelques automates de base que nous associerons pour modéliser IO1. Ils sont donnés en figure A.1.

Remarquons que pour $i \in \{0,1,2\}$, nous avons $|C_i| = |C_0| + i$. en d'autres termes ces automates codent les distances entre deux villes. Observons également que $C_i \cap C_0 = C_i$. C'est à dire que C_0 est l'élément neutre pour les intersections de C_i. Nous notons A' la transformation d'un automate A consistant à ajouter le symbole prime à chaque symbole de A.

Étant donné une instance de E-TSP (V, d, k), nous considérons un ensemble d'automates $\mathcal{A}_{i,j,m}$ avec $i, j \in V$ et $m \leq n$ où n est le nombre de villes dans V. Nous considérons également un automate minimal quelconque D de taille $6 \times n + 3$.

Ces automates définissent les langages rationnels suivants :

$$L(\mathcal{A}_{i,j,1}) = V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-2} C_0 V_i + V'_{V\setminus\{1\}} D'$$

et pour $n > m > 1$,

$$L(\mathcal{A}_{i,j,m}) = (V_{V\setminus\{i,j\}} C_0)^{m-1} V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-m} + V'_{V\setminus\{m\}} D'$$

et

$$L(\mathcal{A}_{i,j,n}) = V_j C_0 (V_{V\setminus\{i,j\}} C_0)^{n-2} V_i C_{d(i,j)} V_j + V'_{V\setminus\{n\}} D'$$

Intuitivement, l'automate $\mathcal{A}_{i,j,m}$ correspond au choix d'aller de la ville i à la ville j à la m^{ieme} étape du tour, ce qui est équivalent au choix $\pi(m) = i$ et $\pi(m+1) = j$.

Nous considérons également un automate dit témoin $\mathcal{A} = (V_V C_0)^n V_V$. Nous remarquons que $|\mathcal{A}_{i,j,m}| = |\mathcal{A}| + d(i,j) + |D'|$.

La réduction que nous proposons a la forme $(V, d, k) \mapsto ((\mathcal{A}_{i,j,m})_{i,j,m}, 2|D'|, |\mathcal{A}| + k)$.

Pour la correction du codage, il faut remarquer que s'il existe un tour défini par π de longueur exactement k alors

$$\bigcap_{1 \leq m \leq n} \mathcal{A}_{\pi(m),\pi(m+1),m} = V_{\pi(1)} C_{d(\pi(1),\pi(2))} V_{\pi(2)} C_{d(\pi(2),\pi(3))} \cdots C_{d(\pi(n),\pi(1))} V_{\pi(1)}$$

qui a la taille $|\bigcap_{1 \leq m \leq n} \mathcal{A}_{\pi(m),\pi(m+1),m}| = |\mathcal{A}| + \sum_{1 \leq m \leq n} d(m, m+1)$.

Donc, si le E-TSP admet une solution alors son codage est une solution pour (IO1) (Nous ne traitons pas la taille des automates intermédiaires qui fera l'objet de considérations minutieuses dans la prochaine preuve).

Pour la réciproque, nous considérons l'ensemble $\mathfrak{A}$ d'automates $(\mathcal{A}_{i,j,m})_{i,j,m}$ fermé par intersection. Si $A \in \mathfrak{A}$ n'est pas vide, il a les propriété suivantes :

(i) $A = A_1 + A_2$ avec
- $A_1 = \emptyset$ ou
- $A_1 = V_{\alpha_1} C_{\beta_1} V_{\alpha_2} C_{\beta_2} \cdots V_{\alpha_n} C_{\beta_n} V_{\alpha_{n+1}}$, $\alpha_i \subseteq V$, $\beta_i \in \{0, 1, 2\}$,

et $A_2 = V_S D'$ avec $S \subseteq \{1..n\}$;

(ii) dans (i), si $\alpha_j = \{k\}$ pour un j, alors aucun autre α_ℓ ne contient k pour $\ell \leq n$,

(iii) Dans (i), $\beta_i = 0$ si et seulement si $i \in S$,

(iv) Dans (i), si $\beta_i \neq 0$, alors $\alpha_i = \{k\}, \alpha_{i+1} = \{\ell\}$ sont des singletons et $\beta_i = d(k, \ell)$.

(v) Dans (i), $\alpha_1 = \alpha_{n+1}$

Partant de (i), nous constatons que $|A| = |A_1| + |A_2| - 1$ si les deux sous-composants sont non-vides. Sinon, $|A| = |A_1| + |A_2|$. Donc dans le pire des cas, on a $|A| \leq 2 + \sum_{i=1}^{n} |C_{\beta_i}| + |D'| < 2 \times |D'|$ et la borne sur les automates intermédiaires est toujours respectée

Depuis (iii), on note que $V'_S D'$ est vide si et seulement si $\forall j : \beta_j \neq 0$. Enfin de (iv), (ii) et (v) nous conluons le fait F suivant :

(F) pour tout i, l'ensemble $\alpha_i = \{k_i\}$ est un singleton et $\pi : [1..n] \to [1..n]$ qui envoie $i \mapsto k_i$ est une bijection et $k_1 = k_{n+1}$. Ensuite, $|D'| > |\mathcal{A}| + k$, $|A| = |\mathcal{A}| + k$ si et seulement si S est vide.

Le fait **(F)** ci-dessus montre que l'automate construit correspond à un tour acceptable. □

Proposition 12. ***(IO2)*** *est NP-complet.*

Démonstration. Nous réduisons le TSP vers IO2. Soit (V, d, k) une instance du TSP et 2 est la distance maximale entre deux villes et $|V| = n$. Ici encore, nous construisons pour chaque distance $d(i, j)$ n automates selon les positions possibles de cette distance dans un tour. Nous construisons donc n^3 automates $\mathcal{A}_{i,j,p}$. Techniquement, par rapport à la preuve précédente, nous devons avoir un contrôle plus fort sur les tailles de tous les automates d'intersection car la condition que nous devons vérifier s'applique à chacun des automates intermédiaires. Nos automates sont composés de trois parties :

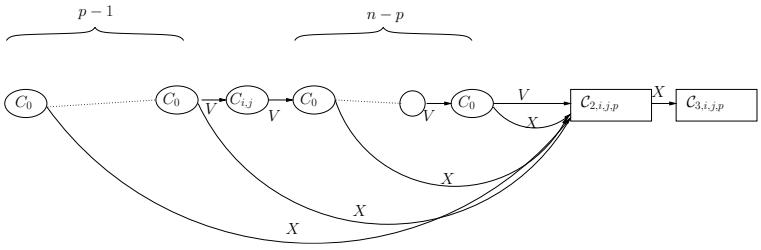


FIG. A.2 – L'automate $\mathcal{A}_{i,j,p}$ et détail du premier composant.

1. le premier composant que nous appelons $\mathcal{C}_{1,i,j,p}$ est présenté sur la figure A.2. Il est chargé de compter la distance totale du tour entrepris, comme pour (IO1). La différence est qu'ici on indice pas les V par des ensembles de villes. Les composants suivants seront chargés de vérifier la validité du tour.

   Les états finaux des sous-composants C_0 sont connectés à l'état initial du composant suivant par un arc étiqueté par X. Donc si les n distances du tour sont instanciées (comme dans IO1) alors seul le dernier V connectera ce composant au suivant.
2. Le second composant ($\mathcal{C}_{2,i,j,p}$) est chargé de vérifier que les villes du tour forment une chaîne cohérente. Il est représenté sur la figure A.3. Le point clé consiste à observer que s'il est intersecté avec $\mathcal{C}_{2,j,k,p+1}$ alors l'automate résultat est de la même taille. Par contre si les villes ne coïncident pas, le résultat est plus grand de $2n$ états.
3. Le troisième composant ($\mathcal{C}_{3,i,j,p}$), présenté figure A.4 empêche et de choisir une position plus d'une fois et de considérer une position p sans avoir considérer toutes les positions précédentes d'abord. Sinon, il grossit de $4n$ états.

Finalement nous aurons besoin d'un dernier automate, T, comme sur la figure A.5

Observons maintenant les automates dont nous disposons. La taille de $\mathcal{A}_{i,j,p}$ est $|\mathcal{A}_{i,j,p}| = |\mathcal{C}_{1,i,j,p}| + |\mathcal{C}_{2,i,j,p}| + |\mathcal{C}_{3,i,j,p}|$. Plus particulièrement :

$$
\begin{aligned}
|\mathcal{C}_{1,i,j,p}| &= 2n + d(i,j) \\
|\mathcal{C}_{2,i,j,p}| &= \begin{cases} 6n+2 & \text{si } p = n \\ 4n+2 & \text{sinon} \end{cases} \\
|\mathcal{C}_{3,i,j,p}| &= 3(p-1)(4n) + 2(n-p+1)(4n) + 2n = 2n(4n+2p-1)
\end{aligned}
$$

$$
|\mathcal{A}_{i,j,p}| = \begin{cases} 2 + d(i,j) + 4n(2+p+2n) & \text{si } 1 \leq p < n \\ 2 + d(i,j) + 2n + 4n(2+3n) & \text{sinon} \end{cases}
$$

Rappelons que nous voulons prouver que $i_1, i_2, \ldots, i_n, i_1$ est un tour pour le TSP avec une longueur inférieure à k si et seulement si chaque automate intermédiaire de l'intersection

$$
\bigcap_{1 \leq i \leq m, \alpha(i) \in I \subset [1..n]^3} \mathcal{A}_{\alpha(i)} \cap T \bigcap_{m+1 \leq j \leq n^3, \beta(j) \in [1..n]^3 \setminus I} \mathcal{A}_{\beta(j)}
$$

est un automate de taille inférieure à $B = 2 + k + 4n(1+2n)$.

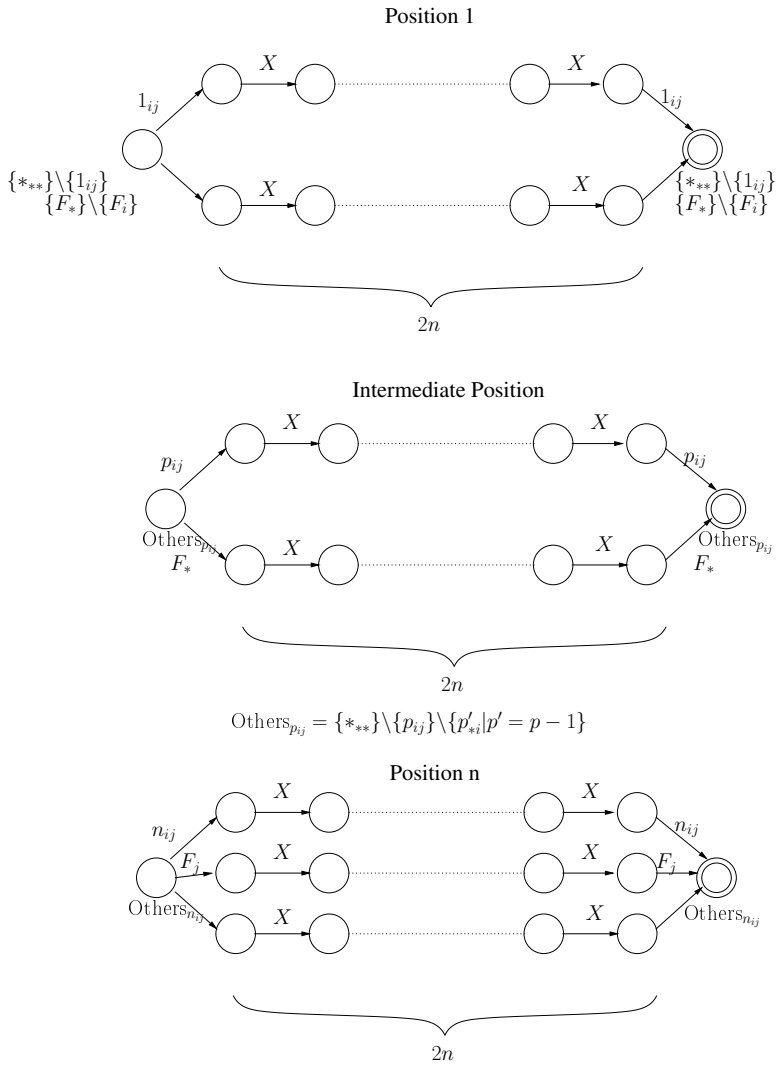


FIG. A.3 – Le deuxième composant de l'automate $\mathcal{A}_{i,j,p}$

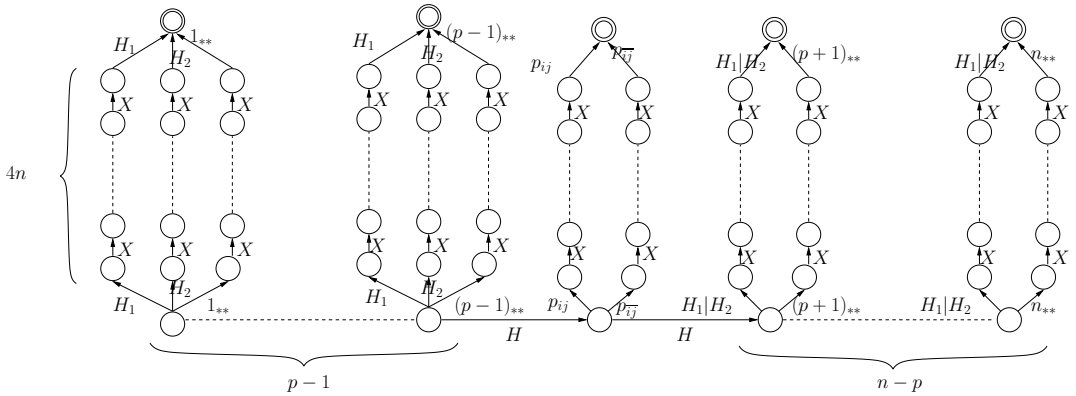


FIG. A.4 – Le troisième composant de l'automate $\mathcal{A}_{i,j,p}$

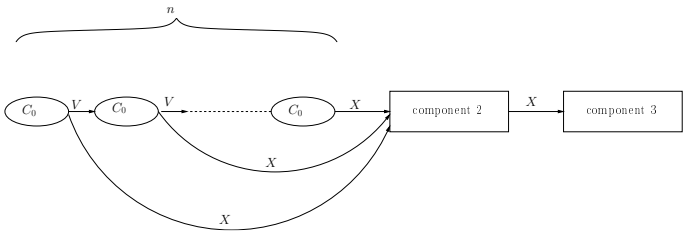


FIG. A.5 – L'automate T

Quelques observations Sans perte de généralité, nous supposons que $k \leq 2n$. Sinon, le TSP est trivial (tous les tours sont solutions). Ceci implique que parmi les automates $(\mathcal{A}_{i,j,p})_{i,j,p}$ considérés, seuls les automates $(\mathcal{A}_{i,j,1})_{i,j}$ sont plus petit que B.

i) $|\mathcal{A}_{i,j,1}| = 2 + d(i,j) + 4n(1+2n) < B$

ii) sinon,

$$\begin{array}{rcl} |\mathcal{A}_{i,j,p}| & \geq & 2 + d(i,j) + 4n(1+p+2n) \\ & \geq & 2 + d(i,j) + 4n(2+2n) + 4n(p-1) \\ & > & 2 + d(i,j) + 4n(1+2n) + k > B \end{array}$$

Nous remarquons ensuite que :

iii) $|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$

iv)

$$|\mathcal{C}_{1,i,j,p} \cap \mathcal{C}_{1,k,l,q}| = \begin{cases} 2n + d(i,j) + d(k,l) & \text{si } p \neq q \\ 2n + \max(d(i,j), d(k,l)) & \text{sinon} \end{cases}$$

v)

$$|\mathcal{C}_{2,i,j,p} \cap \mathcal{C}_{2,k,l,q}| = \begin{cases} |\mathcal{C}_{2,i,j,p}| & \text{si } q = p+1 \text{ et } j = k \\ |\mathcal{C}_{2,i,j,p}| + 2n & \text{sinon} \end{cases}$$

vi)

$$|\mathcal{C}_{3,i,j,p} \cap \mathcal{C}_{3,k,l,q}| = \begin{cases} |\mathcal{C}_{3,i,j,p}| & \text{si } q = p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n|p-q| & \text{si } q > p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n & \text{si } q \leq p \end{cases}$$

En général, pour un suite de villes préfixe d'un tour $i_1, i_2, \ldots, i_k$ avec $k \leq n+1$ (si $k = n+1$ nous imposons $i_k = i_1$) de notre instance du TSP, nous avons

$$\begin{array}{rcl} |\bigcap_{1\leq p\leq k} \mathcal{A}_{i_p,i_{p+1},p}| & = & |\bigcap_{1\leq p\leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\bigcap_{1\leq p\leq k} \mathcal{C}_{2,i_p,i_{p+1},p}| + |\bigcap_{1\leq p\leq k} \mathcal{C}_{3,i_p,i_{p+1},p}| \\ & = & |\bigcap_{1\leq p\leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| \\ & = & 2n + (\sum_{1\leq p\leq k} d(i_p, i_{p+1})) + 4n + 2 + 2n(4n-1) \\ & = & 2 + (\sum_{1\leq p\leq k} d(i_p, i_{p+1})) + 4n(1+2n) \end{array}$$

Correction de la réduction. Nous montrons premièrement que s'il existe un tour $i_1, i_2, \ldots, i_n, i_1$ avec une distance totale inférieur à k alors toutes les intersections $A_{i_1,i_2,1} \cap \ldots \cap A_{i_j,i_{j+1},j}$ pour j allant de 1 à n sont de taille inférieur à B tel que nous l'avons défini. Nous le faisons par induction sur le nombre d'intersections intermédiaires.

Comme nous l'avons déjà dit plus haut, l'automate initial doit être $A_{i_1,i_2,1}$ car tous les autres $A_{i_1,i_2,p}$ sont trop grands. Ensuite, par application de l'égalité définie au paragraphe précédent :

$$\begin{array}{rcl} |A = \bigcap_{1\leq p\leq n} \mathcal{A}_{i_p,j_{p+1},p}| & = & 2 + (\sum_{1\leq p\leq n} d(i_p, i_{p+1})) + 4n(1+2n) \\ & \leq & 2 + k + 4n(1+2n) \end{array}$$

Nous avons donc montré que les n premières intersections codent directement le tour de notre instance du TSP. Maintenant, observons que $A \cap T = \emptyset$ car tous les C_i du premier composant sont différents de C_0. Il s'en suit que si l'instance du TSP considérée a une solution, la suite $\mathcal{A}_{i_1,i_2,1}, \ldots, \mathcal{A}_{i_n,i_1,n}, T, S$ où S est une suite quelconque sur $\{(\mathcal{A}_{i,j,p})_{i,j,p}\} \backslash \{\mathcal{A}_{i_k,i_{k+1},k} : k \leq n\}$ est une solution de IO2.

Complétude de la correction À partir des automates que nous avons définis, toute suite d'intersections qui les fait tous intervenir est de la forme :

$$A = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i}) \cap T \cap (\bigcap_{(\alpha_j)_j \in [1..n]^3 \backslash I} \mathcal{A}_{\alpha_j})$$

Supposons que ni A ni aucun automate intermédiaire de cette intersection n'est plus grand que B. C'est vrai en particulier pour $A' = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i})$. Notons $m = |I|$. Nous pouvons déduire que :

- α_1 est de la forme $(x_1, y_1, 1)$ sinon $\mathcal{A}_{\alpha_1}$ est trop grand.
- si $\alpha_i = (x_i, y_i, p)$ alors $\alpha_{i+1} = (x_{i+1}, y_{i+1}, p+1)$ et $m \leq n$ sinon le composant trois serait trop grand. De ces deux premières observations on déduit que α_i est de la forme (x_i, y_i, i)
- si $\alpha_i = (x_i, y_i, i)$ et $\alpha_{i+1} = (x_{i+1}, y_{i+1}, i+1)$ alors $y_i = x_{i+1}$, sinon le composant deux serait trop grand
- si $m = n$ alors α_m est de la forme (x_m, x_1, m), sinon le composant deux serait trop grand.
- Finalement, $m \geq n$ sinon $|A' \cap T| > B$. On a donc $m = n$. Remarquons que ceci implique à son tour $|A' \cap T| = 0$.

Nous avons montré que A' code un tour $i_1, i_2, \ldots, i_n, i_1$ dans notre instance du TSP. De plus, la taille de A' d'après la construction ci-dessus est :

$$\begin{array}{rcl} |A'| = |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| & \leq & B \\ |\mathcal{C}_1| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| & \leq & B \\ |\mathcal{C}_1| + 4n + 2 + 2n(4n-1) & \leq & B \\ |\mathcal{C}_1| + 2 + 2n(1+4n) & \leq & 2 + k + 4n(1+2n) \\ |\mathcal{C}_1| & \leq & k + 2n \\ 2n + d(i_1, i_2) + \cdots + d(i_n, i_1) & \leq & k + 2n \\ d(i_1, i_2) + \cdots + d(i_n, i_1) & \leq & k \end{array}$$

Et donc le tour codé par notre intersection est une solution pour l'instance considérée. □

Proposition 13. ***(IO3)*** *est NP-complet.*

Démonstration. Le codage est le même que précédemment pour (IO2) sauf pour le premier composant des automates $\mathcal{A}_{i,j,p}$.

Les distances non-instanciées avant la position p sont modélisées par le sous-composant $C_\perp$ qui en quelque sorte efface la distance précédente par intersection.(Notons que $C\perp \cap C_{i\in\{0,1,2\}} = C_\perp$)

Le langage défini par notre automate $\mathcal{A}_{i,j,p}$ est donc

$$L(\mathcal{A}_{i,j,p}) = (V(C_\perp + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{p-1}VC_{d(i,j)}(V(C_0 + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{n-p}V(\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p})$$

Nous devons également modifier la borne à ne pas dépasser. Cette borne doit tenir compte des composants deux et trois que nous ajoutons systématiquement ($B \geq n \times (|\mathcal{C}_{1,i,j,1}| + |\mathcal{C}_{2,i,j,1}| + |\mathcal{C}_{3,i,j,1}|)$) ainsi que des parties de $\mathcal{C}_{1,i,j,1}$ qui ne sont pas effacées par les intersections de $C_\perp$.

Nous posons donc $B = k + n(2 + 2n + |\mathcal{C}_2| + |\mathcal{C}_3|) = k + n(2 + 8n + 8n^2)$

Le reste de la preuve est similaire.

□

Ce dernier problème correspond à notre méthode de sélection lexicale. Si le problème de décision (avec une borne B donnée) est un NP-complet alors trouver la meilleure borne (la plus petite) est encore plus difficile.

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Alain LECOMTE, Professeur, UFR SDL, Université Paris 8, Saint-Denis

Monsieur Aarne RANTA, Professeur, Chalmers University of Technology & Göteborg University, Sweden

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur LE ROUX Joseph

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"La coordination dans les grammaires d'interaction"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Informatique** »

Fait à Vandoeuvre, le 01 octobre 2007
Le Président de l'I.N.P.L.,
F. LAURENT

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55