De la reconnaissance de formes linguistiques à l'analyse syntaxique

Sébastien Paumier

Université de Marne-la-Vallée
U.F.R. d'Informatique

2003

Thèse
pour obtenir le grade de
docteur de l'Université de Marne-la-Vallée
Discipline : Informatique
Spécialité : Informatique linguistique

DE LA RECONNAISSANCE DE FORMES LINGUISTIQUES À L'ANALYSE SYNTAXIQUE

Sébastien Paumier

Volume 1

présentée et soutenue publiquement
le 4 juillet 2003

Directeurs de thèse : †Professeur Maurice Gross
Professeur Éric Laporte

Jury : Christian Choffrut (rapporteur)
Franz Guenthner (rapporteur)
Éric Laporte
Jee-Sun Nam
Dominique Perrin

Ce n'est pas dans la science qu'est le bonheur, mais dans l'acquisition de la science.
Edgar Poe

Remerciements

Je souhaite dédier cette thèse à la mémoire de Maurice Gross, qui a su me faire partager son goût pour cette discipline passionnante, née de la rencontre entre la linguistique et l'informatique. Durant les deux trop courtes années durant lesquelles j'ai eu le privilège de le côtoyer, sa porte a toujours été ouverte pour répondre à mes questions. À travers les nombreuses discussions que j'ai eu avec lui, il m'a enseigné ce qu'était une démarche scientifique rigoureuse, et m'a appris à avoir un regard à la fois critique et ouvert. Pour tout cela et pour bien d'autres choses, j'aurais voulu pouvoir le remercier de vive voix.

J'adresse un immense merci à Éric Laporte, pour avoir accepté d'encadrer mes travaux, pour avoir toujours trouvé le temps de discuter, de répondre à mes questions et de me conseiller, et pour m'avoir soutenu lorsque cela était nécessaire.

Je remercie vivement Christian Choffrut et Franz Guenthner qui m'ont fait l'honneur d'être les rapporteurs de cette thèse, ainsi que Dominique Perrin et Jee-Sun Nam qui ont accepté d'être membres de mon jury.

Un très grand merci à Christian Leclère, qui a eu l'infinie patience de répondre à toutes mes questions de novice en linguistique. Merci également à Annie Meunier, Tita, Antoinette Renouf et Jacques Labelle, pour leur disponibilité et leur bonne humeur. Je tient à exprimer ma gratitude à Mirella Connena pour l'attention particulière qu'elle a eue à Bari, ainsi qu'à Elisabete Ranchhod, Paula et Cristina qui m'ont si bien accueilli à Lisbonne. Je remercie également Harald Ulland pour la collaboration fructueuse que j'ai avec lui.

Que soient remerciés ici mes colocataires de bureau, Matthieu et Takuya, pour leur collaboration, leur bonne humeur et tous les fous rires occasionnés durant cette thèse qui m'ont permis de décompresser un bon coup de temps en temps.

Un grand merci à tous les autres membres du LADL et de l'IGM, pour leur disponibilité de chaque instant et l'ambiance fabuleuse qu'ils savent si bien entrenir grâce à leurs solides traditions festives. Quelques mentions spéciales en vrac, pour Rémi qui m'a aidé à me dépatouiller avec les mystères des interfaces graphiques en Java, pour Laurent et Julien qui ont apporté au laboratoire une ambiance sonore très sympathique bien qu'un tantinet bestiale, pour Julien Cervelle qui m'a soulagé de la rédaction d'un sujet de projet de compilation quand j'étais en pleine bourre de fin de rédaction, pour Anastasia qui m'a apporté une aide inestimable pour les formalités administratives, et enfin pour Patrice, qui lorsqu'on arrive à le capturer, se

révèle d'une aide précieuse pour appréhender les subtilités angoissantes de la communication homme-machine.

Je remercie également toute l'équipe belge, Cédrick Fairon, Anne Dister, Patrick Watrin, Laurent Kevers, Michel Thomas et Claude Devis, pour leur soutien, leur bonne humeur, leur invitation, leur chouchen ;) , leurs remarques et leurs rapports de bugs si précieux.

Un grand merci à tous les utilisateurs d'Unitex qui m'ont soutenu par leurs encouragements et leur enthousiasme, en particulier : l'équipe du CIS qui m'a chaleureusement accueilli à Munich, Alexis et Oto qui m'ont fait l'honneur de traduire ma prose, ainsi qu'Agata, Claude et Olivier qui ont eu le courage de se plonger dans mon code.

Un grand merci à tous mes autres collègues que je n'ai pas cité ici, mais qui m'ont soutenu et aidé durant cette thèse.

Pour leur soutien constant, je remercie mes parents, Juanito, Titou, Claire et mes grands-parents, ainsi que Marie-Hélène, Jean-Pierre, Olivier, Séverine, Raphaël, Oriane et Tite'Mélie.

Pour leur amitié et tous les moments que nous avons passé ensemble, et grâce auxquels ils m'ont permis de me vider la tête quand j'en avais besoin, un grand merci à Ti'Pierre, Charles (tu as vu, je n'ai rien mis ;)), Laetitia et Olga.

Et enfin, bien sûr, pour son indéfectible soutien et ses encouragements permanents, un gigantesque merci à ma femme, Anne-Cécile.

Résumé

La plupart des descriptions des langues naturelles consistent en une accumulation de règles décrivant le comportement des différents éléments du langage. Cependant, si beaucoup de règles générales ont été établies, leurs exceptions sont rarement étudiées, ce qui fait que ces descriptions sont incomplètes, voire erronées lorsque le nombre de cas particuliers n'est pas négligeable. Pour remédier à cela, un examen minutieux des phrases élémentaires du français a été mené au LADL. Ces travaux ont abouti à une description très fine des propriétés syntaxiques de ces phrases, codée sous la forme de matrices appelées tables de lexique-grammaire. En 1993, Emmanuel Roche a montré que l'on pouvait exploiter les données de ces tables pour effectuer l'analyse automatique de phrases simples. Nous avons étudié un moyen d'étendre ces travaux, de façon à traiter exhaustivement les données du lexique-grammaire, afin de pouvoir analyser n'importe quelle phrase simple du français. Le traitement complet de toutes ces constructions est une opération qui prendra beaucoup de temps, ce qui nous a conduit à nous poser la question de la maintenance de données à long terme. Nous avons ainsi pris le parti d'utiliser un formalisme de description très simple, celui des grammaires locales, afin que les données soient représentées d'une façon la plus claire possible, et qu'elles puissent être maintenues facilement. Dans un premier temps, nous avons testé la puissance de description de notre modèle, à travers l'examen de diverses constructions. Bien que simple, il s'est avéré bien adapté à la description de structures syntaxiques, et a mis en évidence le fait que la distinction entre la recherche de motifs et l'analyse syntaxique n'est qu'une question d'échelle de description. En contrepartie de cette simplicité, nous avons dû faire face aux problèmes informatiques soulevés par l'exploitation de nos grammaires, en particulier à cause des ordres de grandeur atteints par les données accumulées. Nous avons donc étudié dans un second temps différentes méthodes permettant de manipuler ces données, les unes mettant en jeu des transformations opérant sur les grammaires, les autres concernant les programmes destinés à les appliquer. Les résultats que nous avons obtenus montrent que notre modèle est viable, et que l'accumulation des grammaires décrivant toutes les structures de phrases simples du français est réalisable, ce qui devrait permettre, à terme, d'obtenir un analyseur syntaxique exact pour ces constructions.

Abstract

Most of natural language descriptions are made of sets of rules modelling the behavior of words. However, whereas many general rules have been established, exceptions to these rules are not often studied. Consequently, these rules are incomplete, and even inaccurate when the number of particular cases is too large. To solve this problem, the LADL team has studied in detail basic sentences of French. This work led to a very fine description of the syntactic properties of these sentences, stored in matrices called lexicon-grammar tables. In 1993, the proof was made by Emmanuel Roche that these data could be used to perform automatic parsing. We have studied a way to extend this work, in order to take into account the whole data contained in lexicon-grammar tables, so that we could analyse any basic sentence of French. As this study will take a long time, we had to address the issue of maintenance of data through a long period of time. In fact, we tried to make the formalism to design our grammars as simple as possible, so that they would be easily maintained. In a first step, we verified that this formalism was powerful enough, through the examination of several syntactic structures. We have shown that this formalism, though simple, was adapted to syntactic description and parsing, which suggests that the difference between pattern matching and syntactic analysis is just a matter of scale. In return, we had to solve computational problems, mainly related due to the huge amount of data we had to deal with. So, in a second step, we studied methods to handle these data in reasonable time, either by transforming grammars or by optimizing programs. Our results show that our model is reliable, and so, that it is possible to build an exploitable set of grammars describing all the basic sentences of French. They show the way for efficient syntactic parsers for these constructions.

Table des matières

Introduction

L'étude des langues naturelles est une activité pratiquée depuis plusieurs millénaires, comme en témoignent les écrits des grammariens grecs et latins. À cette époque, le langage était très lié à la logique, car on accordait une grande importance aux raisonnements et aux démonstrations, ce qui a donné naissance à la rhétorique. Ce rapprochement de la langue et des mathématiques se renforça à partir du XVI^{e} siècle, grâce à un courant de pensée qui mêlait la philosophie et les sciences, et dont le *Discours de la méthode* de Descartes est l'une des plus célèbres illustrations. Il faudra cependant attendre le milieu du XX^{e} siècle pour que ce lien s'accentue encore et que les langues soient étudiées d'un point de vue formel. Les travaux de Zellig S. Harris permirent d'établir un modèle mathématique du langage, dans lequel les objets élémentaires sont des phrases basiques, auxquelles s'appliquent des opérateurs permettant d'obtenir des phrases plus complexes.

Cette conception du langage a conduit un grand nombre de chercheurs à tenter de décrire les langues naturelles de la même façon que les langages formels. Cependant, bien qu'ils aient utilisé des outils mathématiques, la majorité des linguistes a travaillé sans tenir compte de principes scientifiques élémentaires, comme le fait qu'une théorie doive être vérifiée expérimentalement sur des données significatives, et que ces expériences doivent être reproductibles. À cause de cela, beaucoup de phénomènes ont été généralisés hâtivement, bien souvent pour obtenir des règles générales qui puissent rendre compte simplement et élégamment des langues naturelles. Cependant, l'apparente généralité de ces règles disparaît dès lors que l'on s'intéresse aux nombreuses exceptions auxquelles elles sont sujettes.

Critiquant l'absence de vérifications expérimentales de telles règles, Maurice Gross entreprit avec son équipe du LADL l'examen exhaustif des phrases simples du français, afin de disposer de données fiables et chiffrées sur lesquelles il serait possible de faire des expériences scientifiques rigoureuses. Pour cela, chaque verbe fut étudié de manière à tester s'il vérifie ou non des propriétés syntaxiques comme le fait d'admettre une complétive en position sujet. 6000 verbes furent ainsi examinés à l'aide d'environ 300 propriétés. Il ressortit de cette étude que pour 6000 verbes simples, on comptait environ 12000 emplois différents. De plus, il apparut que chacun de ces 12000 emplois présentait un comportement syntaxique différent. Ce résultat est fondamental, car il met clairement en évidence le fait qu'on ne puisse pas décrire le français avec des règles générales, ce qui s'avéra aussi pour toutes les autres langues étudiées selon cette méthodologie rigoureuse.

Les résultats de cette étude ont été codés dans des matrices appelées tables de lexique-grammaire. Ces tables constituent donc une description précise du comportement syntaxique

de chaque verbe du français. Ayant été construites selon un formalisme clair, ces tables présentent une structure idéale pour être exploitées informatiquement. En 1993, Emmanuel Roche montra qu'il était possible d'utiliser ces ressources pour effectuer l'analyse syntaxique automatique de phrases du français. D'autres travaux furent menés par Jean Senellart pour exploiter de façon similaire les descriptions des expressions figées. Ces différentes études ont montré que la finesse des informations contenues dans les tables de lexique-grammaire permettait d'obtenir des analyses beaucoup plus fines que celles obtenues avec peu ou pas de ressources linguistiques. Toutefois, ces travaux n'ont pas exploré la totalité des phrases simples du français car, d'une part, ils n'ont pas traité toutes les tables de lexique-grammaire, et d'autre part, parce qu'ils ont laissé de côté certaines constructions.

Notre objectif à long terme est d'utiliser toutes les ressources des tables de lexique-grammaire pour obtenir un système capable d'analyser n'importe quelle structure de phrase simple. Étant donné qu'il a fallu près de 15 ans aux membres du LADL pour accumuler toutes ces données, il est plus que probable que ce travail demandera beaucoup de temps et nécessitera la collaboration de plusieurs personnes. De plus, la maintenance de données informatiques sur de longues périodes est un réel problème, notamment en raison des évolutions technologiques rapides. Or, le modèle proposé par Roche nous a semblé inadapté, d'une part parce qu'il n'est pas suffisamment lisible pour que plusieurs personnes puissent aisément travailler sur les mêmes grammaires, et d'autre part parce que les grammaires ne peuvent pas être dissociées des programmes destinés à les appliquer, ce qui nuit à leur réutilisabilité. Ces raisons nous ont donc conduit à abandonner ce modèle pour celui, encore plus simple, des grammaires locales utilisées pour la recherche de motifs.

Notre étude se divise donc en deux parties : l'une linguistique, l'autre informatique. Dans le premier chapitre, nous aborderons les problèmes liés à la conversion des tables de lexique-grammaire en un ensemble de grammaires locales. Nous montrerons qu'il est possible de reculer les limites de la recherche de motifs jusqu'à la confondre avec l'analyse syntaxique. Pour y parvenir, nous présenterons une extension d'un modèle existant, les graphes patrons, nous permettant de gérer des descriptions de structures syntaxiques de façon générique. Nous montrerons ainsi comment engendrer un ensemble de grammaires devant permettre d'effectuer l'analyse syntaxique des phrases simples du français. Enfin, nous testerons la viabilité de notre approche en examinant comment elle permet de rendre compte de divers phénomènes linguistiques.

Dans la mesure où la taille des grammaires que nous engendrons est considérable, il était nécessaire de s'assurer que de telles quantités de données pourraient être exploitées informatiquement. C'est pourquoi, dans le second chapitre, nous discuterons des problèmes algorithmiques liés à l'application de nos grammaires. Nous présenterons tout d'abord le programme de recherche de motifs que nous avons mis au point, en montrant tout le bénéfice que l'on peut retirer d'une représentation segmentée des textes. Nous présenterons ensuite deux méthodes permettant d'accélérer les calculs en transformant les grammaires, l'une basée sur une variante de la mise en forme normale de Greibach, l'autre fondée sur une approximation d'un transducteur à états finis. Nous évoquerons également la possibilité de diviser l'analyse en plusieurs passes, en exploitant le fait que certaines grammaires sont appelées un grand nombre

de fois. Enfin, nous montrerons une méthode simple permettant de garantir que les données accumulées pourront être exploitées, quelle que soit leur quantité.

Au cours de notre étude, nous avons été amené à réécrire un certain nombre d'outils informatiques. À cette occasion, nous avons apporté des solutions à quelques problèmes concernant d'autres langues que le français. L'objet de notre troisième chapitre est de présenter ces problèmes et les méthodes que nous avons utilisées pour les résoudre. Nous parlerons en premier lieu des difficultés rencontrées en thaï du fait de l'absence de séparateur entre les mots. Nous présenterons un critère simple permettant de ramener les ambiguïtés de segmentation d'un texte en mots à un niveau comparable à celui des ambiguïtés entre mots simples et mots composés dans les langues à séparateurs. Nous présenterons ensuite le cas des mésoclises en portugais, et nous montrerons comment nous avons utilisé nos outils déjà existants pour générer dynamiquement des grammaires de normalisation, permettant d'expliciter les constructions mises en jeu. Enfin, nous discuterons du problème de l'analyse des mots polylexicaux en norvégien. Nous montrerons comment, à partir d'un dictionnaire électronique exhaustif et de quelques règles simples, il est possible d'analyser automatiquement ces mots, dont la structure est trop productive pour qu'ils puissent être traités par dictionnaire.

Notations

Nous reprendrons ici les notations usuelles du LADL.

Adj	Adjectif
Adj-n	Substantif morphologiquement relié à un adjectif
Adv	Adverbe
Dét	Déterminant
Ddéf	Déterminant défini
Dind	Déterminant indéfini
DU	*du* ou *de la*
GN	Groupe nominal
Inf	Infinitive
LE	*le*, *la* ou *les*
Loc	Préposition locative
Modif	Modifieur (relative, adjectif ou complément de nom)
N	Substantif quelconque. Peut également désigner un groupe nominal argument d'un verbe. Dans ce cas, ce symbole est accompagné par un indice indiquant la position de l'argument dans la phrase de base. Par exemple, N_0 désigne le sujet. Une structure comme N_1 V N_0 doit donc être comprise comme une phrase obtenue par inversion du sujet et du premier complément de la phrase de base.
Nhum	Substantif humain
N-hum	Substantif non humain
Nnr	Substantif non restreint
Npc	Nom de partie du corps
P	Phrase à l'indicatif
Psubj	Phrase au subjonctif
Ppv	Pronom préverbal
Prép	Préposition
Que P	Complétive introduite par *que*
UN	*un*, *une* ou *des*
V	Verbe. Lorsque ce symbole est accompagné d'un exposant i, cela signifie que le verbe est coréférent à l'argument d'indice i (*i.e.* V^1 indique que le verbe est coréférent à N_1)
Vaux	Syntagme décrivant un auxiliaire verbal, éventuellement composé (*devoir*, *pouvoir commencer à*, etc.)

V-a	Adjectif morphologiquement relié à un verbe
V-ant	Verbe au participe présent
V-n	Substantif morphologiquement relié à un verbe
Vpp	Verbe au participe passé
W	Suite quelconque de compléments
*	Devant une phrase, indique qu'elle n'est pas acceptable, ou alors dans un sens différent de celui envisagé
?	Devant une phrase, indique qu'elle est acceptable bien que peu naturelle
?	Devant une phrase, indique qu'elle est très douteuse
=	Signe permettant d'expliciter le contenu d'une catégorie ou d'une structure (*i.e.* N_0 = *Que P*)
ε	Symbole de l'expression vide

On utilisera également le signe + comme opérateur de disjonction. Par exemple, la séquence

Paul mange (ε + du gâteau)

représente les deux phrases

Paul mange
Paul mange du gâteau

La présence de l'étoile devant une séquence à l'intérieur d'une disjonction indique que la ou les phrases construites avec cette séquence sont inacceptables. Ainsi, la séquence

*(Paul + *Qu'il fasse beau) mange (*que Marie vienne + du gâteau)*

représente les quatre phrases

**Paul mange que Marie vienne*
Paul mange du gâteau
**Qu'il fasse beau mange que Marie vienne*
**Qu'il fasse beau mange du gâteau*

Chapitre 1

Lexique-grammaire

> La simplicité est la sophistication suprême.
> *Léonard de Vinci*

Ce chapitre traite de l'aspect linguistique de nos travaux. Nous commencerons par y présenter le cadre théorique du lexique-grammaire dans lequel s'inscrivent nos recherches. Nous définirons ensuite nos objectifs en décrivant de quelle manière nous comptons utiliser les données du lexique-grammaire du français à des fins d'analyse syntaxique. Nous discuterons des problèmes liés à l'élaboration et à la maintenance de grandes quantités de ressources linguistiques, et nous présenterons un moyen simple de gérer un ensemble de tables de lexique-grammaire. Nous détaillerons ensuite les problèmes que soulève notre approche au travers de l'étude de plusieurs phénomènes syntaxiques, et nous montrerons qu'elle semble adaptée à la description des phrases simples du français.

1.1 Cadre théorique

1.1.1 Théorie transformationnelle et grammaire générative

Les fondements du lexique-grammaire sur lequel nous basons notre travail remontent à la théorie transformationnelle de Zellig Harris qui considère que l'analyse d'une phrase met en jeu une série de transformations appliquées à une ou plusieurs phrases élémentaires ([44]). Par exemple, la phrase

Ce livre a été déchiré par Paul

est analysée comme étant obtenue par passivation de la phrase

Paul a déchiré ce livre

Ce principe s'appuie sur une conception mathématique du langage où les phrases élémentaires sont des objets auxquels s'appliquent des opérateurs ([45], [46]), ce qui a très naturellement

conduit à examiner les langues naturelles avec les outils mathématiques des langages formels.

Dans [15], Chomsky tente de déterminer quel formalisme est le mieux adapté pour rendre compte de ces suites de transformations. Pour cela, il prend comme point de départ le fait qu'une analyse est une suite de dérivations allant de la phrase élémentaire vers la phrase complexe (la notion de dérivation est ici la même que dans les langages formels). Une grammaire décrivant de telles règles est dite *générative*. Selon ce principe, la phrase précédente s'analyse de la manière suivante :

Paul a déchiré ce livre $\Longrightarrow^{passivation}$ *Ce livre a été déchiré par Paul*

Dans ce système, deux phrases sont considérées comme liées si l'une peut être dérivée de l'autre, ce qui sous-entend que les transformations sont orientées. Cette hypothèse peut poser problème lorsque l'une des phrases intermédiaires de la dérivation est impossible, car cela peut alors empêcher de dériver une phrase correcte d'une autre phrase correcte. Considérons les phrases élémentaires suivantes [1] :

(1a) Paul croit que Marie est heureuse
(1b) Paul voit que Marie est heureuse

Si l'on réduit la complétive *que Marie est heureuse*, on obtient les phrases

*(2a) *Paul croit Marie être heureuse*
(2b) Paul voit Marie être heureuse

Si l'on efface ensuite le verbe *être*, on obtient les phrases

(3a) Paul croit Marie heureuse
(3b) Paul voit Marie heureuse

On constate donc que l'inacceptabilité de (2a) ne doit pas être considéré comme rédhibitoire, car il serait alors impossible de dériver (3a) de (1a). Ce cas de figure neutralise donc le procédé de dérivation par transformations successives. Pour contourner ce problème, une possibilité est de poser la séquence d'opérations qui permet de dériver (1b) en (3b) comme une règle, et de dire que la phrase (2a) est une exception à cette règle. Ces procédés sont courants en grammaire générative, car ils permettent de conserver à la grammaire son caractère général. Ce point est très important aux yeux de Chomsky, car il est lié à l'idée que la grammaire explique la langue (voir chapitre intitulé *The explanatory power of linguistic theory* dans [15]). Une autre façon de traiter ces problèmes consiste à adapter l'appareillage formel à des règles que l'on crée de plus en plus complexes. Cette approche a conduit à un niveau d'abstraction tel que de nombreux travaux ne considèrent même plus les données qu'ils sont censés expliquer ([35]).

[1] Cet exemple est tiré de [34].

1.1.2 La linguistique traitée comme une science expérimentale

La théorie transformationnelle de Harris sera utilisée de façon différente par Maurice Gross. Il constate que les linguistes se soucient fort peu de tester la validité de leurs règles ou hypothèses au delà de quelques exemples choisis, ce qu'il reproche à la grammaire générative dans [35]. Il constate également l'absence d'exhaustivité dans la plupart des démarches, comme par exemple donner des listes exactes des exceptions aux règles. Ainsi, dans [31] qui est un ouvrage de référence, ne sont indiqués que quelques exemples de verbes transitifs directs ne tolérant pas la construction passive :

La Meuse prend sa source en France
Nous avons un verger
Il peut tout sur les peuples

Se basant sur les méthodes utilisées en sciences expérimentales, Maurice Gross décide alors de remédier à cet état de fait en étudiant la langue selon les principes élémentaires suivants :
- les expériences doivent porter sur des quantités de données significatives ;
- elles doivent être reproductibles ;
- les exceptions aux règles doivent être parfaitement décrites ;
- les modifications du cadre formel doivent être très soigneusement justifiées et extrêmement rares.

Selon une telle approche, la théorie ne peut être formulée qu'à partir des données observées. Cependant, de telles données n'avaient jamais été accumulées en linguistique. L'une des raisons principales venait du fait que la quantité de données était telle que les linguistes ont jugé la tâche irréalisable.

1.1.3 La phrase simple comme unité de sens

Afin d'examiner exhaustivement les données, Maurice Gross prend comme point de départ l'étude des phrases simples du français. Il reprend ainsi l'idée que l'unité minimale de sens est la phrase et non le mot. Le principe qu'il adopte est donc de répertorier les phrases simples et d'étudier les transformations qu'elles peuvent subir. À l'exception de quelques cas particuliers comme les mots-phrases tels que *oui* [2] ou les interjections telles que *parbleu*, les phrases comportent toujours un verbe. Les phrases simples ont donc été indexées par leurs verbes. Les propriétés étudiées pour chacune de ces phrases sont essentiellement des propriétés formelles portant sur la syntaxe plutôt que sur la sémantique, comme par exemple l'existence de la construction passive. Cette attitude est nécessaire si l'on veut garantir la reproductibilité des tests ([34]). Toutefois, certaines propriétés sémantiques ont été prises en compte lorsqu'elles pouvaient être testées de façon claire. Ainsi, l'une de ces propriétés concerne la nature, humaine ou non, des arguments du verbe. Dans une phrase comme

**Max épluche Luc*

[2] *oui* peut être considéré comme une phrase car il en possède certaines propriétés, comme le fait de pouvoir apparaître dans une complétive :

Paul pense que oui

il est clair que l'impossibilité vient de ce que le deuxième argument (ici *Luc*) ne peut être humain [3]. Les propriétés utilisées dans l'élaboration du lexique-grammaire se trouvent résumées dans [53].

1.1.4 Un codage formel et simple

Contrairement à d'autres types de descriptions, le formalisme a été ici réduit à sa plus simple expression. Concrètement, le lexique-grammaire des phrases simples du français est une matrice de dimension 14000×300 dont les lignes sont indexées par les emplois des verbes simples et les colonnes par les propriétés. C'est une matrice codant des propriétés syntaxiques de façon binaire : lorsqu'un verbe vérifie une propriété, la case située à l'intersection des ligne et colonne correspondantes contient le signe $+$, le signe $-$, sinon. Cette matrice peut également contenir des éléments lexicaux, comme par exemple le verbe à l'infinitif. Naturellement, il apparaît de fortes dépendances entre certaines propriétés. Par exemple, les propriétés mettant en jeu l'argument N_2 sont évidemment non vérifiées lorsque le verbe n'a pas de deuxième complément. Certaines familles de verbes ont donc pu être établies sur la base de propriétés communes. Ainsi, la matrice générale a été présentée sous la forme de plusieurs sous-matrices, appelées *tables de lexique-grammaire*, dans lesquelles les colonnes ne contenant que des $+$ ou que des $-$ ont été ignorées. Par exemple, la propriété N_0 *V Que P* qui est la propriété définitionnelle de la table 6, n'apparaît pas dans cette table car la colonne correspondante ne contiendrait que des $+$. Les tables de lexique-grammaire des verbes simples du français sont présentées dans [8], [9], [34] et [43]. Dans leur version la plus actuelle, elles sont au nombre de 59. La figure 1.1 présente un extrait de la table 32NM qui concerne essentiellement des verbes admettant un complément direct N_1 numérique, mais pas le passif.

1.1.5 Conclusions sur la langue

Plusieurs conclusions ont découlé de l'examen exhaustif des verbes simples du français qui a ainsi été réalisé. Tout d'abord, il est apparu que pour 5000 verbes, on obtenait environ 14000 emplois différents. Ces différences ont été mises en évidence par des différences de propriétés. Par exemple, le verbe *diviser* de la phrase

Max divise le tas en trois paquets

est à distinguer de celui de la phrase

Max divise quatre par deux

car dans le premier cas, le verbe n'admet pas de complément numérique introduit avec la préposition *par*. Il a donc été possible d'affirmer que des propriétés syntaxiques permettaient de distinguer les emplois d'un verbe. Plus généralement, il a été montré que, malgré l'existence de similitudes, il n'existait pas deux verbes possédant exactement le même comportement syntaxique. Cette conclusion est lourde de conséquences, car elle anéantit tout espoir de

[3] Sauf en cas de métonymie : *Max a épluché Freud (= les écrits de Freud)*

N0 = Npc	N0 = N-hum	N0 = Nnr	N0 = V-a	32NM	N1 V	N1 = Nhum	N1 = N-hum	N1 = le fait Qu P	N1 = V-a	N1 = Dnum Nmes	Ppv = le	N-1 V N0	N0 V Adj	N0 V Dnum V-a	N0 V à N1	N-1 V N0 (<E>+à) N1	Exemple
-	+	-	-	accepter	-	-	+	-	-	-	+	-	-	-	-	-	Ce salon§accepte§vingt personnes
-	+	-	-	accueillir	-	-	+	-	-	-	+	-	-	-	-	-	Ce salon§accueille§vingt personnes
-	+	-	-	accuser	-	-	+	-	-	+	+	-	-	-	-	-	Max§accuse§80 kilos
-	+	-	-	accuser	-	-	+	+	-	-	-	-	-	-	-	-	Max§accuse§ses trente ans
-	+	-	-	admettre	-	-	+	-	-	-	+	-	-	-	-	-	On§admet§50 personnes dans cette salle
-	+	-	-	affecter	-	-	+	-	-	-	-	-	-	-	-	-	Ces cristaux§affectent§une forme géométrique
-	+	-	-	afficher	-	-	+	-	-	+	-	-	-	-	-	-	Les valeurs ont§affiché§un repli
-	+	-	-	aimer	-	-	+	+	-	-	+	-	-	-	-	-	La plante§aime§l'eau
-	+	-	-	approcher	+	-	+	-	-	+	+	-	-	-	-	-	Cette maison§approche§les deux millions
-	+	-	-	arpenter	-	-	-	-	-	+	+	-	-	+	+	+	Ce terrain§arpente§30 arpents
-	+	-	-	atteindre	-	-	+	-	-	+	+	-	-	-	-	-	Max§atteint§80 kilos
+	+	+	-	avoir	-	-	+	-	-	+	+	-	-	-	-	-	Max§a§(une soeur+une voiture+des sous)
-	+	-	-	avoisiner	-	-	+	-	-	+	+	-	-	-	-	-	Ce sac§avoisine§les 20 kg
-	+	-	-	battre	+	-	+	-	-	-	+	-	-	-	-	-	La montre§bat§les secondes
-	+	-	-	cacher	-	-	+	-	-	-	-	-	-	-	-	-	Son calme§cache§(son+une grande)angoisse
-	+	-	-	caler	-	-	+	-	-	+	+	-	+	-	-	-	Ce bateau§cale§80 cm
+	+	-	-	calibrer	-	-	-	-	+	+	+	+	-	-	-	+	Ces tuyaux§calibrent§dix centimètres
-	-	-	-	chausser	-	-	+	-	-	+	-	-	+	-	-	-	Max§chausse§du 40
+	+	-	-	chiffrer	-	-	-	-	+	+	+	+	+	-	-	-	Cette maison§chiffre§deux millions
-	+	-	-	cocoter	-	-	+	-	-	-	-	-	+	-	-	-	Ida§cocote§l'ail

FIG. 1.1 – Extrait de la table 32NM

formuler des règles générales qui pourraient expliquer la langue. Des travaux similaires sur d'autres langues ([11], [23], [24], [58], [59], [66], [76], [79] et [80]) ont confirmé ce caractère irrégulier qui semble partagé par toutes les langues.

Les irrégularités observées dans les possibilités de transformations sont telles qu'il apparaît impossible de procéder autrement qu'en énumérant ces possibilités. Expliciter systématiquement l'équivalence de deux phrases en utilisant une suite de dérivations orientées est rendu impossible par l'absence non prédictible de constructions [4]. On ne peut pas non plus dire que deux phrases sont équivalentes si elles dérivent d'une même phrase élémentaire qui servirait de référence, car quelle que soit la construction envisagée, il existe toujours un verbe pour lequel elle n'est pas possible. La seule façon d'exprimer que deux phrases sont équivalentes est donc de dire qu'elles appartiennent à une même classe, sans chercher à préciser de quelle façon les relier. La nécessité de ce principe de *classes d'équivalence* montre qu'il est illusoire de chercher l'explication de la langue dans la grammaire. Au contraire, elle met en évidence le fait qu'une langue comporte une très grande quantité d'accidents lexicaux et syntaxiques et qu'il n'est pas possible d'en rendre compte par des règles générales. Le codage de ces phénomènes au cas

[4] On trouvera des discussions sur les problèmes soulevés lorsque l'on considère les transformations comme des dérivations orientées dans [40] et [70].

par cas est donc indispensable.

1.2 Objectif

1.2.1 Analyse exacte contre analyse approximative

Il existe deux grands courants en traitement automatique des langues : les systèmes statistiques et les systèmes utilisant des ressources linguistiques. On trouvera quelques descriptions de techniques à base de statistiques dans [12], [13], [14], [18], [19], [77].

Si l'on se réfère à la procédure d'évaluation GRACE 1998 qui mettait en compétition des systèmes d'étiquetage de texte, c'est le système GREYC ([81]), basé sur des règles très simples et sans utilisation de dictionnaire[5], qui a remporté la première place avec une précision de 94,5%. Cet ordre de grandeur de 95% de réussite se retrouve dans plusieurs travaux n'utilisant pas ou très peu de ressources linguistiques.

Toutefois, ce qui est présenté comme une performance doit être relativisé. Un système qui commet environ 5% d'erreur se trompe 1 mot sur 20, ce qui équivaut à une faute par phrase. Ce taux d'erreur est donc loin d'être négligeable. En outre, les systèmes qui ne sont pas basés sur des ressources linguistiques consistantes ont le grave défaut de ne pas rendre compte des cas peu fréquents. On voit donc mal comment ils pourraient surmonter les 5% d'erreurs résiduelles.

Nous pensons que seule l'utilisation de descriptions linguistiques exactes et exhaustives permettra de franchir cette barrière. Quelques travaux ont été entrepris dans cette voie, dont entre autres [21], [22], [42] et [83]. Toutefois, aucun d'eux ne repose sur des données d'une finesse comparable à celle du lexique-grammaire. Nous nous proposons donc d'exploiter les données codées par les tables de verbes du lexique-grammaire afin de produire les analyses syntaxiques exactes des phrases simples du français.

1.2.2 Nécessité d'une évolution du modèle DELSYN

Nos travaux s'inspirent fortement de ceux menés par Emmanuel Roche sur les verbes simples ([68]) et de ceux menés par Jean Senellart sur les expressions figées ([72]). Dans [68], Emmanuel Roche montre qu'il est possible d'effectuer l'analyse syntaxique exacte des phrases simples du français. Pour cela, il code les structures des phrases élémentaires ainsi que les règles des transformations permettant d'en dériver des constructions plus complexes dans un dictionnaire syntaxique baptisé DELSYN. Les règles décrites dans ce dictionnaire sont des fonctions destinées à s'appliquer soit à une phrase, soit au résultat de l'application d'une autre règle. L'analyse d'une phrase s'effectue donc en deux étapes :

- construction de l'automate représentant les ambiguïtés lexicales de la phrase ;
- application répétée des règles contenues dans le dictionnaire DELSYN sur cet automate jusqu'à obtention d'un point fixe.

[5] L'auteur se targue de n'utiliser que 200 mots en guise de ressources lexicales.

Les conclusions de Roche nous montrent qu'il est tout à fait possible de faire de l'analyse syntaxique en utilisant de grandes quantités de données. Nous pensons toutefois que la méthode qu'il a utilisée présente un défaut. En effet, bien que reposant sur un formalisme très simple si on le compare à d'autres ([16] donne un aperçu de la complexité que l'on peut atteindre), les règles n'en sont pas moins codées avec un système d'opérateurs propres à l'auteur. Nous voyons à cela deux conséquences importantes.

Tout d'abord, ces notations nuisent à la lisibilité des règles. Cela introduit des problèmes de maintenance des grammaires dus à des risques d'erreurs humaines. Ce manque de lisibilité est également un obstacle au travail coopératif, car cela complique la manipulation des grammaires par une autre personne que leur auteur [6]. Or, si l'on considère qu'il a fallu environ 15 ans à une équipe de chercheurs pour construire les tables des verbes simples, on peut raisonnablement supposer que l'exploitation informatique de la totalité de ces données ne pourra se faire que par une approche coopérative.

Le second problème lié au formalisme proposé par Roche vient de ce que les données sont fortement liées aux méthodes utilisées pour les appliquer. Ce point nous semble important pour deux raisons dont la première est la pérennité des données et des programmes. Contrairement à l'accumulation de données linguistiques, l'élaboration de données informatiques sur un long intervalle de temps pose des problèmes de stabilité de représentation. Ces problèmes de maintenance, dus aux évolutions technologiques rapides en informatique, ne peuvent être résolus que par l'adoption de formats standards de données, et par la garantie que les programmes destinés à utiliser ces données pourront s'adapter à ces évolutions, même à long terme. Nous pensons que l'adoption de pratiques de développement coopératif représentées par le phénomène du *logiciel libre*, associée à un choix judicieux des langages de programmation utilisés, permettra de garantir la maintenance à long terme des programmes. La maintenance des données à long terme ne nous paraît possible qu'au prix d'une simplification extrême du formalisme, nécessitant d'alléger encore la représentation proposée par Emmanuel Roche.

La seconde raison qui nous pousse à abandonner la représentation de type DELSYN est la non-réutilisabilité des données. En effet, la dépendance existant entre les données accumulées et les programmes destinés à les appliquer est un obstacle à l'utilisation de ces données dans un autre but que celui prévu par l'auteur, en l'occurrence l'analyse syntaxique des phrases simples. Il nous semble préjudiciable de ne pas pouvoir fournir des données utilisables dans d'autres travaux que les nôtres.

1.2.3 Plan de travail

Notre ambition est de convertir les tables de verbes simples en un ensemble de grammaires locales représentant les différentes structures syntaxiques possibles pour chaque verbe. Depuis l'introduction de ce type de grammaires par le biais du système INTEX ([73]), l'expérience a montré qu'elles étaient parfaitement adaptées à la représentation des phénomènes linguistiques. De plus, ce formalisme permet de dissocier entièrement les données des programmes,

[6] Cela se rapproche fortement des problèmes que connaissent les programmeurs devant reprendre le code de quelqu'un d'autre.

de sorte que les grammaires ne contiennent que des données linguistiques et ce, sous une forme lisible et conviviale. Par ailleurs, ces grammaires présentent l'avantage de pouvoir être directement appliquées sur corpus, ce qui permet de les tester au fur et à mesure de leur construction.

Nous avons donc divisé notre étude en deux parties disjointes : l'une linguistique et l'autre informatique. L'aspect linguistique de cette étude, objet de ce chapitre, traite de la conversion des tables de verbes simples en grammaires locales. Nous y développerons la stratégie que nous avons adoptée pour effectuer cette conversion. Nous détaillerons les adaptations des tables que nous avons réalisées et nous discuterons les questions liées à la maintenance des grammaires construites. Enfin, nous discuterons autour de certains phénomènes linguistiques que nous avons examinés afin de tester la puissance de description de notre modèle, et donc, sa validité.

1.3 Conversion des tables en grammaires

1.3.1 Mécanisme des graphes patrons

Afin de convertir les tables de verbes en grammaires locales, nous avons utilisé le mécanisme de graphes de référence ou *graphes patrons*. Ce principe a été introduit dans [68] et a été mis en pratique avec quelques variantes décrites dans [65], [72] et [75].

Le principe est le suivant : on construit un graphe qui décrit des structures pouvant être vérifiées par les entrées de la table. Ce graphe fait référence aux colonnes de la table grâce à des variables (`@A`=colonne 1, `@B`=colonne 2, etc). On génère ensuite pour chaque ligne de la table une copie de ce graphe dans laquelle les variables sont remplacées en fonction du contenu des cellules situées à l'intersection des colonnes correspondantes et de la ligne traitée. Si une cellule de la table contient le signe `+`, la variable correspondante est remplacée par `<E>`. Si la cellule contient le signe `-`, la boîte contenant la variable correspondante est supprimée, ce qui détruit du même coup les chemins passant par cette boîte. Dans tous les autres cas, la variable est remplacée par le contenu de la cellule.

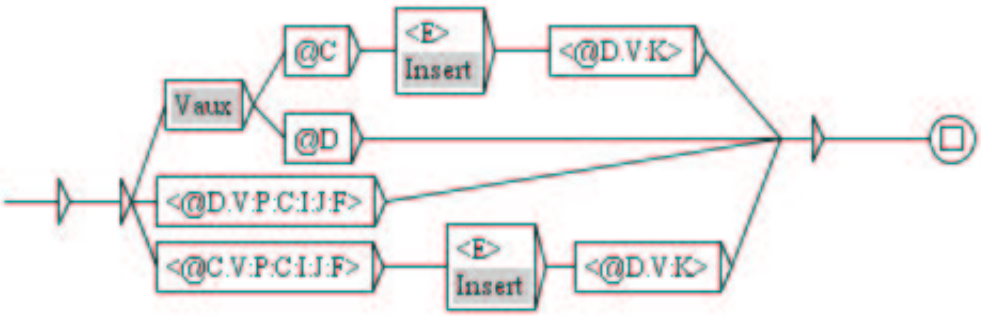

FIG. 1.2 – Un graphe patron simple

La figure 1.2 montre un exemple de graphe patron très simple décrivant un syntagme verbal à l'actif. Les variables `@C` et `@D` font référence à des colonnes de la table 31I, présentée sur la figure 1.5, qui décrivent respectivement l'auxiliaire du verbe (*être* ou *avoir*) et la forme canonique du verbe, c'est-à-dire son infinitif. Si l'on prend *pleuvoir* comme exemple de verbe de la table 31I, `@C` serait donc remplacé par `avoir` et `@D` par `pleuvoir`. Les formes du type `<@D.V:K>` représentent des motifs faisant référence à des informations qui sont contenues dans les dictionnaires, telles que les codes grammaticaux et flexionnels [7]. Le motif `<pleuvoir.V:K>` reconnaîtra ainsi le verbe *pleuvoir* au participe passé (code grammatical `V` pour verbe, code flexionnel `K` pour participe passé), et le motif `<pleuvoir.V:P:C:I:J:F>` reconnaîtra les formes de ce verbe conjuguées au présent (`P`), au conditionnel (`C`), à l'imparfait (`I`), au passé simple (`J`) et au futur (`F`). Les lignes grisées `Vaux` et `Insert` sont des appels à des sous-graphes décrivant respectivement des auxiliaires verbaux (*pouvoir*, *continuer à*, etc) [8] et des insertions (*vraiment*, *lundi matin*, etc) [9]. Le symbole `<E>` représente le mot vide epsilon. Le fait que `<E>` et `Insert` apparaissent dans une même boîte indique que l'on peut reconnaître l'une ou l'autre de ces expressions, ce qui signifie ici que l'appel au sous-graphe `Insert` est facultatif.

L'opération de conversion produit un graphe pour chaque entrée de la table. L'union de tous les graphes générés constitue une grammaire reconnaissant une ou plusieurs structures syntaxiques pour les verbes de la table considérée. La figure 1.3 montre un graphe patron qui décrit différentes structures de phrases mettant en jeu des verbes impersonnels [10]. Il est destiné à être combiné avec la table 31I, présentée sur la figure 1.5.

Ce graphe patron explicite des structures élémentaires de phrases déclaratives à l'actif, correspondant aux propriétés syntaxiques de la table 31I. Par exemple, la propriété *il tombe V-n Loc N*, codée dans la table par la 10^{e} propriété (`@J`), est explicitée par la partie du graphe présentée sur la figure 1.4.

La variable `@J` renvoie à la colonne de cette propriété. De cette manière, le chemin passant par cette variable sera conservé si et seulement si le verbe vérifie la propriété. La figure 1.6 montre le graphe que l'on obtient pour l'entrée *neiger*.

Contrairement aux méthodes de conversion proposées dans [68], [72] et [75], nous avons choisi de ne pas compiler l'ensemble des grammaires obtenues en un unique automate ([63] et [64]), mais de créer un ensemble de sous-graphes. En effet, cette représentation compilée ne permet pas de maintenir des appels à des sous-graphes : chaque appel à un sous-graphe est remplacé par le sous-graphe lui-même et ce, avec une limite de profondeur en cas de récursion. Cette méthode est efficace pour de petites grammaires, mais n'est plus adaptée lorsque leur complexité augmente.

[7] Pour une description complète de la syntaxe des graphes, se reporter au manuel d'Unitex, joint en annexe.
[8] Voir section 1.3.7.1.
[9] Voir section 1.3.7.5.
[10] Ce graphe a été construit manuellement, au tout début de notre étude.

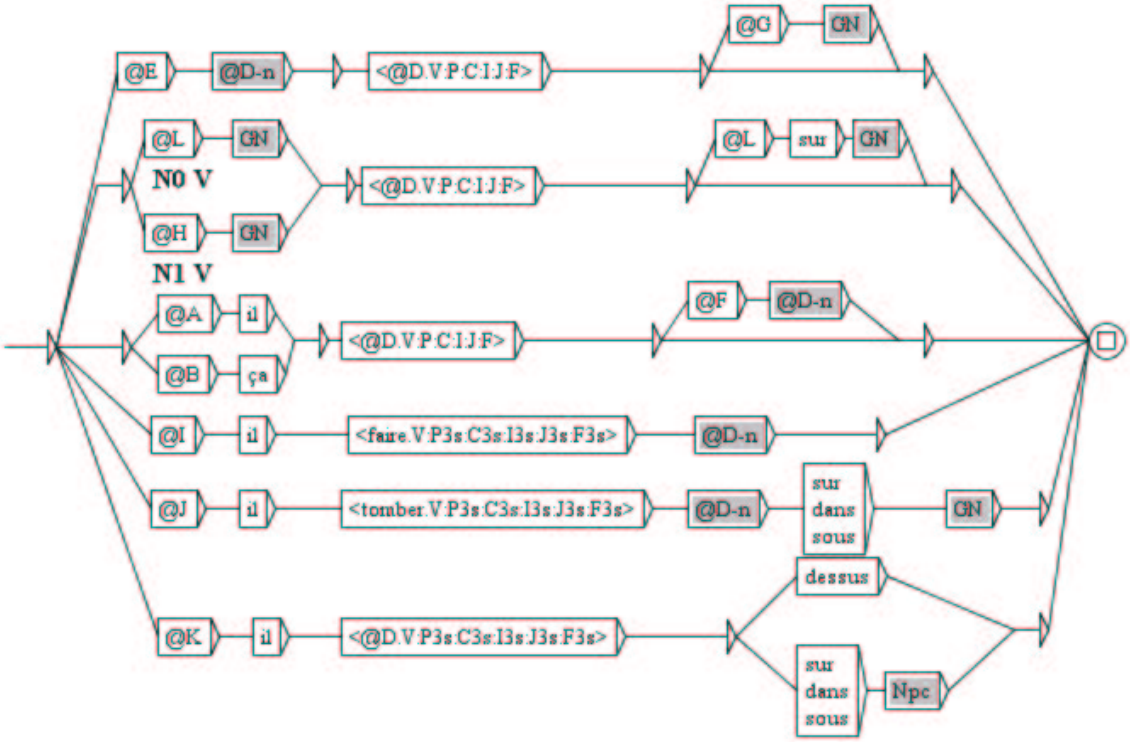

FIG. 1.3 – Exemple de graphe patron

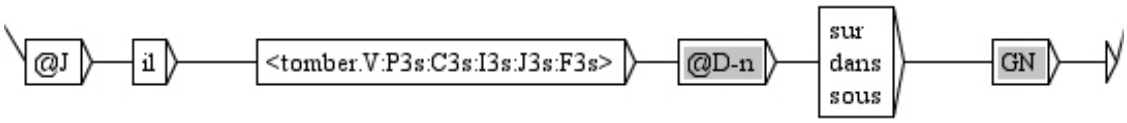

FIG. 1.4 – Structure correspondant à la propriété *il tombe V-n Loc N*

La première raison est que cette méthode peut entraîner une explosion de la taille de l'automate généré. Certaines grammaires comme celle des groupes nominaux ou celle des insertions vont être appelées à maintes reprises. Le fait de remplacer chaque appel par la grammaire elle-même va faire augmenter considérablement la taille de l'automate. Nous avons mesuré que la grammaire des adverbes de temps du français réalisée par Maurice Gross était représentée par un transducteur de 50 méga-octets. Cette expérience montre clairement que cette méthode ne peut fonctionner qu'à petite échelle.

D'autre part, la compilation de tous les graphes en un unique automate est contraignante car elle oblige à disposer de tous les sous-graphes. Il est donc impossible de compiler une grammaire A faisant appel à une grammaire B si B n'est pas définie, ce qui pose problème lors de la construction de grammaires importantes. Considérons par exemple une grammaire décrivant des constructions N_0 V N_1. Si l'on veut pouvoir appliquer cette grammaire, il faut

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	N0 = :il	N0 = :ça	Aux	**31I**	V-n V W	:il V V-n W	V-n V N1	N1 V	:il fait V-n	:il tombe V-n Loc N	:il lui V Loc N1pc	N0 V W	Exemple
2	-	+	avoir	barder	-	-	-	-	-	-	-	-	Ca va§barder§
3	+	+	avoir	brouillasser	+	+	-	-	+	-	-	-	Il§brouillasse§
4	+	+	avoir	bruiner	+	+	-	-	-	+	+	+	Il§bruine§
5	+	+	avoir	brumasser	+	+	-	-	+	-	-	-	Il§brumasse§
6	+	+	avoir	brumer	+	+	-	-	+	-	-	-	Il§brume§
7	+	+	avoir	cailler	-	-	-	-	-	-	-	-	Il§caille§dehors
8	-	+	avoir	chauffer	-	-	-	-	-	-	-	-	Ca va§chauffer§(E+entre eux)
9	-	+	avoir	chier	-	-	-	-	-	-	-	-	Ca§chie§dans le bureau
10	-	+	avoir	coller	-	-	-	-	-	-	-	-	Ca ne§colle§pas entre eux
11	+	+	avoir	crachiner	+	+	-	-	-	+	+	+	Il§crachine§
12	+	+	avoir	déluger	+	+	-	-	-	+	+	+	Il§déluge§
13	-	+	avoir	douiller	-	-	-	-	-	-	-	-	Les impôts,ça§douille§cette année
14	+	+	avoir	floconner	+	+	-	-	-	+	+	+	Il§floconne§une neige fine sur la ville
15	+	+	avoir	flotter	+	+	-	-	-	+	+	-	Il§flotte§sur la campagne
16	+	+	avoir	fraîchir	-	-	+	+	+	-	-	+	(Ca+il)§fraîchit§
17	+	+	avoir	geler	-	-	+	+	+	-	-	-	Il§gèle§
18	+	+	avoir	givrer	-	-	+	+	+	-	-	-	Il a§givré§cette nuit
19	+	+	avoir	grêler	+	+	-	-	-	+	+	+	Il§grêle§(E+de gros grêlons)
20	+	+	avoir	grésiller	+	+	-	-	-	+	+	-	Il§grésille§
21	+	+	avoir	grisailler	+	+	+	+	+	-	-	-	Il§grisaille§aujourd'hui
22	+	+	avoir	lancequiner	+	+	-	-	-	+	+	+	Il§lancequine§sur la plaine
23	+	+	avoir	neiger	+	+	-	-	-	+	+	+	Il§neige§
24	+	+	avoir	pincer	-	-	-	-	-	-	-	-	Ca§pince§ce matin
25	+	+	avoir	pleuvasser	+	+	-	-	-	+	+	+	Il§pleuvasse§
26	+	+	avoir	pleuvoir	+	+	-	-	-	+	+	+	Il§pleut§
27	+	+	avoir	pluvioter	+	+	-	-	-	+	+	+	Il§pluviote§
28	+	+	avoir	souffler	+	+	+	-	-	-	+	+	Il§souffle§un vent du Nord
29	+	+	avoir	tonner	+	+	-	-	+	-	-	+	Il§tonne§

FIG. 1.5 – Table 31I

disposer d'une description des groupes nominaux. Or, la construction d'une telle grammaire étant complexe, il est naturel de se contenter dans un premier temps de faire référence à cette grammaire sans la décrire réellement. De cette façon, on peut éviter de devoir travailler sur les deux grammaires simultanément, car, si l'on génère des graphes pour chaque entrée de la table, il n'est pas nécessaire de posséder tous les sous-graphes au moment de la conversion de la table : la présence de ces sous-graphes ne sera requise qu'au moment d'appliquer la grammaire à un texte.

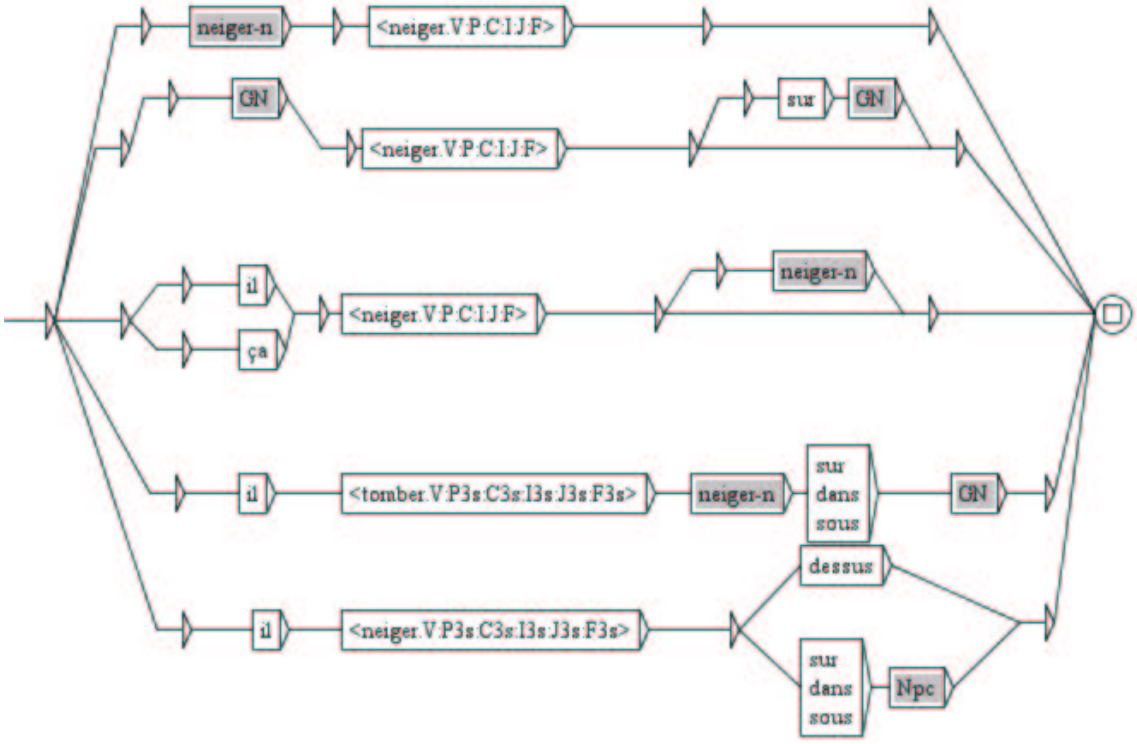

FIG. 1.6 – Grammaire générée pour le verbe *neiger*

Enfin, le fait de générer des graphes au lieu d'un automate compilé permet d'itérer le procédé. De cette manière, on peut construire des graphes patrons qui génèrent d'autres graphes patrons. Nous verrons en 1.3.4 tout le bénéfice que l'on peut retirer de ce principe.

L'intérêt majeur de cette description par tables et graphes patrons est qu'elle peut être raffinée simplement et à volonté. En effet, la description des constructions au moyen de graphes permet d'éditer les grammaires de façon très intuitive. Il est donc facile d'ajouter ou de modifier des constructions dans les grammaires. Lorsqu'un nouveau phénomène nécessite d'être codé, il suffit d'ajouter une nouvelle propriété aux tables concernées et de tenir compte de cette propriété dans les grammaires. Le raffinement de la description ne dépend donc que de la précision des grammaires et de la finesse des propriétés codées dans les tables, et ne requiert aucune complication du formalisme.

1.3.2 Verbes traités

Nous avons choisi de centrer notre étude sur les verbes simples ne comprenant qu'un seul mot. Les constructions figées ont été écartées car leurs propriétés transformationnelles sont mal connues, notamment en ce qui concerne la mobilité des constituants. Par exemple, on peut avoir les infinitives équivalentes :

s'y prendre mal

mal s'y prendre

alors que pour l'expression

s'y prendre comme un manche

on ne peut pas avoir

**comme un manche s'y prendre*

L'absence de codage systématique des propriétés de ces constructions ne nous permet donc pas d'en déduire des grammaires complètes. Nous avons exclu quelques verbes pour lesquels nous ne disposions pas de descriptions suffisantes : des verbes pronominaux tels que *se morfondre* ou *se fâcher*, des verbes se construisant avec *en* (*en découdre*), des verbes composés tels que *avoir beau* ou *présenter (bien+mal)*, ainsi que quelques autres verbes simples constitués de plus d'un mot. En effet, de même que pour les expressions figées, les propriétés spécifiques à ces constructions ne sont pas codées. Nous avons retiré de nos tables les verbes ainsi écartés.

1.3.3 Organisation des constructions

Notre but étant de construire un ensemble de grammaires locales décrivant toutes les phrases simples du français, il apparaît nécessaire d'organiser ces grammaires si l'on veut qu'elles soient utilisables. Comme nous l'avons présenté dans [63], nous avons distingué les formes *autonomes* des formes *non-autonomes*. Les formes autonomes sont celles qui peuvent constituer seules des phrases correctes. Elles comprennent les constructions suivantes :

- déclaratives : *Paul lit un livre*
- interrogatives : *Que lit Paul ?*
- impératives : *Lisez ce livre*
- extractions : *C'est Paul qui lit ce livre*
- extrapositions : *Il sort beaucoup de monde du métro*

Les formes non-autonomes sont toutes les formes qui ne peuvent apparaître qu'à l'intérieur d'autres constructions pour pouvoir former des phrases acceptables. Elles comprennent les constructions :

- infinitives : *avoir lu une revue*
- relatives : *que Paul lira*
- complétives : *que Paul lise ce livre*
- participes présents : *lisant ce livre*
- participes passés : *lu à la petite fille*

Cette répartition en différents types de construction nous semble très naturelle car les grammaires que l'on construira pour ces constructions pourront être facilement réutilisées. Ainsi, des schémas de phrases tels que N_0 *V Vinf* ou N_0 *V Que P* pourront être facilement décrits si l'on dispose des grammaires *Vinf* et *Que P*. De même, les grammaires des relatives ainsi que celles des participes pourront être intégrées aisément dans les descriptions des groupes nominaux.

La plupart de ces constructions peuvent être soumises à des variations telles que la passivation, la négation et l'ajout d'auxiliaires verbaux. Ces transformations peuvent se combiner commme dans l'exemple suivant :

Ce ne peut être ce livre qui a été lu par Paul

Naturellement, ces transformations ne sont pas toujours possibles. Ainsi, certains verbes transitifs comme *mesurer* n'admettent pas toujours le passif :

La feuille mesure 30 centimètres
**30 centimètres sont mesurés par la feuille*

Il y a également des restrictions sémantiques sur certains verbes qui interdisent la négation :

Ce film vaut bien un oscar
**Ce film ne vaut pas bien un oscar*

Paul s'est laissé dire que Léa sortait avec Max
**Paul ne s'est pas laissé dire que Léa sortait avec Max*

Le phénomène inverse existe également avec certains verbes à négation obligatoire :

Paul ne peut pas sentir Léa
**Paul peut sentir Léa*

Cette réunion n'en finit pas
**Cette réunion en finit*

Cependant, ces restrictions concernent la plupart du temps des verbes constitués de plus d'un mot, constructions que nous avons écartées de notre étude. Nous avons donc décidé de ne pas les prendre en compte. Toutefois, elles pourraient tout à fait être traitées pour peu qu'on les code dans les tables.

Nous avons choisi de classer les constructions par forme de base (N_0 V, N_0 V N_1, etc) et par type (déclarative, infinitive, etc). Par convention, nous nommerons les grammaires de la façon suivante :

forme de base(type de construction)

Par exemple, *N0V(Vinf)* désignera la grammaire des infinitives basées sur les constructions N_0 V.

Nous avons intégré les extractions dans les phrases déclaratives, car toutes ces formes peuvent être utilisées comme phrase dans des constructions *Que P*. La grammaire *N0VdeN1(P)* reconnaîtra ainsi les phrases :

beaucoup de monde sort du métro

c'est du métro que sort beaucoup de monde

Ces constructions sont déclinées à l'indicatif et au subjonctif, afin d'obtenir les grammaires *P* et *Psubj*. Lorsqu'elle est possible, la construction passive fait l'objet de grammaires distinctes. On a ainsi des grammaires pour les phrases déclaratives passives, les infinitives passives, etc. Voici des exemples de phrases reconnues par la grammaire *N0VN1(Vinf_passif)* :

être lu par Paul
avoir été aimé de Marie

Les transformations par négation, insertion d'auxiliaires verbaux et insertion d'adverbes ont été autorisées pour toutes les constructions, et sont codées directement dans les grammaires correspondantes. Par exemple, la grammaire *N0V(P)* peut reconnaître des phrases comme :

Paul ne peut pas réfléchir
les fleurs commenceront bientôt à bourgeonner

Les autres types de transformations tels que la pronominalisation ou l'inversion des compléments sont représentés par des chemins parallèles dans les grammaires. Ainsi, la grammaire *N0VN1(Interro)* reconnaîtra les phrases :

qui a mangé ce gâteau ?
qui pourrait l'avoir mangé ?

1.3.4 La super-table

Lors de nos premières expériences, nous avons construit des grammaires particulières pour chaque table, comme par exemple la figure 1.3 de la page 30. Cependant, il s'est rapidement avéré que cette approche présentait des défauts. Beaucoup de structures étant communes à plusieurs tables, il fallait transposer ces structures pour chaque table : en effet, une même propriété n'est pas codée dans la même colonne pour toutes les tables. Cette redondance entraînait d'importants problèmes de maintenance, car la moindre modification devait être répercutée sur toutes les tables concernées, ce qui multipliait les risques d'erreurs en même temps que la charge de travail.

Par ailleurs, cette transposition des structures devait également tenir compte des propriétés qui sont constantes dans certaines tables. Ainsi, le fait d'admettre un humain comme sujet (propriété $N_0 = N_{hum}$) étant définitionnel de la table 31H, cette propriété n'est pas codée dans cette table puisque cela correspondrait à une colonne ne contenant que des plus. À moins d'harmoniser les numéros de colonnes, ce qui nous aurait obligé à rétablir les colonnes constantes, il n'était donc pas possible d'éviter de transposer les structures manuellement pour chaque table.

Nous avons résolu ce problème de maintenance en ajoutant un niveau d'abstraction à notre modèle : les graphes patrons génériques. L'idée est de générer les graphes patrons propres à chaque table à partir d'une seule description générique. Chaque graphe patron ainsi généré

peut alors être utilisé pour générer les grammaires propres à chaque entrée de table. Ce modèle comporte donc trois niveaux de graphes :

1. *graphes génériques* : graphes patrons décrivant des structures de façon générique ;
2. *graphes de tables* : graphes patrons spécialisés pour une table donnée ;
3. *graphe d'entrées* : graphes lexicaux engendrés pour des entrées particulières.

Pour obtenir une architecture générique, nous utilisons une super-table qui va permettre de générer les graphes de tables à partir de graphes génériques. Ainsi, il n'y a plus besoin de décrire chaque structure qu'une seule fois, dans un graphe générique qui sera ensuite décliné pour toutes les tables. Pour que les graphes de tables correspondant à une propriété P ne soient générés que pour les tables qui admettent cette propriété, il suffit de placer une référence à celle-ci comme première boîte du graphe générique. Cette transition jouera le rôle d'une condition : si une table ne vérifie pas la propriété, alors la boîte sera supprimée et donc, le graphe de table ne sera pas généré, car il ne pourrait rien reconnaître.

La génération des graphes d'entrées peut être résumée par l'algorithme suivant :

```
      Générer tous les graphes d'entrées :
0     Pour chaque graphe générique
1           Générer les graphes de tables
2           Pour chaque table
3                 Générer les graphes d'entrées
4           Fin Pour
5     Fin Pour
6     Fin.
```

Dans la pratique, on construit une super-table qui prend en abscisse les propriétés et en ordonnées les tables de lexique-grammaire. L'interprétation des cases de cette table est la suivante :

- si la table T vérifie toujours la propriété P, alors $table[P,T]$ contient le signe $+$;
- si la table T ne vérifie jamais la propriété P, alors $table[P,T]$ contient le signe $-$;
- si dans la table T, la propriété P est codée dans la colonne C, alors $table[P,T]$ contient $@C$.

La figure 1.7 montre un extrait de la super-table sur lequel on peut voir le codage des verbes avec leur auxiliaire ainsi que la définition des propriétés de N_0. Les points d'exclamation que l'on trouve dans presque toutes les cases de la colonne C correspondent à des négations. En effet, dans presque toutes les tables, les verbes n'admettent qu'un seul auxiliaire. Dans ce cas, c'est la propriété $Aux = avoir$ qui est codée, de sorte que la propriété $Aux = être$ est la négation de la première. La table 2 possède elle deux colonnes distinctes, car certains verbes comme *accourir* ou *disparaître* peuvent prendre les deux auxiliaires.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Table	Aux = avoir	Aux = être	V	N0 = Nhum	N0 = N-hum	N0 = Nnr	N0 = le fait que p	N0 = que P subj	N0 = V1 W	N0 = V2 W	N0 = Vinf W	N0 = V-n	V-n (N0)	N0 = il	N0 = ça	N0 = Npc	N0 = Npl obl
2	2-@%	@E	@F	@D	@A	-	@B	@C	-	-	-	-	-	-	-	-	-	-
3	3-@%	@D	!@D	@E	@A	-	@B	@C	-	-	-	-	-	-	-	-	-	-
4	4-@%	@E	!@E	@F	@A	-	@B	@C	@D	-	-	-	-	-	-	-	-	-
5	6-@%	@E	!@E	@F	@A	-	@B	@C	-	@D	-	-	-	-	-	-	-	-
6	7-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	-	-	-	-	-	-	-	-
7	8-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	-	-	-	-	-	-	-	-
8	9-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	@E	-	-	-	-	-	-	-
9	11-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	@E	-	-	-	-	-	-	-
10	12-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	@E	-	-	-	-	-	-	-
11	13-@%	@F	!@F	@G	@A	-	@B	@C	-	@D	@E	-	-	-	-	-	-	-
12	31H-@%	@C	!@C	@D	+	@A	-	-	-	-	-	-	@B	@N	-	-	-	-
13	31I-@%	@C	!@C	@D	-	-	-	-	-	-	-	-	-	-	@A	@B	-	-
14	31R-@%	@E	!@E	@F	@A	-	@B	-	-	-	-	@C	@D	@K	-	-	-	-

FIG. 1.7 – Extrait de la super-table

Le symbole *@%* que l'on rencontre dans la première colonne de la super-table est un symbole particulier qui est remplacé lors de la génération de graphes par le numéro de la ligne courante. Considérons par exemple le graphe patron de la figure 1.9.

Si on l'applique à une table où les verbes sont codés dans la colonne D, et que le verbe *baisser* apparaît dans cette table à la 12^{e} ligne, on obtiendra le graphe de la figure 1.10.

Nous avons introduit ce nouvel opérateur dans le mécanisme des graphes patrons pour pouvoir faire référence de façon non ambiguë à chaque entrée de chaque table : puisque ce symbole prend une valeur différente pour chaque entrée de la table, nous pouvons l'utiliser pour caractériser de façon précise chaque entrée dans une table. En effet, il peut arriver que plusieurs emplois d'un même verbe figurent dans une même table, comme c'est le cas dans la table 35R où l'on trouve les 4 emplois suivants pour le verbe *tirer* :

l'escrimeur a tiré contre son adversaire
cette couleur tire sur le bleu
Max tire sur la (sonnette + corde)
Max tire sur sa (cigarette + pipe)

En associant ce numéro de ligne au nom de la table, on obtient une référence unique pour chaque entrée. C'est la raison pour laquelle la première colonne de la super-table contient des séquences du type *31H-@%*. Nous verrons plus loin que la garantie d'unicité ainsi obtenue permet de simplifier la construction des grammaires.

La figure 1.11 montre en exemple le graphe générique décrivant les infinitives dérivées de la construction N_0 V. On peut voir que la première boîte de ce graphe contient @AS, ce qui est une référence à la colonne codant la propriété N_0 V dans la super-table.

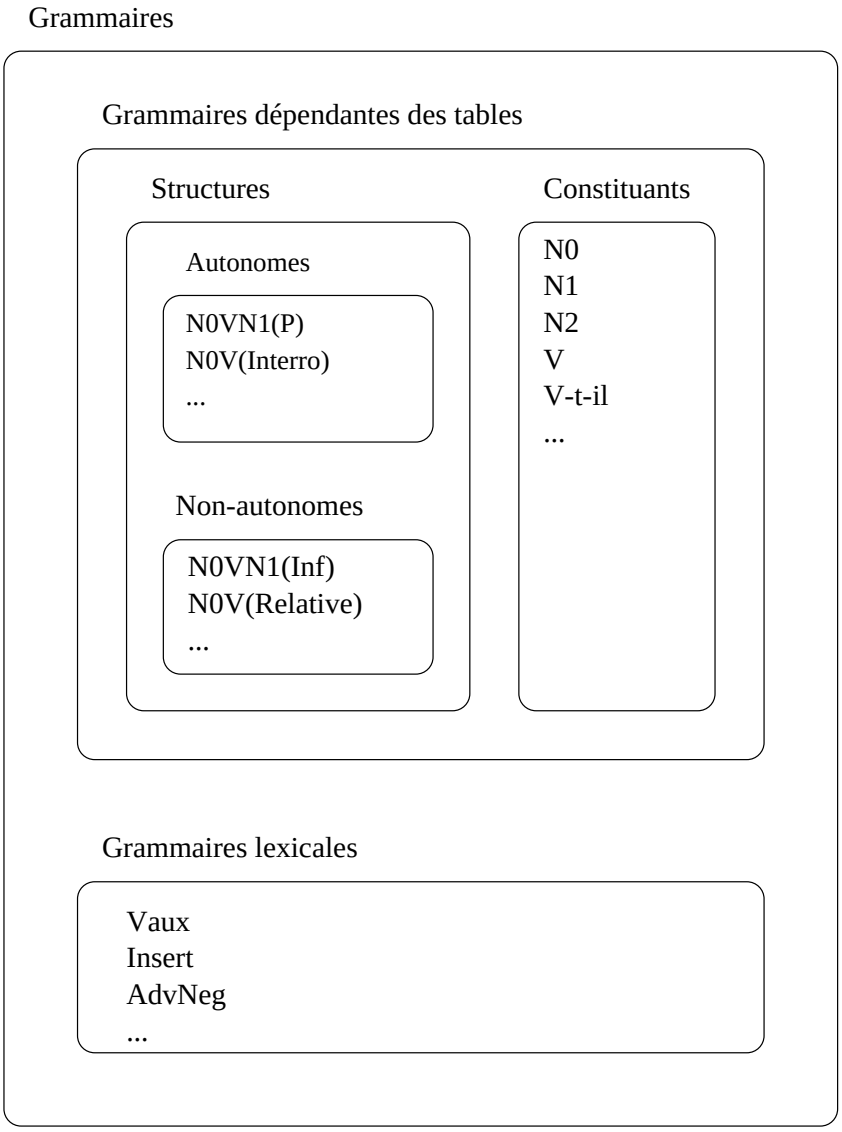

FIG. 1.8 – Organisation des différentes grammaires

FIG. 1.9 – Un graphe patron élémentaire

1.3.5 Adaptation des tables

Dès la construction de nos premières grammaires, il nous est apparu qu'il fallait adapter les tables de lexique-grammaire si l'on voulait les exploiter informatiquement, comme nous

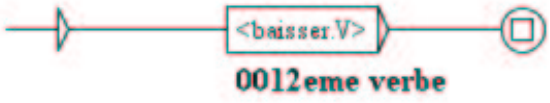

FIG. 1.10 – Graphe obtenu à partir de celui de la figure 1.9

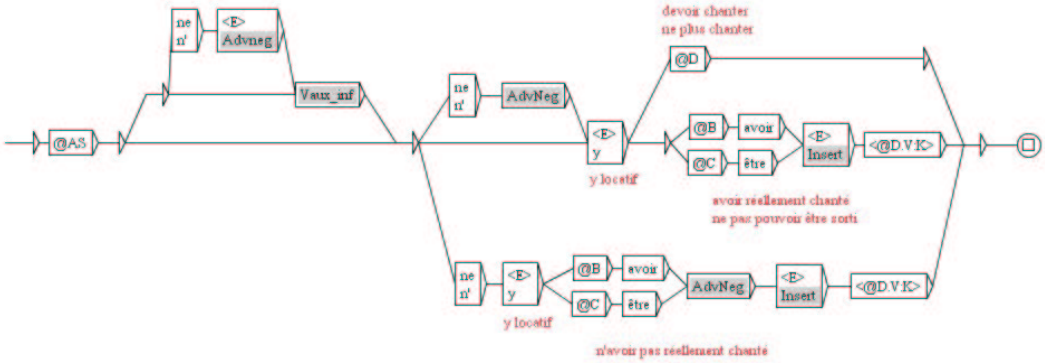

FIG. 1.11 – Graphe générique *super-N0V(Vinf)*

l'avons discuté dans [63]. En effet, les tables ont été constituées dans un but d'étude et de classification des phrases. Pour y parvenir, les auteurs des tables ont pris en compte des propriétés telles que le nombre de compléments, leur pronominalisation, le fait d'admettre une complétive comme sujet, etc (voir [43] et [53] pour une description des propriétés utilisées). Les propriétés qui ont été retenues soulèvent plusieurs problèmes lorsque l'on veut utiliser ces données pour faire de l'analyse automatique. Tout d'abord, certaines propriétés ne peuvent pas être exploitées. Il s'agit en général de propriétés sémantiques. Par exemple, la propriété N_2 *apparition*, utilisée dans la table 37M6, traduit le fait que le complément N_2 apparaît pendant le procès. Ainsi, dans les phrases

Léa brode sa robe d'un motif
Max macule le mur de taches de vin

le motif et les taches de vin n'existent pas avant que les actions ne se soient déroulées. De telles propriétés pourraient peut-être être utilisées pour décrire des constructions correctes, mais dans l'état actuel de nos connaissances, nous ne pouvons pas les exploiter. Un autre problème, plus gênant, vient de ce que beaucoup d'éléments nécessaires à l'exploitation informatique des tables n'ont pas été codés. Considérons le cas des auxiliaires utilisés pour conjuguer les verbes aux temps composés. Le codage de ces auxiliaires présente peu d'intérêt linguistique lorsqu'il s'agit de classer des phrases, mais il est indispensable si l'on veut analyser correctement les phrases aux temps composés. Nous avons donc ajouté aux tables le codage des auxiliaires *être* et *avoir*.

De plus, les phrases ayant servi de tests lors de l'élaboration des tables sont pour la plupart

des phrases basiques, essentiellement des phrases déclaratives au présent ou au passé composé. Certaines constructions comme par exemple la formation de l'impératif n'ont donc pas été examinées, et rien ne permet de penser que ces constructions ne sont pas soumises elles aussi à des restrictions qu'il faudrait coder. Les possibilités de combinaison des transformations n'ont pas non plus été étudiées, et il est problable que l'on découvre certains blocages nécessitant de raffiner la description.

Une autre source de problèmes est que certaines propriétés n'ont pas la même interprétation dans toutes les tables. Par exemple, la propriété N_0 *V* N_1 *de* N_2 n'a pas le même sens dans les phrases suivantes :

Max accuse Léa de ce crime
Max enduit le mur de ce crépi
Max extrait un clou de cette porte

Même si la structure semble être la même dans les trois constructions, elles n'ont pas les mêmes propriétés syntaxiques. Par exemple, il n'y a que dans la seconde phrase que *de* peut commuter avec la préposition *avec* :

**Max accuse Léa avec ce crime*
Max enduit le mur avec ce crépi
**Max extrait un clou avec cette porte*

Il est donc nécessaire de distinguer toutes les propriétés qui ont des sens différents selon les tables où elles apparaissent si l'on veut pouvoir analyser correctement les constructions correspondantes.

Nous avons également ajouté deux types de colonnes en prévision d'une utilisation ultérieure. Nous avons ainsi codé les *V-n* lorsqu'ils existaient (voir section 1.3.6.2), et nous avons mis en regard de chaque verbe une traduction en anglais. Nous avons pu ainsi distinguer les différents emplois d'un même verbe par leurs traductions dans la plupart des cas. Ainsi, chacun des trois emplois du verbe *tenir* dans la table 32H est associé à une construction différente en anglais :

Max tient Luc (ε+en son pouvoir) → *to have under control*
Max tient Luc (ε+dans les sondages) → *to equal*
Max tient un rhume → *to have*

La dernière catégorie de modifications que nous avons effectuées sur les tables est liée au format de celles-ci. Ces données ayant été accumulées sur un long laps de temps, les évolutions technologiques successives ont entraîné de nombreuses conversions : passages des tables écrites sur papier aux cartes perforées, puis aux bandes magnétiques, puis aux disquettes. À cela s'est ajouté le fait que les données, une fois informatisées, ont subi des changements de format au fur et à mesure que les outils utilisés pour les éditer évoluaient. Ces conversions successives ont entraîné de nombreuses erreurs de codage dont certaines, telles que l'inversion de deux colonnes, ont pu fausser des tables entières. À l'heure actuelle, il existe plusieurs standards

permettant de représenter ces données, indépendamment des logiciels utilisés. Cela nous assure que les données des tables peuvent désormais être relativement stables dans le temps. Nous pouvons donc considérer que la correction des tables dans leur version actuelle est une étape qui peut être effectuée une fois pour toutes. Nous avons ainsi corrigé un certain nombre d'erreurs, mais nous pensons qu'il sera nécessaire dans l'avenir de réviser intégralement les codages des tables.

1.3.6 Arguments du verbe

1.3.6.1 Graphes génériques des arguments

Un grand nombre de propriétés mettent en jeu les arguments des verbes. Afin de simplifier la description des constructions faisant appel à ces arguments, nous avons décidé de construire pour chaque verbe la grammaire de chacun de ses arguments. Cette tâche est rendue très simple par l'emploi de la super-table, car il nous a suffi d'écrire 3 graphes génériques pour rendre compte des structures N_0, N_1 et N_2. La figure 1.12 montre le graphe générique décrivant la structure N_0.

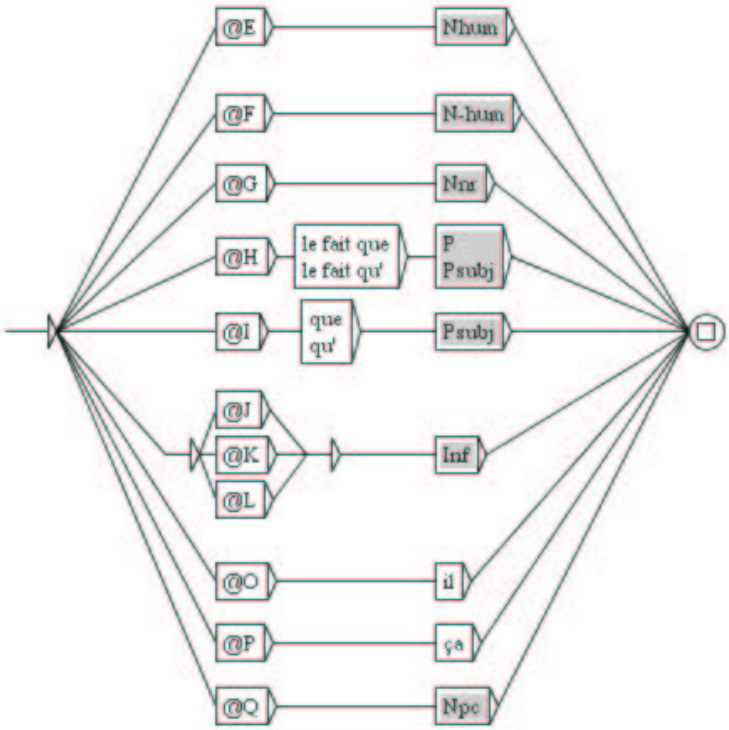

FIG. 1.12 – Graphe générique *super-N0*

La construction de l'infinitive en position N_0 est codée par trois propriétés différentes :

– $N_0 = V^1\ W$: indique que l'infinitive a pour sujet N_1 ;

- $N_0 = V^2\ W$: indique que l'infinitive a pour sujet N_2 ;
- $N_0 = V\ W$: indique la possibilité d'avoir une infinitive sans préciser son sujet.

Il suffit donc qu'une de ces propriétés soit vérifiée pour que l'on puisse avoir une infinitive en position N_0. Cela se traduit par la mise en parallèle de ces 3 propriétés dans le graphe générique (@J, @K, @L), ce qui est équivalent à un OU logique de ces 3 propriétés. Le ET logique pourrait être obtenu en faisant se suivre les 3 boîtes au lieu de les mettre en parallèle.

Nous avons choisi d'utiliser ces descriptions compactes des arguments car, bien souvent, l'ensemble des constructions qui sont possibles pour un N_i le sont quelle que soit la position de ce N_i dans la construction. Considérons par exemple certains verbes de la table 37E qui admettent la construction $N_1\ V$. Ces verbes peuvent vérifier les propriétés N_1 = *Nhum* et N_1 = *N-hum*. Lorsqu'un verbe vérifie $N_1\ V$, les N_1 valides dans les constructions $N_0\ V\ N_1$ le sont aussi dans les constructions $N_1\ V$. Ainsi le verbe *saigner* qui admet aussi bien les humains que les non-humains en position N_1 :

Max saigne le patient
Max saigne le cochon

autorise les deux constructions :

le patient saigne
le cochon saigne

Toutefois, il existe des cas où toutes les possibilités ne sont pas admises quel que soit le rôle de l'argument. Par exemple, pour certains verbes le passif peut s'exprimer avec *de* ou *par* en fonction de la nature de l'argument N_0. Ainsi, pour les phrases

l'attitude de Léa déçoit Luc
Max déçoit Luc

on obtient les constructions passives suivantes :

Luc est déçu (de+par) l'attitude de Léa
*Luc est déçu (*de+par) Max*

Dans ces cas-là, il est nécessaire de distinguer les différentes constructions possibles pour les N_i considérés. Cela ne pose aucun problème car il est tout à fait possible de ne faire appel qu'aux propriétés mises en jeu. Ainsi, la construction passive en N_1 *être Vpp de* N_0 pourra être conditionnée pour certains verbes par la propriété N_0 = *N-hum*.

Pour résumer, nous avons choisi de décrire des N_i génériques pour simplifier l'écriture de nos grammaires en partant de la constatation que la position des arguments leur impose peu de restrictions. Étant donné qu'il est possible d'affiner la description des arguments lorsque cela est nécessaire, notre choix d'une description globale des N_i est une simplification mais n'affecte pas la qualité des grammaires.

Dans le même ordre d'idées, on observe des contraintes sur les pronoms personnels. Ainsi, pour le verbe symétrique *épouser* qui tolère les constructions $N_0\ V\ N_1$ et $N_1\ V\ N_0$, on peut avoir la phrase

j'ai épousé Marie

mais pas

**Marie a épousé je*

Afin de ne pas alourdir les grammaires, nous n'avons provisoirement pas tenu compte de ces restrictions. Nous pensons en effet qu'il suffira d'inclure les différents pronoms dans les grammaires des groupes nominaux concernés, car les formes incorrectes qui pourraient être reconnues n'introduiront que rarement des ambiguïtés. Ce point est intéressant car il met en lumière l'opposition qu'il peut y avoir entre les grammaires d'analyse et de génération. La problématique est du même ordre que celles du type : faut-il interdire les expressions comme *le 30 février* ? Étant donné que notre cadre est celui de l'analyse, nous pensons que nous pouvons nous permettre ces approximations. En effet, nos grammaires sont destinées à être appliquées sur des textes contenant des formes supposées correctes et, dans de tels corpus, des phrases telles que

**Marie a épousé je*

ne devraient pas apparaître. Il est toutefois nécessaire de vérifier que les approximations que l'on souhaite faire n'introduiront pas d'ambiguïté ni de bruit. Par exemple, une grammaire qui décrirait comme approximation d'une date horaire un nombre suivi du mot *heures* reconnaîtrait de façon incorrecte la séquence *140 heures* dans la phrase

ce professeur en est déjà à 140 heures cette année

Nous voyons donc qu'une trop grande permissivité peut introduire des erreurs. La limite entre approximation correcte et sur-reconnaissance n'est pas toujours évidente à déterminer, mais il est nécessaire de tenter de s'en rapprocher si l'on veut obtenir des grammaires de bonne qualité.

1.3.6.2 Propriétés N_i = *V-n*

Certains verbes vérifient des propriétés du type N_i = *V-n*. Cela signifie qu'un substantif morphologiquement lié au verbe peut apparaître comme argument. Ainsi, la phrase

la colle colle

nous montre qu'au moins un des emplois du verbe *coller* vérifie la propriété N_0 = *V-n*. Les *V-n* peuvent également apparaître en position N_1 :

les deuxième année bizutent les bizuts

ainsi qu'en position N_2 :

l'administration a alloué 1000 € à cet allocataire

Il arrive également que des verbes admettent plusieurs *V-n*, comme c'est le cas dans les phrases :

les joueurs jouent à ce jeu
le loueur loue une location à un locataire

Nous avons ajouté dans les tables des colonnes qui explicitent les *V-n* lorsqu'ils existent. La figure 1.13 montre quelques-uns des *V-n* que nous avons codés dans la table 36DT. On peut noter que l'acceptation des propriétés N_i = *V-n* a parfois été poussée, de sorte que certains *V-n* ne sont possibles que dans des interprétations très particulières. Ainsi, le verbe *administrer* de la table 36DT admet *ministre* comme *V-n* en position N_0 à cause de phrases comme :

le ministre du culte lui a administré l'extrême onction

<OPT>V-n (N0)	<OPT>V-n (N1)	<OPT>V-n (N2)	<OPT>Exemple
-	-	-	Max§aboule§du fric à Ida
-	-	-	Max a§accepté§un lit de Marie
-	-	-	Paul§accorde§un prêt à Luc
-	achat	-	Max a§acheté§un lit à Luc
-	acquisition	-	Paul a§acquis§ce tableau au marchand
-	-	-	Jean a§acquitté§cette dépense à Max
-	adjoint	-	Luc a§adjoint§un collaborateur à Eva
juge	-	-	Max a§adjugé§le prix à Paula
ministre	-	-	Max a§administré§une(drogue+claque)à Marie
-	-	adresse	Paul a§adressé§un paquet à Jacques
-	fermage	fermier	Le maître a§affermé§ce pré à ce paysan
fermier	fermage	-	Ce fermier a§affermé§ce pré au maître

FIG. 1.13 – Quelques *V-n* de la table 36DT

Une première application de ces propriétés pourrait être de décrire les transformations liant des phrases telles que

Paul a acquis ce tableau
ce tableau est une acquisition de Paul

Toutefois, plusieurs problèmes s'opposent à une exploitation immédiate de ces propriétés. Tout d'abord, ces données sont incomplètes car les *V-n* en *eur* ont été écartés à cause de leur grande productivité. On peut ainsi voir sur la figure 1.13 que les mots *acheteur* et *acquéreur* n'ont pas été pris en compte, ce qui nous empêche de rendre compte de relations comme celle liant les phrases :

Paul a acquis ce tableau
Paul est l'acquéreur de ce tableau

Pour résoudre ce problème, il faudrait donc compléter les tables. On peut noter que le même problème se pose pour les *V-a*, sans lesquels il n'est pas possible d'expliciter des relations du type

Léa rit du costume du Max
le costume de Max est risible

Une autre difficulté vient de ce que les propriétés N_i = *V-n* se prêtent mal à la production de structures syntaxiques. En effet, l'utilité de ces propriétés est plutôt de mettre en relation des phrases telles que :

Max troue le mur $\Leftrightarrow$ *Max fait un trou dans le mur*

Une seconde application de ces propriétés pourrait donc être de représenter la distribution des arguments des verbes, grâce à des phrases classificatrices du type :

un N est une tartine

Cependant, de nouveaux problèmes se posent. Ainsi que nous l'avons discuté dans [63], les *V-n* désignent parfois non pas un mot mais une classe de mots. Ainsi, la propriété N_2 = *V-n* pour le verbe *beurrer* ne désigne pas seulement le substantif *beurre*, mais également d'autres substantifs désignant des substances avec lesquelles on peut beurrer :

N_0 *beurrer une tartine de (beurre+margarine+saindoux)*

Dans cet exemple, la classe est assez restreinte pour des raisons sémantiques qui rendent très douteuses des phrases telles que :

*?N_0 *beurrer une tartine de (confiture+miel)*

Ce n'est cependant pas toujours le cas, comme le montre l'exemple du verbe *tartiner* qui admet une classe de substantifs très ouverte pour la propriété N_1 = *V-n*. Ainsi, le nom *tartine* est un nom classifieur auquel peuvent se substituer un grand nombre de substantifs :

N_0 *tartiner (une tartine+une biscotte+une tranche de pain+un quignon+un quart de baguette+...)*

Pour de telles classes de noms, il devient difficile d'établir des listes précises. Les classes d'objets développées par l'équipe de Gaston Gross (voir [32]) pourraient sans doute aider à résoudre ce problème, mais ce sera un très long travail de construire toutes les classes nécessaires, et ce, avec une précision suffisante. En effet, si la description n'est pas assez fine, nous pourrons avoir des problèmes d'ambiguïtés structurelles :

N_0 tartiner un quignon de confiture
N_0 tartiner une tranche de pain

Ces deux phrases sont théoriquement ambiguës entre les constructions N_0 V N_1 *de* N_2 et N_0 V N_1 avec un N_1 complexe. Cependant, aucune d'elles n'est ambiguë pour un humain car *un quignon de confiture* ne constitue pas un N_1 acceptable pour le verbe *tartiner* [11], et *pain* n'est pas un N_2 valide pour ce verbe. Ce problème d'ambiguïté apparaît dans de nombreuses constructions et déborde du cadre des seuls *V-n*. Le résoudre nécessiterait une description très fine de toutes les classes de noms possibles, ce qui représente un travail considérable. De plus, au regard d'exemples tels que

Léa a tartiné une tranche de pain de mie de 3 cm de confiture

il n'est pas sûr que cela soit suffisant. Dans l'exemple précédent, outre les ambiguïtés artificielles que représentent les interprétations :

**Léa a tartiné (une tranche de pain de mie de 3 cm de confiture)$_{N_1}$*
**Léa a tartiné (une tranche)$_{N_1}$ de (pain de mie de 3 cm de confiture)$_{N_2}$*
**Léa a tartiné (une tranche de pain)$_{N_1}$ de (mie de 3 cm de confiture)$_{N_2}$*

on a une ambiguïté naturelle entre les phrases

Léa a tartiné (une tranche de pain de mie)$_{N_1}$ de (3 cm de confiture)$_{N_2}$
Léa a tartiné (une tranche de pain de mie de 3 cm)$_{N_1}$ de (confiture)$_{N_2}$

La construction de telles classes pourra donc lever certaines ambiguïtés, mais il sera nécessaire d'apporter des solutions complémentaires si l'on veut résoudre les ambiguïtés dans le cas général. En outre, il est probable que de nombreuses classes telles que "argument N_1 du verbe *tartiner*" ne pourraient pas être réutilisées. Enfin, même si chaque classe pouvait être décrite avec la précision souhaitée, un nouveau problème surgirait du fait que les classes sont en nombre potentiellement infini. En effet, il est possible de créer de nouvelles classes en combinant plusieurs restrictions comme le montre l'exemple suivant :

Paul recouvre la table de bois

où l'on a une ambiguïté entre les deux interprétations

Paul recouvre (la table de bois)$_{N_1}$

[11] Ce n'est même pas un groupe nominal correct.

Paul recouvre (la table)$_{N_1}$ de (bois)$_{N_2}$

L'ambiguïté naît du fait que *bois* peut être interprété comme un matériau susceptible d'entrer dans la constitution d'une table. Cette ambiguïté disparaît dans la phrase

Paul recouvre la table de trombones

mais on la retrouve dans la phrase

Paul recouvre la chaîne de trombones

On voit donc que les restrictions imposées par le verbe et celles imposées par le N_1 peuvent se combiner et lever ou non l'ambiguïté de la phrase N_0 *recouvrir* N_a *de* N_b, selon que le nom N_b appartient ou non à l'ensemble des matériaux pouvant constituer le N_1 ainsi qu'à l'ensemble des N_2 valides pour le verbe *recouvrir*. On peut d'ailleurs noter que ce type d'ambiguïté n'est pas exclusif de la relation *N de* $N_{materiau}$ et qu'il peut mettre en jeu d'autres types de constructions *N de N* :

Paul recouvre le pot de trombones
Léa sort la clé de la mallette

Nous voyons donc qu'il est possible d'engendrer de nouvelles classes en combinant des restrictions et qu'en conséquence, les classes possibles ne sont peut-être pas énumérables.

En conclusion, nous avons constaté que la reconnaissance des *V-n* est un problème compliqué qui soulève de nombreuses interrogations pour l'instant sans réponse. Nous avons codé des informations concernant les *V-n* dans les tables de verbes et dans la super-table, mais ces données ne sont ni complètes, ni exploitées pour l'instant dans nos grammaires.

1.3.6.3 Grammaire de test

Comme nous avons pu le voir à la section 1.3.6.1, nous utilisons plusieurs sous-graphes de groupes nominaux : `Nhum`, `N-hum`, `Nnr`, etc. Nous n'avons pas défini le contenu de ces sous-graphes, car cela aurait nécessité de résoudre préalablement le problème de la description des groupes nominaux, ce qui sort du cadre de notre étude. Nos graphes *super-N0*, *super-N1* et *super-N2* sont donc des structures qui décrivent, entre autres, les types de groupes nominaux acceptés par chaque verbe. Pour utiliser concrètement nos grammaires, il faudra donc ajouter les graphes manquants une fois que ceux-ci seront disponibles. Toutefois, afin de pouvoir tester nos grammaires, nous avons construit une description grossière des groupes nominaux et nous l'avons utilisée à la place de chacun des sous-graphes manquants. Nous présenterons dans cette section les différents graphes qui composent cette grammaire.

Le graphe *GN* est le graphe principal de notre grammaire. Il décrit des séquences d'un ou plusieurs groupes nominaux simples. Nous avons inclus dans ce graphe les pronoms personnels *je*, *tu*, *on* et *l'on*, car contrairement aux autres pronoms sujets, ceux-ci ne peuvent se combiner avec d'autres éléments pour former des groupes nominaux complexes :

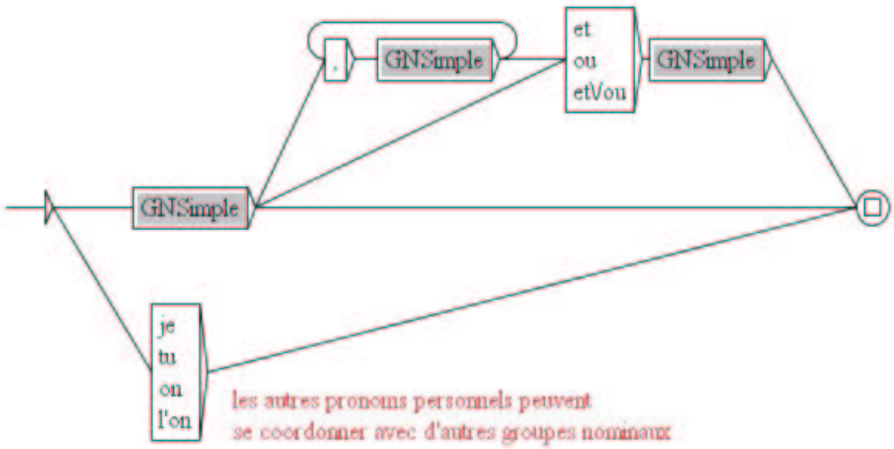

FIG. 1.14 – Graphe GN

**je et cet homme avons longuement discuté*
**c'est on ou Max qui a pris la clé*

Le graphe de la figure 1.15 décrit plusieurs sortes de groupes nominaux, dont plusieurs formes pronominales telles que *celui-ci*, *toi* ou *nous-mêmes*. Nous avons également codé les expressions composées de *de* suivi d'un adjectif et d'un nom pluriels comme *de braves garçons*. La troisième partie de cette grammaire reconnaît des approximations grossières de groupes nominaux, ce qui fait que le bruit généré par cette grammaire est important.

Notre graphe *Det*, présenté sur la figure 1.16, décrit quelques déterminants parmi les plus courants. On y trouve les articles définis et indéfinis, les pronoms possessifs, les déterminants partitifs, etc. Nous avons également intégré dans notre grammaire diverses constructions afin de reconnaître des séquences telles que : *un certain N*, *tous les autres N*, *mon ou mes N*, *les quelques N*, etc. Ces ajouts ont été essentiellement motivés par des silences lors de l'application de notre grammaire. Il s'agit donc d'une approximation. Notons que cette grammaire intègre également des déterminants numériques. Nous utilisons ainsi la grammaire *Dnum*, due à Max Silberztein et présentée dans [73]. Nous utilisons également la grammaire d'expressions numériques *DetNum* développée par Matthieu Constant (voir [20]). Cette grammaire est très intéressante car elle intègre des prédéterminants tels que *environ*, *à vue de nez*, *en dessous de* ou encore *de l'ordre de*, ainsi que des déterminants numériques basés sur des grandeurs. Elle peut ainsi reconnaître des expressions telles que : *à peu près 2 kilos de*, *à peine plusieurs dizaines de*, *pas loin de trois quarts de*, etc.

Notre grammaire *Adj*, présentée sur la figure 1.17, se contente de reconnaître une séquence à valeur adjectivale entourée ou non par des virgules. La grammaire *AdjSansVirgule* qui décrit réellement ces séquences est présentée sur la figure 1.18. Ce graphe reconnaît des adjectifs seuls comme *joyeux* ou *mal embouché*, mais aussi des constructions plus complexes mettant en jeu plusieurs adjectifs comme *ni jeune ni vieux* ou *riche et célèbre*. Nous décrivons également la

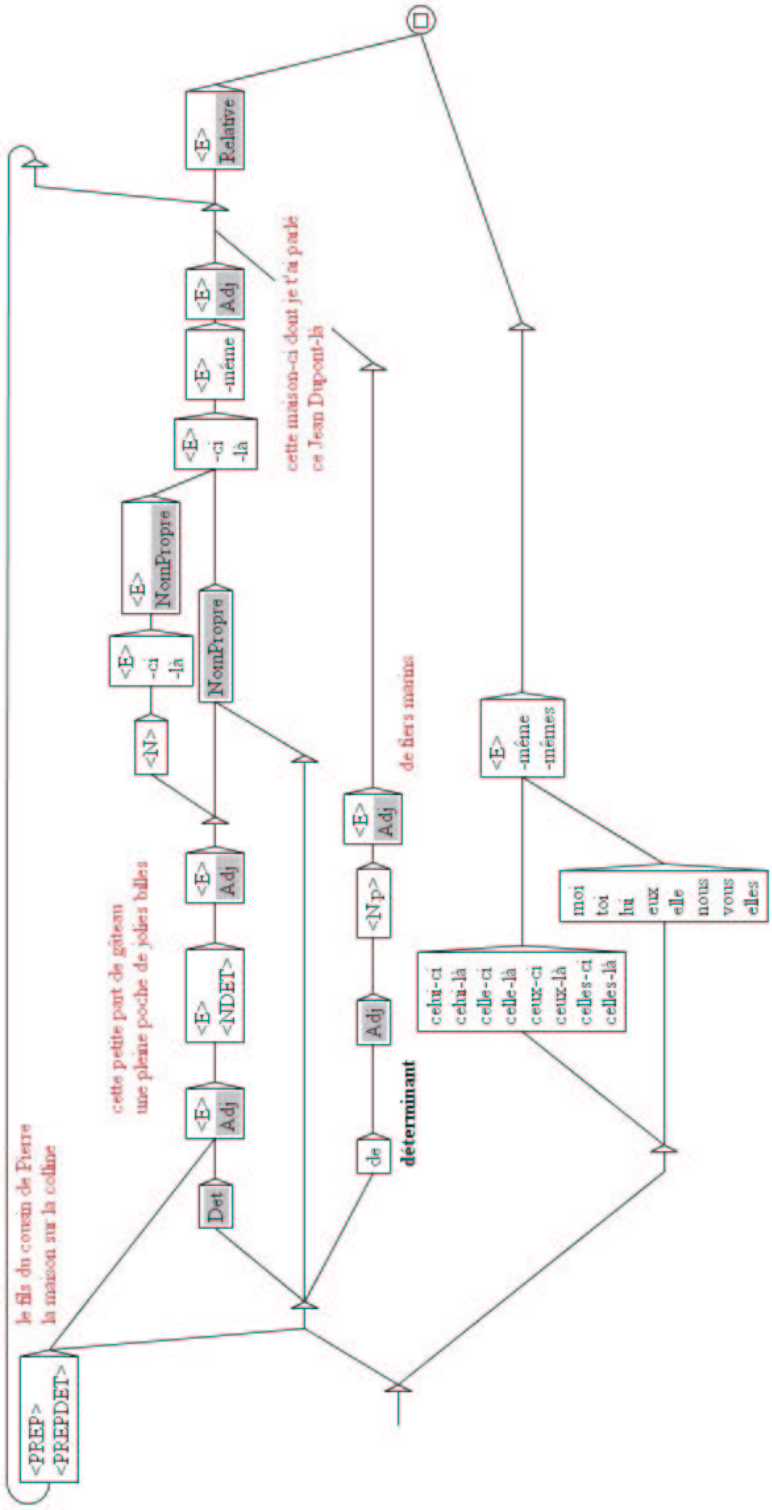

FIG. 1.15 – Graphe GNSimple

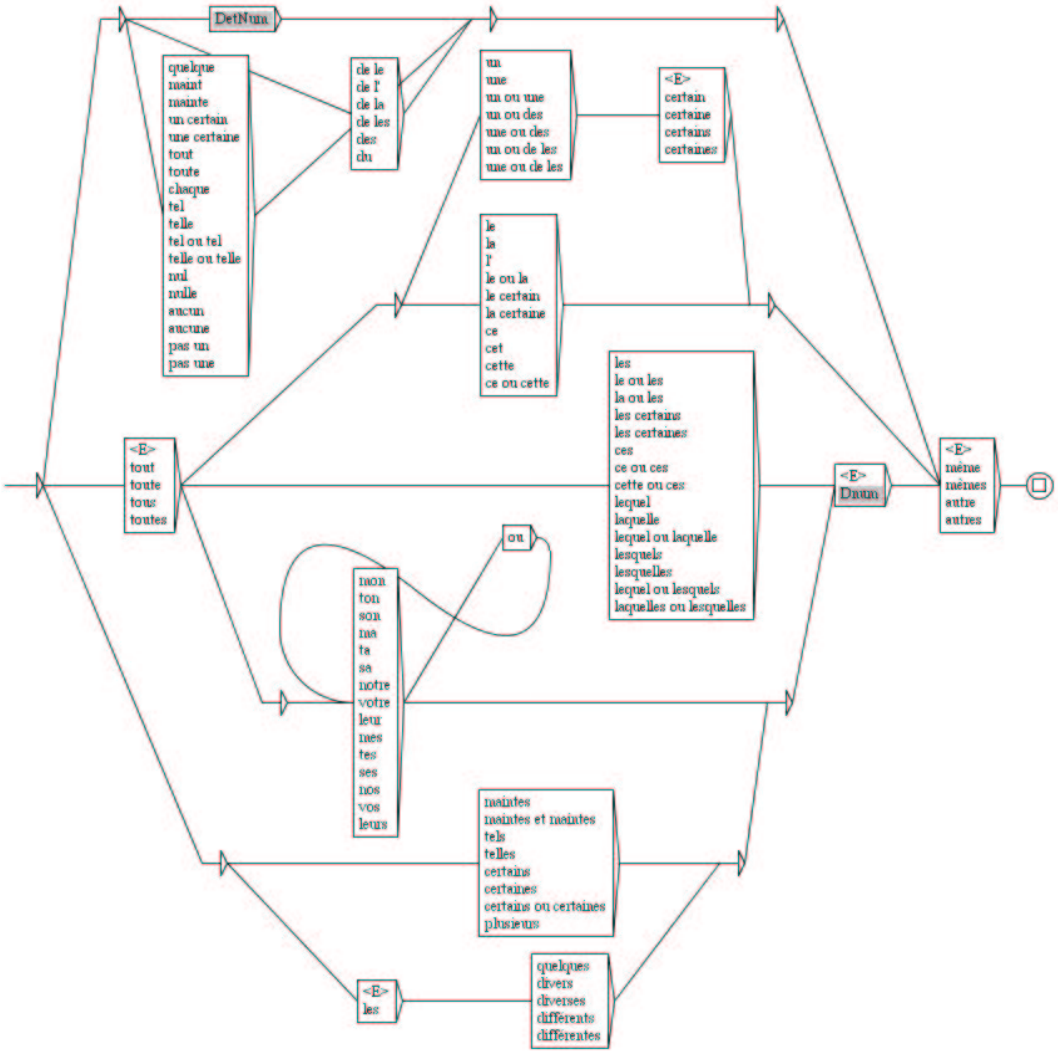

FIG. 1.16 – Graphe Det

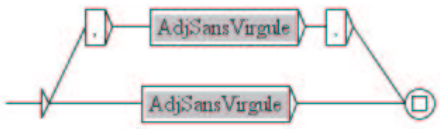

FIG. 1.17 – Graphe Adj

possibilité de reconnaître des syntagmes adjectivaux contenant des groupes nominaux, des complétives et des infinitives tels que :

tout beau pour aller danser

pas assez coupant pour que ce soit dangereux
trop vieux pour un athlète

En réalité, les graphes `Completive` et `Infinitive` ne sont pas définis, car cela supposerait que notre description des verbes soit achevée. Nous les avons inclus dans notre grammaire des adjectifs uniquement pour mémoire.

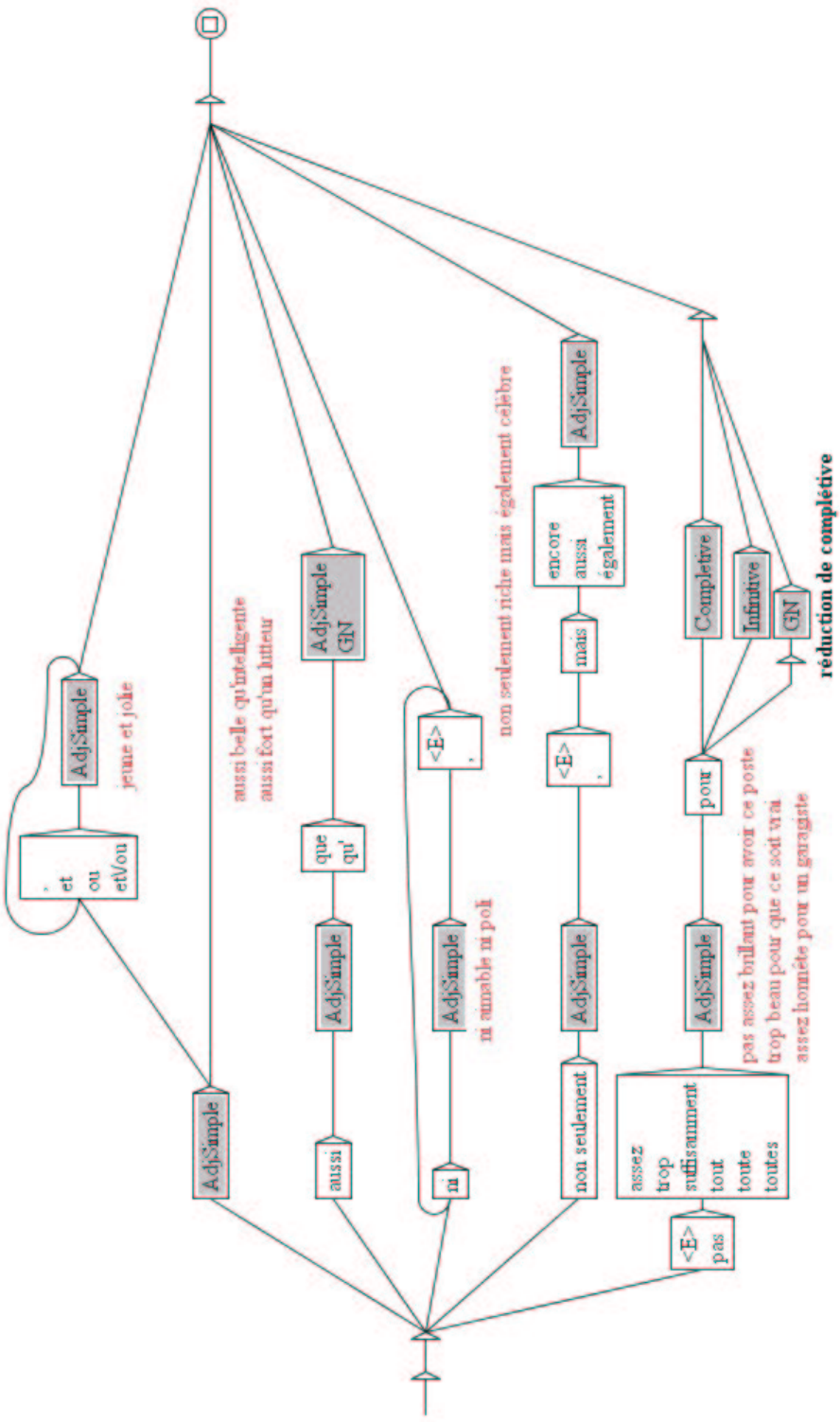

FIG. 1.18 – Graphe AdjSansVirgule

Note : on peut se demander pourquoi la grammaire possède une boîte qui contient *tout*, *toute* et *toutes*, mais pas *tous*. Cela est dû à une bizarrerie de comportement de l'adverbe *tout*. En effet, cet adverbe est soumis à des règles spéciales : il s'accorde quand il est placé devant un adjectif au féminin commençant par une consonne ou un *h* aspiré :

Luc s'est fait tout beau
Luc et Max se sont fait tout beaux
**Luc et Max se sont fait tous beaux*
Léa s'est fait toute belle
Léa et Ida se sont fait toutes belles
Léa est tout aimable
**Léa est toute aimable*
Léa et Ida sont tout habillées
**Léa et Ida sont toutes habillées*
**Les grilles sont tout hérissées de pointes*
Les grilles sont toutes hérissées de pointes

Notons qu'il serait possible de vérifier cette contrainte a posteriori en même temps que les autres contraintes d'accord, comme nous le verrons en 1.3.7.6.

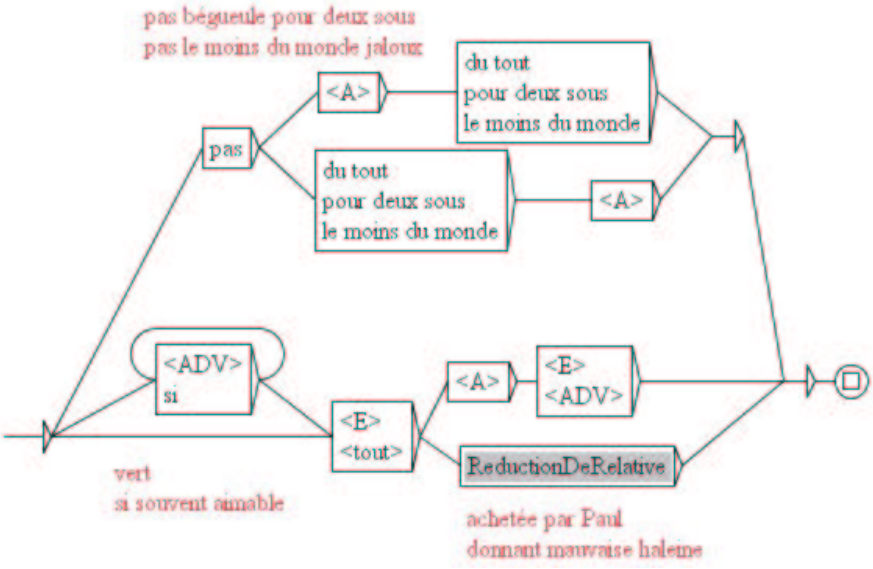

FIG. 1.19 – Graphe AdjSimple

Le graphe *AdjSimple* présenté sur la figure 1.19 présente des constructions élémentaires centrées autour d'adjectifs. Nous avons autorisé des constructions comme *pas Adj du tout* car celles-ci peuvent se combiner avec les structures du graphe de la figure 1.18 :

non seulement pas aimable du tout, mais également malhonnête

ni pas du tout jaloux ni pas du tout confiant

Ce graphe prend comme atomes soit des adjectifs comme *dur* ou *vert-de-gris*, soit des réductions de relatives comme *donné à Paul par Marie*, soit des constructions au participe présent telles que *coûtant une fortune*. De même que pour les appels aux sous-graphes `Completive` et `Infinitive`, l'appel à la grammaire `ReductionDeRelative` ne renvoie à aucune grammaire existante, et n'a été inséré dans le graphe *AdjSimple* que pour indiquer que ces formes doivent être prises en compte.

Enfin, le graphe *NomPropre* présenté sur la figure 1.20 donne une description grossière de noms propres. Cette grammaire a été essentiellement conçue pour décrire les noms propres contenus dans nos corpus de tests et rendre possibles nos expériences. Pour une description plus fine et générale des noms propres, il serait sans doute préférable de reconnaître préalablement les noms propres, par exemple au moyen de procédés tels que ceux décrits dans [26].

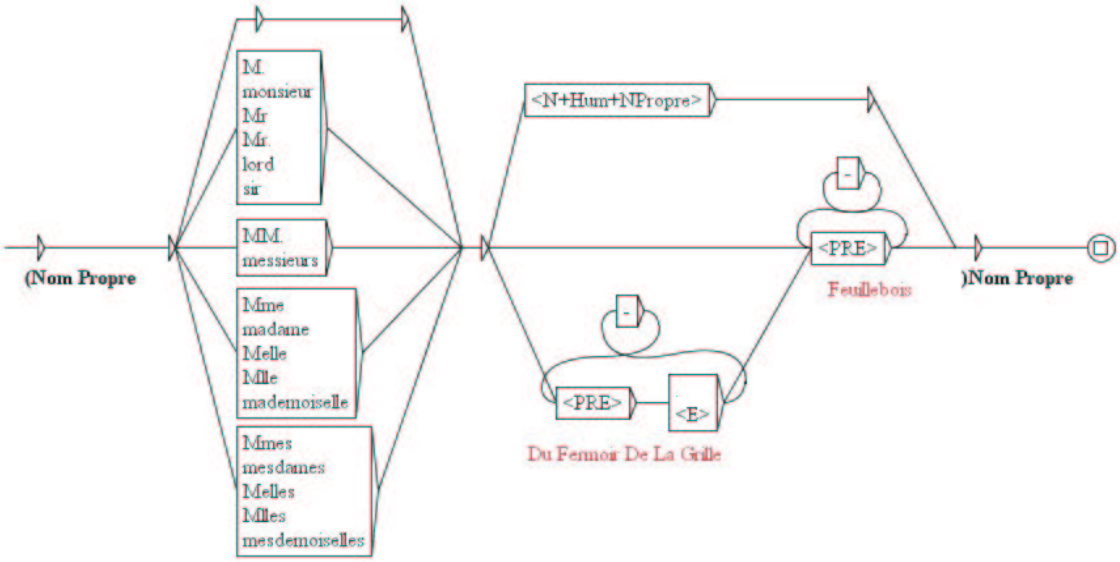

FIG. 1.20 – Graphe NomPropre

1.3.7 Syntagmes verbaux

Nous avons choisi de décrire les syntagmes verbaux de telle sorte que les références à ces syntagmes se limitent à des appels à des sous-graphes. Cette compacité simplifie grandement la construction des grammaires de phrases, comme le montre la figure 1.21 qui présente l'extrait du graphe générique *super-N0V(P)* qui décrit la construction élémentaire N_0 V. On peut y voir que la description de cette structure se résume à deux appels à des sous-graphes : `N0-@A` qui décrit le sujet et `V-@A` qui décrit le syntagme verbal.

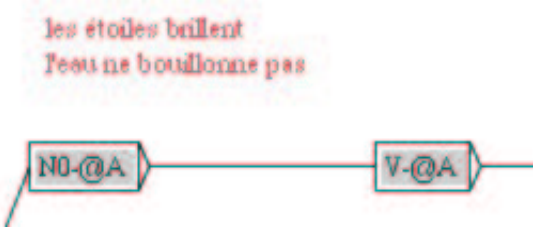

FIG. 1.21 – Extrait du graphe *super-N0V(P)*

Une telle représentation pose de nombreux problèmes. Comment en effet gérer des phénomènes tels que la négation, la passivation, l'introduction d'auxiliaires, etc ? De plus, on peut voir sur la figure 1.21 que cette approche conduit à décrire séparément les verbes et leurs sujets. Dans ces conditions, comment faire pour gérer des contraintes telles que l'accord en genre et en nombre entre ces éléments ? Nous montrerons dans cette section comment notre représentation permet de prendre en compte les différents phénomènes qui affectent le syntagme verbal.

1.3.7.1 Auxiliaires verbaux

L'une des transformations que peut subir un verbe est l'insertion d'auxiliaires. Les auxiliaires élémentaires sont ceux de temps : *avoir* et *être*. Il existe d'autres verbes pouvant jouer le rôle d'auxiliaires. [31] donne une liste de tels verbes : *aller*, *devoir*, *être (en passe+sur le point+près+en train+loin+en voie) de*, *être (ε+après) à* , *être pour*, *faillir*, *manquer (ε+de)*, *ne faire que de*, *paraître*, *sembler*, *passer pour*, *pouvoir*, *sortir de*, *venir (à+de)*. La plupart de ces verbes ont en commun d'apporter une nuance de sens au verbe sans modifier la capacité de sélection des arguments du verbe. Ainsi, dans la phrase

il commence à pleuvoir

le sujet *il* est figé avec le verbe *pleuvoir* et non sélectionné par l'auxiliaire *commencer*. Cela est confirmé par l'impossibilité de changer le pronom personnel sujet :

**elle commence à pleuvoir*

Cependant, d'autres auxiliaires ont un comportement différent, et imposent des contraintes sur le sujet (cf. la propriété $N_0 = N_{nc}$ dans la table 1 de [34]) :

*(Luc+*cette machine) se dépêche de dénoyauter les olives*

Comme cela a été présenté dans [39] et [41] pour le français, [55] pour le portugais et développé plus complètement pour l'anglais dans [38], la liste des verbes qui peuvent présenter un comportement d'auxiliaire peut être considérablement allongée. En effet, de nombreux verbes qui se construisent avec une infinitive peuvent jouer ce rôle. Cependant, ce n'est pas

le cas de tous, comme le montre l'exemple suivant :

Paul a entendu marcher

Dans cette phrase, le sujet de *marcher* n'est pas *Paul*. Le verbe *entendre* ne peut donc pas être considéré comme un auxiliaire [12].

Parmi les verbes qui admettent une infinitive dont le sujet est celui de l'auxiliaire, on trouve des verbes simples (*souhaiter*), accompagnés ou non de pronoms et/ou prépositions (*s'empresser de*), des expressions figées (*être à deux doigts de*) et des expressions nominales à verbe support ou même semi-figées (*avoir (ε+une fâcheuse+une nette) tendance à*). Cette dernière catégorie se caractérise par l'acceptation de variantes pouvant porter sur divers éléments de la construction. Ces variantes sont aisément représentables par graphes, comme le montre la figure 1.22.

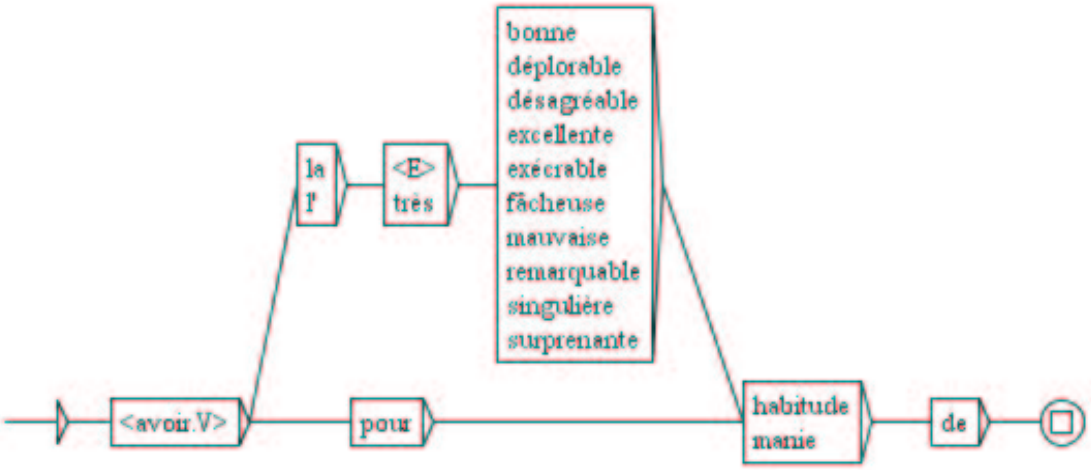

FIG. 1.22 – Variantes de l'expression *avoir l'habitude de*

Les phrases comportant des auxiliaires verbaux peuvent être analysées de deux façons : soit en détaillant les imbrications d'infinitives, soit en considérant toujours le verbe principal comme centre de la construction. On obtient ainsi pour une même phrase des analyses telles que :

(1) $(Paul)_{N_0}$ $(devrait)_V$ $((s'empresser\ de)_V$ $(filer)_{Vinf})_{Vinf}$
(2) $(Paul)_{N_0}$ $(devrait\ s'empresser\ de)_{Aux}$ $(filer)_{Vinf}$

L'analyse (1) suppose que l'on dispose pour chaque auxiliaire des grammaires décrivant les constructions N_0 *V Vinf W* et *Vinf W*. Or, pour les auxiliaires qui ne sélectionnent pas leur

[12] Il existe toutefois un autre emploi d'*entendre* qui peut jouer un rôle d'auxiliaire : *Paul entend faire respecter sa volonté*. *entendre* a ici un sens proche de *avoir l'intention de*.

sujet, une grammaire décrivant une construction telle que N_0 *devoir Vinf W* n'est pas correcte puisque le N_0 de la phrase n'est pas sélectionné par le verbe principal. En conséquence, nous préférons l'analyse (2) qui est construite sur le schéma N_0 *(V+*V_{aux} V^0_{inf}*) W*. Cette analyse permet de représenter plus facilement les contraintes entre le sujet et le verbe principal. Dans la pratique, cette approche nous permet de regrouper tous les auxiliaires dans une même grammaire *Vaux*. On peut ainsi coder facilement le schéma N_0 *(V+*V_{aux} V^0_{inf}*) W*.

Comme nous l'avons vu, une des caractéristiques des principaux auxiliaires est de ne pas exercer de contrainte sur la sélection des arguments du verbe principal. Cette règle a pour conséquence que ces auxiliaires peuvent se construire avec n'importe quel sujet. Or, si l'on examine les exemples mentionnés plus haut, on observe des blocages tels que :

**il s'empresse de pleuvoir*
**que Paul vienne a pour habitude de m'énerver*

Ces verbes en position auxiliaire exercent une contrainte sur les sujets des infinitives. Peu importe qu'on les qualifie d'auxiliaires ou non : la terminologie est peu normalisée. En tous cas, nous n'avons pas tenu compte des contraintes et nous avons regroupé ces verbes avec les autres auxiliaires car cela permet d'étendre la couverture de nos grammaires sans avoir à décrire explicitement les constructions de ces verbes. Ainsi, pour analyser une phrase telle que

Paul était à deux doigts de commencer à parler

il n'est pas nécessaire de disposer d'une description complète de l'expression *être à deux doigts de* ; il suffit de l'ajouter dans notre grammaire de façon purement lexicale en rajoutant *à deux doigts* dans la liste des termes pouvant apparaître entre *être* et *de*. Le bénéfice est particulièrement flagrant lorsque l'expression n'est pas codée dans les tables, comme c'est le cas d'une expression comme *être à un quart de poil de mouche de*. N'étant pas codée dans les tables, une telle expression ne sera pas prise en compte par notre approche consistant à générer des constructions à partir des tables, alors que nous pouvons l'intégrer facilement dans la grammaire des auxiliaires. De cette manière, on peut donc construire une grammaire des auxiliaires reconnaissant un grand nombre de cas sans qu'il soit nécessaire de disposer de toutes les descriptions des verbes concernés. Cela constitue donc une solution à court terme qui nous a permis de mener nos tests à bien. Au fur et à mesure que l'on disposera des descriptions complètes des verbes, nous pourrons retirer de notre grammaire les verbes qui imposent des contraintes sur le sujet et ainsi tendre vers une représentation plus fine des auxiliaires.

Les grammaires que nous allons présenter ont été réalisées conjointement avec Matthieu Constant et Takuya Nakamura et ont fait l'objet d'une communication au 21^e Colloque International "Grammaires et Lexiques Comparés". Ces grammaires décrivent différents types de constructions verbales pouvant jouer un rôle d'auxiliaire. Les graphes que nous allons présenter montrent les verbes que nous avons pris en compte, conjugués à l'indicatif et sans négation. Pour chacun de ces graphes, il existe des variantes décrivant les mêmes formes avec la négation

et à l'infinitif. En outre, nous avons considéré que toutes les combinaisons d'auxiliaires étaient possibles. Cela n'est évidemment pas vrai comme le montrent des exemples tels que

**Paul vient de venir de manger*
**Léa commence à venir de manger*

Il ne s'agit donc que d'une approximation effectuée dans un but pratique. La description exacte des combinaisons possibles est sans doute réalisable. Toutefois, les blocages rencontrés étant essentiellement de nature sémantique, il est probable que l'on soit amené à examiner une par une toutes les combinaisons possibles. Dans la mesure où des combinaisons comme celles présentées ci-dessus ne se rencontrent pas dans les textes, il n'est pas certain qu'une combinatoire exacte des auxiliaires apporte beaucoup au problème de l'analyse.

On peut noter que les auxiliaires de temps *avoir* et *être* n'ont pas été inclus dans ces grammaires d'auxiliaires. Cela est dû au fait que les auxiliaires de temps varient en fonction des verbes. Le fait d'inclure ces auxiliaires dans le graphe générique des syntagmes verbaux nous permet d'obtenir une grammaire des auxiliaires, indépendante des tables de verbes, et donc, facilement réutilisable dans d'autres applications.

Le graphe présenté sur la figure 1.23 regroupe les verbes simples, non pronominaux et sans préposition, que nous avons pris en compte. On y trouve des verbes considérés classiquement comme des auxiliaires tels que *pouvoir* et *devoir*, mais également d'autres verbes pouvant exprimer une volonté (*désirer*), une appréciation (*détester*) ou encore une apparence (*sembler*).

Les figures 1.24 et 1.25 décrivent respectivement les auxiliaires de type *V à* et *V de* qui permettent d'analyser des phrases telles que

Paul apprend à lire l'égyptien
Luc redoute de rencontrer Léa
Max a oublié de penser à prendre le pain

Nous avons également intégré dans nos grammaires des verbes pronominaux. Les figures 1.26, 1.27 et 1.28 présentent respectivement les auxiliaires basés sur les constructions type *se V*, *se V à* et *se V de*. Voici quelques exemples de phrases reconnues grâce à ces verbes :

Le chèque s'est avéré provenir d'un escroc
Léa s'échine à réparer cette radio
Max se contente de faire la vaisselle

Les verbes des figures 1.23 à 1.28 ont été extraits des tables de verbes 1, 6, 7 et 8 de [34].

Le graphe de la figure 1.29 reconnaît les séquences du type *en être W à*. Ces construction se caractérisent par leur nombre très réduit. En effet, on ne distingue que 4 verbes entrant dans cette construction. Deux de ces emplois partagent la forme *en être à* :

Paul en est à rédiger les remerciements
Max en est à mendier dans le métro

La première forme revêt un aspect de localisation dans le temps qui apparaît clairement lorsque l'on reformule la phrase en :

Paul en est à l'étape qui consiste à rédiger les remerciements

La seconde forme semble elle être réduite d'une des deux autres formes possibles :

Max en est réduit à mendier
Max en est arrivé à mendier

On peut noter que la forme *en être arrivé à*, contrairement à la forme *en être réduit à*, ne porte pas nécessairement d'appréciation négative, comme on peut le voir dans une phrase telle que

Luc en est arrivé à collectionner les puits de pétrole

La figure 1.30 reconnaît des séquences du type *faire N de*. Les noms qui entrent dans ce graphe ont été tirés de la table FN dans [29]. Cette grammaire permet d'analyser des phrases telles que :

Riton a fait la bêtise de se faire plumer de 100 balles
Léa a fait la gaffe de tout dire à Luc

On peut noter que cette grammaire construite autour du verbe support *faire* pourrait être étendue à d'autres constructions basées sur des variantes de ce verbe :

Max a accompli l'exploit de courir pendant 14 heures
Léa a commis le sacrilège d'aimanter le tournevis sacré

Les figures 1.31 et 1.32 décrivent des auxiliaires construits selon les schémas *être (Vpp+A) à* et *être (Vpp+A) de*. La plupart de ces verbes peuvent être analysés comme des passifs d'emplois verbaux à complétive, comme c'est le cas pour des phrases telles que :

Max est autorisé à fumer une cigarette
Paul est humilié de ne pas avoir été promu sergent

Ces constructions ont été tirées des tables 11 et 13 de [34]. Nous avons ajouté à cette grammaire des constructions en *être* qui ne s'analysent pas comme des passifs mais comme des adjectifs. On trouve ainsi des constructions formées avec des participes passés autonomes :

Léa est disposée à parler aux autorités
Max est assuré de réussir son pari

et d'autres issues des tables d'adjectifs de [57] :

Paul est apte à encadrer une colonie de vacances
Ida est impatiente de lire cet article

Les grammaires suivantes sont basées sur le schéma général *avoir N Prep*. Elles ont été élaborées à partir de différentes tables de [49] et [57]. La figure 1.33 décrit les constructions en *avoir (ε+LE) N de* :

Paul a eu la jugeotte de se taire
Max a honte de mettre son nouveau pull
Léa a le souci de faire les choses correctement

Le graphe de la la figure 1.34 décrit des constructions du type *avoir (ε+LE+de LE+UN) N pour*. Cela permet de reconnaître des phrases comme :

Ida a toujours eu du talent pour faire les bouquets
Luc a de grandes dispositions pour devenir un athlète

Pour des raisons de présentation, nous avons séparé les constructions *avoir N à* en deux grammaires présentées sur les figures 1.35 et 1.36. Ces grammaires regroupent des auxiliaires construits sur le schéma *avoir (ε+LE+de LE+UN) N à* :

Léa a un intérêt certain à accepter ce rendez-vous
Max a plaisir à constater l'échec de Paul
Luc a du mérite à supporter les jérémiades de Léa

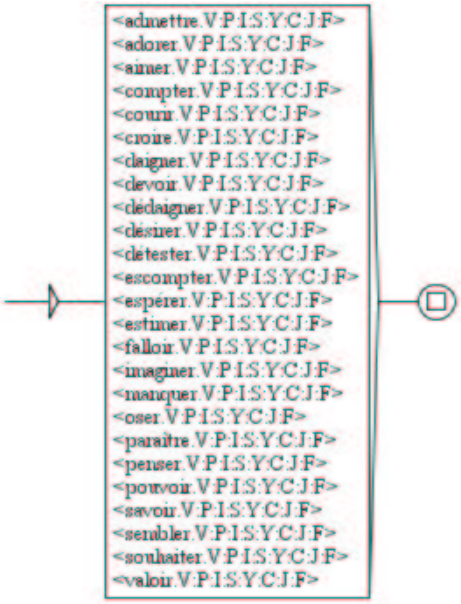

FIG. 1.23 – Auxiliaires *V*

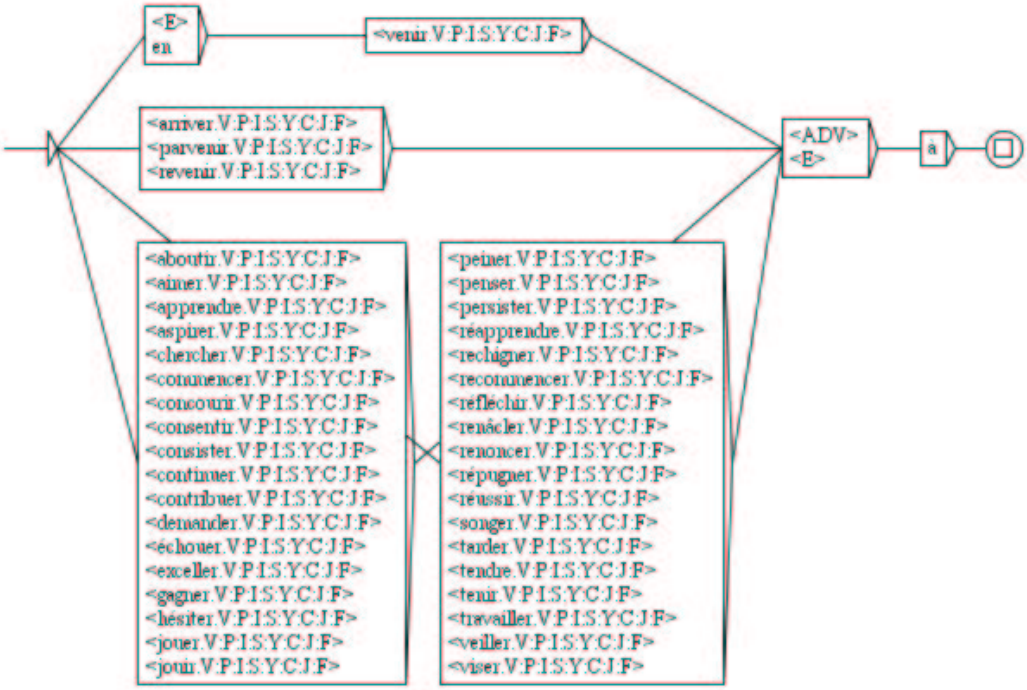

FIG. 1.24 – Auxiliaires *V à*

FIG. 1.25 – Auxiliaires *V de*

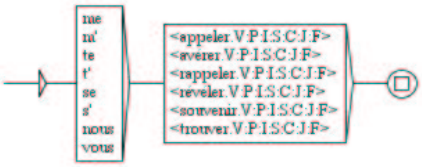

FIG. 1.26 – Auxiliaires *se V*

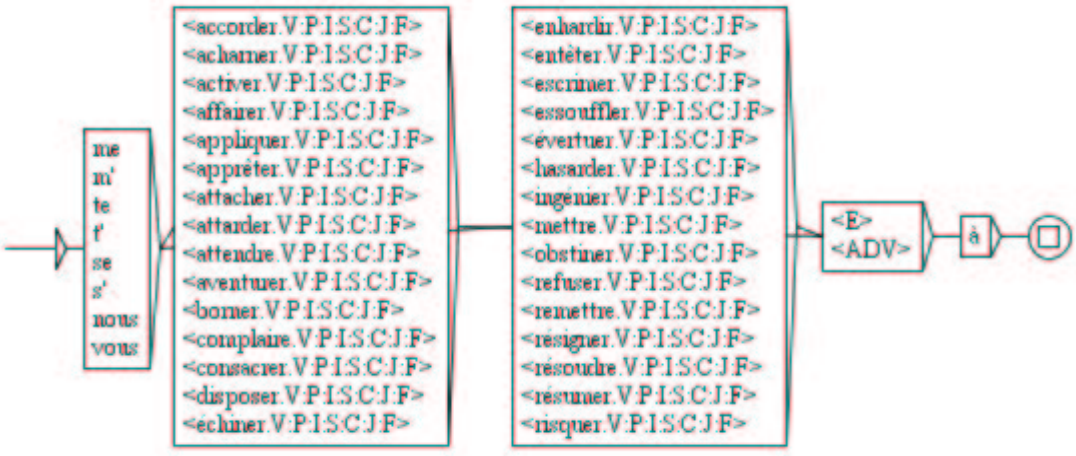

FIG. 1.27 – Auxiliaires *se V à*

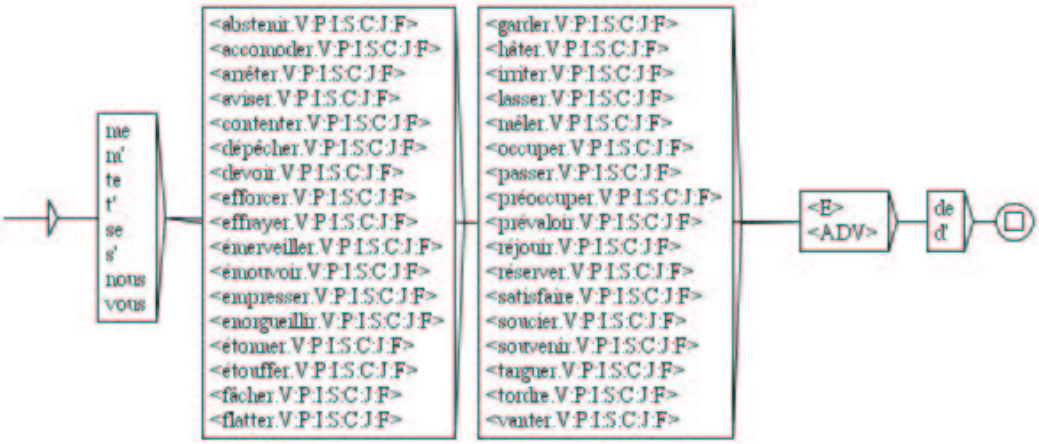

FIG. 1.28 – Auxiliaires *se V de*

FIG. 1.29 – Auxiliaires *en être (ε+réduit) à*

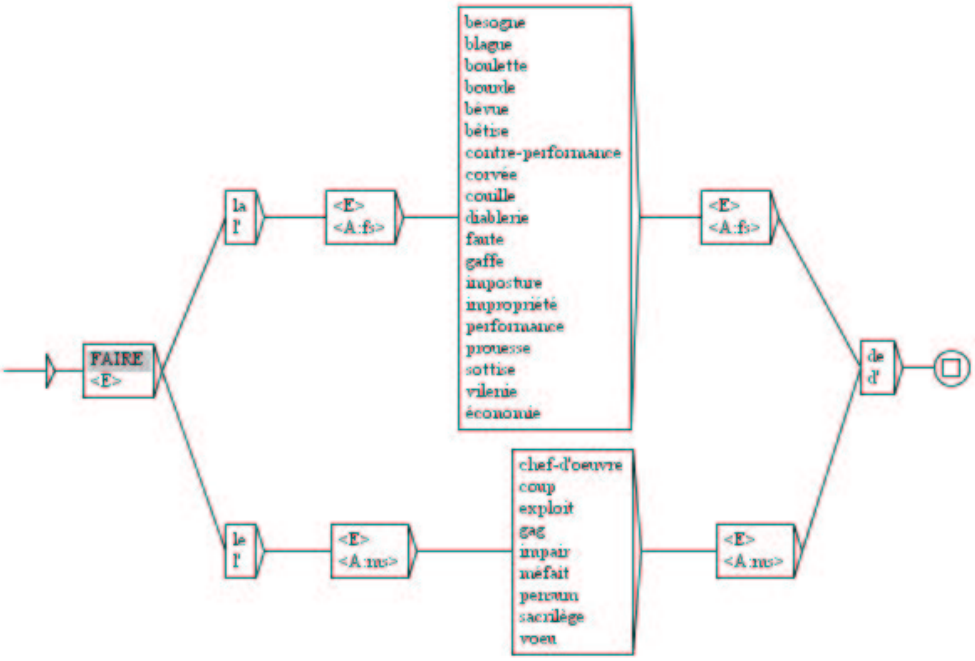

FIG. 1.30 – Auxiliaires *faire N de*

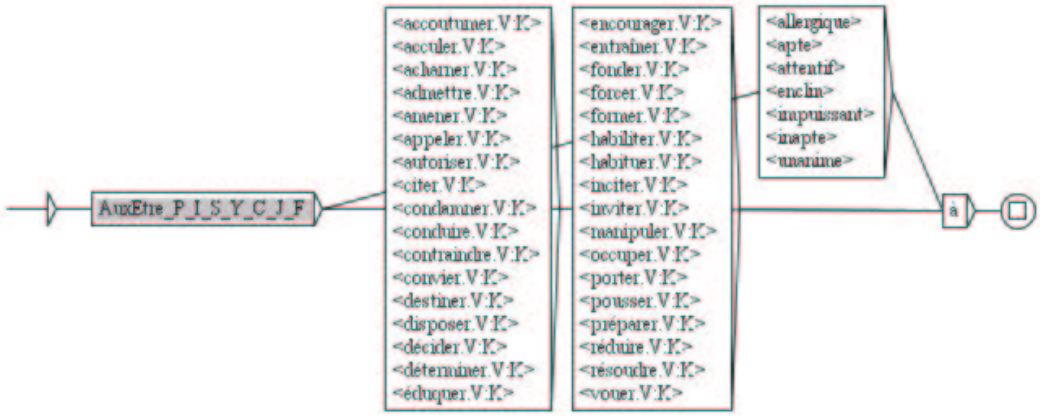

FIG. 1.31 – Auxiliaires *être (Vpp+A) à*

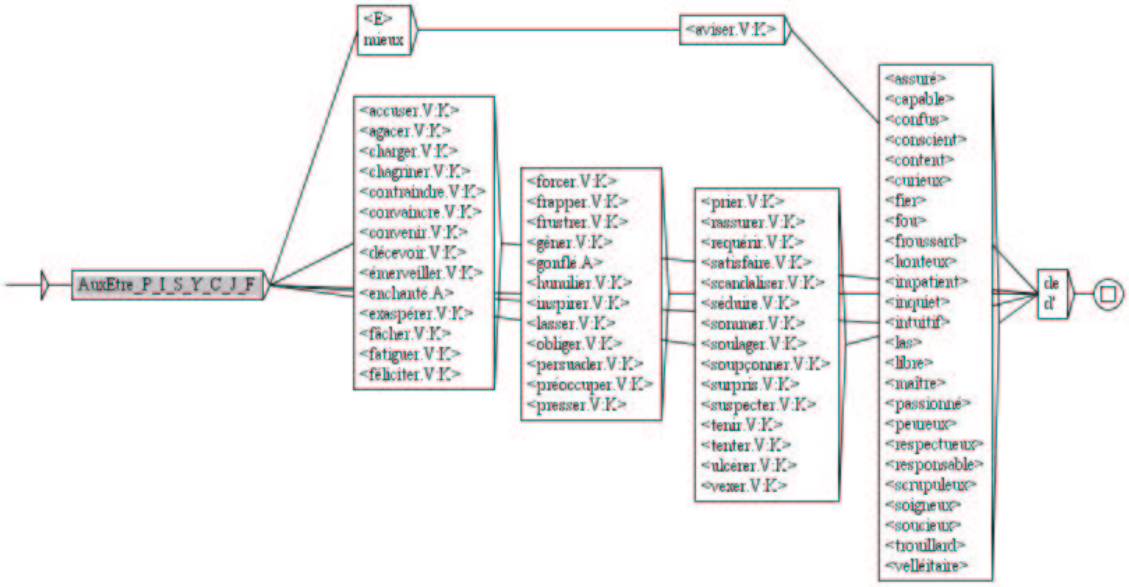

FIG. 1.32 – Auxiliaires *être (Vpp+A) de*

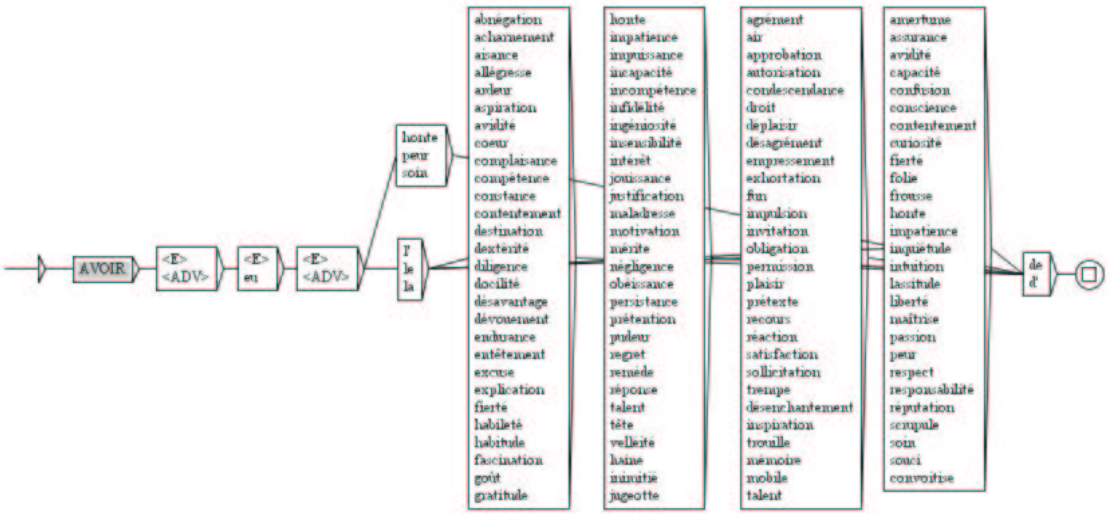

FIG. 1.33 – Auxiliaires *avoir N de*

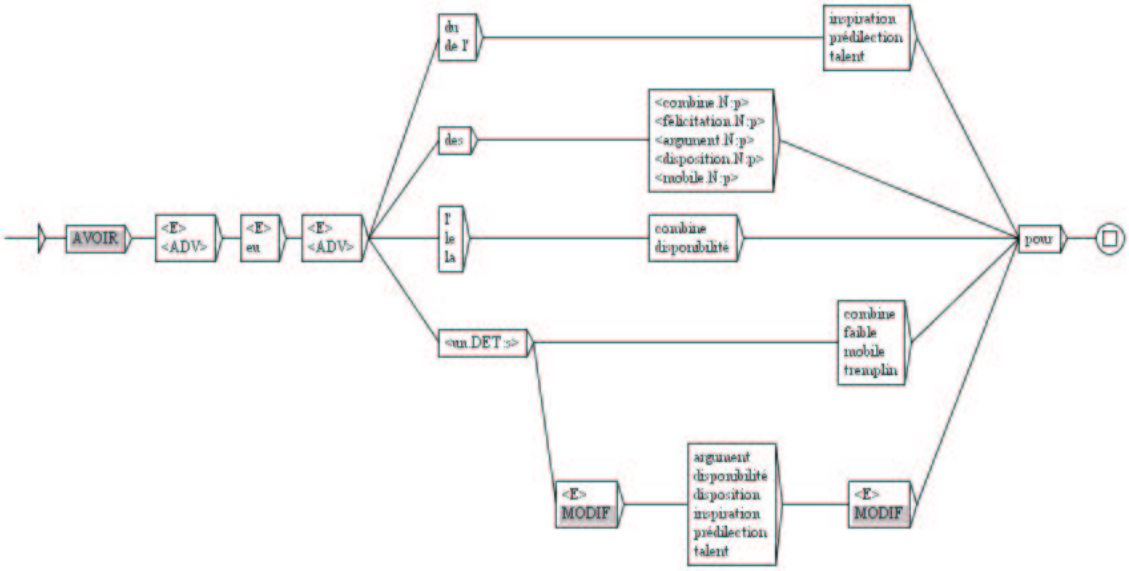

FIG. 1.34 – Auxiliaires *avoir N pour*

FIG. 1.35 – Auxiliaires *avoir UN N à*

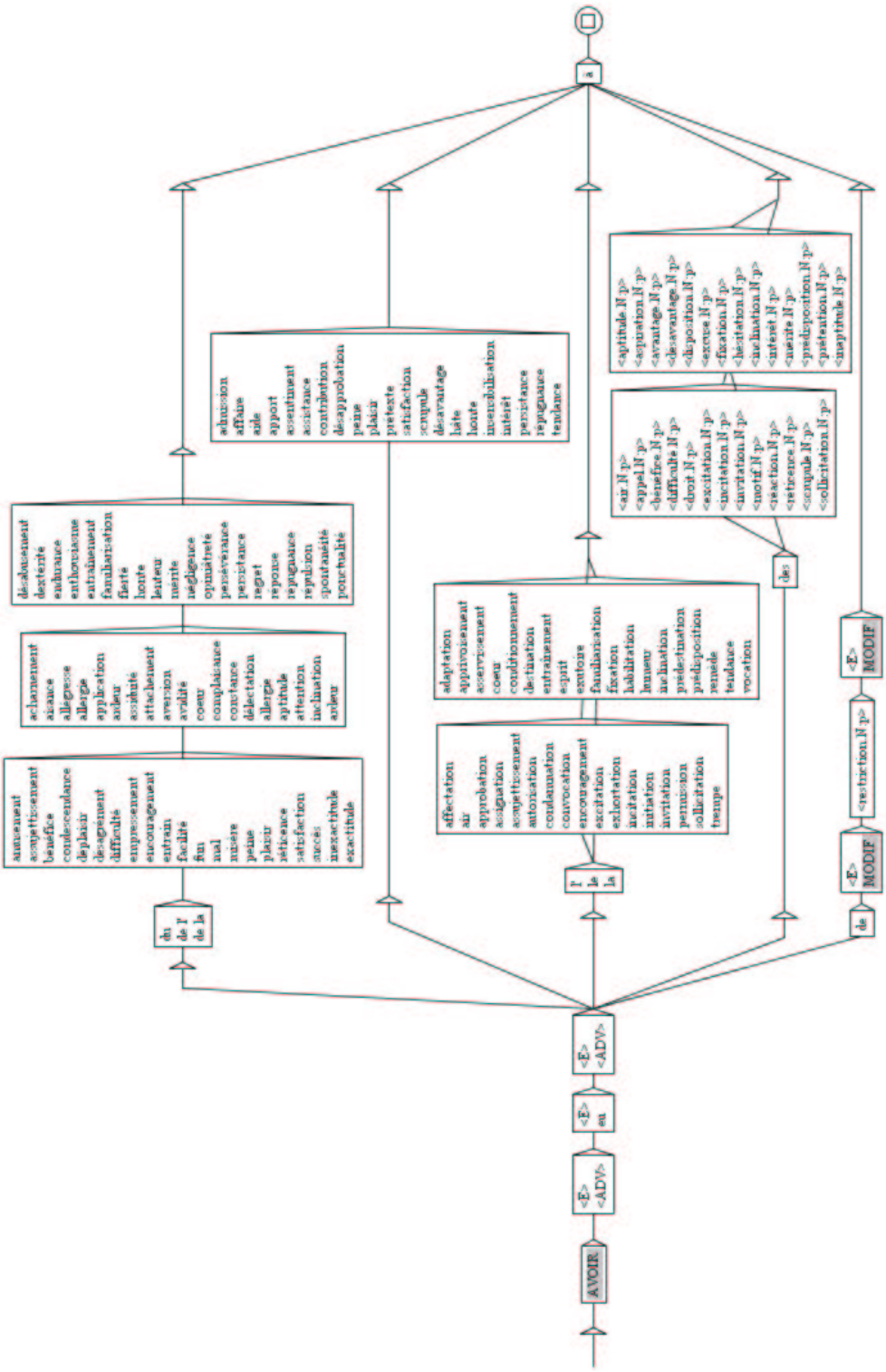

FIG. 1.36 – Auxiliaires *avoir (ε+LE+de LE) N à*

1.3.7.2 Négation

À l'exception de quelques verbes présentant des blocages sémantiques, la négation est une transformation qui peut s'appliquer de façon très générale à toutes les constructions :

Paul ne lit guère de romans policiers
ne pas manger de fruits nuit à la santé
n'est-ce plus Paul qui dirige cette revue ?
ne sortez jamais de cette enceinte !

Nous avons choisi de traiter les négations du type *ne V (ε+AdvNeg)*. Ce choix nous distingue de l'approche de Roche dans [68], car il cherche à expliciter systématiquement le second membre de la négation. Ainsi, il détaille des structures telles que *pas* UN_0 *ne* V_0 W pour analyser des phrases comme

pas le moindre étudiant n'est arrivé à l'heure

Selon lui, la solution consiste dans ce cas à décrire les structures de groupes nominaux dont le déterminant est *le moindre*, ce qui permet d'éviter de reconnaître des phrases erronées telles que

**pas l'étudiant n'est arrivé à l'heure*

Il existe d'autres cas où l'absence d'un adverbe négatif (*pas, point, guère, etc*) est compensé par un autre élément de la phrase. Ainsi, dans l'exemple suivant, la négation est portée par l'adverbe *à aucun moment* :

à aucun moment Paul n'a songé à sa famille

Il arrive également que l'adverbe négatif soit absent, le plus souvent pour des raisons stylistiques :

il est fort regrettable qu'il ne se soit marié avant ses trente ans
l'art moderne ne peut se résumer à quelques gribouillages

On voit sur ces quelques exemples que le problème ne peut pas être résolu uniquement avec des contraintes sur les déterminants des groupes nominaux. C'est pourquoi nous avons choisi de considérer l'adverbe négatif comme facultatif. Ainsi, nous autorisons des structures *ne V* en faisant l'hypothèse que les différents éléments de la phrase susceptibles de porter la marque de la négation sont décrits correctement. De cette manière, nous analysons la phrase

pas le moindre étudiant n'est arrivé à l'heure

en considérant que *pas le moindre étudiant* constitue le syntagme sujet d'une structure N_0 *ne V*. Cette approche suppose que les descriptions des groupes nominaux et des adverbes incluent des formes négatives.

Ce choix de description des structures négatives pose un problème de sur-reconnaissance. En effet, la liberté combinatoire que nous avons adoptée autorise l'analyse de phrases incorrectes telles que :

**jamais Paul n'a pas mangé de gâteau*
**Luc veut qu'il ne parte*
**Léa n'a pas parlé à pas un de ses amis*

Nous pourrions tenter de remédier à ce problème en contraignant les combinaisons. Ainsi, si l'on codait séparément les adverbes négatifs tels que *jamais*, on pourrait imposer qu'une phrase en *ne V pas* ne contienne pas de tels adverbes. Nous voyons cependant deux objections à cela. La première tient au fait que ces contraintes ne sont pas toujours identifiables clairement. Ainsi, si l'on considère les phrases

**Léa n'a pas parlé à pas un de ses amis*
**Léa n'a pas parlé à aucun de ses amis*
**Léa n'a plus parlé à pas un de ses amis*
Léa n'a plus parlé à aucun de ses amis

on constate que toutes les combinaisons ne sont pas incorrectes.

La seconde raison se rapporte à la discussion sur les grammaires d'analyse. Dans la mesure où la sur-reconnaissance de formes invalides n'affecte pas l'analyse des formes correctes, nous ne pensons pas que cet effort de description combinatoire soit nécessaire.

Comme le montre la figure 1.37, nous avons regroupé un certain nombre d'adverbes négatifs dans une grammaire nommée *AdvNeg*. Cette grammaire n'est pas exhaustive car elle ne comprend que des adverbes pouvant apparaître à toutes les positions, en particulier dans des constructions infinitives où la négation est de la forme *ne AdvNeg V* :

ne (pas+jamais+point trop+pas même) manger

Nous avons ainsi écarté des constructions telles que *ne V pas des masses* qui ne peuvent apparaître qu'à droite du verbe :

ne pas swinguer des masses est typique des requiems
**ne pas des masses swinguer est typique des requiems*

La négation en *ne ... rien* pose un problème particulier. En effet, cette forme de négation n'est possible que lorsque le verbe est transitif :

Paul ne mange rien
**Luc n'agonise rien*

Nous pouvons traiter ce phénomène en décrivant simplement *rien* comme un groupe nominal négatif. Un argument en faveur de cette solution est que *rien* peut apparaître à diverses positions dans les phrases :

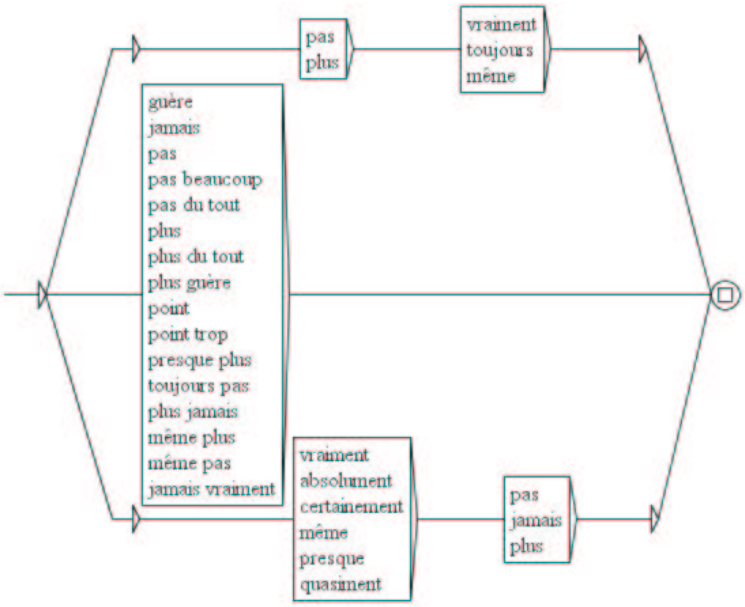

FIG. 1.37 – Grammaire *AdvNeg*

rien ne pourra me faire changer d'avis
Léa ne pense à rien
Max n'accuse Paul de rien

Il se pose toutefois un problème pour les structures N_0 *ne V rien W* lorsque le verbe se combine avec des auxiliaires. Dans ce cas, *rien* peut venir s'insérer à l'intérieur du syntagme verbal :

Léa n'a rien mangé
Léa n'a rien pu manger
Léa peut ne rien avoir mangé
Léa peut n'avoir rien mangé

Nous pouvons résoudre le problème en décrivant un syntagme verbal particulier contenant obligatoirement *ne ... rien*. Il suffit alors d'autoriser les constructions N_0 $V_{ne\ rien}$ W pour toutes les entrées qui tolèrent les constructions N_0 V N_1 W et pour lesquelles on peut avoir un N_1 non humain. Cette dernière condition permet d'écarter des phrases incorrectes telles que :

**ce vin n'a rien enivré*

Cela se complique encore lorsque l'on a des séquences plus complexes telles que *rien de W*, car la partie *de W* reste, elle, à droite du verbe principal :

Léa n'a rien mangé (du tout+de ce plat+de ce que Luc a cuisiné)

Léa n'a rien pu manger (du tout+de ce plat+de ce que Luc a cuisiné)
Léa peut ne rien avoir mangé (du tout+de ce plat+de ce que Luc a cuisiné)
Léa peut n'avoir rien mangé (du tout+de ce plat+de ce que Luc a cuisiné)

Toutefois, il semble qu'à part la séquence figée *rien ... du tout* on puisse considérer les *de W* comme des groupes nominaux à valeur partitive. On pourrait donc peut-être tenter d'analyser ces formes en prenant en compte la structure N_0 $V_{ne\ rien}$ *(du tout+*$GN_{Det=partitif}$*)* W. On pourrait également étendre l'analyse aux formes basées sur des relatives et reconnaître ainsi des phrases telles que :

Je n'ai rien mangé qui n'ait été goûté au préalable
Je n'ai rien écrit que Paul ait censuré

Notons que la structure *ne ... rien ... que* ne s'analyse pas comme une variante de *ne ... rien*, mais comme une variante de la structure *ne ... que*. Celle-ci n'est pas une négation mais une locution exprimant la restriction et signifiant *seulement* ([31]). Elle peut d'ailleurs se combiner avec la négation pour donner des phrases comme

Paul ne mange pas que des pommes

Contrairement aux adverbes de négation, *que* peut se déplacer sur n'importe quel complément ou adverbe de la phrase :

Paul ne lit de romans qu'à condition qu'ils soient policiers
Paul ne mange des pommes que le mardi

Nous n'avons pas pris en compte cette transformation dans nos grammaires car cela aurait nécessité de dupliquer tous nos graphes. En effet, cette transformation est aussi générale que la négation, mais le fait qu'elle repose sur une structure non connexe nous empêche de construire une grammaire décrivant ce phénomène de façon locale. Ainsi, si nous avons pu cantonner la description de la négation à l'intérieur de celle des syntagmes verbaux, ce n'est pas le cas pour cette transformation qui doit être traitée au niveau de la phrase entière. Une solution pour résoudre ce problème consisterait à ajouter un niveau supplémentaire à nos graphes patrons, de façon à générer automatiquement les graphes génériques correspondant aux différentes formes de la structure *ne ... que*. Pour une structure de base N_0 V N_1 *à* N_2, on pourrait ainsi décliner automatiquement les graphes génériques pour les formes :

N_0 $V_{neg\ obligatoire}$ *que* N_1 *à* N_2
N_0 $V_{neg\ obligatoire}$ N_1 *qu'à* N_2

Notons enfin que pour gérer complètement ce phénomène, il faudrait tenir compte de la possibilité de faire porter *que* sur un adverbe, ce qui nous amènerait à décrire des formes telles que :

N_0 $V_{neg\ obligatoire}$ *que Adv* N_1 *à* N_2 *(Paul ne donne que très rarement des livres à Léa)*
N_0 $V_{neg\ obligatoire}$ N_1 *que Adv à* N_2 *(?Paul ne donne des livres que très rarement à Léa)*

N_0 $V_{neg\ obligatoire}$ N_1 à N_2 que Adv (Paul ne donne des livres à Léa que très rarement)

Concrètement, nous gérons la négation dans les grammaires des syntagmes verbaux. La figure 1.38 montre le graphe générique *super-V*. On peut y voir que les négations sont codées pour chacune des formes de verbes possibles : temps simples et composés, constructions avec auxiliaires. Le sous-graphe `AdvNeg` renvoie au graphe de la figure 1.37. Le sous-graphe `Vaux` renvoie à la grammaire des auxiliaires présentée à la section 1.3.7.1. Le sous-graphe `Insert` renvoie à la grammaire des insertions qui sera présentée dans 1.3.7.5.

Lorsque la négation porte sur un auxiliaire, elle est gérée à l'intérieur de la grammaire *Vaux*. On peut ainsi reconnaître diverses formes de négation :

Paul n'a pas pu prendre ce train
Léa ne doit pas voir Max
après cet uppercut, ce boxeur aurait dû ne pas pouvoir se relever

Il est également possible de reconnaître des combinaisons de plusieurs négations :

vous ne pouviez pas ne pas l'avoir vu
ces étudiants n'ont pas fait la bêtise de ne pas passer leurs examens

1.3.7.3 Passif

Le passif a fait l'objet de grammaires séparées car il introduit un nouveau syntagme dans la phrase : l'agent. Pour chaque structure tolérant la construction passive, nous avons donc construit une grammaire qui rend compte des différentes constructions possibles. La structure la plus simple est la construction N_1 *être Vpp par* N_0 qui correspond à la forme active N_0 *V* N_1 :

Paul lit ce discours $\Leftrightarrow$ *ce discours est lu par Paul*
Léa a aimanté ce tournevis $\Leftrightarrow$ *ce tournevis a été aimanté par Léa*

Le passif peut être également introduit sous d'autres formes. Certains verbes admettent ainsi la construction passive N_1 *être Vpp de* N_0 :

Marie aime Max $\Leftrightarrow$ *Max est aimé de Marie*
le départ de Léa déçoit Luc $\Leftrightarrow$ *Luc est déçu du départ de Léa*

On trouve également une forme de passif dans laquelle l'agent est une complétive. La structure de cette construction est N_1 *être Vpp de ce que P*. Cela donne des phrases telles que :

Que Max vienne agace Luc $\Leftrightarrow$ *Luc est agacé de ce que Max vienne*
que sa femme parte a anéanti Paul $\Leftrightarrow$ *Paul a été anéanti de ce que sa femme parte*

Nous remarquons que l'agent peut presque toujours être omis pour chacune des formes présentées. Cela peut donner des phrases bizarres comme

?Paul est tué

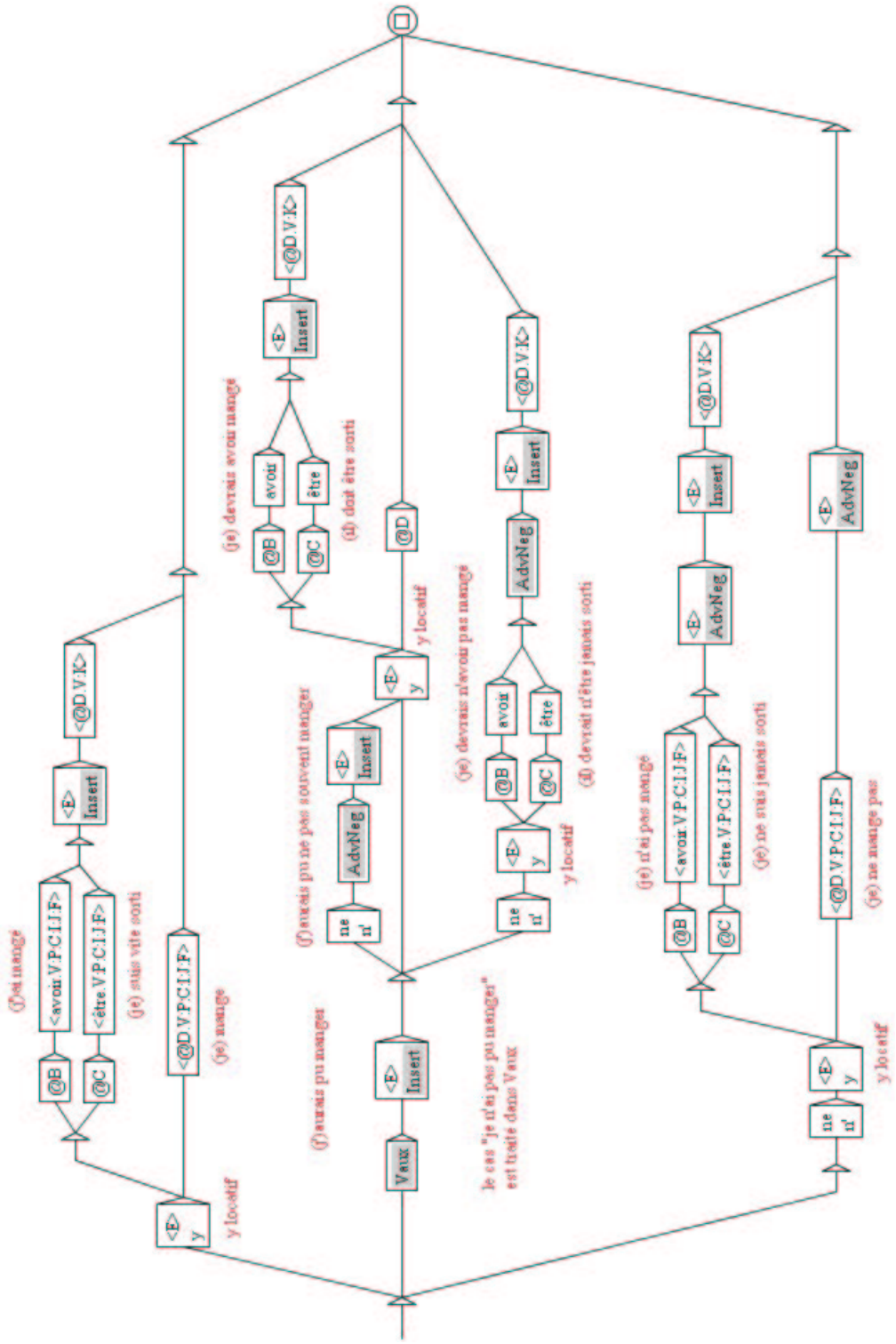

FIG. 1.38 – Graphe générique *super-V*

?cette statue est regardée
?Max a été concerné

mais il est possible de rendre ces phrases correctes en leur ajoutant des modifieurs :

dans cette pièce, Paul est tué au troisième acte
cette statue est regardée environ 1000 fois par jour
Max a été concerné pour diverses raisons

Pour des raisons pratiques, nous avons systématiquement autorisé l'omission de l'agent dans toutes les constructions passives. Toutefois, [56] a montré qu'une cinquantaine de verbes ne toléraient pas l'effacement de l'agent :

*Marie est dévorée (*ε+par la jalousie)*
*Luc est brûlé (*ε+par la passion)*
*Marie est importunée (*ε+par le bruit des pelleteuses)*

Étant donné que cette particularité syntaxique n'est pas décrite dans les tables, nous n'avons pas tenu compte de ces exceptions. Toutefois, si l'on codait cette propriété, il serait très facile de décrire ce phénomène dans nos grammaires.

Comme le montre [7], il existe d'autres structures exprimant la voix passive, construites à partir des verbes pronominaux *se faire*, *se laisser* et *se voir*. Tout comme les constructions passives présentées précédemment, la structure *se voir* se combine avec un participe passé ou un infinitif. Voici quelques exemples de phrases construites selon cette structure :

chaque année, cette région se voit envahie (par des+de) touristes
Luc s'est vu anéanti de ce que sa femme le quitte
Paul s'est vu secourir par Léa

En revanche, les verbes *se faire* et *se laisser* se construisent avec l'infinitif :

Paul s'est (fait+laissé) écarteler
Léa se laisse agacer par Max
Max s'est fait aimer de Léa

Nous avons tenu compte de cette différence de construction en séparant ces verbes en deux grammaires, l'une décrivant *se voir*, et l'autre, présentée sur la figure 1.39, décrivant *se faire* et *se laisser*.

La figure 1.40 montre le graphe générique *super-Vpassif* qui décrit les différents syntagmes verbaux passifs que nous avons pris en compte. Comme on peut le voir, cette grammaire reconnaît les formes passives affirmatives et négatives, avec ou sans auxiliaires. Nous pouvons remarquer que nous avons ici un exemple de la puissance de description des graphes. En effet, cette grammaire décrit aussi bien des formes courantes telles que

Paul a été dénoncé par Léa

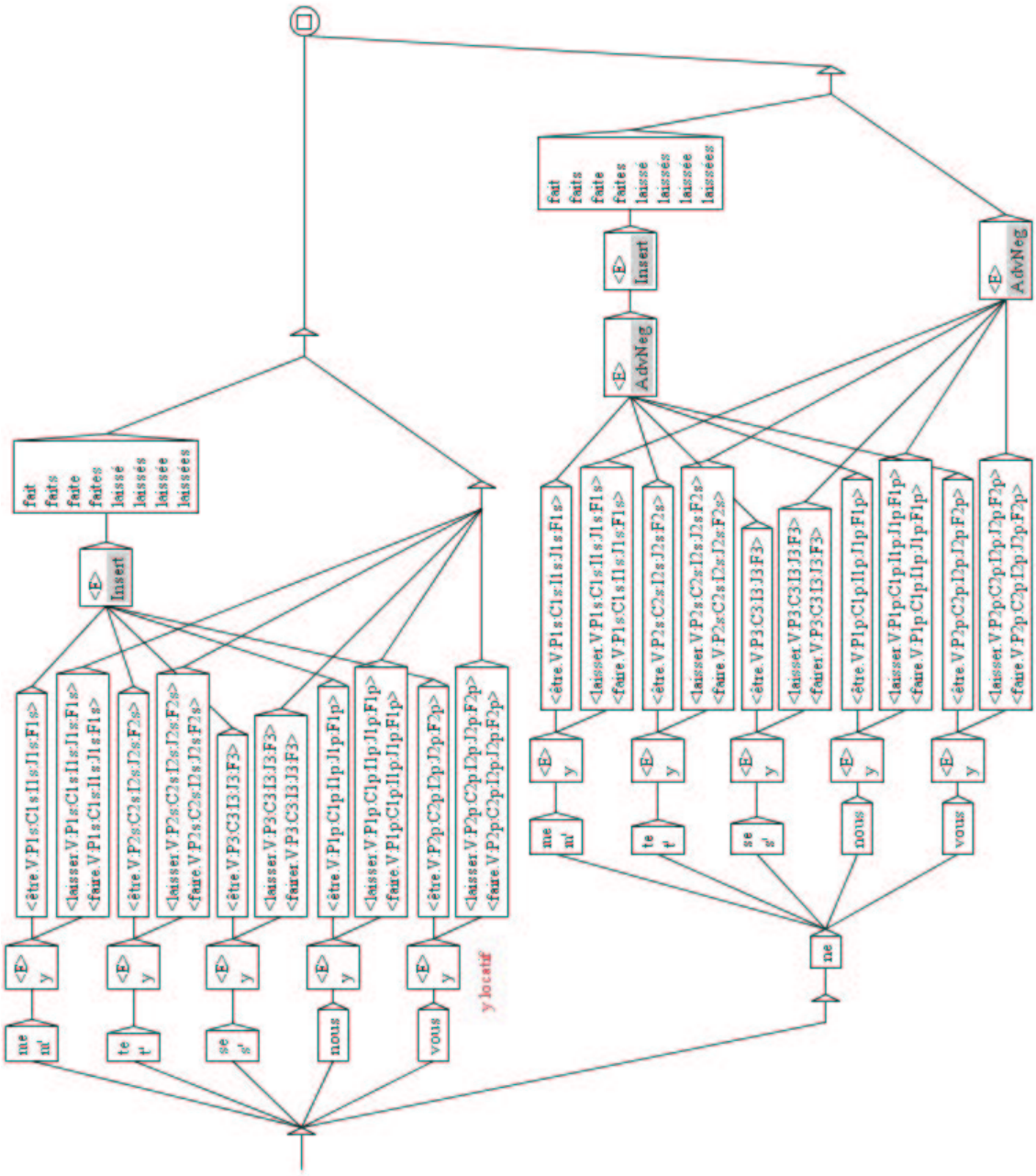

FIG. 1.39 – Graphe *SeLaisser*

Max ne peut pas être dénoncé

que des formes littéraires comme

la Lombardie pourrait n'être jamais envahie par ces barbares

voire même des formes très soutenues :

Cynthia aurait pu ne se point voir aimée de Steve

On peut donc constater que la finesse linguistique des tables de lexique-grammaire, combinée à la souplesse de description des graphes, permet un contrôle simple et puissant de la qualité des grammaires. Notre approche montre ici pleinement l'avantage qu'il y a à utiliser des grammaires de recherche de motifs pour faire de l'analyse syntaxique. En effet, des systèmes tels que HPSG, qui utilisent des formalismes plus complexes, n'ont pas cette souplesse qui nous permet d'adapter très simplement nos grammaires pour gérer des structures particulières telles que N_1 *ne se point voir Vpp de* N_0.

Comme nous l'avons expliqué précédemment, les constructions passives ont fait l'objet de grammaires séparées. La grammaire *super-N0VN1(Ppassif)* présentée sur la figure 1.41 montre l'ensemble des constructions déclaratives passives que nous avons décrites à partir de la construction de base N_0 V N_1. On peut y voir des appels à la grammaire des syntagmes verbaux passifs via le sous-graphe `Vpassif-@A` [13]. Les structures avec agents en *par*, *de* et *de ce que P* sont respectivement conditionnées par les variables *@BA*, *@BB* et *@BC* qui correspondent dans la super-table aux propriétés *[passif par]*, *[passif de]* et N_1 *être Vpp de ce que P*. Comme nous l'avons dit, l'omission de l'agent est toujours autorisée, ce qui explique que les chemins correspondants dans le graphe ne soient pas conditionnés par une variable. Nous pouvons également voir sur cette grammaire que nous ne traitons pas le passif uniquement sur la forme N_0 V N_1, mais également sur d'autres formes déclaratives, obtenues par extraction. Ces constructions seront examinées en détail dans la section 1.3.8.2. De même, les phrases passives pour les formes infinitives, interrogatives et impératives seront présentées dans les sections consacrées à ces constructions.

Nous n'avons considéré que des structures simples de passif mais qui, combinées avec d'autres transformations peuvent donner des phrases complexes telles que

n'est-ce pas ce tournevis qui a été aimanté par Max ?
que ce soit par Max qu'Ida soit fouettée !

Il ne s'agit toutefois que d'une première tentative d'analyse des structures passives, car il existe bien d'autres formes que nous n'avons pas traitées. Ainsi, il est montré dans [37] que *par les soins de* peut introduire un agent du passif dans certaines phrases telles que

ce texte a été révisé par les soins de Luc

[13] Nous expliquerons pourquoi le nom du sous-graphe est `Vpassif-@A` à la section 1.3.8.1

De même, [27] recense un certain nombre de formes passives que nous n'avons pas gérées. Parmi celles-ci, on rencontre par exemple des phrases issues d'extrapositions :

il a été insisté sur ce point (ε+par de nombreux orateurs)
$\Leftrightarrow$ *de nombreux orateurs ont insisté sur ce point*

Notre approche par graphes patrons permettrait d'analyser de telles constructions. L'exhaustivité de la description des formes passives est uniquement fonction de la précision des grammaires que l'on construit.

1.3.7.4 Pronominalisation

Nous allons aborder ici trois aspects du phénomène de pronominalisation. Le premier d'entre eux est la forme de pronominalisation la plus répandue : la pronominalisation d'un argument du verbe. Nous avons pris en compte 4 types de pronominalisations que nous avons codés dans la super-table :

$ppv = LE$:	*je mange ce gâteau*	$\Leftrightarrow$	*je le mange*
$ppv = LUI$:	*ils pardonnent à Luc*	$\Leftrightarrow$	*ils lui pardonnent*
$ppv = EN$:	*il prend de la drogue*	$\Leftrightarrow$	*il en prend*
$ppv = Y$:	*nous pensons déménager*	$\Leftrightarrow$	*nous y pensons*

Contrairement aux formes EN et Y qui sont invariables, les formes LE et LUI admettent des variantes : LE admet *le, l', la* et *les* ; LUI admet *lui* et *leur*. De plus, ces deux formes ont en commun la possibilité d'être remplacées par des pronoms personnels réfléchis : *me, m', te, t', se, s', nous* et *vous*, ce qui donne des phrases telles que :

nous nous mangerons
ils se pardonnèrent

Notons que lorsqu'un verbe est employé avec un pronom réfléchi, son auxiliaire devient *être*. Ainsi, alors que l'on a

je vous ai apporté des bières

il est impossible d'obtenir la phrase

**je m'ai apporté des bières*

La phrase correcte est

je me suis apporté des bières

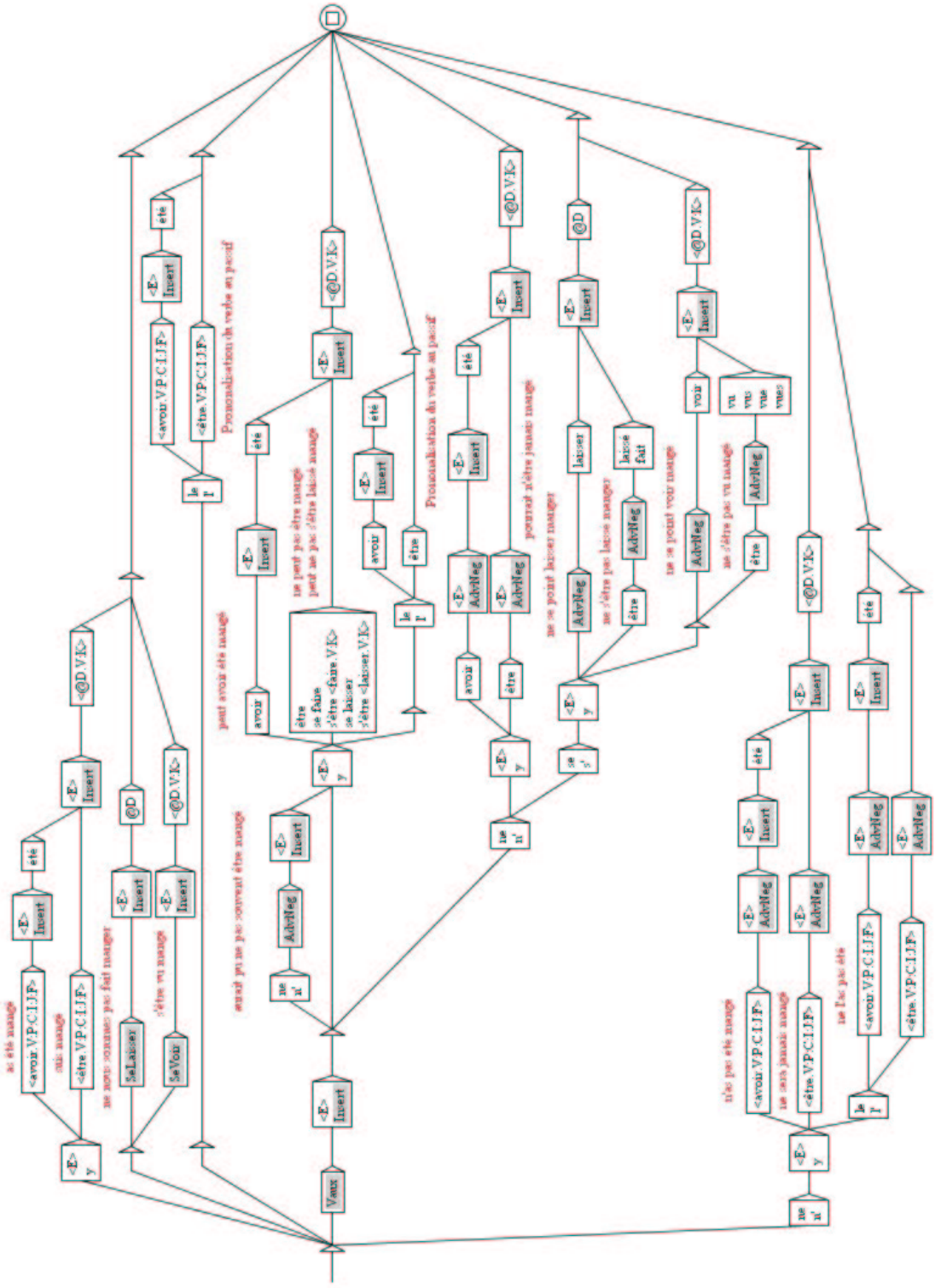

FIG. 1.40 – Graphe générique *super-Vpassif*

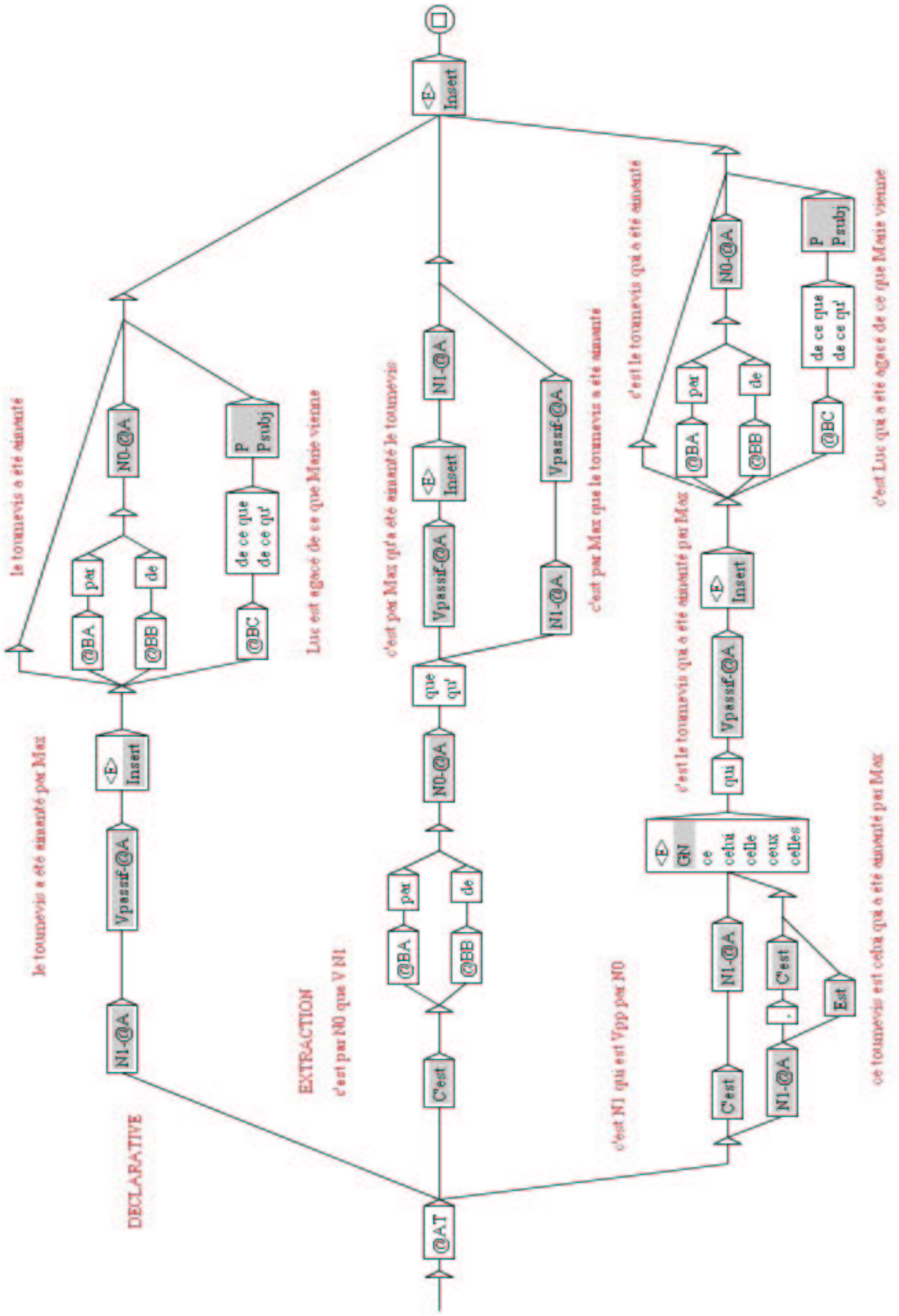

FIG. 1.41 – Graphe générique *super-N0VN1(Ppassif)*

Ceci est un phénomène très régulier. Cela affecte aussi bien les pronoms correspondant à des datifs lexicaux, c'est-à-dire les datifs qui correspondent à une phrase où le référent est explicité :

je me suis apporté une bière ⇔ *?j'ai apporté une bière à moi*

que les datifs étendus pour lesquels le pronom n'est pas la réduction d'un complément essentiel du verbe (voir [52]) :

Paul s'est construit une armoire
Léa s'est sifflé un pichet de vin

Étant donné que ce phénomène survient plutôt à l'oral qu'à l'écrit, nous ne l'avons pas traité. Cependant, il serait tout à fait possible de le faire, à condition de coder au préalable dans les tables les verbes qui tolèrent cette construction. Notons que dans des textes familiers ou oraux, on peut également trouver un datif éthique, c'est à dire un pronom qui fait référence à un élément extérieur au procès :

ce type te descendrait une bouteille de gnôle sans broncher

Le pronom *te* ne se rapporte ici ni au verbe, ni à l'un de ses compléments ; il apporte une valeur emphatique à la phrase en prenant à témoin l'interlocuteur. Nous n'avons pas traité ce type de pronom.

La pronominalisation d'un argument du verbe nécessite la description d'un syntagme verbal particulier. En effet, si l'on se contentait d'autoriser les pronoms personnels dans le syntagme verbal, la grammaire reconnaîtrait des phrases incorrectes comme :

**Paul la regarde Léa*
**Luc en reprend de la tarte*

Pour éviter cela, nous avons construit une grammaire nommée *super-V[N1=ppv]* décrivant un syntagme verbal dans lequel se trouve un pronom qui représente l'argument N_1. Une fois que l'on dispose d'une telle grammaire, il suffit d'ajouter un chemin $N_0\ V_{[N_1=ppv]}$ à la grammaire décrivant les constructions basées sur la structure $N_0\ V\ N_1$. Le grammaire obtenue peut alors analyser les phrases où l'argument N_1 est pronominalisé. Le procédé peut facilement être étendu aux pronominalisations des autres arguments ; il suffit de décrire les syntagmes verbaux correspondant (*super-V[N2=ppv]*, *super-V[N1=ppv,N2=ppv]*, ...) et d'intégrer dans les grammaires les constructions correspondantes ($N_0\ V_{[N_2=ppv]}\ N_1$, $N_0\ V_{[N_1=ppv,N_2=ppv]}$, ...).

La figure 1.42 montre le graphe *super-V[N1=ppv]*. Ce graphe est appelé dans la grammaire *super-N0VN1(P)* présentée sur la figure 1.43. On peut voir sur ce graphe le chemin décrivant la forme $N_0\ V_{[N_1=ppv]}$.

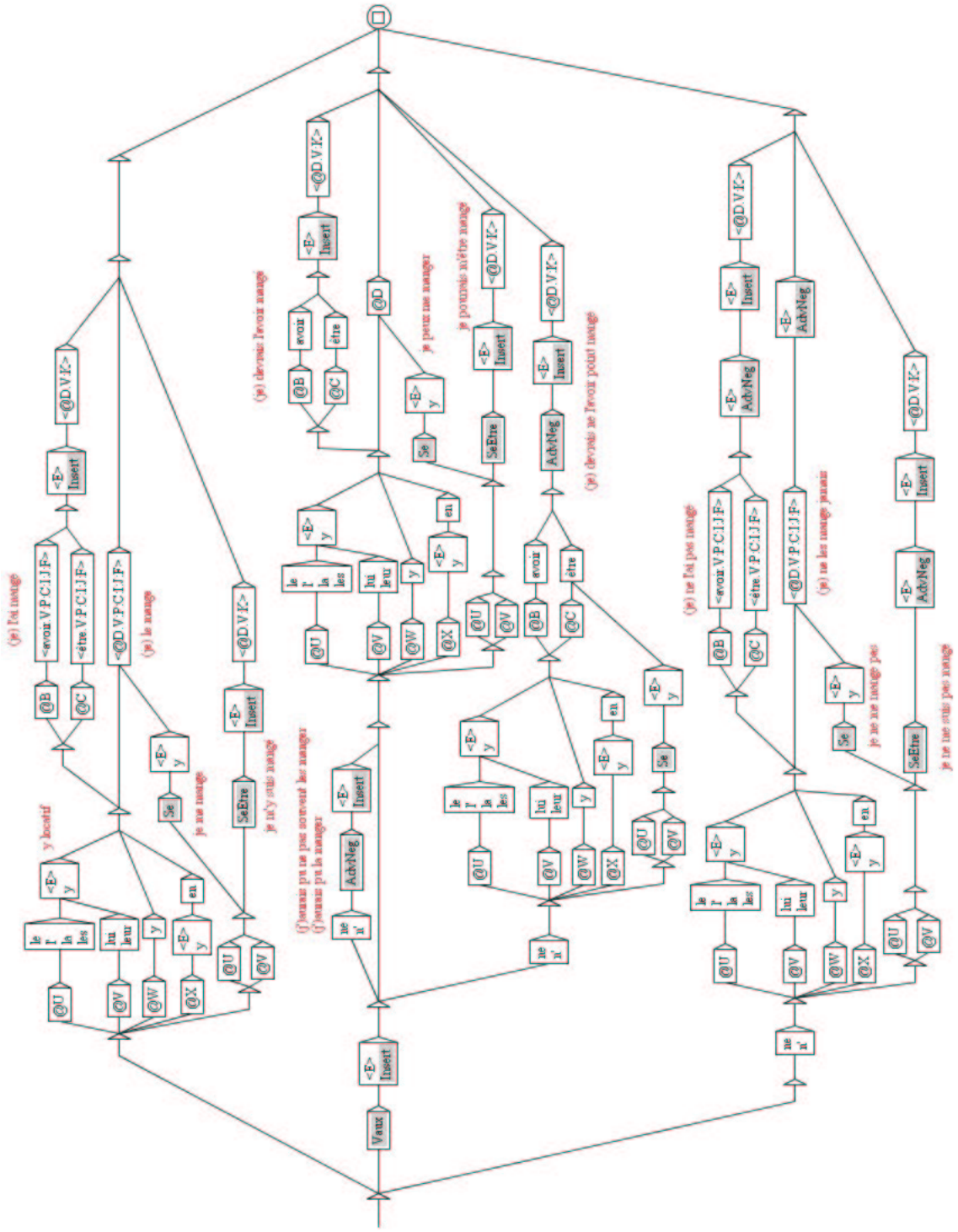

FIG. 1.42 – Graphe générique *super-V[N1=ppv]*

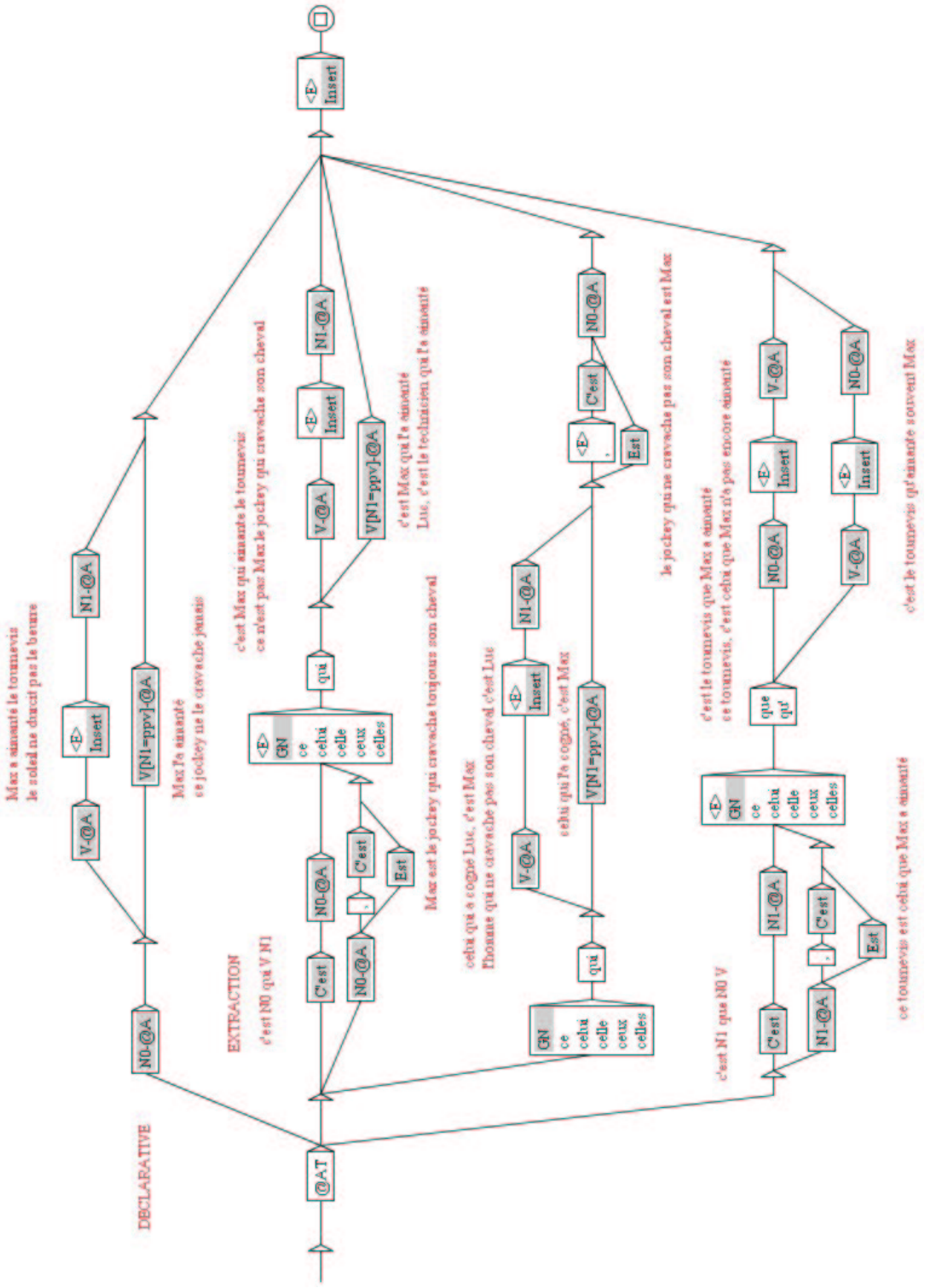

FIG. 1.43 – Graphe générique *super-N0VN1(P)*

Le deuxième type de pronominalisation que nous avons traité concerne la réduction en *y* de compléments non essentiels de lieu. Ces compléments peuvent apparaître dans presque toutes les phrases. En voici un exemple :

Paul a donné un bouquet à Léa dans le jardin ⇔ *Paul y a donné un bouquet à Léa*

Ce phénomène peut se combiner avec les pronominalisations normales :

Paul lui y a donné un bouquet
Paul le lui y a donné

Toutefois, un blocage intervient lorsque la phrase contient une autre pronominalisation en *y* :

Paul réfléchit à ce problème dans le jardin
Paul y réfléchit à ce problème
Paul y réfléchit dans le jardin
**Paul y y réfléchit*

Nous n'avons donc pas autorisé la présence simultanée dans nos grammaires de deux pronoms *y*. La même restriction s'applique d'ailleurs à la préposition *en* :

Max sort des fiches du tiroir
Max en sort des fiches
Max en sort du tiroir
**Max en en sort*

Notons cependant que la pronominalisation en *en* ne peut affecter que des arguments du verbe, ce qui la distingue de celle en *y*.

En raison de la généralité du phénomène, nous avons donc autorisé dans nos grammaires toutes les combinaisons de *y* avec d'autres pronoms, à l'exception de *y y* comme nous l'avons indiqué plus haut. Cette pronominalisation d'un complément locatif non essentiel se retrouve dans tous les graphes décrivant des syntagmes verbaux. On peut donc la voir, entre autres, sur les graphes *super-V*, *super-Vpassif* et *super-V[N1=ppv]* présentés respectivement sur les figures 1.38, 1.40 et 1.42.

Le dernier cas de pronominalisation que nous avons traité est celui de la pronominalisation du verbe au passif, et plus généralement, de l'adjectif employé avec un verbe support. En effet, on peut observer une pronominalisation en *le* du verbe principal :

ce tournevis a été aimanté par Max ⇔ *ce tournevis l'a été par Max*

Notons que pour être naturelle, cette construction nécessite toujours que l'on dispose d'un contexte :

Vous dites que votre site est visité par les étudiants ? Le nôtre l'est cent fois par jour !

Les pronominalisations de verbe au passif sont détaillées dans le graphe des syntagmes verbaux passifs, présenté sur la figure 1.40.

1.3.7.5 Problème des insertions

Une des caractéristiques des adverbes est qu'ils peuvent s'insérer à presque toutes les positions dans une phrase, comme le montrent les exemples suivants :

récemment, Paul a offert une bague à Léa
Paul, récemment, a offert une bague à Léa
Paul a récemment offert une bague à Léa
Paul a offert récemment une bague à Léa
Paul a offert une bague récemment à Léa
Paul a offert une bague à Léa récemment

Il n'est cependant pas possible d'insérer un adverbe à n'importe quelle position. Il est par exemple interdit d'effectuer des insertions entre une particule préverbale et le verbe :

**Paul lui récemment a offert une bague*

De même, on ne peut pas insérer d'adverbe entre *à* et N_1 [14] dans une construction N_0 *V à* N_1 :

**Paul parle à récemment Léa*

Ces restrictions nous amènent à la conclusion qu'il faut décrire précisément dans nos grammaires les emplacements auxquels peuvent intervenir des insertions.

Nous avons étendu nos insertions à des expressions plus complexes que des adverbes constitués de mots simples. Pour cela, nous avons adopté la définition d'*adverbe généralisé* donnée dans [36]. Nous regroupons ainsi sous le terme adverbe :
- adverbes simples : *récemment*, *hier* ;
- adverbes composés : *à ces mots*, *tôt ou tard* ;
- compléments circonstanciels : *dans une heure et douze minutes*, *dans la rue St Paul* ;
- propositions subordonnées circonstancielles : *puisque vous le dites.*

Nous avons également pris en compte la possibilité d'insérer des incises, afin de pouvoir analyser des phrases telles que :

Tout va bien, a affirmé le porte-parole de l'entreprise
le butin s'évaluerait, a-t-il déclaré, à plusieurs milliers d'euros

Pour cela, nous nous basons sur la description des incises par grammaires locales donnée par Cédrick Fairon dans [25].

[14] Sauf lorsqu'il s'agit d'un prédéterminant : *Paul parle à environ cent personnes.*

Notre description des insertions s'appuie donc sur une grammaire comprenant les différentes formes d'adverbes que nous avons autorisées, ainsi qu'un appel à la grammaire décrivant les incises. Notre grammaire, nommée *Insert*, est présentée sur la figure 1.44.

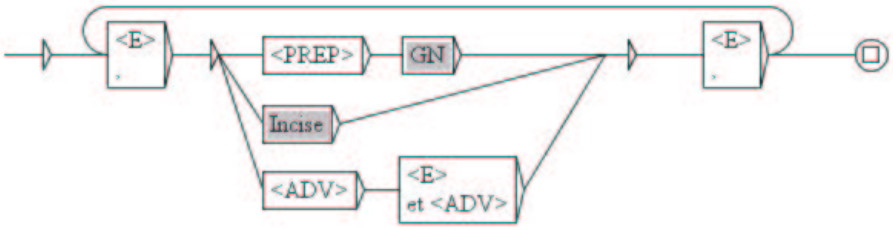

FIG. 1.44 – Grammaire des insertions *Insert*

Les symboles `<PREP>` et `<ADV>` reconnaissent respectivement les prépositions et adverbes recensés dans les dictionnaires électroniques. Cela inclut aussi bien les formes simples que les formes composées, de sorte que `<ADV>` peut reconnaître des adverbes tels que *tôt ou tard*. Le chemin reconnaissant une préposition suivie par un groupe nominal est une approximation grossière des adverbes libres tels que *à environ deux heures de marche* ou *depuis cet endroit*.

Cette grammaire est utilisée dans la plupart des graphes patrons : à chaque position où une insertion peut apparaître, on place une boîte contenant l'appel au sous-graphe `Insert` ainsi que le mot vide, car les insertions sont facultatives.

Notre approche des insertions est donc purement distributionnelle, car nous ne tenons pas compte des coréférences pouvant exister entre les insertions et les autres éléments de la phrase. De même, nous n'avons pas abordé le problème de la portée des adverbes. Cette position nous distingue de Roche qui discute dans [68] de l'opposition entre analyses transformationnelle et distributionnelle pour le problème des insertions. Il met en lumière le fait que certains adverbes ne sont pas indépendants des phrases dans lesquelles ils s'insèrent à travers l'exemple suivant :

Luc a continué de s'avancer, sur ses gardes
**Luc a continué de s'avancer, sur mes gardes*

Roche souligne que, pour rendre compte des blocages de ce type, il est nécessaire d'expliciter les coréférences entre les adverbes et les éléments sur lesquels ils portent. Il précise également que pour de telles phrases, seule une analyse transformationnelle permet de rendre compte des structures sous-jacentes. Ainsi, selon Roche, l'exemple ci-dessus devrait s'analyser en :

Luc a continué de s'avancer. Luc est sur ses gardes

Nous partageons ces conclusions, mais nous n'avons pas cherché à les prendre en compte dans nos grammaires, car nous ne sommes pas en mesure de gérer ces problèmes de coréférences dans l'état actuel de nos connaissances (ces problèmes seront discutés dans la section suivante). Nos grammaires peuvent donc reconnaître des phrases telles que

**Luc a continué de s'avancer, sur mes gardes*

De même, si nous pouvons localiser des insertions, nous ne gérons aucunement la portée des adverbes. Nos grammaires ne différencient donc pas les adverbes des phrases suivantes :

Max est parti de son plein gré
Max est parti sur les chapeaux de roue

En conclusion, notre grammaire des insertions ne résout pas les problèmes de l'analyse des adverbes, mais permet uniquement d'en tenir compte de façon à ne pas bloquer l'analyse de phrases contenant des insertions. L'analyse de la structure et de la portée des adverbes reste donc un domaine très ouvert dans lequel, à l'exception de celle présentée dans [25], peu d'expériences pratiques ont été menées.

1.3.7.6 Problèmes de coréférence

L'analyse d'une phrase nécessite de vérifier certaines coréférences, dont les plus élémentaires sont les accords en genre et en nombre, ainsi que les accords sujet-verbe. Faute de respecter ces règles, nous ne pourrions pas rejeter des phrases incorrectes comme :

**la maisons est jolie*
**Paul irez se promener*

Toutefois, le lecteur pourra être surpris de ce que nous n'ayons tenu compte de telles règles dans nos grammaires. Nous avons ainsi défini de manière indépendante les grammaires décrivant les groupes nominaux et celles décrivant des syntagmes verbaux, ce qui fait que nos grammaires peuvent accepter des phrases erronées. Deux raisons motivent notre décision.

Tout d'abord, les contraintes de coréférences ne sont pas toujours claires. Si l'on considère le cas de l'accord sujet-verbe, on peut se trouver face à des situations ambiguës telles que :

un grand nombre de personnes (prend+prennent) le métro
*un grand nombre de gens (*tond sa+tondent leur) pelouse) le dimanche*

Ces exemples de sujets collectifs nous montrent qu'il n'est pas toujours aisé de formuler les contraintes d'accord entre le sujet et le verbe. Ce problème de décidabilité se pose également pour d'autres types de coréférences. Nous avons vu plus haut que la phrase

**Luc a continué de s'avancer, sur mes gardes*

devait être rejetée car *mes* ne se rapporte pas au sujet. Ici, la situation semble claire car la coréférence se traduit par une modification d'un adjectif possessif. Cependant, il ne s'agit que d'un cas particulier, car la coréférence peut s'exprimer de façon purement sémantique, et ainsi être syntaxiquement neutre. Considérons la phrase suivante :

César a gracié le gladiateur d'un signe du pouce

d'un signe du pouce est un adverbe qui est coréférent au sujet et au verbe, ce qui explique l'inacceptabilité de la phrase :

**la rivière a débordé de son lit d'un signe du pouce*

Les règles qui régissent cet adverbe ne sont pas formalisables de manière triviale. Il semble nécessaire que le sujet possède un pouce pour que la phrase puisse être acceptable. Cette première condition pourrait sans doute être résolue par une contrainte $N_0 = hum$. Toutefois, il resterait à vérifier que l'action désignée par le verbe soit compatible avec cet adverbe, faute de quoi, on pourrait accepter des phrases telles que :

**César rêve à Cléopâtre d'un signe du pouce*

Nous voyons donc que, dans le cas général, les coréférences portant sur des adverbes ne peuvent être traitées, à moins que l'on ne dispose d'un codage précis des combinaisons autorisées. Malheureusement, ce problème rejoint celui de la description des arguments *V-n* (voir section 1.3.6.2), et nous ne pensons pas qu'il soit possible d'y apporter une réponse dans le cas général.

Cependant, même dans les cas où l'on dispose de contraintes claires, nous ne pensons pas qu'il soit toujours judicieux de les expliciter dans les grammaires, à moins qu'elles ne puissent être décrites de façon strictement locale. Ainsi, nous prenons en compte des contraintes telles que l'impossibilité de reconnaître deux pronoms *y* successifs, ou la restriction qui porte sur l'auxiliaire de conjugaison. Pour des contraintes plus complexes à vérifier, comme c'est le cas de l'accord sujet-verbe, il nous semble plus raisonnable de traiter la question d'une façon détournée. Dans [71], Senellart propose d'intégrer les contraintes d'accord dans les grammaires en les notant sous forme de formules logiques placées dans les sorties associées aux boîtes. Nous n'avons pas retenu cette solution car d'une part, elle complique le formalisme, ce que nous voulons éviter à tout prix, et d'autre part, parce que cette technique nous prive de l'usage des transductions, qui pourront nous être utiles, par exemple pour étiqueter des segments reconnus par les grammaires. Notre idée consiste à produire dans un premier temps toutes les analyses potentielles puis à vérifier a posteriori les contraintes que l'on souhaite tester. Considérons ainsi la phrase :

le gentil président la porte

En appliquant nos grammaires, nous obtenons les deux analyses suivantes :

(le gentil)$_{N_0}$ *(président)*$_V$ *(la porte)*$_{N_1}$
(le gentil président)$_{N_0}$ *(la)*$_{PRO}$ *(porte)*$_V$

Il est alors facile de vérifier l'accord sujet-verbe puisque les syntagmes de la phrase sont délimités pour chaque analyse. On peut donc aisément supprimer la première analyse qui ne satisfait pas cette contrainte. Cette approche est basée sur le fait qu'une phrase possède rarement un grand nombre d'analyses potentielles. En effet, la plupart des ambiguïtés que l'on rencontre en ne considérant que peu de mots disparaissent lorsque l'on considère la phrase

entière. On retrouve ici un principe très général : l'ambiguïté diminue lorsque le contexte augmente. À titre d'exemple, nous avons indiqué sur la figure 1.45 l'automate qui rend compte de toutes les ambiguïtés lexicales de la phrase : *le gentil président la porte*. Nous pouvons constater qu'il y a 108 découpages possibles pour cette séquence, mais que seules les 2 analyses décrites plus haut correspondent à des structures de phrases. Cet état de fait nous permet d'affirmer que la vérification de contraintes telles que les accords sujet-verbe n'aura qu'un rôle d'appoint et ne servira qu'à retirer qu'un petit nombre d'analyses parmi celles proposées. Ce point est important car il montre que le fait d'ignorer dans un premier temps les coréférences n'engendre pas une explosion du nombre d'analyses potentielles, ce qui indique que notre approche est viable.

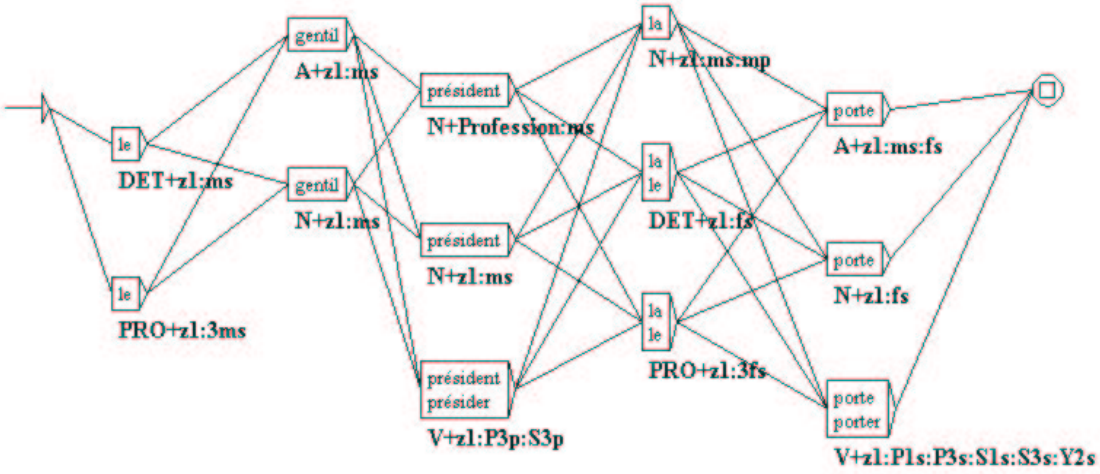

FIG. 1.45 – Ambiguïtés lexicales de la phrase *le gentil président la porte*

De telles vérifications pourront être effectuées à l'aide du programme de levée d'ambiguïtés ELAG ([51]), qui sera incorporé aux outils informatiques que nous utilisons pour appliquer nos grammaires. Pour cela, il suffirait de modifier nos grammaires pour quelles insèrent les limites potentielles des constituants dans leurs analyses. Il ne resterait alors plus qu'à ajouter ces limites de constituants aux symboles manipulables par ELAG, pour pouvoir écrire des règles testant la validité des coréférences. Cette méthode permet de préserver la simplicité d'écriture des grammaires, car les structures décrites peuvent être décomposées en sous-graphes. Comme nous pouvons le voir sur la figure 1.43, cela permet de conserver une grande lisibilité, car une structure telle que N_0 V N_1 peut être décrite au moyen d'un chemin très simple appelant des sous-graphes `N0-xxx`, `V-xxx` et `N1-xxx`. Nous pouvons remarquer que cette méthode implique que l'analyse se déroule en plusieurs passes. L'organisation des différentes étapes de l'analyse sera présentée dans la section 2.5.

1.3.8 Génération et description des constructions

Après avoir présenté les principes et les différentes sous-grammaires que nous avons utilisés, nous allons maintenant détailler le contenu de nos grammaires principales. Nous commence-

rons par décrire l'étape de génération des graphes, puis nous commenterons les grammaires décrivant les formes déclaratives, infinitives, impératives et interrogatives pour les structures de base $N_0\ V$ et $N_0\ V\ N_1$.

1.3.8.1 Génération des graphes

Ainsi que nous l'avons expliqué en 1.3.4, nous générons automatiquement nos graphes en deux étapes :

1. génération des graphes de tables
2. génération des graphes d'entrées

Cette opération doit s'effectuer pour chaque graphe générique. Comme nous avons pu le voir, nous décrivons séparément les différents éléments de nos grammaires, en particulier les arguments du verbe. Il faut donc s'assurer que les grammaires d'arguments appelées dans la grammaire d'un verbe donné sont bien celles prévues pour ce verbe. Nous résolvons ce problème de sélection en mettant à profit la garantie d'unicité que nous offre la combinaison *(nom de table, numéro d'entrée)* (voir 1.3.4).

Le principe est de caractériser de façon unique les grammaires se rapportant à une entrée donnée, ce qui peut être fait en les nommant judicieusement. Une fois que l'on dispose de cette garantie, il suffit de faire en sorte que les grammaires d'une entrée X appellent bien les sous-grammaires correspondant à cette entrée.

Pour mettre cette idée en pratique, nous utilisons une caractéristique de notre mécanisme de génération de graphes. L'astuce est basée sur le fait que le programme `Table2Grf`, que nous utilisons pour générer les graphes, permet de paramétrer le nom des fichiers produits à l'aide des variables `@xxx`. En d'autres termes, si l'on fournit au programme le modèle de nom de fichier `graphe-@%.grf`, le programme va générer des graphes nommés `graphe-0001.grf`, `graphe-0002.grf`, etc. En plaçant dans la colonne A de la super-table le nom de chaque table suivi par `-@%`, nous pouvons ainsi générer des noms de grammaires uniques.

Considérons par exemple le graphe de la figure 1.46 :

FIG. 1.46 – Exemple de graphe générique

Si l'on décline ce graphe à l'aide de la super-table, on obtient pour l'entrée correspondant à la table 31H le graphe de la figure 1.47. On peut y voir que la chaine `A@` a été remplacée par `31H-@%`. La variable `@%` n'a pas été interprétée lors de cette étape, mais le sera lorsque l'on déclinera ce graphe pour la table 31H.

FIG. 1.47 – Graphe patron pour la table 31H

La figure 1.48 montre le graphe final que l'on obtient pour la 27e entrée de la table 31H. Si l'on a généré correctement les sous-graphes `N0-31H-0027` et `V-31H-0027`, les grammaires du sujet et du syntagme verbal se combineront pour analyser de façon correcte les constructions N_0 V du 27e verbe de la table 31H.

FIG. 1.48 – Graphe obtenue pour la 27e entrée de la table 31H

1.3.8.2 Phrases déclaratives

Nous avons regroupé différentes constructions dans nos grammaires de phrases déclaratives. Ainsi, en plus des phrases élémentaires telles que :

les étoiles ne brillent plus
Max regarde un film

nous avons pris en compte certaines formes obtenues par extraction. Ainsi, à partir d'une phrase élémentaire en N_0 V

Max agonise

nous obtenons plusieurs variantes, parmi lesquelles :

c'est N_0 GN qui V :	*c'est Max le patient qui agonise*
N_0 (c'est+est) GN qui V :	*Max est le patient qui agonise*
GN qui V, c'est N_0 :	*le patient qui agonise, c'est Max*

Lorsque l'on considère la structure de base N_0 V N_1, il est également possible de faire porter l'extraction sur N_1, ce qui donne des variantes telles que :

N_0 V N_1 :	*Max a aimanté ce tournevis*
N_1 (c'est+est) GN que N_0 V :	*ce tournevis est celui que Max a aimanté*
N_1 (c'est+est) GN que V N_0 :	*ce tournevis est celui qu'a aimanté Max*
c'est N_1 que N_0 V :	*c'est ce tournevis que Max a aimanté*
c'est N_1 que V N_0 :	*c'est ce tournevis qu'a aimanté Max*

Les figures 1.51 et 1.43 montrent respectivement les grammaires *super-N0V(P)* et *super-N0VN1(P)*. On peut y voir que nous utilisons les deux sous-graphes `Est` et `C'est`. Ces graphes sont présentés sur les figures 1.49 et 1.50.

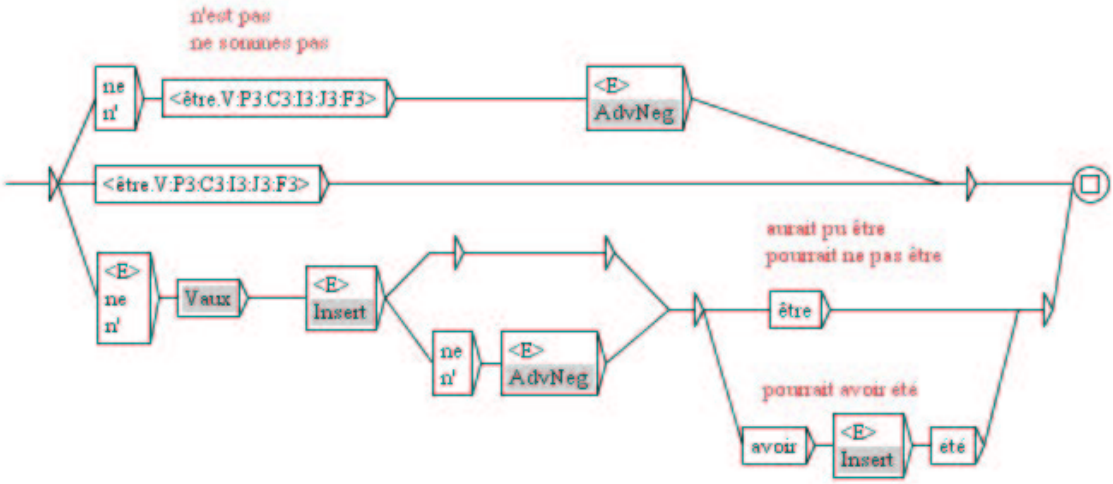

FIG. 1.49 – Graphe *Est*

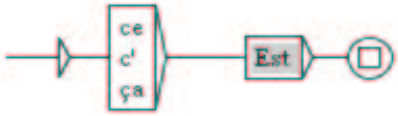

FIG. 1.50 – Graphe *C'est*

En plus de ces formes déclaratives, nous disposons de la grammaire *super-N0VN1(Ppassif)*, déjà présentée en figure 1.41, qui décrit les formes de phrases passives dérivées de la structure N_0 V N_1. Notons que les variantes par extraction sont également décrites pour ces constructions, ce qui permet de reconnaître des phrases telles que

ce tournevis est celui qui a été aimanté par Max
c'est par Max que ce tournevis a été aimanté

En considérant toutes les combinaisons de transformations possibles, on peut donc analyser des phrases relativement complexes telles que :

ce n'est pas ce tournevis celui qui devait y être aimanté
c'est par Max que ce tournevis l'a été

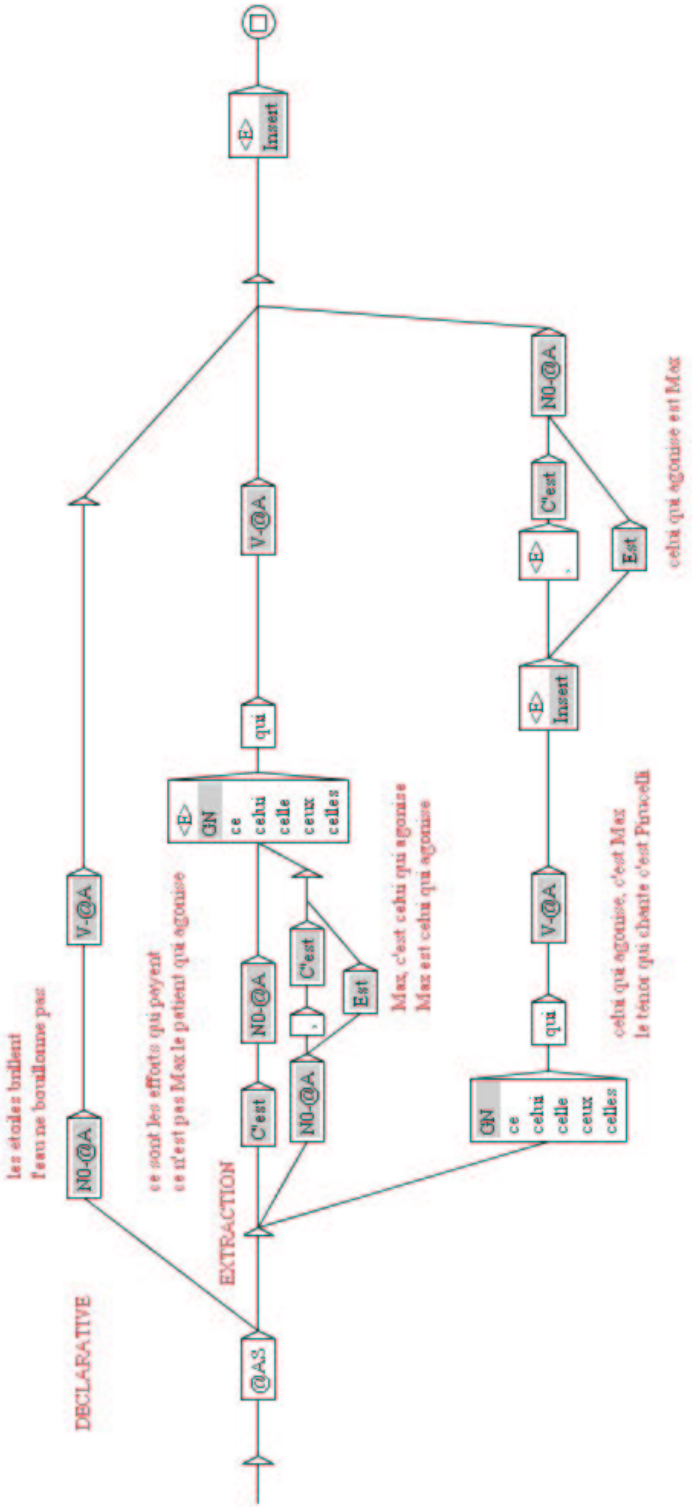

FIG. 1.51 – Graphe générique *super-N0V(P)*

1.3.8.3 Infinitives

Les infinitives dérivées de la construction N_0 V sont décrites par le graphe *super-N0V(Vinf)*, présenté sur la figure 1.11 en page 39. Ce graphe fait appel aux sous-graphes `AdvNeg` (présenté en figure 1.37) et `Vaux_inf`. Ce dernier est une variante du graphe *Vaux* qui reconnaît des séquences d'auxiliaires verbaux à l'infinitif. Ce graphe décrit les infinitives présentes (*dormir*) et passées (*avoir dormi*). Il est également tenu compte de la possibilité de trouver un complément locatif pronominalisé en *y*. En combinant ces transformations, on peut analyser des phrases telles que :

s'empresser de dormir
ne pas y avoir dormi

Les infinitives à la voix active dérivées des formes N_0 V N_1 sont présentées sur le graphe de la figure 1.52. En plus des transformations décrites précédemment, ce graphe décrit la possibilité de pronominaliser le complément N_1. Voici quelques exemples d'infinitives analysables avec cette grammaire :

n'y avoir pas pensé
renoncer à manger des fruits
veiller à ne pas le perturber
se vanter d'y avoir rencontré Marie

Le graphe de la figure 1.53 décrit les infinitives passives dérivées des constructions en N_0 V N_1. Voici quelques exemples de constructions reconnues par ce graphe :

avoir été engueulé par Paul
y être choisi
espérer ne pas être désigné
ne l'avoir jamais été par Max

1.3.8.4 Impératives

Nous avons décrit deux formes de constructions impératives. Dans la première de ces formes, le verbe principal est conjugué à l'impératif et un groupe nominal vocatif, souvent coréférent au sujet [15], est facultatif :

(ε+soldats,) brûlez ces maisons !

Cette forme est compatible avec les transformations par négation, pronominalisation et passivation. On peut ainsi analyser des phrases telles que :

mangez-la
ne soyez pas piégés par Luc
ne le croyez pas !

[15] À la première personne du pluriel, la coréférence n'est pas stricte : *Paul, allons dehors.*

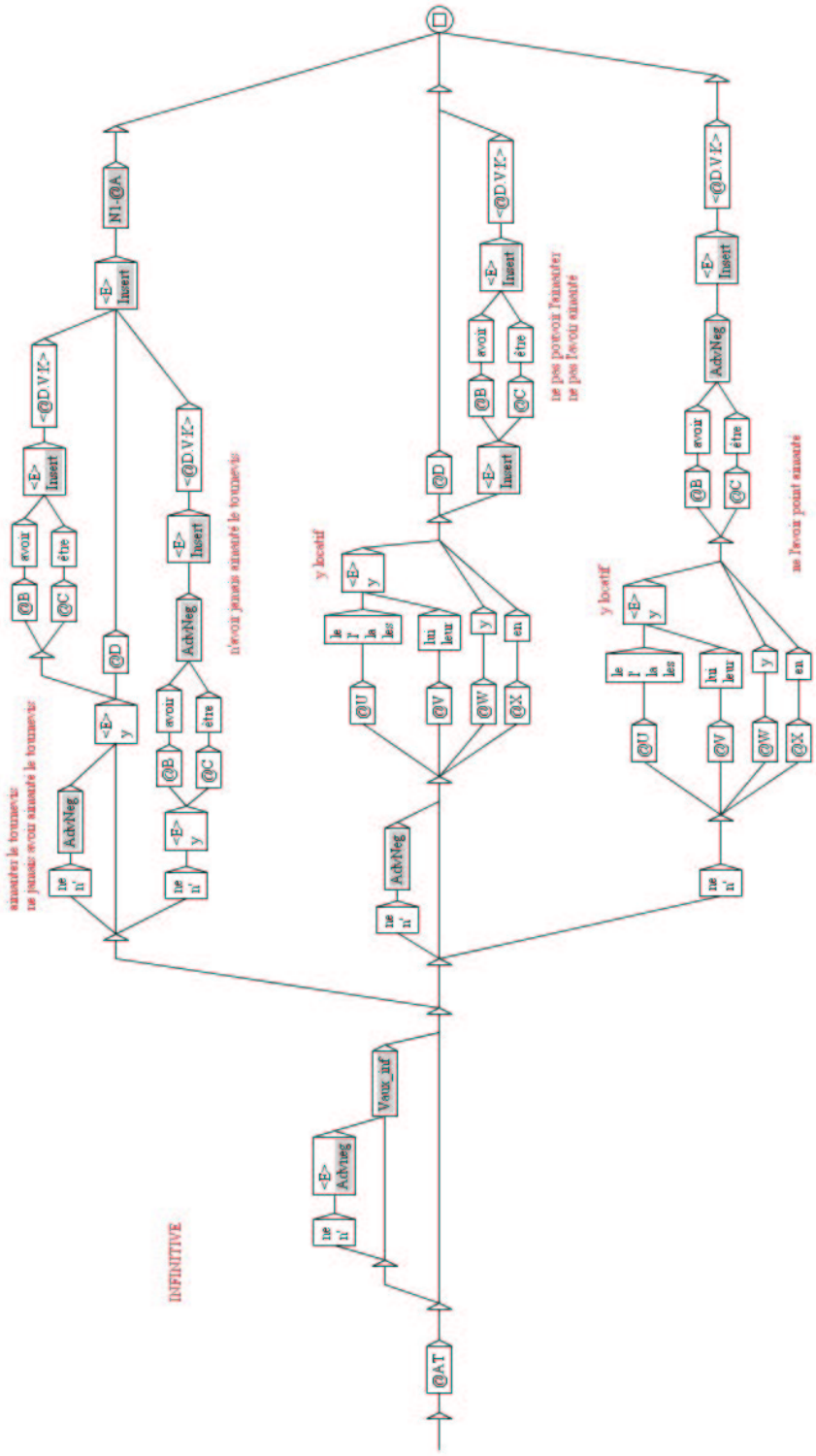

FIG. 1.52 – Graphe générique *super-N0VN1(Vinf)*

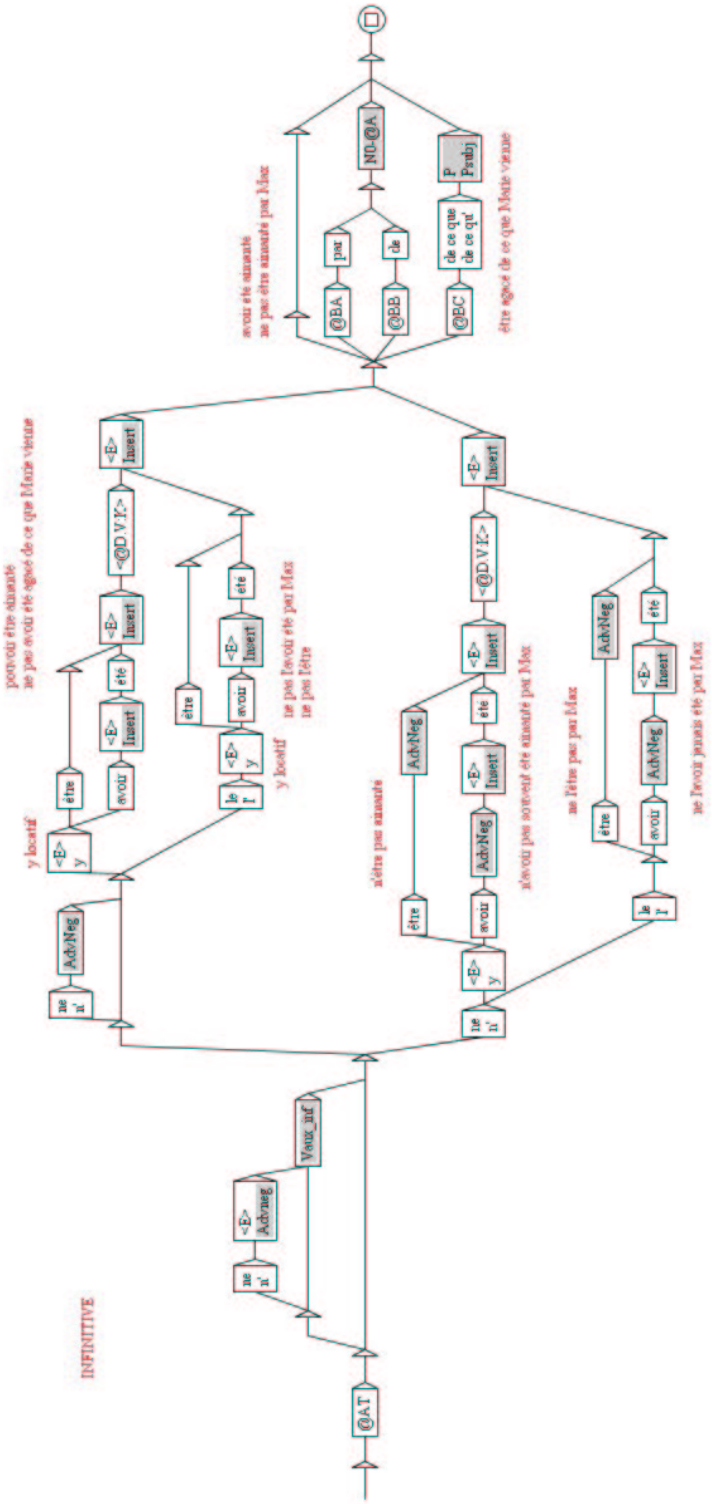

FIG. 1.53 – Graphe générique *super-N0VN1(Vinf_passif)*

Nous n'avons pas décrit dans nos grammaires des impératifs avec sujet :

**les soldats, brûlez la maison ⇔ les soldats brûlent la maison*

Dans les langues à cas, le groupe nominal vocatif est au cas vocatif. En français, il peut être marqué par l'absence de déterminant ou l'emploi d'une interjection :

soldats, brûlez cette maison !
eh vous, les soldats, brûlez cette maison !

Toutefois, on peut rencontrer des phrases impératives dans lesquelles ce groupe nominal a un déterminant mais est sans interjection [16] :

les enfants, rangez vos chambres !
les mecs, venez ici !

Étant donné que nous ne disposons pas d'une description précise des constraintes mises en jeu, nous n'avons pas traité ce phénomène dans nos grammaires.

La seconde forme de construction impérative que nous avons traitée concerne les phrases au subjonctif précédées de *que* :

qu'ils le lisent !
que les plus lents partent les premiers !

Il semble que toutes les formes de phrases déclaratives, une fois mises au subjonctif, permettent de construire des phrases impératives. Ainsi, pouvons obtenir des phrases impératives dérivées à partir d'extractions, de phrases passives, de phrases avec pronominalisation, etc :

que ce soit Max qui mange ce gâteau !
que cette annonce soit lue par tous !
qu'elle le soit par tout le monde !
?que ce soit Max qui ait pris la clé

Nous avons donc décrit simplement ces formes en faisant appel aux graphes *super-N0V(Psubj)*, *super-N0VN1(Psubj)* et *super-N0VN1(Psubj_passif)* qui décrivent les mêmes séquences que les grammaires de formes déclaratives, mais dans lesquelles le verbe principal est au subjonctif. La figure 1.57 montre le graphe *super-N0VN1(Psubj)*. On peut y voir des appels aux sous-graphes `Vsubj-@A`, `Vsubj[N1=ppv]-@A`, `C'est_subj` et `Est_subj`, qui sont les variantes au subjonctif des graphes *V-@A*, *V[N1=ppv]-@A*, *C'est* et *Est*.

Nous avons également tenu compte de l'inversion du sujet, qui peut intervenir dans des phrases comme :

que les enfants chantent !

[16] Il semble que dans ce cas-là, ce soit la virgule qui joue le rôle d'interjection.

que chantent les enfants !

Comme cela n'apparaît pas dans les constructions *Psubj*, nous avons ajouté à nos grammaires des chemins permettant de reconnaître ces variantes. Notons que le sujet semble être rejeté à la fin de la phrase lorsque le verbe admet plusieurs arguments :

que revêtent leurs armures les chevaliers !
**que revêtent les chevaliers leurs armures !*

Nous n'avons pas décrit de restrictions sémantiques sur les *Psubj* pouvant entrer dans la formation de phrases à sens impératif, ce qui autorise des phrases telles que :

?pense venir demain !
?ressemble à ton frère !

Les constructions impératives que nous avons prises en compte sont décrites dans les graphes *super-N0V(Imper)*, *super-N0VN1(Imper)* et *super-N0VN1(Imper_passif)*, respectivement présentés sur les figures 1.54, 1.55 et 1.56.

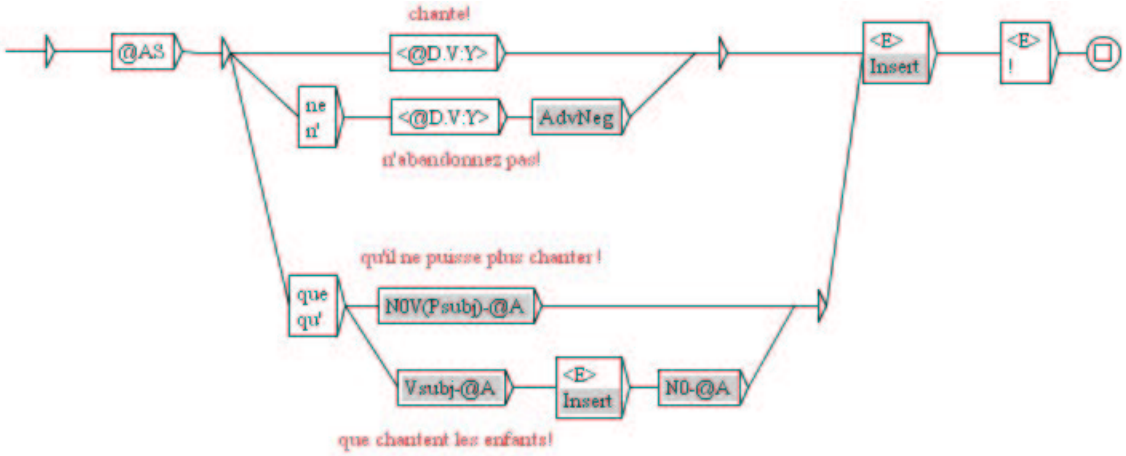

FIG. 1.54 – Graphe générique *super-N0V(Imper)*

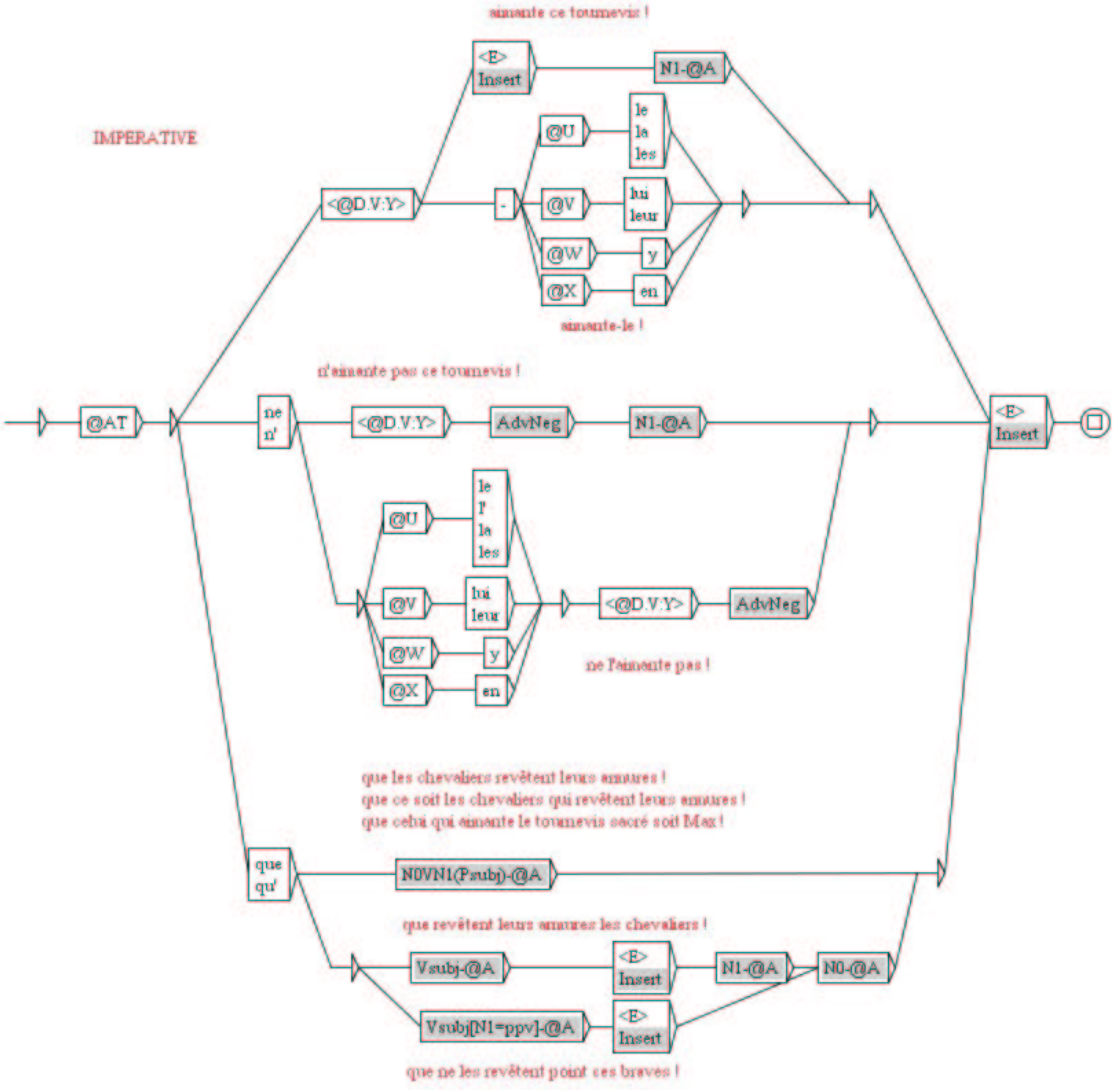

FIG. 1.55 – Graphe générique *super-N0VN1(Imper)*

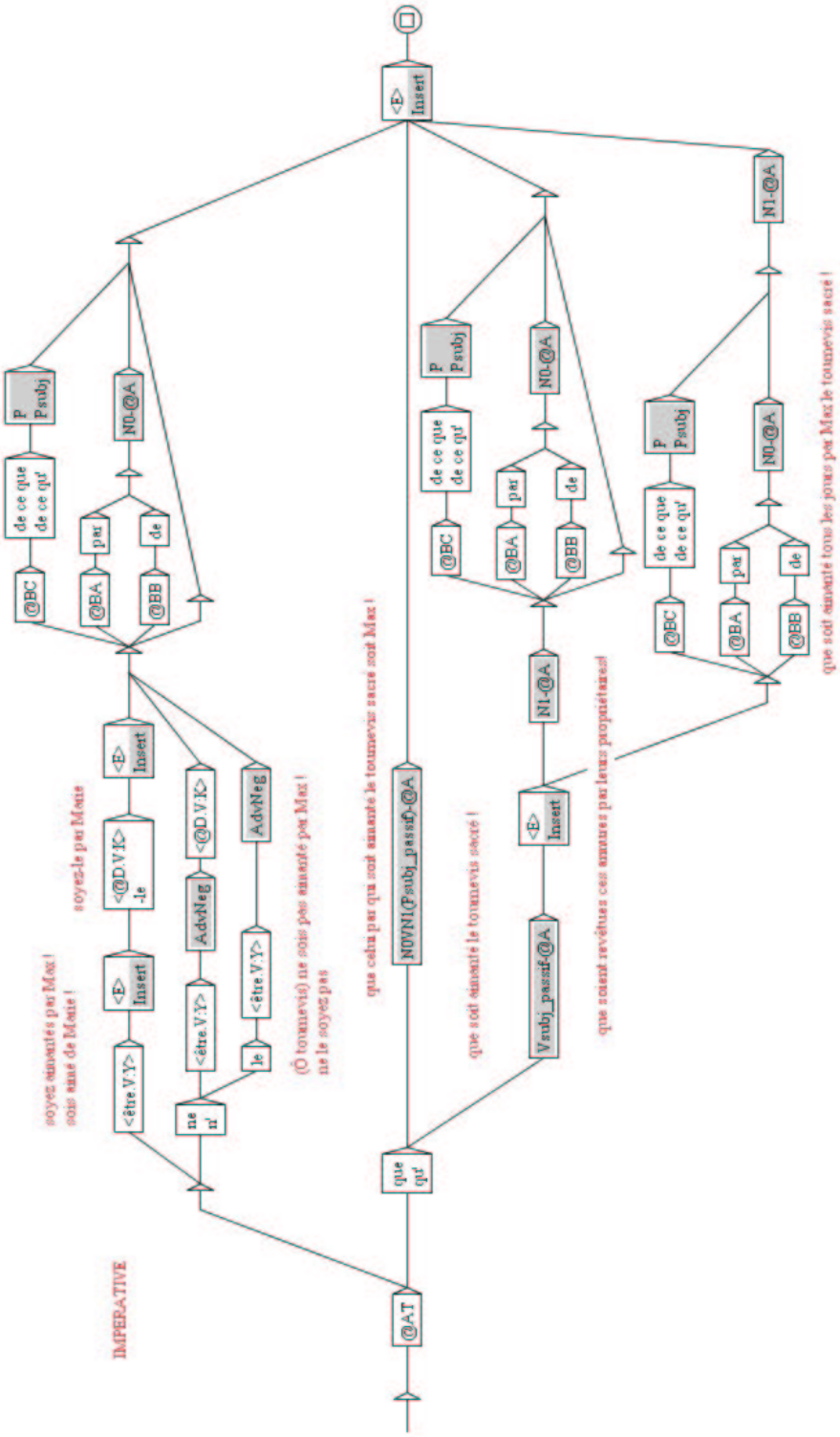

FIG. 1.56 – Graphe générique *super-N0VN1(Imper_passif)*

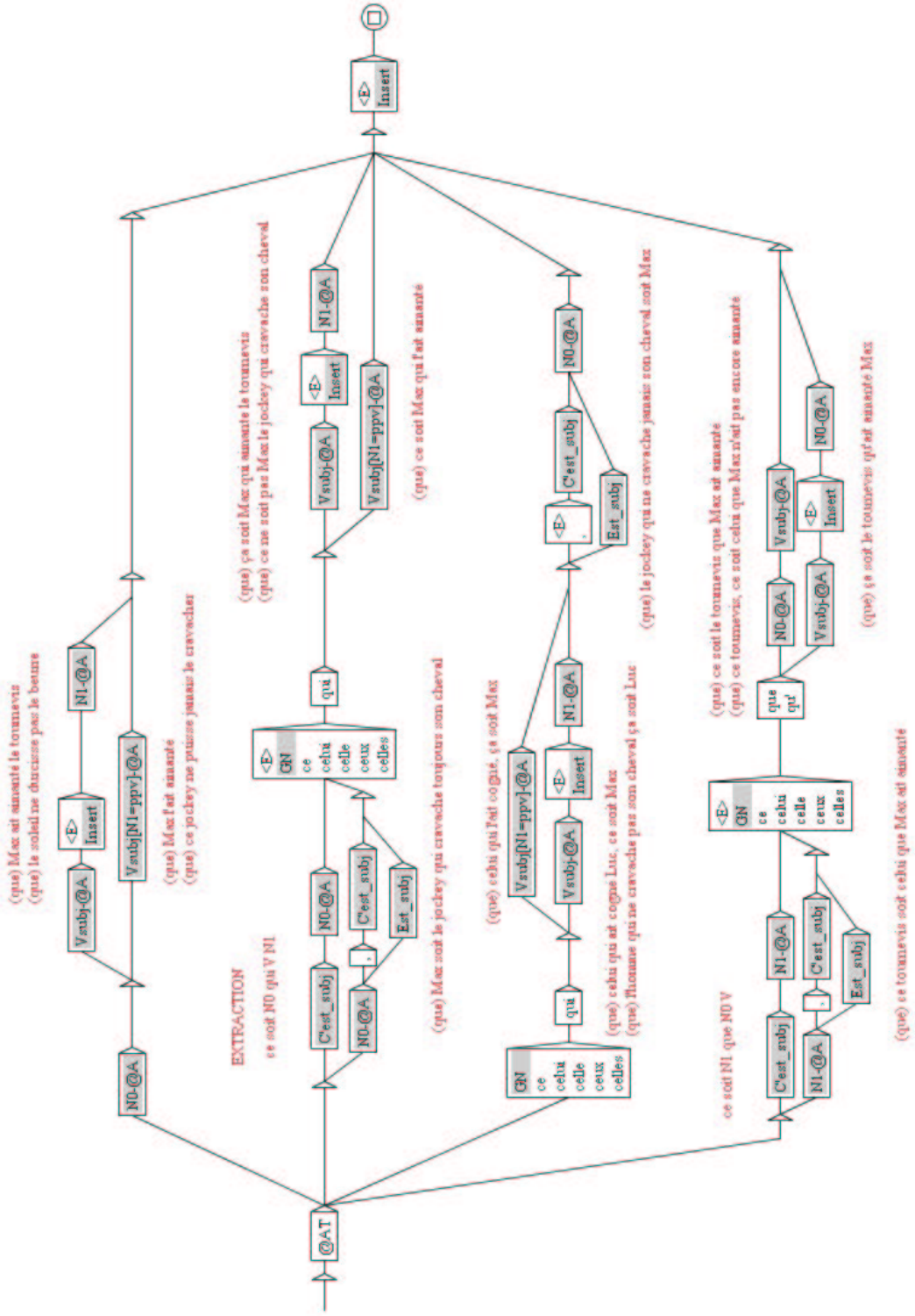

FIG. 1.57 – Graphe générique *super-N0VN1(Psubj)*

1.3.8.5 Propositions relatives et interrogatives

Les propositions relatives et les phrases interrogatives sont très semblables : ces deux structures extraient un élément de la phrase. Dans le cas d'une interrogative, cet élément devient un pronom interrogatif, tandis que dans le cas d'une relative, il devient le pivot de celle-ci. Cet élément peut être un complément essentiel ou non, et l'extraction peut porter sur la totalité du complément ou seulement sur une partie, comme le montrent les exemples suivants :

Luc lit un livre dans la chambre du haut

qui lit un livre dans la chambre du haut ?
... qui lit un livre dans la chambre du haut ...

où Luc lit-il un livre ?
... où Luc lit un livre ...

dans quelle chambre Luc lit-il un livre ?
... dans laquelle Luc lit un livre ...

Dans ce dernier exemple, seule une partie du complément est extraite, car *la chambre* ne fait pas partie de la relative.

Bien qu'il y ait quelques différences, ces structures présentent de telles similarités que nous pensons que la description précise de l'une des deux constructions devrait pouvoir être transposée aisément pour la seconde. Nous avons choisi de nous focaliser sur les phrases interrogatives, car elles présentent des particularités absentes des relatives telles que la répétition du sujet ou son inversion lorsqu'il est pronominalisé. En d'autres termes, il nous a semblé que la description des interrogatives était plus complexe que celle des relatives, et donc qu'il serait plus facile d'adapter les grammaires dans le sens *Interrogatives* $\rightarrow$ *Relatives*.

Comme nous venons de le voir, l'interrogation peut porter soit sur un complément, soit sur une partie d'un complément. Lorsque c'est la totalité d'un complément qui est concernée, la description est simple car c'est un pronom interrogatif simple qui est utilisé. Le problème se limite donc à choisir le bon pronom de façon à éviter d'analyser des phrases telles que :

**qui butinent les abeilles ?*
**que discute avec Paul ?*

Le problème se complique lorsque la focalisation porte sur une partie d'un complément. En effet, il faut pouvoir déterminer quelles parties des compléments peuvent être concernées et de quelles façons elles se pronominalisent, ce qui n'est pas facile comme le montre l'exemple suivant :

il a volé les bijoux de la reine d'Angleterre
*les bijoux (de qui+de quelle reine+de la reine de quel pays+*de la reine d'où) a-t-il volé ?*

Ce problème est lié à celui de la description des groupes nominaux. Nous pensons que le seul moyen de le résoudre serait de décrire les groupes nominaux dont un des éléments est

pronominalisé. Ainsi, si nous disposions d'une grammaire $GN_{interrogatif}$ capable de reconnaître les expressions *(quels bijoux)+(les bijoux (de qui+de quelle reine+de la reine de quel pays))*, la description des phrases interrogatives ci-dessus se résumerait à la structure suivante :

(que+$GN_{interrogatif}$*) V-t-il ?*

Étant donné que nous ne disposons pour l'instant d'aucune description de tels groupes nominaux, nous avons laissé ces cas de côté dans nos grammaires.

Nous avons décrit trois types de questions :

1. questions portant sur toute la phrase
2. questions portant sur un argument du verbe
3. questions portant sur un complément non essentiel

Le premier type regroupe les questions pour lesquelles la réponse peut être *oui* ou *non*, ce qui inclut les questions sur les formes obtenues par extraction dans *c'est ... (que+qui)*. Cette première catégorie regroupe des phrases telles que :

a-t-il chanté ?
Luc chante-t-il ?
est-ce Luc qui chante ?
est-ce que Luc chante ?

Le second type englobe les questions pour lesquelles la réponse est un argument du verbe. Cela concerne les questions en *qui, que, qui est-ce que, par qui*, etc. Voici quelques exemples de telles questions :

qu'a-t-il mangé ?
qu'est-ce que Luc a mangé ?
qui est-ce qui a mangé ce gâteau ?
par qui ce gâteau a-t-il été mangé ?

La dernière catégorie comprend les questions auxquelles la réponse est un complément non essentiel. Nous n'avons pris en compte que les questions élémentaires en *où, pourquoi, quand* et *comment*. Toutefois, nous pourrions aisément étendre nos grammaires si l'on disposait d'une grammaire des adverbes interrogatifs reconnaissant des expressions telles que *à quelle heure, à combien de kilomètres d'ici, pour quelle raison*, etc. La construction d'une telle grammaire rejoindrait celle d'une grammaire des groupes nominaux interrogatifs évoquée plus haut.

La figure 1.58 montre le graphe des phrases interrogatives dérivées de la structure N_0 *V*. Nous avons restreint les questions en *(qui+qui est-ce qui)* en utilisant la variable @E qui renvoie à la propriété $N_0 = Nhum$. Cela nous permet d'éviter de reconnaître des phrases telles que :

**qui contient de l'huile ?*

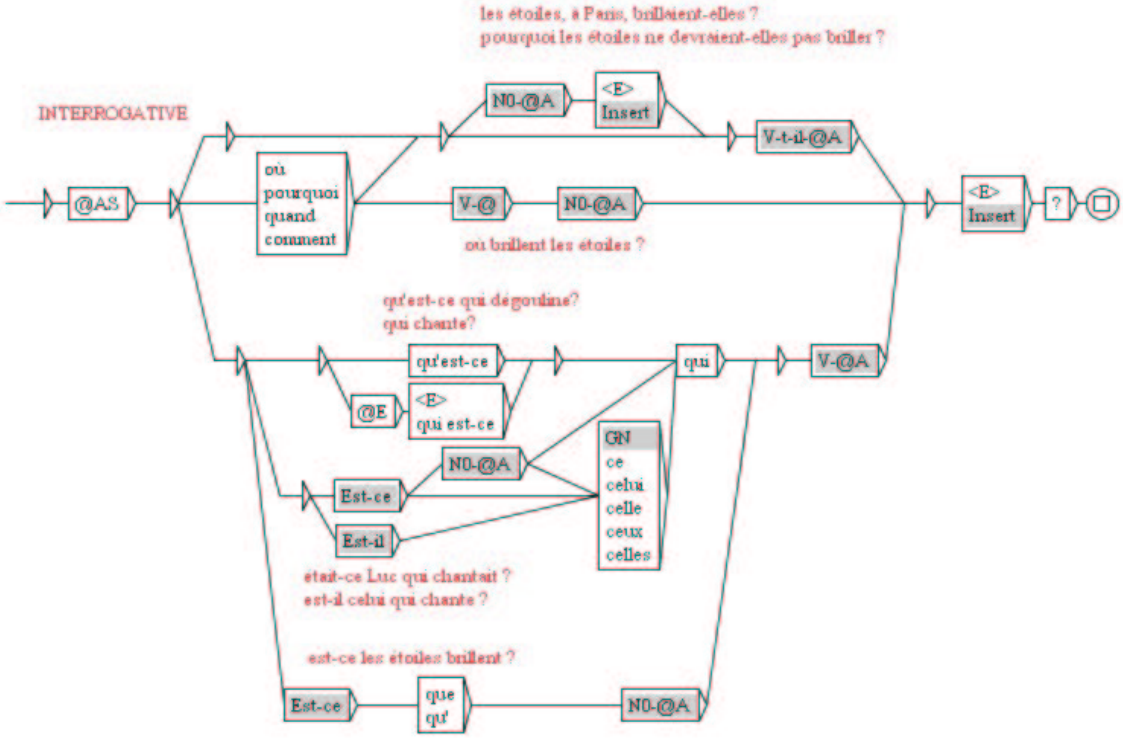

FIG. 1.58 – Graphe générique *super-N0V(Interro)*

On peut voir sur ce graphe que nous utilisons le sous-graphe `V-t-il-@A`. Le graphe générique correspondant, présenté sur la figure 1.61, décrit les syntagmes verbaux où le sujet est pronominalisé et placé après le verbe. Les figures 1.62 et 1.63 décrivent les variantes de ce graphe qui reconnaissent les syntagmes verbaux interrogatifs pour le passif, et ceux traitant la pronominalisation de l'argument N_1. Nous utilisons également les graphes `Est-ce` et `Est-il` qui sont présentés sur les figures 1.59 et 1.60.

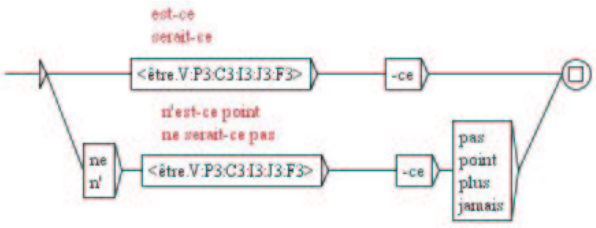

FIG. 1.59 – Graphe *Est-ce*

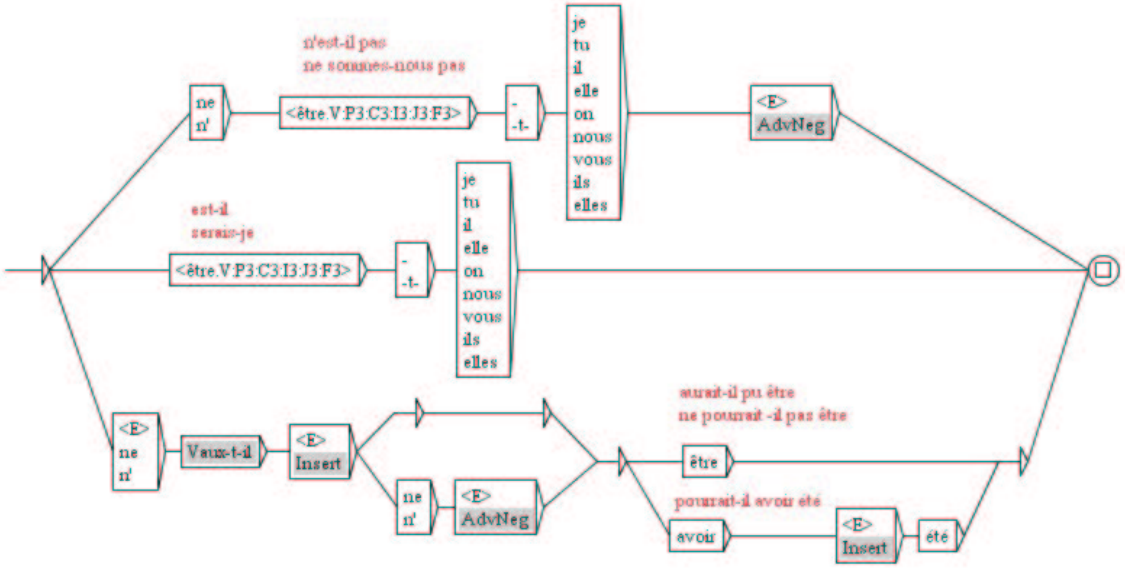

FIG. 1.60 – Graphe *Est-il*

Les graphes *super-N0VN1(Interro)* et *super-N0VN1(Interro_passif)* décrivent les phrases interrogatives actives et passives pour la structure N_0 V N_1. Ces graphes, présentés sur les figures 1.64 et 1.65, permettent d'analyser des phrases telles que :

où Max a-t-il trouvé ce livre ?
Luc l'a-t-il mangé ?
pourquoi aurait-il été pris par Léa ?
qui n'a pas été agacé par Paul ?
ce technicien, est-ce celui par qui le système devait être contrôlé ?
n'était-ce point d'Iseult que Tristan était aimé ?

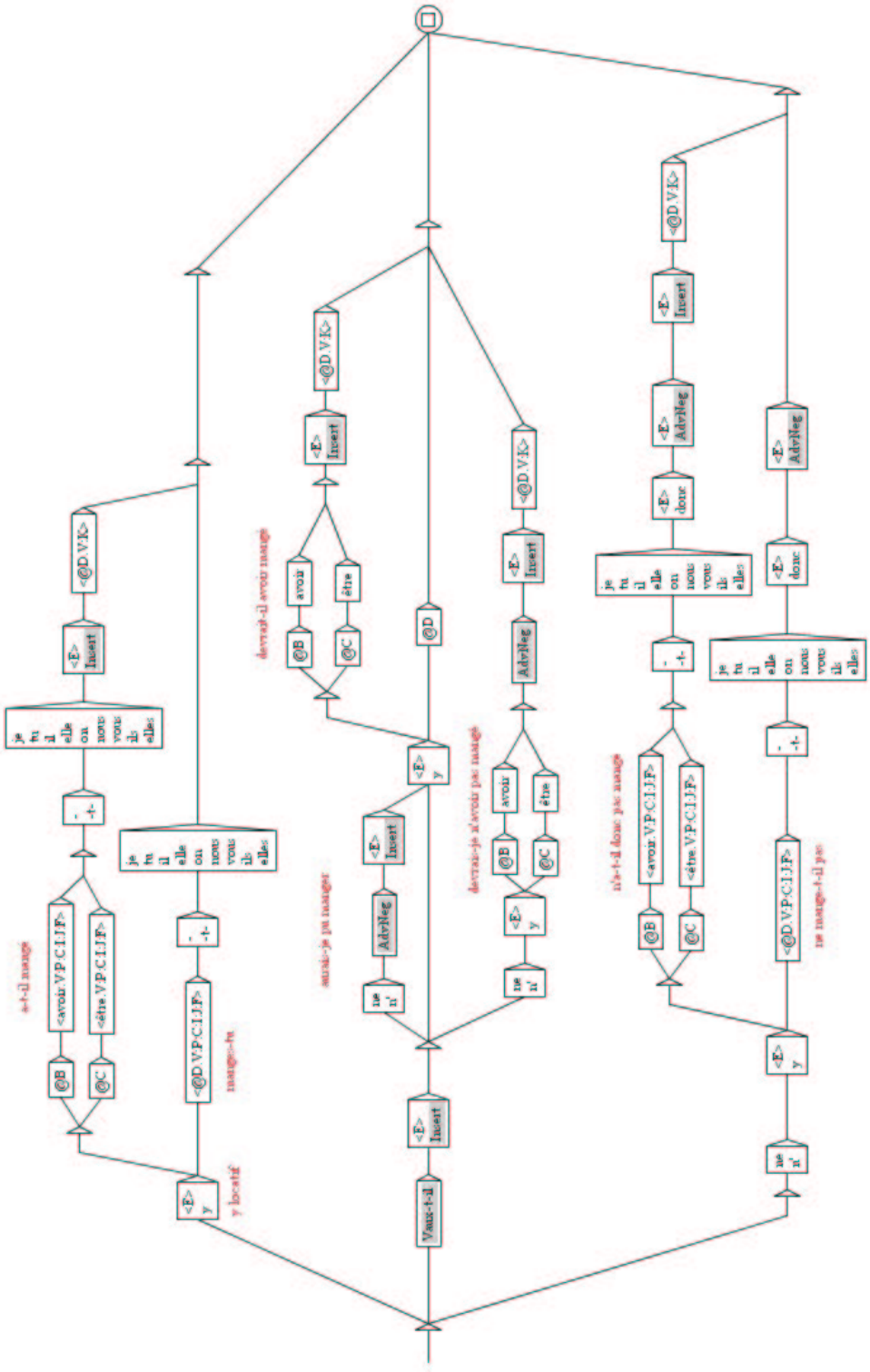

FIG. 1.61 – Graphe générique *super-V-t-il*

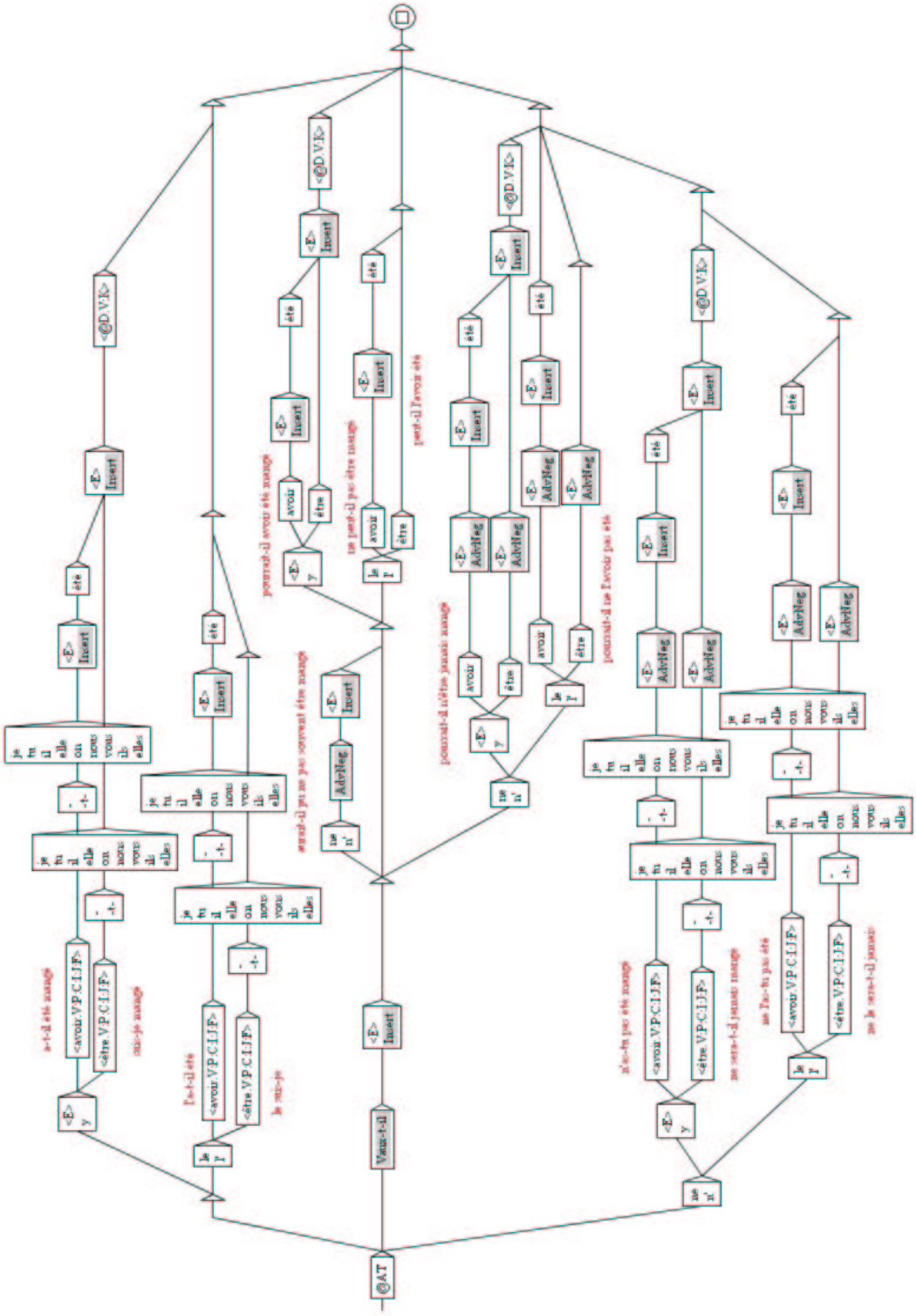

FIG. 1.62 – Graphe générique *super-Vpassif-t-il*

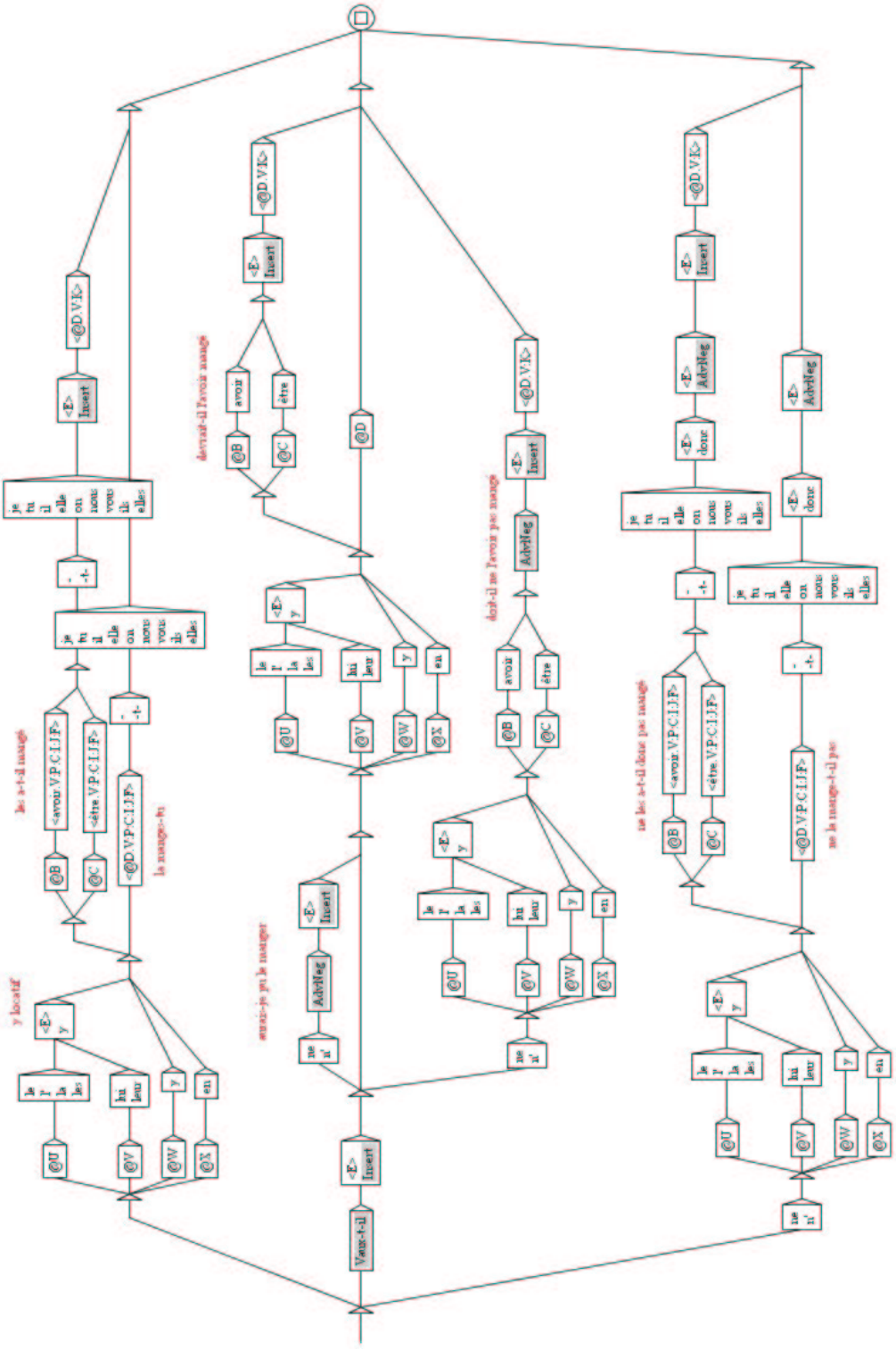

FIG. 1.63 – Graphe générique *super-V-t-il[N1_ppv]*

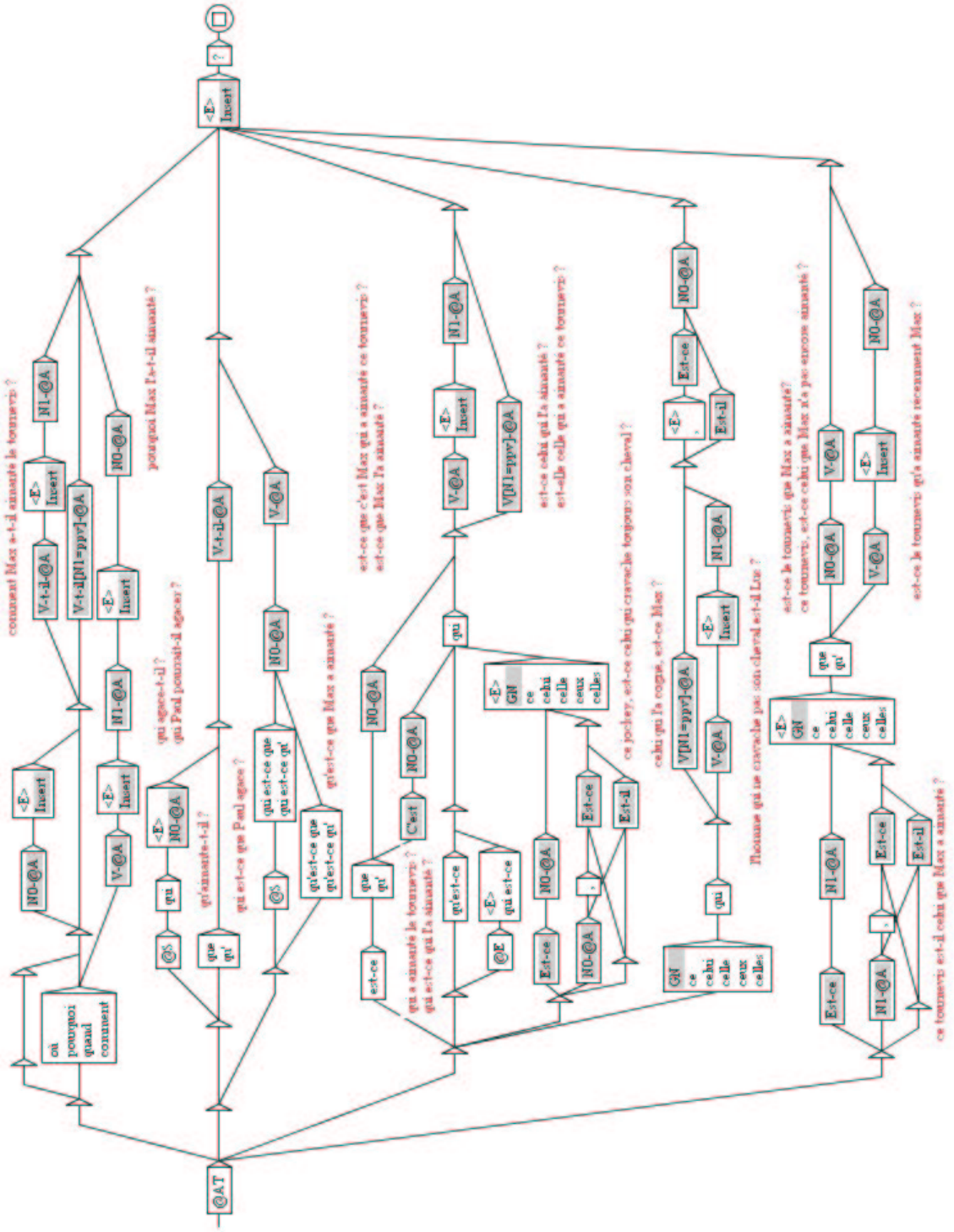

FIG. 1.64 – Graphe générique *super-N0VN1(Interro)*

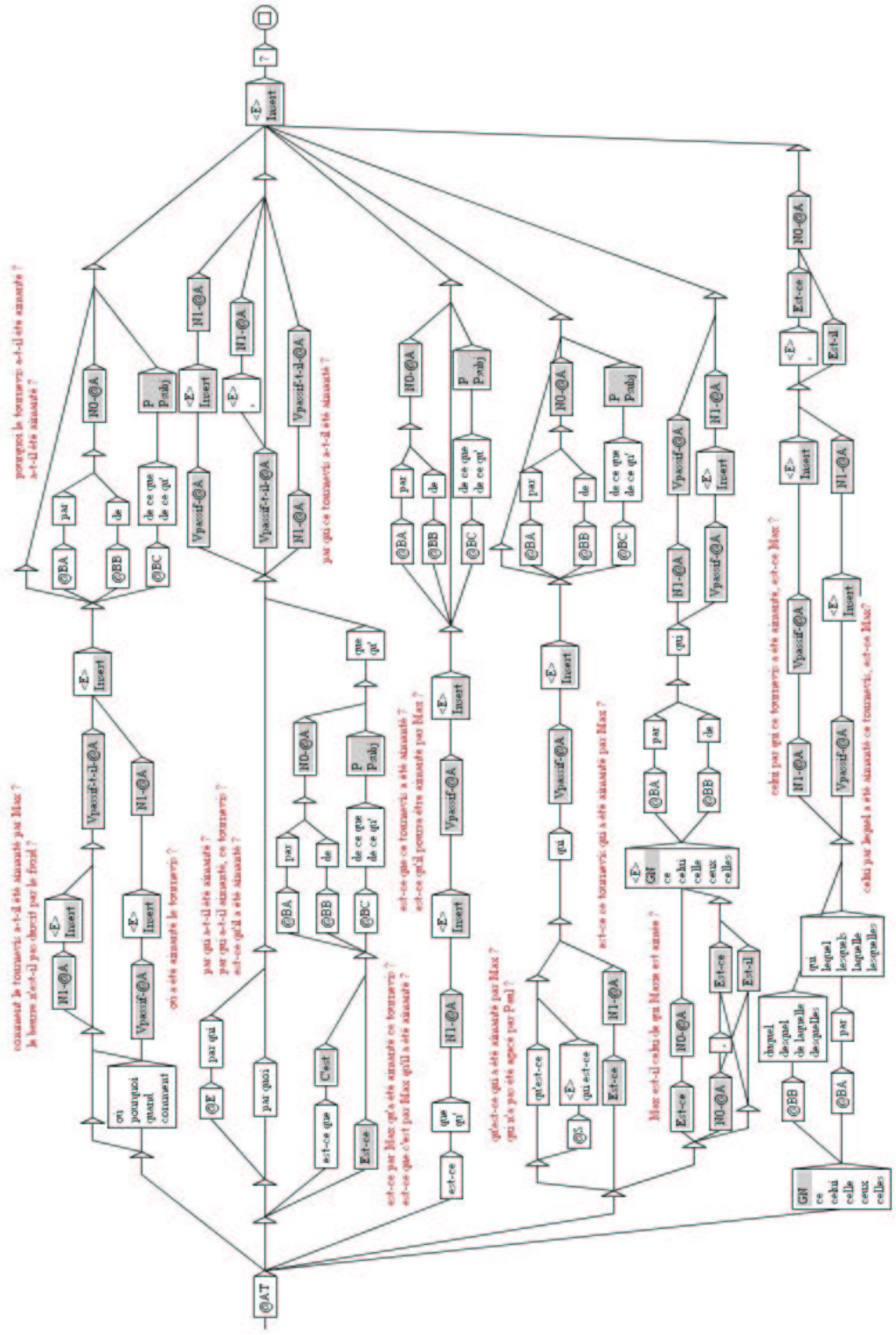

FIG. 1.65 – Graphe générique *super-N0VN1(Interro_passif)*

1.4 Conclusion

Des travaux antérieurs ont montré qu'il était tout à fait possible d'exploiter informatiquement les données du lexique-grammaire, en particulier à des fins d'analyse syntaxique automatique. Nous avons donc cherché une approche permettant d'envisager une exploitation exhaustive de ces données, ce qui pose des problèmes de maintenance sur une longue période de temps.

Pour cela, nous avons simplifié à l'extrême le formalisme de description de nos grammaires, ce qui nous a permis de séparer les données et les programmes destinés à les utiliser. Nous avons ensuite examiné diverses constructions de façon à tester la validité de notre modèle, et nous sommes arrivés à la conclusion que l'utilisation de graphes génériques et de grammaires locales permettait de décrire l'ensemble des constructions considérées. Toutefois, nous avons observé que, pour être tout à fait valide, ce modèle devait être complété par des vérifications d'accords, ce qui peut être fait à une étape ultérieure de l'analyse, au moyen d'un outil tel qu'ELAG. Ces grammaires étant élaborées manuellement, leur validité ne dépend que du soin apporté à leur conception, ce qui signifie que la précision des grammaires peut être affinée à volonté.

Nous avons donc obtenu un modèle permettant de décrire pratiquement les différentes constructions de phrases simples. Dans ce cadre, la description exhaustive des constructions du français consiste simplement à dessiner des grammaires et à compléter les tables du lexique-grammaire. Étant donné que nous avons dissocié les données des programmes, ces tâches de description pourront être effectuées par des linguistes n'ayant pas à se soucier des aspects pratiques de leur exploitation par programme.

Néanmoins, pour que cette solution soit entièrement viable, il faut s'assurer que ces données peuvent être utilisées pratiquement. Nous avons généré 98000 graphes à partir des graphes génériques que nous avons présentés, ce qui représente 290 méga-octets. Sachant qu'à long terme, ces données vont s'accroître considérablement, on peut se demander s'il est envisageable de traiter de telles quantités d'informations en temps raisonnable. Les solutions que nous avons étudiées pour remédier à ce problème sont présentées au chapitre suivant.

Chapitre 2

Outils informatiques

Toute chose est nombre.
Pythagore

Comme nous l'avons présenté dans le chapitre précédent, notre but est de faire de l'analyse syntaxique exacte en utilisant des descriptions linguistiques précises représentées par des grammaires. Notre analyseur syntaxique se constitue donc de ces grammaires et des programmes destinés à les appliquer. L'objet de ce chapitre est de présenter les différentes méthodes que nous avons envisagées pour exploiter ces grammaires.

Nous présenterons tout d'abord le fonctionnement de notre programme de recherche de motifs, et nous montrerons tout le bénéfice que l'on peut retirer du principe de découpage en unités lexicales sur lequel nos méthodes reposent. Nous discuterons ensuite de deux techniques d'optimisation basées sur des transformations appliquées aux grammaires. Nous montrerons comment il est possible d'améliorer l'application des grammaires en découpant l'analyse en plusieurs passes. Enfin, nous évoquerons une piste de recherche basée sur une génération des grammaires en fonction du lexique présent dans le texte. Nous conclurons ce chapitre par une discussion sur les résultats expérimentaux obtenus.

2.1 Programme de recherche de motifs

Dans cette section, nous allons présenter le fonctionnement du programme que nous utilisons pour appliquer des grammaires à des textes. Par appliquer, nous entendons repérer dans le texte toutes les occurrences des motifs décrits par les grammaires. Nous avons consacré notre mémoire de DEA à l'élaboration d'un programme baptisé AGLAE dont le but était d'effectuer cette opération pour des données volumineuses. Nous avons repris et amélioré ce programme pour l'utiliser avec les grammaires issues des tables de lexique-grammaire. Nous commencerons par présenter le programme d'origine en rappelant ses principes essentiels, puis nous décrirons les modifications qui lui ont été apportées.

2.1.1 Présentation d'AGLAE

Cette section reprend en partie les explications données dans [61] et [62]. Le logiciel AGLAE peut être téléchargé librement sur la page web située à l'adresse suivante :
http://ladl.univ-mlv.fr/tools/aglae/aglae.html

2.1.1.1 Objectif d'AGLAE

Le traitement automatique des langues naturelles fait l'objet de méthodes très diverses. L'approche conçue au LADL et adoptée à l'IGM repose sur la description de propriétés linguistiques, notamment dans les grammaires représentées au moyen de graphes. Pour être efficace, cette approche doit s'appuyer sur des descriptions les plus exhaustives possibles, ce qui conduit à examiner des corpus volumineux. Il faut donc être en mesure de traiter de grands textes dans des temps raisonnables. Une fois constituées, les collections de graphes doivent pouvoir être appliquées rapidement, ce qui n'est pas trivial étant donné leur taille. Il faut donc disposer de méthodes permettant de manipuler simultanément de grands textes et d'importantes grammaires. Le logiciel AGLAE a donc été conçu pour résoudre ce problème. Ce programme constitue un outil additionnel au système INTEX développé par Max Silberztein (voir [73], [74] et [75] pour une présentation de ce système).

2.1.1.2 Les corpus

Nous n'avons pas choisi de représenter le texte au moyen d'un index. Cette méthode est courante dans les applications commerciales car elle présente l'avantage d'accélérer la recherche par mot clé. Dans [71], Jean Senellart montre comment on peut utiliser une structure d'index pour accélérer l'application de grammaires. Sa méthode consiste à localiser les zones du texte dans lesquelles peuvent apparaître des motifs reconnus par la grammaire, en examinant en priorité les mots les plus rares qu'elle contient. Toutefois, cette méthode ne nous semble plus adaptée lorsque la taille des textes et des grammaires augmente. En effet, plus le texte est volumineux et plus il contient de mots. Par conséquent, les mots contenus dans la grammaire voient leur probabilité d'apparaître augmenter, ce qui nuit à la propriété de rareté sur laquelle repose l'algorithme. De plus, la complexité de l'application d'une grammaire augmente avec sa taille, car la représentation des graphes que nous avons choisie ne permet pas de garantir le déterminisme (ce point sera discuté en 2.1.1.3). En conséquence, l'économie réalisée en ne démarrant les calculs qu'aux endroits favorables risque de devenir négligeable devant le temps consommé par ces calculs au delà d'une certaine taille.

Nous avons donc décidé de conserver une structure de texte linéaire. Étant donné que nous traitons des données linguistiques, nous avons segmenté le texte en prenant comme unité des *tokens* représentant des unités lexicales.

Définition : on appelle *token* tout mot t appartenant au langage $L^+ \cup \{\texttt{\{S\}}\} \cup M$, où L représente l'ensemble des lettres, M représente l'ensemble des caractères n'étant pas des lettres, et $\{\texttt{\{S\}}\}$ désigne le mot formé des caractères `{`, `S` et `}`.

Autrement dit, un token peut être :

- soit une séquence de lettres (les caractères considérés comme des lettres sont définis au moyen d'un fichier alphabet) ;
- soit la séquence de caractères `{S}` représentant le séparateur de phrases ;
- soit un caractère non alphabétique.

Chaque token différent se voit attribuer un entier. Le texte est alors représenté par une suite d'entiers. Considérons le texte suivant : *{S}Un chat est un chat.* La segmentation de ce texte produit les tokens suivants :

Token	{S}	Un	*espace*	chat	est	un	.
Entier	0	1	2	3	4	5	6

Le fichier représentant le texte contiendra donc la suite d'entiers :

0 1 2 3 2 4 2 5 2 3 6

On notera qu'il est tenu compte des différences entre minuscules et majuscules : *Un* et *un* sont codés comme deux tokens disctincts [1]. Comme nous le verrons, ce principe de représentation des tokens par des entiers permettra d'optimiser un certain nombre d'opérations ultérieures.

2.1.1.3 Compilation des graphes

La représentation FST (Finite-State Transducer) utilisée dans INTEX pour compiler les grammaires repose sur une transformation des graphes où chaque appel à un sous-graphe est remplacé par le sous-graphe lui-même, comme on peut le voir sur l'exemple suivant :

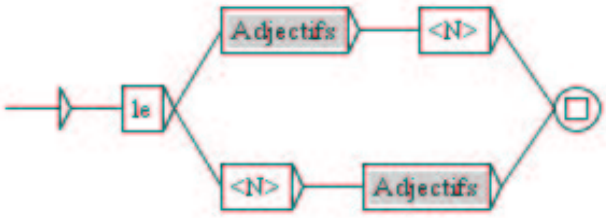

FIG. 2.1 – Graphe principal

La compilation en FST du graphe de la figure 2.1 produit un résultat équivalent au graphe de la figure 2.3. Comme nous l'avons déjà souligné en 1.3.1, le problème de redondance qui en résulte peut faire exploser la taille des fichiers obtenus. Afin d'éviter ce problème, nous avons conservé la structure en sous-graphes. L'avantage est la taille réduite de représentation, l'inconvénient est qu'elle n'est pas déterministe, ce qui peut ralentir notablement les calculs. Nous nous sommes contenté dans un premier temps de déterminiser chaque graphe et sous-graphe, en considérant les appels à des sous-graphes comme des symboles de l'alphabet. Nous

[1] À titre indicatif, on trouve environ 30 000 tokens différents dans un corpus journalistique équivalent à 3 méga-octets de texte en ASCII.

FIG. 2.2 – Graphe *Adjectifs*

verrons en 2.2.1 comment nous avons par la suite tenté de remédier à ce problème de non-déterminisme. Le format FST2 que nous avons mis au point pour représenter nos graphes compilés est décrit dans le manuel d'Unitex joint en annexe.

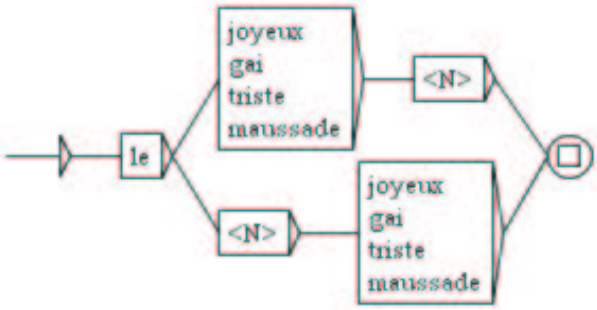

FIG. 2.3 – Graphe équivalent au graphe de la figure 2.1

Notre compilation a ceci d'intéressant qu'elle transforme chaque sous-graphe en un automate dont les transitions sont étiquetées soit par des appels à des sous-graphes, soit par des motifs, soit par des unités lexicales. On pourra donc remplacer ces dernières par des entiers comme nous l'avons fait pour le texte. Notons que le fait de manipuler des entiers au lieu de chaînes de caractères nous permet déjà de réduire le nombre d'opérations que nous effectuerons sur les unités lexicales d'un facteur représentant le nombre moyen de lettres par mot. Nous avons calculé à partir de l'année 1994 du journal *Le Monde*, qui représente 120 méga-octets de texte, que ce nombre valait 5,7 pour le français.

2.1.1.4 Analyse descendante

En conservant la structure en sous-graphes dans les grammaires, notre compilateur ne produit pas des transducteurs à états finis, mais des *réseaux de transitions récursifs* (RTN : Recursive Transition Networks). Ces objets sont également désignés sous le nom de *grammaires algébriques étendues* ([2], [10]), *automates finis avec appels récursif* ([28]), ou encore *diagrammes syntaxiques*. Cette structure nous a naturellement conduit à adopter une méthode d'analyse descendante semblable à celles utilisées en compilation ([1]). Notons d'ailleurs que ce que l'on nomme diagrammes syntaxiques sont en fait des grammaires décrivant des langages de programmation et utilisés pour écrire des compilateurs.

Toutefois, l'utilisation de l'analyse descendante est soumise à deux conditions :
- la grammaire ne doit pas être récursive à gauche ;
- la grammaire ne doit pas comporter de boucle infinie par ε.

Nous considérons ici tous les types de récursivité à gauche, c'est-à-dire tous les cas où il est possible qu'un graphe s'appelle lui-même, de façon directe ou indirecte, sans avoir avancé dans l'entrée. Dans le cas de grammaires linguistiques, les récursions à gauche sont d'une part assez rares, et d'autre part faciles à éliminer manuellement, car cela ne rend pas les grammaires illisibles et ne bouleverse pas leur structure. La grammaire de la figure 2.4 montre un exemple de récursivité à gauche destinée à analyser des conjonctions de groupes nominaux élémentaires tels que *la cigale et la fourmi*. Cette récursion peut être évitée en transformant cette grammaire en celle présentée sur la figure 2.5. Bien que ces récursions puissent être éliminées automatiquement, nous ne l'avons pas fait afin de ne pas modifier la structure de la grammaire. Nous avons donc pris le parti de détecter, lors de la compilation d'une grammaire, les éventuelles récursions à gauche et de signaler une erreur le cas échéant. Il incombe alors à l'utilisateur de modifier ses graphes.

FIG. 2.4 – Graphe *GN*

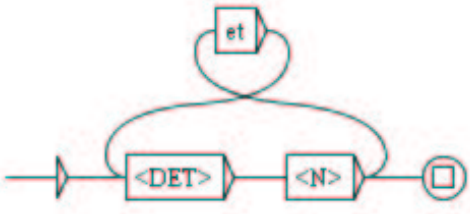

FIG. 2.5 – Graphe *GN* modifié

L'autre type d'erreur empêchant d'effectuer une analyse descendante est la présence dans la grammaire de boucles infinies par ε. Ces boucles peuvent avoir pour origine soit des ε-transitions munies de sorties [2], soit des appels à des sous-graphes reconnaissant le mot vide. La figure 2.6 montre un graphe contenant ces deux types de boucle.

Il existe un algorithme permettant de supprimer les ε-transitions dans un automate. Nous l'appliquons sur toutes les ε-transitions dépourvues de sorties car cela ne modifie pas la grammaire. En revanche, nous ne touchons pas à celles munies de sorties car une boucle sur une

[2] Les boucles par des ε-transitions sans sortie ne sont pas gênantes, car l'algorithme de suppression des ε-transitions permet de les retirer.

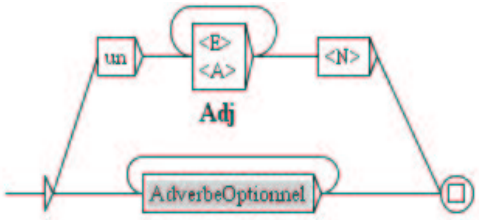

FIG. 2.6 – Graphe contenant deux boucles par ε

ε-transition avec sortie serait absurde : cela revient à vouloir produire une infinité d'éléments en sortie sans rien lire en entrée. Les boucles par des sous-graphes reconnaissant le mot vide ne pourraient être éliminées qu'en modifiant simultanément plusieurs graphes dans la grammaire. Nous avons choisi de ne pas chercher à résoudre ces problèmes de boucles mais plutôt de les signaler à l'utilisateur.

Afin d'obtenir la garantie que les grammaires utilisées sont compatibles avec les exigences de l'analyse descendante, nous avons intégré au compilateur de graphes un module de détection d'erreurs. Ce module vérifie que la grammaire qui lui est fournie ne reconnaît pas le mot vide et ne contient ni récursion à gauche ni boucle infinie. En cas d'erreur, le programme fournit une description du problème rencontré qui permet à l'utilisateur de localiser et corriger la partie incorrecte de la grammaire.

2.1.1.5 Traitement des étiquettes

Une fois la grammaire compilée et chargée, nous effectuons deux prétraitements sur les étiquettes portées par ses transitions afin d'accélérer l'exploration de la grammaire lors de son application au texte. Ces prétraitements dépendent du contenu du texte.

Le premier de ces traitements consiste à remplacer chaque transition étiquetée d'un token ou d'une entrée de dictionnaire par la liste de toutes ses variantes. Dans le cas d'une entrée de dictionnaire comme `<manger.V>`, on extrait du dictionnaire du texte toutes les formes fléchies reconnues par ce motif. Ainsi, si le dictionnaire du texte contient les entrées suivantes :

```
mange,manger.V+z1:P1s:P3s:S1s:S3s:Y2s
mangea,manger.V+z1:J3s
mangeait,manger.V+z1:I3s
```

le motif `<manger.V>` sera remplacé par la liste de mots *mange*, *mangea* et *mangeait*. Une fois ce remplacement effectué, on substitue à chaque mot l'ensemble des variantes minuscules/majuscules qu'il peut avoir dans le texte. Si le texte contient les tokens *mange*, *MANGE*, *mangea* et *Mangeait*, le motif `<manger.V>` sera donc finalement remplacé par la liste de ces 4 tokens. Les tokens sont ensuite remplacés par les entiers qui leur ont été attribués lors de la segmentation du texte.

Le second prétraitement porte sur les motifs sur contenant pas de lemme. Les méta-

symboles tels que <MAJ> ou <DIC>[3] sont remplacés par des codes spéciaux. Nous pré-calculons pour chaque token T du texte quels métas il vérifie, et nous codons ces informations sur un entier meta[T] dont chaque bit correspond à un méta. Lors de l'exploration de la grammaire, la comparaison entre un méta M et un token T s'effectue donc très simplement au moyen d'un test binaire (meta[T] & MASQUE) où MASQUE est le masque binaire correspondant au méta M. Cette opération est rendue très simple par le fait que les entiers codant les tokens indexent des tableaux.

Les formes du type <N:ms>, que nous appellerons *filtres lexicaux*, sont extraites de la grammaire et numérotées. Lors du chargement des dictionnaires du texte, on vérifie pour chaque token quels filtres il vérifie. Prenons par exemple un graphe contenant les filtres <V>, <N+z1> et <N:fs:ms> désignant respectivement les verbes, les noms du langage courant et les noms singuliers au féminin ou au masculin. Numérotons ces filtres de la manière suivante :

<V> = 0, <N+z1> = 1 et <N:fs:ms> = 2

Si les dictionnaires du texte contiennent les lignes

```
garde,garde.N+z1:fs
garde,garde.N+z1:ms:fs
garde,garder.V+z1:P1s:P3s:S1s:S3s:Y2s
```

alors le code associé au token *garde* est l'ensemble $\{0, 1, 2\}$. Nous pourrions donc tester si un token est reconnu par un filtre lexical au prix d'une consultation dans un tableau de bits à deux dimensions, indexé par les mots et les numéros de filtre. Toutefois, ce tableau étant très creux, il nous a paru préférable de le comprimer, ce qui ralentit légèrement la procédure de test, mais permet d'économiser beaucoup de place. Pour cela, nous codons chaque ligne du tableau en en extrayant tous les suites de 32 bits non nulles en associant à chaque entier ainsi obtenu son numéro dans la ligne. Nous obtenons ainsi des couples *(numero d'entier,valeur)*, que nous stockons dans une liste chaînée, triée selon les numéros d'entier. Chaque ligne du tableau se trouve ainsi codée par une liste chaînée. Ainsi, pour tester si le filtre p est vérifié par le token n, il nous suffit d'explorer la liste chaînée numéro n : si l'on trouve l'entier numéro $p/32$ et que son bit numéro p *modulo* 32 vaut 1, alors le test est positif. Dans la plupart des cas, les listes ne contiennent pas plus d'un élément, ce qui réduit le nombre maximum d'opérations à 3 : un test pour savoir si la liste est vide, une comparaison sur le numéro de l'entier, et un test pour tester le bit considéré. Ce mécanisme est donc très efficace pour représenter et manipuler nos ensembles de numéros de filtres. Notons que cette représentation est plus efficace qu'une simple liste répertoriant les numéros des filtres vérifiés. En effet, comme notre représentation permet de regrouper 32 filtres dans un même entier, elle permet de tester 32 filtres différents en un même nombre d'opérations, contrairement à une représentation par liste. La méthode par liste simple n'est donc plus efficace que lorsque le nombre de filtres vérifiés est inférieur ou égal à 1. Or, il n'est pas rare qu'un mot soit reconnu par plus d'un filtre : par exemple, le mot *agréable* peut être reconnu par les filtres <A>, <A:s>, <A:ms>, <A:f>, etc. Notre méthode est donc en général plus rapide.

[3] Les symboles <MAJ> et <DIC> désignent respectivement les mots ne contenant que des majuscules et les mots contenus dans le dictionnaire.

2.1.2 D'AGLAE à Unitex

À l'issue de notre DEA, nous disposions d'une version d'AGLAE qui diminuait très nettement les temps d'application des grammaires par rapport au programme équivalent d'INTEX. Au cours de notre étude sur le lexique-grammaire des verbes du français, nous avons été amené à traiter d'une part, des grammaires de plus en plus importantes et de nature essentiellement lexicale, et d'autre part, des grammaires générées automatiquement depuis les tables et possédant des structures particulières. Nous avons donc développé nos propres outils pour mener à bien nos expériences. Étant donné que le système INTEX était un logiciel propriétaire et que l'effort nécessaire pour compléter nos outils était peu important, nous avons réécrit l'ensemble des fonctions élémentaires afin de constituer un nouveau système baptisé Unitex, basé sur le principe du logiciel libre. Nous avons profité de ce changement pour intégrer la gestion du standard Unicode ainsi que la prise en charge des langues sans délimiteur de mots telles que le chinois ou le thaï (voir section 3.1).

Dans la mesure où le programme chargé d'appliquer les grammaires sur des textes constitue le moteur de notre analyse syntaxique, nous allons maintenant décrire les tranformations que nous avons apportées à AGLAE afin de l'exploiter dans nos recherches. Nous commencerons par présenter notre méthode de gestion des mots composés, puis nous décrirons la méthode que nous avons adoptée pour accélérer l'exploration des grammaires.

2.1.2.1 Gestion des mots composés

AGLAE ne gérait que les mots simples, de sorte qu'il était impossible de bénéficier des informations contenues dans les dictionnaires de mots composés. Nous avons remédié à cette situation en traitant les filtres lexicaux d'une façon similaire à la méthode employée pour les mots simples.

Si le filtre contient un lemme, on le remplace par la liste des mots composés qu'il peut reconnaître. Ainsi, le filtre `<pomme de terre>` sera remplacé par *pomme de terre* et *pommes de terre* si ces deux variantes sont présentes dans le dictionnaire des mots composés du texte. Nous dérivons ensuite de ces deux séquences toutes les suites de tokens que l'on peut obtenir en tenant compte des variantes minuscules/majuscules. Ainsi, si le texte contient les tokens *pomme*, *Pommes*, *de*, *DE*, *terre* et *Terre*, le filtre `<pomme de terre>` sera remplacé dans la grammaire par l'équivalent de l'extrait de graphe présenté sur la figure 2.7.

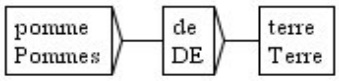

FIG. 2.7 – Combinaisons de tokens reconnaissables par le filtre `<pomme de terre>`

Dans le cas d'un filtre lexical ne contenant pas de lemme, nous avons déjà vu que ces filtres étaient numérotés et que l'on notait pour chaque mot simple quels filtres il vérifiait. Nous gérons les mots composés reconnus par des filtres d'une façon semblable. La différence

réside dans le fait que les mots simples du texte peuvent servir à indexer un tableau, puisqu'ils sont codés par des entiers. Puisqu'on ne peut pas utiliser le même procédé pour les mots composés, la solution que nous avons adoptée consiste à représenter l'ensemble des mots composés reconnus par au moins un filtre sous la forme d'un automate dont les transitions représentent des tokens et dont les états contiennent des ensembles de numéros de filtres. Supposons que l'on ait les trois filtres suivants :

`<A>` = 0, `<ADV>` = 1 et `<PREP>` = 2

Si le dictionnaire de mots composés du texte contient les lignes

```
à venir,.A
dans ce cas,.ADV
dans l'ombre de,.PREP
dans l'ombre,.A
dans l'ombre,.ADV
```

on obtiendra un automate semblable à celui présenté sur la figure 2.8.

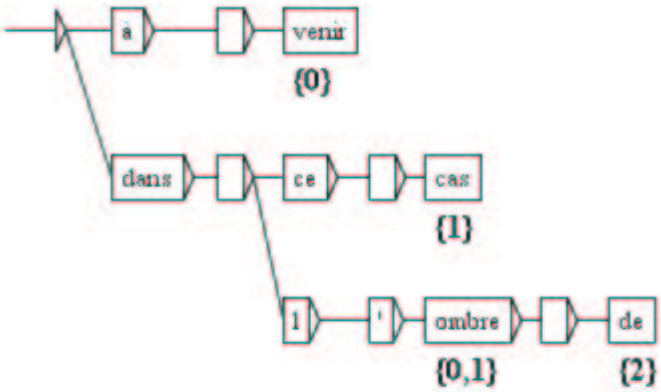

FIG. 2.8 – Représentation d'un automate de mots composés

Nous intégrons dans cet automate les variantes minuscules/majuscules afin d'être en mesure de repérer toutes les occurrences de mots composés pouvant être reconnues par au moins un des filtres lexicaux de la grammaire. Nous trions ensuite les transitions sortant de chaque état de façon à pouvoir les parcourir efficacement. Lorsque l'on est en présence d'un filtre susceptible de reconnaître des mots composés lors de l'exploration de la grammaire, il suffit alors de parcourir simultanément le texte et l'automate et de tester si l'on aboutit à un état terminal dans celui-ci. Il ne reste alors plus qu'à vérifier si le numéro du filtre que l'on teste appartient à l'ensemble des filtres vérifiés par le mot composé trouvé.

2.1.2.2 Optimisation de l'analyseur

Dans la première version d'AGLAE, nous avions accéléré l'application d'une grammaire à un texte en tirant parti d'une caractéristique des grammaires linguistiques. En effet, on trouve très fréquemment dans les graphes des boîtes contenant plusieurs lignes. Cela correspond dans la grammaire compilée à plusieurs transitions ayant les mêmes états de départ et d'arrivée. Une fois les étiquettes remplacées par leurs variantes possibles, ce nombre de transitions peut devenir très important. Ainsi, la boîte de la figure 2.9 peut théoriquement être équivalente à plus d'une centaine de transitions car chacun de ces verbes peut posséder jusqu'à 39 formes fléchies différentes.

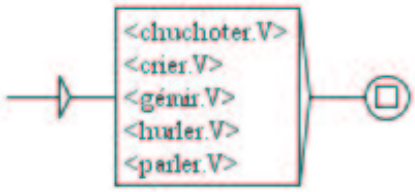

FIG. 2.9 – Exemple de graphe

Nous avions tiré parti de cette constatation en regroupant ces transitions dans des classes de mots. Une classe de mots est définie comme un ensemble de tokens tels qu'il existe dans la grammaire deux états a et b, pour lesquels on a une au moins une transition $(a, (t, s), b)$ pour chaque token t, s désignant soit un symbole de l'alphabet de sortie, soit le mot vide ε.

Le principe est de donner un numéro à chaque classe ainsi formée, et de noter pour chaque mot à quelles classes il appartient. On remplace ensuite toutes les transitions relatives à une classe par une seule transition, étiquetée par le numéro de la classe. De cette façon, lors de l'exploration de la grammaire, il suffit de tester si le token courant dans le texte appartient à l'une des classes de mots définies pour le noeud courant de la grammaire, ce qui peut être fait rapidement au moyen d'une consultation d'un tableau d'entiers.

Cette méthode nous a permis d'accélérer sensiblement les calculs. Cependant, elle présente deux défauts. Premièrement, elle consomme de la mémoire, car il faut stocker pour chaque mot l'ensemble des classes auxquelles il appartient. Deuxièmement, une fois les étiquettes remplacées par les listes de tokens correspondantes, la grammaire peut ne pas être déterministe car deux classes distinctes peuvent avoir une intersection non vide. Considérons par exemple le graphe de la figure 2.10. Ce graphe va conduire à la construction d'une classe pour les tokens issus du filtre lexical `<être.V>`, et une autre pour ceux provenant du filtre `<suivre.V>`. Or, le token *suis* peut être reconnu par les deux filtres. Il y a donc non-déterminisme car on doit tester toutes les classes pour explorer les chemins.

Nous avons amélioré les performances de notre programme en utilisant une autre technique qui est basée sur une propriété de notre représentation des textes. En effet, après avoir traité les

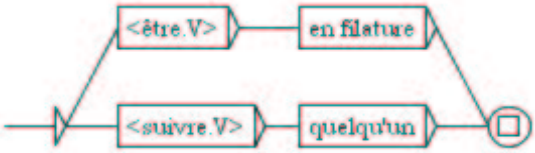

FIG. 2.10 – Graphe devenant ambigu après remplacement des étiquettes

étiquettes de la grammaire comme nous l'avons expliqué en 2.1.1.5, les noeuds de la grammaire contiennent soit des appels à des sous-graphes, soit des métas (`<MIN>`, `<MOT>`, etc), soit des filtres lexicaux (`<N:ms>`, `<ADV>`, etc), soit des listes de tokens. Les trois premières catégories de transitions doivent toutes être explorées. En revanche, les tokens sont par construction des éléments disctincts les uns des autres. Lorsque l'on va comparer une liste de tokens avec le token courant dans le texte, nous sommes donc sûr qu'au plus un token pourra correspondre. Nous avons tiré parti de cette propriété en représentant les listes de tokens par des tableaux. Nous trions ensuite chaque tableau suivant l'ordre des entiers représentant les tokens. Il est possible de rencontrer plusieurs transitions étiquetées par un même token mais possédant des sorties différentes et/ou des états d'arrivées différents, comme c'est le cas avec le graphe de la figure 2.10. Dans ce cas, nous ne codons qu'une seule fois ce token dans le tableau trié, mais nous lui associons plusieurs informations. Considérons par exemple le graphe de la figure 2.11.

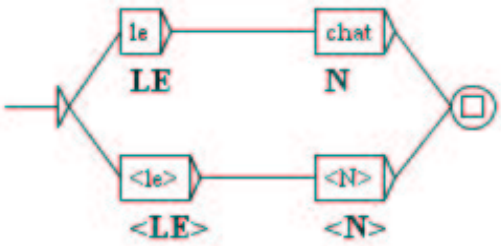

FIG. 2.11 – Exemple de graphe

En supposant que `<le>` soit remplacé par *le*, *LA* et *Les*, l'état initial de la grammaire serait représenté par le tableau 2.1. Cette opération nous permet de réduire le coût du non-déterminisme, car on ne testera qu'une fois un token même si plusieurs couples *(sortie, état d'arrivée)* lui sont associés.

Le parcours de ces tableaux triés peut alors s'effectuer très rapidement en recherchant par dichotomie si le token courant du texte est présent dans le tableau du noeud courant de la grammaire. Les avantages de cette méthode par rapport à la précédente sont une économie de mémoire et une réduction des effets du non-déterminisme. L'inconvénient relatif est que l'on effectue $1 + \log(n)$ opérations, où n est le nombre de tokens distincts dans la classe, alors que l'on n'effectuait qu'une opération pour chaque classe de mots sortant du noeud. On est

Token	(sortie, état d'arrivée)
le	(LE, 1) (<LE>, 2)
LA	(<LE>, 2)
Les	(<LE>, 2)

TAB. 2.1 – Tableau de tokens pour l'état initial du graphe de la figure 2.11

sûr que la méthode par dichotomie est moins efficace lorsqu'il n'y a qu'une seule classe de mots. En revanche, dans le cas où chaque token forme une classe constituée d'un seul élément, on obtient une situation très défavorable pour la méthode par classes de mots, car on doit effectuer autant de tests qu'il y a de tokens, ce qui n'apporte aucun gain par rapport à une exploration naïve des transitions. Ces deux cas extrêmes sont représentés par les graphes de figures 2.12 et 2.13. Le premier graphe conduit à la formation d'une seule classe de mots, ce qui conduira à n'effectuer qu'une seule opération contre $1 + \log(6)$ pour la méthode par dichotomie. En revanche, l'état initial du second graphe contiendra 6 classes de mots, ce qui donnera 6 tests au lieu de $1 + \log(6)$ pour l'autre méthode.

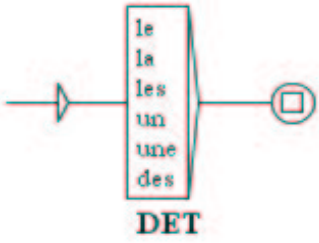

FIG. 2.12 – Cas favorable aux classes de mots

Dans la pratique, la diversité des grammaires fait qu'elles oscillent entre ces deux tendances. On distingue toutefois trois grandes familles de grammaires. La première catégorie regroupe les grammaires très riches lexicalement ; les grammaires d'auxiliaires verbaux présentées à la section 1.3.7.1 en sont un exemple. Dans de telles grammaires, le nombre de tokens par classe de mots est élevé, et il n'y a en général qu'une seule classe par état, ce qui rend la méthode par classes de mots très efficace pour ces grammaires. La seconde catégorie constitue l'autre extrême, c'est à dire les grammaires où chaque classe ne contient que très peu de mots, et où il y a plusieurs classes pour chaque état. Cette catégorie englobe les grammaires générées à partir des tables. Le dernier type de grammaire regroupe celles dont la complexité est intermédaire. C'est par exemple le cas de la grammaire présentée sur la figure 1.60 (page 104). Nous avons constaté que, dans l'ensemble, le gain moyen était plus important lorsque l'on utilisait la méthode par dichotomie. Afin de mesurer la portée de l'optimisation ainsi obtenue, nous avons comparé dans [60] notre représentation des grammaires avec deux autres modèles.

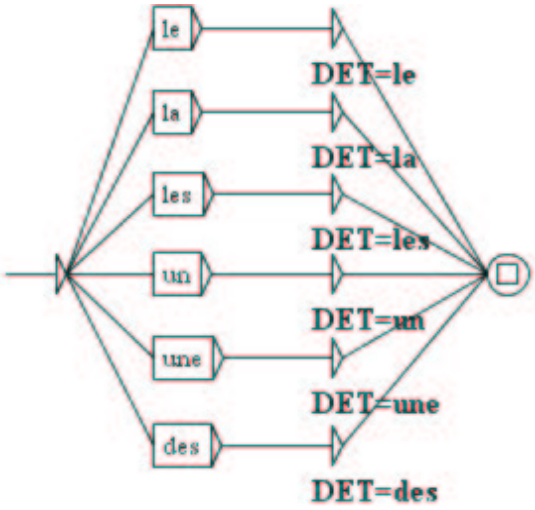

FIG. 2.13 – Cas défavorable aux classes de mots

Nous avons tout d'abord considéré une représentation semblable à la nôtre, mais dans laquelle les tokens ne sont pas codés par des entiers. On peut néanmoins appliquer une méthode par dichotomie semblable à celle que nous utilisons pour parcourir les transitions étiquetées par des tokens, ce qui nous donne un nombre d'opérations égal à $1+\log(n)$ (n représentant le nombre de tokens distincts) que multiplie la complexité de la fonction de comparaison de deux tokens. Or, lorsque les tokens sont représentés par des chaînes de caractères, la comparaison s'effectue en un nombre d'opérations dépendant des chaînes considérées. Dans le meilleur des cas, c'est-à-dire lorsque la comparaison peut se faire en n'examinant que les premiers caractères des chaînes, on doit effectuer au moins une comparaison d'entiers. Dans le cas moyen, ce nombre d'opérations est plus élevé, ce qui montre que cette méthode est moins efficace que la nôtre, puisque nous effectuons la comparaison de deux tokens quelconques en une seule comparaison entière.

Nous avons ensuite comparé notre méthode avec une représentation d'une liste de n tokens sous forme d'un automate déterministe de caractères. En supposant que les transitions de cet automate soient triées pour pouvoir être explorées par dichotomie, le test de l'appartenance à l'automate d'un token de longueur m s'effectue en $O(m \times (1+\log(t)))$ opérations si le mot appartient à l'automate, t représentant le nombre moyen de transitions sortant de chaque état de l'automate. Avec notre méthode, ce test est effectué en $1+\log(n)$ opérations, où n représente le nombre de tokens à tester. Les valeurs m, n et t étant indépendantes les unes les autres, il faut examiner des valeurs typiques pour pouvoir établir une comparaison.

Si le nombre de tokens n est inférieur à une certaine limite dépendante des tokens considérés, typiquement 16 tokens, la relation $1+\log(n) < O(m \times (1+\log(t)))$ se vérifie car $1+\log(n)$ est inférieur à m, c'est-à-dire au nombre de lettres du token à tester, et la constante multiplicative dans $O(m \times (1+\log(t)))$ vaut au moins 1. Pour des listes de mots de taille raisonnable, notre méthode est donc plus efficace.

Quand n augmente, il devient très difficile de déterminer la valeur de $O(m \times (1 + \log(t)))$, car le nombre moyen t de transitions sortant d'un état dépend des tokens codés par l'automate. Nous pouvons toutefois faire l'estimation suivante.

La plupart des listes contiennent moins de 10 éléments. Les listes les plus importantes que nous ayons rencontrées dans des grammaires sont des listes de noms propres, comme par exemple la liste des noms de pays. Même ces listes ne dépassent pas les 500 éléments. Notons qu'il s'agit là d'une valeur maximum qui n'est que rarement atteinte dans la pratique, le nombre de mots que l'on peut trouver dans une liste étant bien inférieur à 500. Cela s'explique par une raison linguistique simple : les mots ne sont pas regroupés au hasard dans une même liste. Le but d'une grammaire linguistique est de décrire un ensemble cohérent d'expressions. Si des mots apparaissent dans une même liste, cela signifie donc qu'ils sont liés, la nature de ces liens pouvant être diverse : même catégorie grammaticale, relation de synonymie, etc. Ce phénomène est illustré par le graphe de la figure 2.14, dans lequel les noms de mois constituent une classe de mots évidente. Si les éléments d'une classe sont trop nombreux, par exemple l'ensemble des noms, ils n'apparaîtront vraisemblablement pas dans une liste à l'intérieur d'une grammaire, mais plutôt comme des entrées dans un dictionnaire, et la grammaire fera référence à ces entrées par l'intermédiaire d'un code grammatical.

FIG. 2.14 – Graphe décrivant les noms de mois

Étant donné que dans la valeur m dans l'expression $O(m \times (1 + \log(t)))$ représente le nombre de lettres du token que l'on teste, nous pouvons considérer qu'en moyenne ce nombre est proche du nombre moyen de lettres par mot. Or, comme nous l'avons indiqué en 2.1.1.3, ce nombre vaut 5,7 pour le français[4]. Si nous considérons un nombre de tokens n suffisamment important, nous pouvons supposer que $1 + \log(t)$ aura une valeur supérieure à 1. En d'autres termes, lorsque le nombre de tokens augmente, il y a de fortes chances pour que le nombre moyen de transitions par état dans l'automate soit supérieur à 1. Pour vérifier cette intuition, nous avons mesuré le nombre moyen de transitions par états sur 5 séries de 16 mots. Afin

[4]Cette valeur peut varier pour d'autres langues, mais les ordres de grandeur restant les mêmes, cela n'affecte pas le raisonnement.

d'obtenir des distributions de mots relativement aléatoires, nous avons pris 5 textes français, et nous en avons extrait les 16 premiers tokens distincts. Nous avons ensuite modifié ces listes en ajoutant à chaque ligne la séquence `,.TEST` afin de donner à nos listes des structures de dictionnaires DELAF. Nous avons ensuite utilisé notre programme de compression de dictionnaire pour qu'il construise les automates déterministes minimaux associés aux formes fléchies de nos 5 listes de 16 mots. Nous avions au préalable modifié ce programme pour qu'il calcule les nombres d'états et de transitions de chaque automate. Nous avons obtenu les nombres moyens de transitions par état suivants :

1,269
1,343
1,257
1,291
1,350

Ces résultats confirment donc notre intuition et nous permettent de supposer que la valeur $1 + \log(t)$ est supérieure à 1 pour un nombre de tokens supérieur à la quinzaine[5]. En prenant pour t la moyenne des 5 résultats obtenus, et pour m le nombre moyen de lettres par mot, on obtient :

$$m \times (1 + \log(t)) = (5,7 \times (1 + \log(\tfrac{1,269+1,343+1,257+1,291+1,350}{5}))) \approx 8.$$

La valeur t ne varie pas beaucoup lorsque l'on augmente le nombre de tokens (on obtient la valeur 1,775 en prenant comme liste de mots la totalité des mots simples du français). De l'inéquation approximative $1 + \log(n) < 8$, nous déduisons donc que la valeur maximum de n telle que $1 + \log(n) < O(m \times (1 + \log(t)))$ est approximativement comprise entre 100 et 150, en prenant 1 comme constante multiplicative. Au delà de ces valeurs, notre méthode est donc moins performante qu'une représentation par automate de caractères. Néanmoins, nous avons vu que de telles tailles de listes étaient marginales et qu'en moyenne, la valeur n était plutôt inférieure à 10. Nous pouvons donc en conclure que notre méthode est donc théoriquement plus efficace en moyenne.

Bien que ces calculs ne constituent qu'une estimation, ils semblent indiquer que notre approche est bien adaptée au traitement de listes de tokens. Toutefois, pour mesurer l'efficacité réelle de notre méthode, nous devons prendre en compte le coût des pré-calculs requis par notre méthode, c'est-à-dire la segmentation du texte et les opérations effectuées sur la grammaire.

La segmentation d'un texte est une opération dont la complexité est proportionnelle à sa taille. En pratique, le coût de cette opération peut être ignoré dès lors que les grammaires deviennent importantes, car la durée de la segmentation devient négligeable devant le temps nécessaire à l'application de grammaires. Étant donné que d'une part, le programme a pour vocation de traiter des grammaires non-triviales, et que d'autre part, cette opération ne doit

[5]Rappelons que 16 est approximativement le nombre de tokens en dessous duquel notre méthode est plus efficace car $\log(n)$ est en moyenne inférieur au nombre de lettres du token à tester.

être effectuée qu'une seule fois même si l'on applique plusieurs grammaires, nous pouvons considérer que cette étape n'affecte pas les performances générales.

L'optimisation de la grammaire passe par le remplacement de chaque token par la liste d'entiers équivalente. Cette opération est linéaire en la taille de l'alphabet de la grammaire, ce qui est négligeable devant la complexité de l'application de la grammaire au texte. En effet, l'ordre de grandeur de l'alphabet d'une grammaire complexe est souvent bien inférieur à celui de la complexité de la grammaire. À titre d'exemple, une grammaire de Matthieu Constant, qui décrit des groupes nominaux basés sur des expressions numériques, possèdant un alphabet d'environ 1600 éléments et constituée de 126 sous-graphes, contient environ 100 000 états et 700 000 transitions si on la transforme en un transducteur à états finis équivalent.

Lorsqu'une grammaire contient des références aux informations associées aux tokens dans le dictionnaire du texte, il faut consulter ce dictionnaire. La complexité de cette opération dépend de la représentation des dictionnaires. Cette étape dans l'optimisation de la grammaire pourrait être critique, car notre algorithme parcourt toutes les références aux informations lexicales contenues dans la grammaire. Nous allons cependant montrer qu'il n'en est rien.

Si la grammaire est entièrement explorée durant son application au texte, alors toutes les références aux informations lexicales doivent être traitées. Dans ce cas, il n'y a aucune différence entre notre approche et une consultation du dictionnaire à la volée, c'est-à-dire chaque fois qu'un code grammatical de la grammaire est comparé à un mot du texte : quelle que soit la complexité des consultations, ces opérations sont inévitables.

En revanche, si seule une petite partie de la grammaire est explorée, un algorithme de consultation à la volée n'aura effectué presque aucun accès au dictionnaire, alors que tous les accès possibles auront été faits avec notre méthode lors du prétraitement. Le cas critique est donc celui où l'on applique à un texte une grammaire qui décrit beaucoup de séquences absentes du texte. D'un point de vue linguistique, ce cas est rare, mais n'est pas absurde, car une grammaire peut être plus générale qu'un texte, même si personne n'appliquerait une grammaire de groupes nominaux français à un corpus anglais, ou une grammaire de termes électroniques à un texte poétique. De plus, en effectuant des prétraitements plutôt qu'une consultation dynamique, nous réduisons le non-déterminisme de la grammaire en remplaçant plusieurs motifs par des listes d'éléments lexicaux. Ainsi, nous avons eu recours à une comparaison empirique.

Étant donné qu'il existe très peu de systèmes autorisant des recherches de motifs au moyen de grammaires et de dictionnaires, notre seul point de comparaison était le système INTEX. Une communication personnelle avec Max Silberztein, l'auteur de ce système, nous a permis d'établir que le système fonctionnait par consultation dynamique du dictionnaire et que les grammaires étaient représentées en interne d'une façon proche des automates de caractères. En pratique, nous avons mesuré que le temps consommé par les pré-calculs était compensé par le gain obtenu lors de l'application de la grammaire au texte, dès que celle-ci n'est plus triviale. Le tableau 2.2 présente des résultats obtenus sur un Pentium 500 MHz doté de 128 mégaoctets de mémoire. La grammaire d'expressions numériques contient très peu de références

	Expressions numériques	Syntagmes verbaux	Groupes nominaux économiques
États	107 000	250 000	587
Transitions	703 000	2 000 000	3596
Taille du corpus	0.4 Mo	16 Mo	120 Mo
Temps avec Intex	20 min	Échec	9 min
Temps avec notre programme	22 s	3 min 37 s	1 min 11 s

TAB. 2.2 – Résultats expérimentaux

aux informations contenues dans les dictionnaires, tandis que l'on en trouve beaucoup dans les expressions verbales, et une vingtaine dans les groupes nominaux économiques.

Les expériences que nous avons menées montrent donc que notre méthode est bien adaptée aux recherches de motifs linguistiques dans des textes. Cependant, le gain n'est important que lorsque les grammaires sont d'une certaine taille. Notre méthode n'est donc pas la plus intéressante sur de faibles quantités de données, pour lesquelles les méthodes usuelles de recherche de motifs avec ou sans index sont plus efficaces. Par exemple, la recherche d'un mot situé à la fin d'un long texte sera bien plus efficace si l'on utilise une recherche par index plutôt que notre méthode.

2.2 Optimisations par transformation des grammaires

Comme nous l'avons expliqué précédemment, nous avons conservé la structure en sous-graphes afin d'éviter l'explosion des tailles des grammaires. Nous avons ensuite présenté les techniques que nous avons utilisées pour accélérer l'application de nos grammaires en optimisant certains calculs. Nous allons maintenant présenter deux autres techniques d'optimisation, qui opèrent sur les grammaires et non plus sur le programme de recherche de motifs. Nous présenterons tout d'abord les pistes de recherche que nous avons suivies en nous inspirant de l'algorithme de mise en forme normale de Greibach, puis nous montrerons une méthode simple qui réalise un compromis entre notre représentation avec sous-graphes et les transducteurs à états finis.

2.2.1 Le format FST3

Le non-déterminisme de nos grammaires provient de l'utilisation de motifs tels que `<MOT>` ou `<A:fs>` ainsi que des appels aux sous-graphes. Nous avons vu dans la section précédente que des pré-calculs sur les motifs permettaient de les comparer au texte de façon très efficace. En revanche, nous n'avons jusque-là pas tenté de remédier au problème posé par la présence des sous-graphes. Pour mesurer l'étendue de ce problème, considérons le graphe de la figure 2.15. Il constitue le graphe principal d'une grammaire non publiée de Maurice Gross qui décrit des expressions de dates en français. Comme on le voit, ce graphe est constitué en majeure partie de références à des sous-graphes. Cela signifie que lors de l'application de cette grammaire à un

texte, le programme devra explorer tous ces sous-graphes pour chaque position dans le texte. Or, dans cette grammaire, plusieurs sous-graphes ont des préfixes communs, en particulier des prépositions. Si le mot courant dans le texte n'est pas une préposition, le programme effectuera en vain plusieurs fois les mêmes tests. Le problème est donc dû à l'absence de visibilité du contenu des sous-graphes.

Nous avons donc tenté d'améliorer la situation en suivant le principe de l'algorithme de mise en forme normale de Greibach ([30]). Le principe est de transformer la grammaire de sorte que chaque sous-graphe débute par des transitions autres que des appels à des sous-graphes. En d'autres termes, nous faisons remonter les transitions étiquetées par des symboles terminaux de façon à pouvoir comparer au plus tôt ces transitions avec le texte. Ainsi, dans l'exemple précédent, nous pourrions tester immédiatement si le texte contient une préposition, et n'explorer les sous-graphes que si ce test est positif, ce qui permettrait de réduire les coûts d'exploration de la grammaire.

Le problème qui se pose à nous est que l'algorithme de Greibach s'applique à des grammaires algébriques représentées sous forme de listes de règles de dérivation. La première différence entre ce modèle et nos grammaires est que celles-ci possèdent des automates comme membres droits des règles, ce qui fait d'elles des grammaires algébriques étendues. Ce problème pourrait être résolu en effectuant dans un premier temps la conversion des automates en expressions régulières, puis en transformant ces expressions régulières en une grammaire algébrique non étendue au moyen de simples règles de réécriture telles que celles présentées dans le tableau 2.3. Nous pourrions également effectuer directement la mise en forme normale de Greibach en utilisant la méthode présentée dans [2].

Règle d'origine	Réécriture
$A \rightarrow expr1 + expr2$	$A \rightarrow expr1 \vert expr2$
$A \rightarrow expr1.expr2$	$A \rightarrow BC$ $B \rightarrow expr1$ $C \rightarrow expr2$
$A \rightarrow expr1^{\star}$	$A \rightarrow B$ $B \rightarrow BB \vert expr1 \vert \varepsilon$
$A \rightarrow (expr1)$	$A \rightarrow expr1$
$A \rightarrow a$	$A \rightarrow a$
$A \rightarrow B$	$A \rightarrow B$

TAB. 2.3 – Conversion d'une grammaire algébrique étendue en une grammaire algébrique non étendue

Toutefois, aucune de ces deux solutions ne nous semble satisfaisante, car elles changeraient la nature des objets sur lesquels nous travaillons, puisque nous perdrions les structures d'automates que nous avions, ce qui nécessiterait de modifier tous les programmes manipulant des grammaires. En outre, les algorithmes mis en oeuvre ont été conçus pour s'appliquer sur des

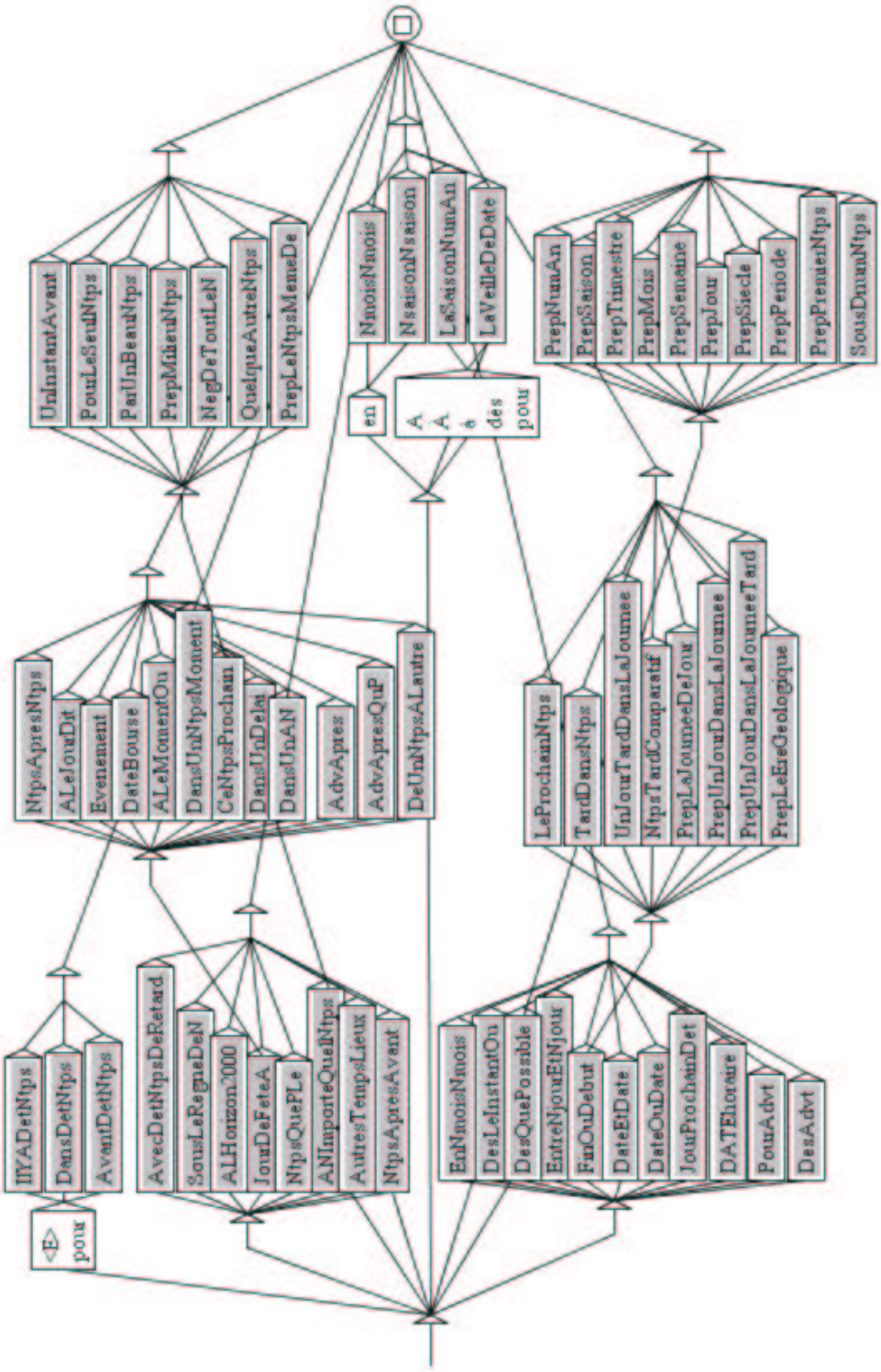

FIG. 2.15 – Graphe *DATE*

grammaires algébriques étendues dans lesquelles les membres droits sont des chaînes ou des expressions régulières. Étant donné que les membres droits de nos grammaires peuvent être des automates quelconques, il est probable que les expressions régulières équivalentes seront très complexes. Ainsi, bien que nous ayons une garantie théorique d'équivalence entre nos grammaires et les grammaires algébriques usuelles, nous aurons en pratique des difficultés pour passer d'un format à l'autre, car les algorithmes nécessaires à la mise en forme normale de Greibach ne sont pas optimisés pour ce type de données.

À cela vient s'ajouter un second problème, car nos grammaires ne sont pas exactement des grammaires algébriques étendues, du fait qu'elles peuvent contenir des sorties. La définition formelle de ces objets, que nous appelons *transducteurs algébriques étendus*, est la suivante :

Définition : un transducteur algébrique étendu au format FST3 est défini par un n-uplet $(E, I, F, \Sigma, \Omega, S, \delta)$. E représente l'ensemble des états. On appelle *état sous-initial* tout état qui apparaît comme étiquette d'au moins une transition du transducteur, ce qui signifie alors que cette transition désigne un appel récursif au sous-transducteur obtenu en prenant cet état comme état initial. I représente l'ensemble des états sous-initiaux du transducteur. F représente l'ensemble des états terminaux. Σ désigne l'alphabet d'entrée, Ω désigne l'alphabet de sortie. S est un état appartenant à I qui désigne l'axiome de la grammaire, c'est-à-dire l'état initial du transducteur. δ est la fonction de transition, et on a :

$$\delta : E \times (I \cup ((\Sigma \cup \{\varepsilon\}) \times (\Omega \cup \{\varepsilon\}))) \rightarrow E$$

Cette définition inclut les grammaires FST2, qui sont les objets de départ de notre transformation. Les grammaires FST2 correspondent en effet aux graphes tels qu'on peut les dessiner avec l'éditeur de graphes d'Unitex : les noms des graphes sont les symboles non terminaux, et les autres étiquettes sont les symboles terminaux, et la sortie associée par défaut à une entrée est le mot vide. Toutefois, les grammaires FST2 sont sujettes à une restriction supplémentaire. En effet, dans une grammaire FST2, les sous-grammaires obtenues à partir des états sous-initiaux sont disjointes et correspondent exactement aux différents sous-graphes qui composent la grammaire.

La solution que nous avons adoptée consiste à transposer le principe de Greibach à ces transducteurs algébriques étendus, tout en conservant la structure en sous-graphes. Notre but est donc de transformer la grammaire de telle sorte que chaque sous-graphe débute par des transitions terminales. Nous dirons alors que la grammaire est en forme normale de Greibach. Nous avons mis au point dans ce but un algorithme qui se divise en 3 étapes.

Étape 1 : construction des pseudo-sous-graphes

Pour chaque sous-graphe G, on construit un pseudo-sous-graphe G' équivalent à G. G' reconnait le même langage que G, et décrit les premières transitions de G suivies par des transitions étiquetées par les états pointés par ces premières transitions. G' contient donc des appels à des sous-transducteurs inclus dans G. Avant l'application de notre transformation, l'ensemble I contient donc exactement les états initiaux des sous-graphes. Nous noterons I_0 cet ensemble

de départ. Nous noterons I_1 l'ensemble des états correspondant aux états initiaux des pseudo-sous-graphes.

```
      Première étape :
0     Initialiser I1 avec l'ensemble vide
1     Pour chaque sous-graphe G, défini par un élément de I0
2           Soit e l'état initial de G
3           Créer un état e' possédant la même finalité que e
4           Ajouter l'état e' dans I, dans I1 et dans E
5           Créer un état final z
6           Ajouter l'état z dans E
7           Pour chaque transition (e, α, f)
8                 Créer un état f', en le définissant comme final si et seulement si
9                  f est final
10                Ajouter l'état f' dans E
11                Créer une transition (e', α, f')
12                Créer une transition (f', f, z)
13                Ajouter l'état f dans I
14          Fin Pour
15    Fin Pour
16    Prendre comme nouvel axiome de la grammaire la copie de l'ancien axiome
17    Fin.
```

Afin d'illustrer les explications, nous avons pris un exemple de grammaire, présenté sur la figure 2.16, à laquelle nous avons appliqué successivement les différentes étapes de notre algorithme. Les conventions de lecture sont les suivantes :

- l'axiome de la grammaire est représenté par l'état dans lequel entre une flèche sans origine ;
- les états finaux sont symbolisés par des doubles cercles ;
- les flèches pleines représentent des transitions étiquetées par des symboles terminaux. Les symboles étiquetant ces flèches sont des symboles d'entrées, car pour simplifier l'exemple, nous avons considéré une grammaire sans sortie, et nous avons omis d'associer systématiquement la sortie ε aux entrées ;
- les flèches en pointillés représentent des transitions étiquetées par des états. Une flèche étiquetée par un état x correspond à un appel récursif au sous-automate obtenu en prenant x comme état initial.

Ainsi, la grammaire de la figure 2.16 est composée de 2 sous-graphes : A, qui est le graphe principal, et B. La flèche en pointillés étiquetée par le numéro 4 indique qu'il s'agit d'une transition par l'état numéro 4 ; en d'autres termes, cela signifie que A fait appel au sous-graphe B.

Après l'étape 1, on obtient la grammaire de la figure 2.17. On peut voir que l'axiome de la grammaire est maintenant l'état initial du sous-graphe A'.

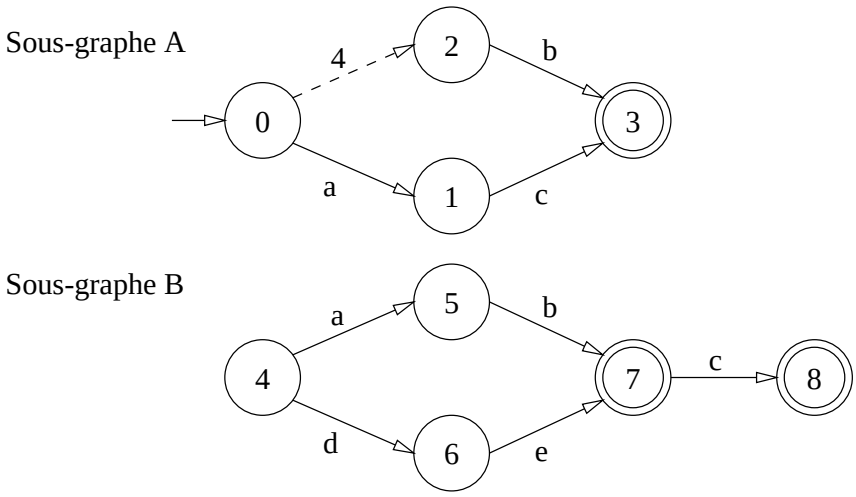

FIG. 2.16 – Grammaire avant l'étape 1

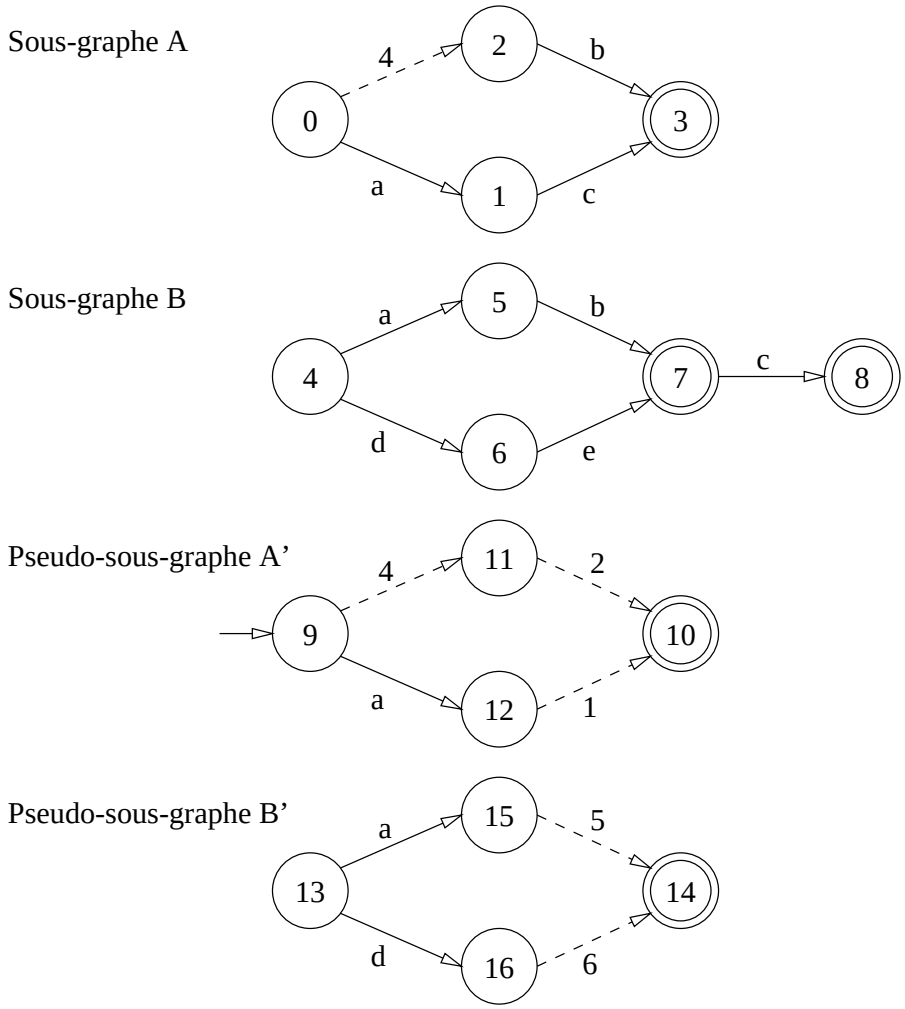

FIG. 2.17 – Grammaire après l'étape 1

Étape 2 : suppression des transitions non-terminales parmi les premières transitions des pseudo-sous-graphes.

```
    Deuxième étape :
0   Pour chaque pseudo-sous-graphe G, défini par un élément de I_1
1       Tant que il existe une transition (e, α, f) avec α ∈ I telles que e
2       soit l'état initial de G ou soit accessible depuis l'état initial de G par
3       des ε-transitions
4           Supprimer la transition (e, α, f)
5           Si l'état α est final Alors créer une transition (e, ε, f)
6           Créer un état non final k
7           Ajouter l'état k dans E
8           Créer une transition (e, ε, k)
9           Pour toutes les transitions (α, β, l)
10              Créer un état non final z
11              Ajouter l'état z dans E
12              Créer une transition (k, β, z)
13              Créer une transition (z, l, f)
14              Ajouter l'état l à I
15          Fin Pour
16      Fin Tant que
17  Fin Pour
18  Fin.
```

Nous allons maintenant éliminer certaines transitions non-terminales, de façon à ce que toutes les premières transitions des pseudo-sous-graphes soient terminales. Nos pseudo-sous-graphes seront alors en forme normale de Greibach aux ε-transitions près. Pour cela, nous remplaçons chaque transition non-terminale par un sous-automate qui explicite les premières transitions de l'état pointé. Nous itérons ensuite le procédé jusqu'à ce qu'il n'y ait plus aucune transition non-terminale au début des pseudo-sous-graphes. Pour que cet algorithme s'arrête, il faut que la grammaire ne soit pas récursive à gauche, et notre algorithme ne s'applique que s'il n'en existe pas. Le compilateur de graphes détecte de telles récursions.

La figure 2.18 montre la grammaire de la figure 2.17 après l'étape 2. Le pseudo-sous-graphe A' a été modifié car il y avait une transition non-terminale qui sortait de son état initial. La transition $(9, 4, 11)$ a donc été remplacée par une ε-transition reliant l'état 9 au groupe d'états 17, 18 et 19. On peut voir que les transitions sortant de l'état 17 correspondent aux premières transitions de l'état 4. Les états 18 et 19 sont reliés à l'état 11 par des transitions étiquetées par les états 5 et 6.

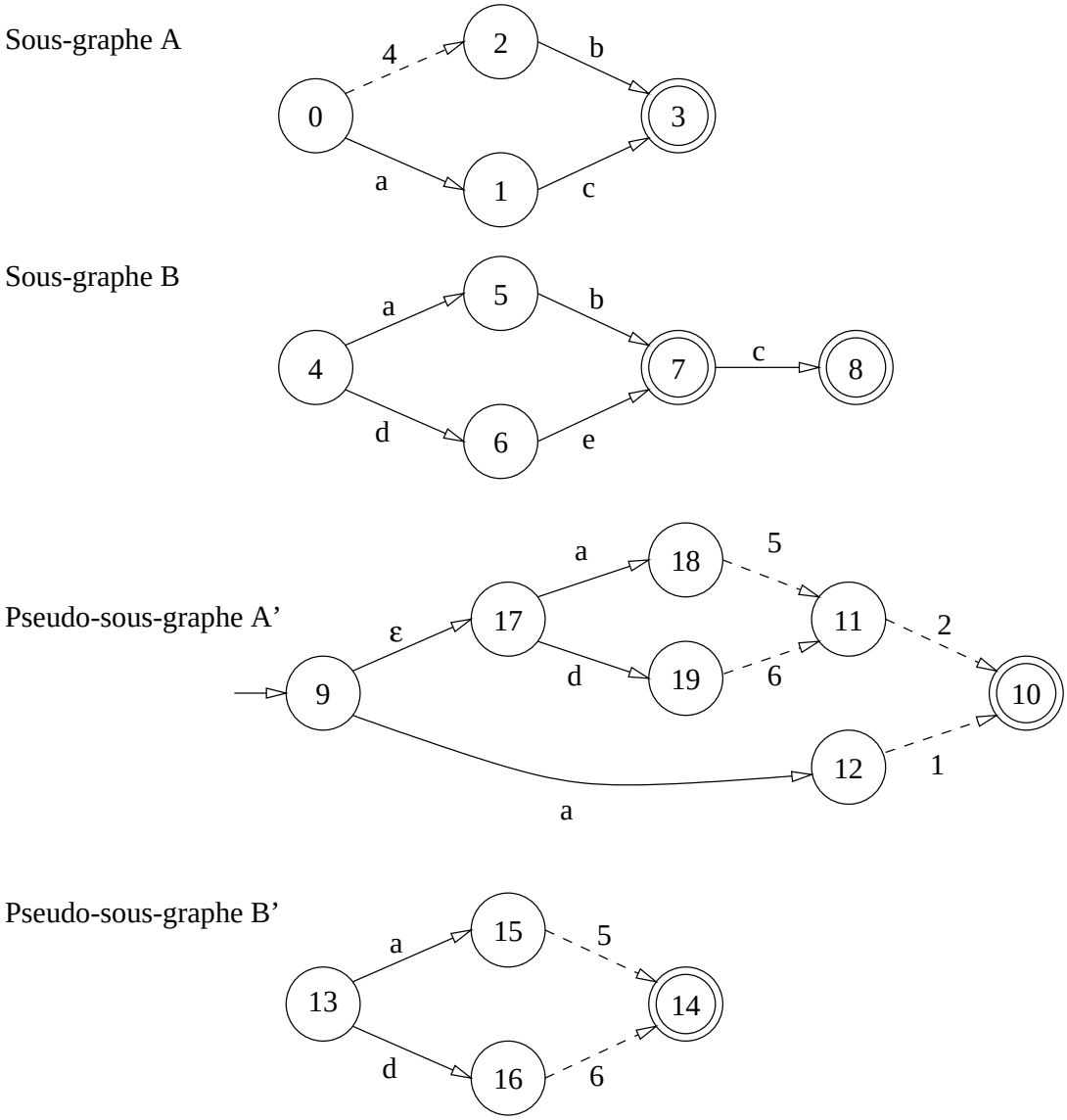

FIG. 2.18 – Grammaire après l'étape 2

Étape 3 : suppression des ε-transitions, déterminisation et modification des sous-graphes

La dernière phase de l'algorithme consiste à nettoyer la grammaire obtenue après l'étape 2. Pour cela, on commence par supprimer les ε-transitions, puis on déterminise les pseudo-sous-graphes. Enfin, on remplace dans les sous-graphes les transitions correspondant à des appels de sous-graphes par des transitions étiquetées par les états initiaux des pseudo-sous-graphes équivalents. Cette dernière opération nous permet de remplacer les appels aux sous-graphes par des appels aux pseudo-sous-graphes équivalents, qui sont eux en forme normale de Greibach.

```
      Troisième étape :
0     Supprimer les ε-transitions dans les pseudo-sous-graphes
1     Déterminiser les pseudo-sous-graphes
2     Pour chaque sous-graphe G, défini par un élément de I_0
3           Pour toutes les transitions (a, α, b) avec α ∈ I_0
4                 Remplacer la transition par une transition (a, β, b) où β est l'état
5                 initial du pseudo-sous-graphe correspondant au sous-graphe pointé
6                 par α
7           Fin Pour
8     Fin Pour
9     Fin.
```

La figure 2.19 montre la grammaire obtenue après l'étape 3. On peut y voir que le pseudo-sous-graphe A' a été déterminisé et que la transition $(0, 4, 2)$ a été transformée en $(0, 14, 2)$, car 4 correspondait au début du sous-graphe B, et 14 correspond au début du pseudo-sous-graphe équivalent B'. On peut noter que toutes les premières transitions des pseudo-sous-graphes sont des transitions terminales. De cette façon, nous avons obtenu une grammaire équivalente à la grammaire d'origine, dans laquelle nous disposons au début de chaque sous-graphe d'une transition terminale, ce qui nous permet de conditionner l'exploration des sous-graphes par une comparaison entre une transition terminale et le contenu de l'entrée. Nous sommes donc à l'abri des situations défavorables telles que celle présentée sur la figure 2.15.

Pour tester cette méthode, nous avons comparé les temps d'application de grammaires transformées et non transformées. Pour cela, nous avons utilisé le programme AGLAE auquel nous avons intégré un programme permettant d'effectuer la transformation des grammaires. Le tableau 2.4 montre les différences de temps de calcul que nous avons mesurées sur un ordinateur pourvu d'un Pentium 500 MHz et de 128 méga-octets de mémoire.

	Phrases économiques (281 sous-graphes, corpus = 4,7 Mo)	Expressions de pourcentages (16 sous-graphes, corpus = 125 Mo)	Expressions de date (94 sous-graphes, corpus = 125 Mo)
FST2	2 min 28 s	11 min 15 s	41 min 40 s
FST3	31 s	1 min 59 s	8 min 11 s

TAB. 2.4 – Comparaison entre les temps d'application de grammaires aux formats FST2 et FST3

Comme on peut le voir, cette transformation des grammaires diminue sensiblement les temps d'application. Le gain de vitesse est donc un point fort de cette méthode, qui présente également l'avantage de ne pas faire exploser la taille de la grammaire transformée. En effet, le fait de conserver des transitions sur des parties des sous-graphes d'origine au lieu de les

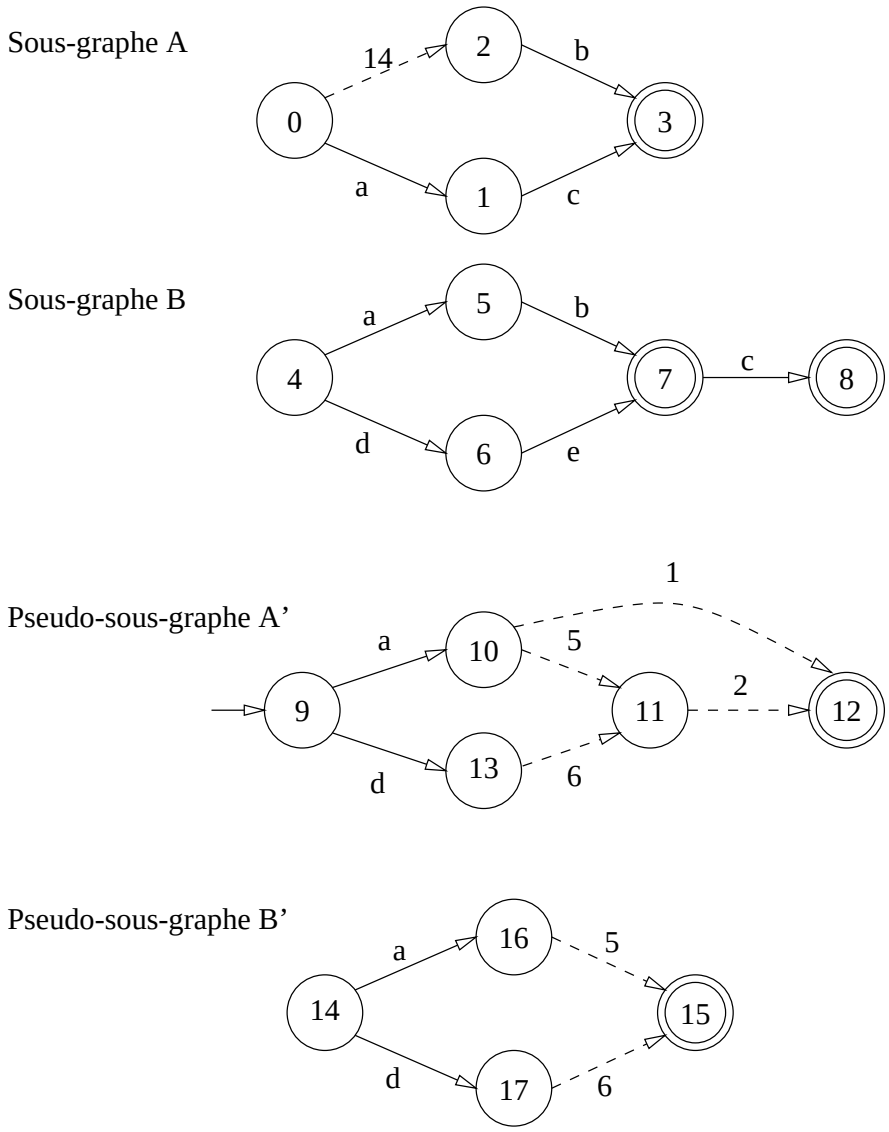

FIG. 2.19 – Grammaire après l'étape 3

dupliquer nous permet de ne créer qu'un petit nombre d'états dans chaque pseudo-sous-graphe. Cette méthode présente toutefois des inconvénients. Tout d'abord, elle ne prend en compte que les transitions apparaissant au début des sous-graphes, laissant de côté les appels à des sous-graphes se situant à des endroits quelconques de la grammaire. Dans certains cas de figure, en particulier lorsqu'il s'agit de grammaires générées à partir de tables de lexique-grammaire, la méthode trouve donc ses limites. Il faudrait donc l'étendre pour gérer des situations plus complexes. Le second inconvénient est que les grammaires transformées sont illisibles pour des humains. En effet, le fait de ne pas dupliquer des parties de sous-graphes mais d'y faire référence, produit des grammaires où les sauts récursifs sont enchevêtrés, à tel point

que même pour une grammaire simple, la grammaire FST3 produite est presque impossible à appréhender pour un humain, car il est très difficile d'en synchroniser le contenu avec la grammaire d'origine. En fait, il s'agit d'un problème très similaire à celui de la localisation d'instructions dans un code optimisé. La complexité des techniques mises en oeuvre dans ce domaine ([78]) nous donne une indication sur la difficulté qu'il peut y avoir à tenter de lire des grammaires FST3. Cet état de fait peut donc avoir des conséquences sur l'écriture de programmes destinés à manipuler de tels objets, car le débogage d'applications portant sur des grammaires non triviales risque d'être une opération particulièrement critique.

En conclusion, nous pensons que cette méthode est bien adaptée aux grammaires lexicales, mais qu'il sera nécessaire de l'améliorer pour gérer des cas figures particuliers, comme celui des grammaires générées automatiquement, et donc, présentant de fortes régularités. Nous pensons également qu'il serait possible d'étendre la transformation de façon à prendre en compte plus d'une transition terminale. Dans ce cas, si l'on veut éviter de créer trop d'états, il faudra peut-être envisager des structures de prédictions telles que les arbres syntaxiques partiels présentés dans [10], ce qui aurait toutefois comme inconvénient de devoir modifier les programmes en plus des grammaires.

2.2.2 Méthode par aplatissement contrôlé

Comme nous l'avons déjà expliqué, nous avons renoncé au format de transducteurs à états finis pour deux raisons : non équivalence avec les transducteurs algébriques et explosion potentielle de la taille. Toutefois, certaines applications comme la levée d'ambiguïtés utilisent des objets à états finis, de façon à pouvoir appliquer des algorithmes classiques comme l'intersection. Il nous fallait donc pourvoir notre système Unitex d'un programme permettant de passer d'une grammaire FST2 à un transducteur à états finis. Nous avons résolu ce problème d'une façon très simple en effectuant un aplatissement contrôlé des grammaires.

Cette opération consiste à remplacer chaque appel à un sous-graphe par le sous-graphe lui-même, jusqu'à une profondeur fixée. Si l'on atteint cette profondeur et que la grammaire contient encore des appels à des sous-graphes, il nous suffit de supprimer ou de laisser ces appels, suivant que l'on souhaite obtenir un résultat strictement à états finis ou non. Ainsi, nous pouvons demander au programme de construire un transducteur à états finis qui ne sera pas nécessairement équivalent à la grammaire d'origine, tout en contrôlant à quelle profondeur s'effectuera éventuellement l'approximation.

Au contraire, si l'on décide de laisser les appels aux sous-graphes restants, on obtient alors une grammaire strictement équivalente à la grammaire d'origine, mais dans laquelle on a supprimé des appels à des sous-graphes, diminuant ainsi le non-déterminisme. Notre programme, baptisé `Flatten`, prend donc en paramètre une profondeur x et une valeur indiquant si l'on souhaite préserver ou non l'équivalence avec la grammaire d'origine. Le tableau 2.5 décrit les résultats que l'on peut obtenir en fonction des résultats souhaités.

Grâce à ce programme, nous pouvons donc obtenir des transducteurs à états finis lorsque cela est nécessaire, mais nous pouvons également améliorer nos grammaires. En effet, en

```
      Aplatissement :
0     p ← profondeur maximum
1     Tant que p > 0 ET que le graphe principal contient des appels à des sous-
2     graphes
3           Pour chaque transition (e, α, f) du graphe principal tq α soit un appel
4           à un sous-graphe ET que cette transition n'ait pas été créée à l'étape p
5                 Supprimer cette transition
6                 Cloner le sous-graphe pointé par α
7                 Soit e' l'état initial de ce clone
8                 Créer une transition (e, ε, e')
9                 Pour chaque état final f' du clone
10                      Créer une transition (f', ε, f)
11                Fin Pour
12          Fin Pour
13          p ← (p − 1)
14    Fin Tant que
15    Si il reste des appels à des sous-graphes dans le graphe principal
16    ET que le résultat doit être un transducteur à états finis
17    Alors on supprime tous ces appels
18    Fin Si
19    Supprimer les ε-transitions
20    Déterminiser le graphe principal
21    Fin.
```

aplatissant nos grammaires, nous pouvons en réduire le non-déterminisme, et ce, sur toute la surface des sous-graphes considérés au contraire de la transformation en FST3 qui n'opère que sur les premières transitions. De plus, la grammaire obtenue possède exactement le même format que la grammaire d'origine contrairement au format FST3 pour lequel nous avions dû généraliser la notion d'appel à des sous-graphes. En revanche, cette méthode n'offre aucune garantie quant à une possible explosion en taille. Il appartient donc au concepteur d'une grammaire d'établir empiriquement la profondeur jusqu'à laquelle il peut aplatir sa grammaire sans que celle-ci explose. Le tableau 2.6 présente quelques comparaisons de vitesse et de taille entre des grammaires FST2 et leurs équivalents aplatis avec différentes limites de profondeur.

Comme on peut le voir, les résultats sont très variables suivant la nature de la grammaire, et peuvent même devenir défavorables au-delà d'une certaine limite comme c'est le cas pour les groupes nominaux humains à la profondeur 2. La raison en est que si le temps de calcul est amélioré, la taille de la grammaire augmente tellement que les prétraitements sur la grammaire deviennent très coûteux. Cependant, dans les cas favorables, le gain peut être appréciable comme c'est le cas pour cette même grammaire à la profondeur 1 puisque l'on passe de 1 min 56 s à 17 s, ce qui représente une accélération par un facteur d'environ 7. En conclusion, cette transformation des grammaires ne peut pas être appliquée automatiquement, car ses effets ne sont pas prédictibles, pouvant même nuire aux performances dans certains cas. Nous

Objet souhaité	Objet obtenu dans	
	le meilleur des cas	le pire des cas
Transducteur à états finis (FST)	FST équivalent	FST non équivalent
Transducteur algébrique (FST2) ou FST, mais équivalent à la grammaire d'origine	FST équivalent	FST2 équivalent

TAB. 2.5 – Résultats possibles du programme `Flatten`

Grammaire / Taille du corpus	FST2	Flatten profondeur=1	Flatten profondeur=2	Flatten profondeur=3
Groupes nominaux généraux / 844 Ko	25 Ko / 19 sec	27 Ko / 19 sec	116 Ko / 18 sec	1149 Ko / 13 sec
Groupes nominaux humains /5,4 Mo	75 Ko / 1 min 56 s	453 Ko / 17 sec	11 Mo / 39 sec	explosion de la taille
Grammaire du mot *health* /5,4 Mo	52 Ko / 12 s	102 Ko / 10 sec	102 Ko / 10 sec	102 Ko 10 sec

TAB. 2.6 – Comparaisons entre grammaires aplaties et non aplaties

suggérons donc de la considérer comme un bonus potentiel, qui doit être testé pour chaque grammaire afin d'en évaluer les effets, bons ou mauvais.

2.3 Une analyse en plusieurs passes

Nous avons vu jusqu'à maintenant des techniques permettant d'accélérer l'application de grammaires aux textes, soit en optimisant le programme de recherche de motifs, soit en appliquant des transformations aux grammaires. Nous allons maintenant envisager une autre façon d'améliorer les temps de calculs, qui est destinée à être appliquée aux grammaires générées à partir des tables de lexique-grammaire. Nous sommes partis de l'observation suivante : dans ces grammaires, on trouve de nombreux appels aux mêmes sous-graphes aux mêmes positions. Par exemple, une grammaire telle que celle présentée sur la figure 1.43 (page 82), engendre une collection de graphes qui ont tous en commun d'avoir un appel au sous-graphe `Insert` entre le verbe et son complément. Notre idée consiste donc à tirer parti de telles redondances en évitant de d'explorer plusieurs fois un même sous-graphe au même endroit.

Pour y parvenir, il suffit d'effectuer une analyse en plusieurs passes. Dans une première phase, nous allons appliquer une ou plusieurs grammaires au texte, en notant les différentes séquences reconnues. Nous allons ensuite appliquer la grammaire principale, dans laquelle nous aurons remplacé les appels aux sous-graphes appliqués en première passe par des références aux séquences reconnues. Ainsi, nous aurons additionné les temps d'application des différentes grammaires au lieu de les combiner de façon exponentielle.

En pratique, nous pouvons appliquer cette méthode avec nos outils en procédant de la

manière suivante (nous supposerons que l'on veut opérer sur le sous-graphe `Insert`) :

1. Retirer toutes les transductions éventuellement présentes dans le sous-graphe `Insert`
2. Ajouter une ε-transduction juste avant l'état terminal du sous-graphe `Insert` qui produise en sortie la séquence `,.Insert`
3. Appliquer cette grammaire modifiée au texte en mode MERGE (insertion des productions dans le texte d'entrée)
4. Construire la concordance des séquences reconnues en prenant des contextes gauches et droits de longueur nulle
5. Fusionner la liste obtenue aux dictionnaires du texte
6. Modifier la grammaire principale en remplaçant tous les appels à `Insert` par le motif `<Insert>`
7. Appliquer cette grammaire au texte.

L'astuce consiste à utiliser les dictionnaires du texte pour stocker les séquences reconnues. Les étapes 2, 3 et 4 servent à produire une concordance ayant exactement la forme d'un dictionnaire DELAF. L'étape 1 sert à s'assurer que la grammaire ne contienne pas de transductions qui puissent interférer avec la production des entrées de dictionnaire. L'étape 5 incorpore les lignes obtenues aux dictionnaires du texte, ce qui fait que la prochaine application d'une grammaire au texte va prendre ces entrées en compte. L'étape 6 consiste simplement à consulter au moyen d'un motif les informations qui ont été stockées dans les dictionnaires.

Comme on le voit, cette méthode est peu coûteuse car elle ne nécessite pas le moindre effort de programmation. Toutefois, nous pourrions envisager d'effectuer les modifications dans les grammaires au moyen de scripts simples afin d'automatiser entièrement le procédé. Notons que cette méthode peut être itérée afin de simplifier les calculs par un mécanisme de cascade de transducteurs. Ce principe de cascade a déjà été exploité en extraction d'information par des systèmes comme Fastus ([4]). Enfin, nous pouvons remarquer que ce principe de décomposition en plusieurs passes permet d'optimiser chaque étape en appliquant la transformation la plus adaptée à chaque grammaire (FST3 ou aplatissement).

Toutefois, étant donné que cette méthode est destinée à fonctionner sur des ensembles de grammaires dont nous n'avons pour l'instant que de très petites approximations, nous n'avons pas pu la tester sur des jeux de données significatifs. Il s'agit donc pour l'instant d'une voie de recherche ouverte qu'il faudra explorer lorsque nous disposerons de masses de données suffisantes.

2.4 Compilation des grammaires en fonction du lexique

Nous avons vu au chapitre 1 que la description de quelques formes de phrases pour les verbes simples d'une trentaine de tables de lexique-grammaire générait 98 000 graphes. En tenant compte d'autres formes de phrases, d'autres verbes simples et d'autres constructions composées, nous devons nous attendre à une augmentation importante de ce chiffre, et ce, pour

les seuls verbes. Si l'on ajoute les grammaires obtenues pour les noms et les adjectifs, il est fort problable que nous aurons à gérer à long terme une base de données de plusieurs millions de graphes. Cet ordre de grandeur dépassant celui auquel nous travaillons actuellement, nous devons nous assurer avant d'entreprendre l'accumulation systématique de graphes que leur quantité ne nuira pas à leur exploitation.

Nous avons envisagé une méthode permettant de s'affranchir de ce problème. Elle repose sur un principe simple : les grammaires générées à partir les tables peuvent être filtrées par le lexique du texte sur lequel on veut les appliquer. En effet, contrairement aux grammaires locales conçues manuellement (grammaires de dates, d'incises, etc), les grammaires générées automatiquement sont centrées autour des entrées des tables, c'est-à-dire autour d'éléments lexicaux. Lors de l'application de ces grammaires à un texte, nous pouvons donc écarter toutes celles qui sont basées sur des éléments absents du texte. Ainsi, si le texte ne contient aucun mot pouvant être une forme du verbe *pleuvoir*, il est inutile de chercher à appliquer toutes les grammaires liées à ce verbe.

L'idée consiste donc à générer une grammaire *ad hoc* pour le texte que l'on souhaite analyser. Pour cela, il suffira de sélectionner dans la base de données tous les graphes concernés par les éléments lexicaux du texte. Considérons par exemple la phrase suivante :

Je suis énervé de devoir partir.

Les ambiguïtés lexicales de cette phrase sont représentées sur l'automate de la figure 2.20. On peut y voir que les mots *suis*, *énervé*, *devoir* et *partir* sont des formes potentielles des verbes *être*, *suivre*, *énerver*, *devoir* et *partir*. Si l'on fait l'union des grammaires de chacun de ces verbes, nous serons sûr de pouvoir analyser cette phrase.

L'avantage de cette méthode est donc de réduire la grammaire aux formes potentielles présentes dans le texte. Toutefois, le gain obtenu peut être maigre si le texte est très gros, car le nombre de formes alors présentes risque alors de devenir important. Il faudra donc peut-être adapter le procédé de façon à découper le texte et à générer une grammaire adaptée à chaque segment. Cela aurait pour conséquence de remettre en cause les programmes d'application des grammaires, mais offrirait en retour la garantie de pouvoir traiter des textes de taille quelconque.

2.5 Architecture générale de l'analyse syntaxique

Après avoir examiné les problèmes soulevés par la description des schémas de phrases simples au moyen de graphes et ceux posés par l'exploitation informatique de ces données, nous allons maintenant proposer une architecture d'analyseur syntaxique. Dans la mesure où nous sommes loin de disposer de toutes les données nécessaires pour pouvoir expérimenter ce modèle de façon significative, il ne n'agit pour l'instant que d'un programme de travail qu'il faudra adapter en fonction des données disponibles.

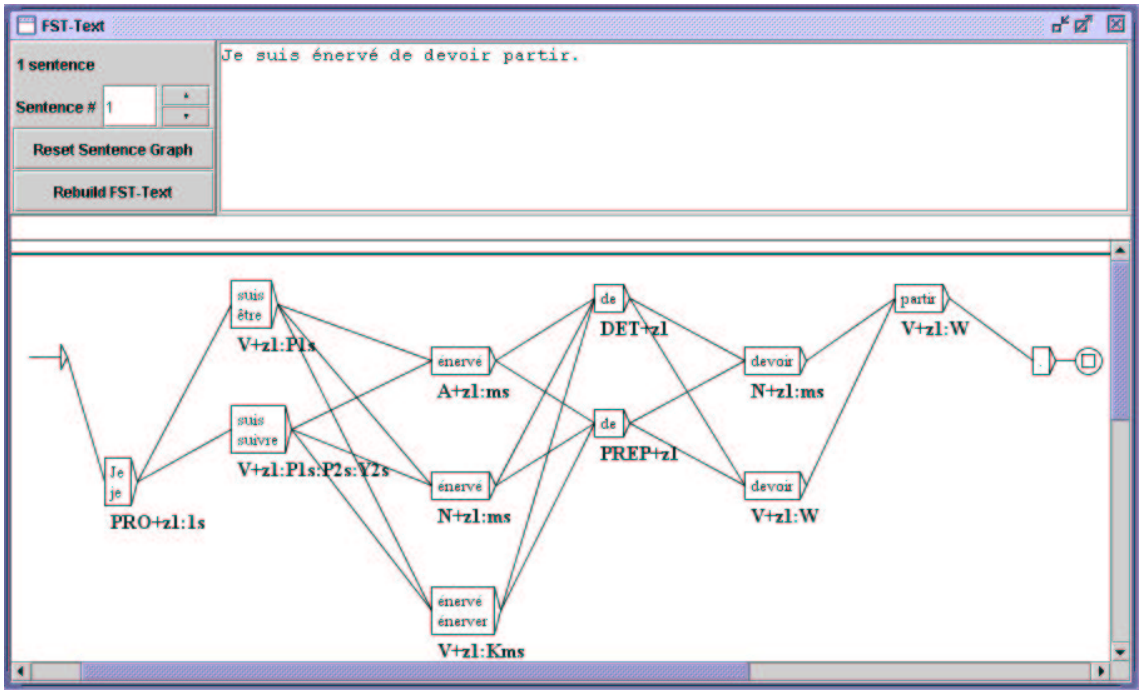

FIG. 2.20 – Ambiguïtés lexicales de la phrase *Je suis énervé de devoir partir.*

Notre architecture est résumée par l'organigramme de la figure 2.21. Selon nous, la meilleure façon de représenter les différentes analyses possibles d'une phrase serait de les montrer sous forme d'automate. Il sera donc nécessaire de passer d'un texte à sa représentation sous forme d'automates de phrases. Toutefois, nous ne sommes pas encore en mesure de déterminer à quelle étape de l'analyse cette conversion devra être effectuée. En effet, les programmes existants à l'heure actuelle fonctionnent sur le texte linéaire et non sur les automates de phrases. Il nous est donc pour l'instant impossible d'effectuer des tests pour déterminer l'étape à laquelle nous devrons travailler sur les automates de phrases.

Il faut également noter que les données linguistiques sur lesquelles s'appuie ce système sont loin d'être exhaustives. Il faudra prendre en compte d'autres structures dans les graphes patrons, étendre ces graphes à d'autres tables, étudier et coder certains phénomènes tels que la mobilité des constituants dans des expressions composées, etc. De plus, certains programmes tels que celui permettant de vérifier des contraintes ne sont pas encore disponibles. Il faudra donc attendre une évolution des données linguistiques et des outils avant de pouvoir tester à grande échelle cette stratégie d'analyse syntaxique, et éventuellement l'étendre à des constructions plus complexes.

2.6 Application des grammaires sur corpus

Bien que nous ne disposions pas de toutes les données nécessaires pour mener des tests exhaustifs, nous avons tenté d'appliquer nos grammaires sur corpus. Nous présenterons nos

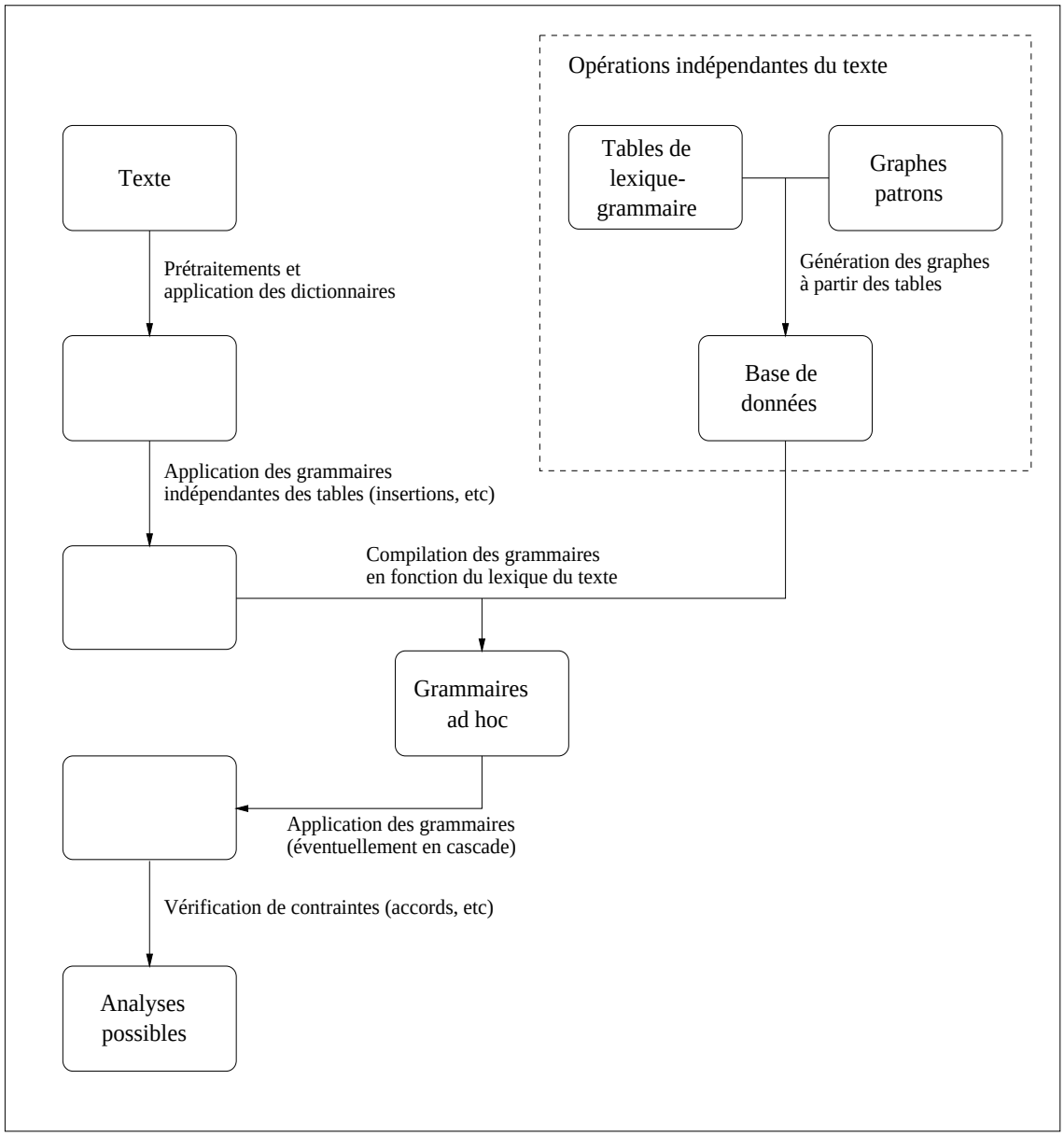

FIG. 2.21 – Architecture générale de l'analyse syntaxique

premiers résultats expérimentaux et nous discuterons de la possibilité d'améliorer ces résultats, à travers l'examen des erreurs rencontrées.

2.6.1 Premiers résultats

Nous avons appliqué nos grammaires sur corpus afin d'évaluer les résultats qu'elles produisaient en reconnaissance. Étant donné que nous avons pris le parti de prendre des grammaires

de groupes nominaux et d'insertions très permissives, nous n'avons quasiment pas de silence dans nos résultats, les exceptions concernant essentiellement des noms propres. Nous pouvons donc estimer que les concordances obtenues contiennent la quasi-totalité des occurrences cherchées, auxquelles s'ajoutent les erreurs.

La figure 2.22 montre un extrait d'une concordance obtenue à l'aide de la grammaire des phrases déclaratives générée à partir de la table 32NM ; le corpus est le roman *Le tour du monde en 80 jours* de Jules Verne. Sur les 461 occurrences trouvées, nous constatons qu'il y a 279 erreurs d'analyse et 182 analyses correctes ou presque correctes. Les erreurs d'analyse se divisent de la manière suivante :

1. bruit causé par les grammaires de groupes nominaux et d'insertions (156 occurrences) :
   {S} Mais il comprenait comment il serait reçu par
   {S} En tenant compte des arrêts, il ne parcourait
   vapeurs qui lui donnaient une apparence spectrale ? {S}Les

2. tentatives d'analyse d'expressions figées (75 occurrences) :
   consul, il a l'air d'un parfait honnête
   {S}Phileas Fogg ne eut pas le temps d'arrêter ce courageux garçon
   {S} Ce Fogg avait la détestable habitude de sauter d'un bateau

3. ambiguïtés entre le verbe *avoir* avec un *y* locatif et la structure *il y a* (26 occurrences) :
   eh bien, il y a du Yankee en vous ". {S}
   Dans la hutte, il y avait un Indien, et dans

4. erreurs dues à des majuscules faisant passer un *A* pour une forme du verbe *avoir* (20 occurrences) :
   FOGG ACHÈTE UNE MONTURE A UN PRIX FABULEUX {S}Le train était
   SUIT, AVEC UNE VITESSE DE VINGT MILLES A L'HEURE, UN COURS D'HISTOIRE

5. interprétation verbale incorrecte d'un mot en minuscules (2 occurrences) :
   laissait voir sur son visage les marques d'un vif mécontentement
   {S}Le fait dont il était question

Parmi les 182 formes correctes ou presque correctes, nous avons dénombré 72 analyses correspondant aux emplois des verbes de la table 32NM, contre 110 analyses correspondant à d'autres emplois des verbes. Sur ces 182 séquences, nous avons également noté que 129 d'entre elles correspondaient à des structures complètement reconnues, et que 53 analyses étaient incomplètes faute d'analyser entièrement un ou plusieurs arguments. Voici quelques exemples correspondant aux 4 cas :

Cas 1 : analyse complète et sens correct du verbe (46 occurrences)

```
Le balancier de l'horloge battait la seconde avec une régularité
mathématique
ce prétendu voyage en quatre-vingts jours pourrait bien cacher quelque
mission secrète
on comptait divers fonctionnaires civils et des officiers de tout grade
Smyth devint un audacieux banquier
Le défilé des bisons dura trois grandes heures
les nids comestibles forment un mets recherché dans le Céleste Empire
Phileas Fogg avait gagné son pari de vingt mille livres
```

Cas 2 : analyse complète et sens différent du verbe (83 occurrences)

```
Aouda accepta l'offre avec reconnaissance
ce est un torrent de chair vivante que aucune digue ne saurait contenir
Sir Francis Cromarty regardait toute cette pompe
cette main tenait un couteau ouvert
```

Sens de ces verbes dans la table 32NM :

accepter	*cette salle accepte 200 personnes*
contenir	*le flacon contient 1 litre*
regarder	*cette affaire regarde la justice*
	la maison regarde le lac
tenir	*ce vase tient trois litres*
	la colère tient Luc

Cas 3 : analyse incomplète et sens correct du verbe (26 occurrences)

```
la dépression accuse douze cents [pieds au-dessous]
Vous avez un bateau prêt [à partir]
on compte treize cent [dix milles]
l'horloge du salon sonna huit heures [quarante]
```

Cas 4 : analyse incomplète et sens différent du verbe (27 occurrences)

```
il aurait atteint la station [d'Omaha]
il fit un geste [pour saisir la carte jouée]
```

Sens de ces verbes dans la table 32NM :

atteindre	*Max atteint 80 kilos*
faire	*la table fait 3 mètres*
	Max fait le clown

FIG. 2.22 – Résultats de l'application de la grammaire *N0VN1(P)* générée pour la table 32NM au texte *Le tour du monde en 80 jours*

2.6.2 Analyse

En examinant les chiffres précédents, on voit que le résultat n'est pas très bon. Toutefois, il est possible de corriger un certain nombre d'erreurs.

Beaucoup d'erreurs dues aux définitions des groupes nominaux et des insertions pourraient être corrigées en améliorant la précision de ces grammaires. Ainsi, certaines analyses telles que

**(En tenant)$_{N_0}$ (compte)$_V$ (des arrêts)$_{N_1}$*

pourraient être écartées si l'on avait une définition correcte des groupes nominaux. En effet, l'erreur dans l'analyse précédente est due au fait que *En tenant* est considéré comme un nom propre suivi d'un adjectif à cause de la majuscule.

Par ailleurs, nous pourrions exercer des contraintes sur les séquences reconnues afin d'éviter des formes incorrectes telles que

**Léa aime il*

En effet, il suffirait d'interdire la présence du pronom *il* ailleurs qu'en position N_0 pour éviter de telles analyses erronées. Les contraintes d'accord permettraient également de filtrer certaines analyses incorrectes comme

**l'éléphant et lui donna une heure de repos*

Ici, la faute pourrait être évitée si l'on déterminait que *l'éléphant et lui* est un groupe nominal pluriel, car constitué d'une coordination de deux termes. Disposant de cette information, il serait aisé de rejeter l'analyse en vérifiant que le verbe est au singulier. Comme nous l'avons déjà évoqué en 1.3.7.6, nous pensons que de telles vérifications pourraient être faites au moyen d'un outil comme le programme ELAG ([51]).

Les erreurs dues à des interférences avec des expressions figées peuvent avoir deux origines. Tout d'abord, une grammaire de verbe simple peut reconnaître une partie d'une expression figée plus longue. Si l'on a une grammaire décrivant la structure N_0 *prendre* N_1, alors on ne reconnaîtra qu'une partie de la phrase

Paul prend le taureau par les cornes

Dans ce cas, le problème sera résolu en lorsque l'on appliquera cette grammaire en concurrence avec celle décrivant la structure N_0 *prendre le taureau par les cornes*, car seule la seconde grammaire permet d'analyser la phrase entière. Le second cas d'erreur survient lorsque les deux grammaires peuvent reconnaître la phrase entière :

(Paul)$_{N_0}$ (a)$_V$ (du retard)$_{N_1}$
(Paul)$_{N_0}$ (a du retard)$_V$

Nous sommes alors en présence d'un cas d'ambiguïté dans lequel, ne pouvant choisir une analyse plutôt que l'autre, la bonne solution consiste à présenter les deux. Nous ne considérons pas ces cas comme des erreurs à corriger mais plutôt comme des analyses multiples qu'il convient de toutes prendre en compte. Le problème est exactement le même en cas de concurrence entre plusieurs emplois d'un même verbe. Nous estimons donc qu'en l'absence d'indice syntaxique permettant de trancher, une phrase telle que

il gagna la station de Pauwell

devra être présentée comme ambiguë entre les différents emplois de *gagner* qui tolèrent la construction $N_0\ V\ N_1$.

Malgré tout, ces résultats sont très encourageants, car ils ont été obtenus avec des grammaires de groupes nominaux et d'insertions peu précises. Nous pensons qu'une description beaucoup plus fine de ces éléments permettra d'améliorer sensiblement les résultats. En effet, nos grammaires sont des squelettes de phrases simples que nous avons construit manuellement. Nous sommes donc sûr que les structures qui seront reconnues seront correctes, à condition que les arguments et les insertions soient analysés correctement. Il est donc tout à fait naturel que les approximations que nous avons faites ne permettent pas d'obtenir des résultats idéaux, et cela ne remet pas en cause nos descriptions des constructions des verbes simples. En fait, les erreurs d'analyse proviennent du fait que nous avons testé nos grammaires sur du texte réel et non sur un corpus de phrases *ad hoc* telles que *Luc mange une pomme.*

2.7 Conclusion

Nous avons présenté diverses façons d'appliquer les grammaires nécessaires à notre modèle d'analyse syntaxique. Nous avons tout d'abord décrit le fonctionnement de notre programme de recherche de motifs par grammaires, et nous avons montré comment l'on pouvait tirer parti d'une segmentation en unités lexicales pour optimiser certains calculs. Puis, nous avons décrit deux transformations qui, appliquées aux grammaires, permettent d'accélérer la recherche de motifs. Nous avons ensuite présenté une stratégie d'analyse en plusieurs passes au moyen d'une cascade de transducteurs mettant à profit certaines régularités des graphes que nous générons à partir des tables de lexique-grammaire. Enfin, nous avons montré une méthode simple permettant de nous affranchir des ordres de grandeurs qu'atteindront à long terme les bibliothèques de grammaires, obtenant ainsi la garantie que ces données ne seront pas accumulées en vain.

Nous sommes donc arrivés à la conclusion qu'il est possible d'exploiter les tables de lexique-grammaire pour faire de l'analyse syntaxique en les convertissant en une bibliothèque de graphes, et nous avons proposé un modèle d'architecture qui devrait permettre d'y parvenir. Les premiers résultats que nous avons obtenus nous ont montré que l'on pouvait appliquer de telles grammaires à des textes réels, et ce, même en utilisant de grossières approximations pour reconnaître certains éléments. Nous pensons donc que l'affinement des données existantes et l'accumulation de données supplémentaires permettra, à terme, d'effectuer l'analyse syntaxique exacte des phrases simples du français.

Chapitre 3

Quelques problèmes de segmentation

> Il faut avoir beaucoup étudié
> pour savoir peu.
> *Montesquieu*

Ainsi que nous l'avons dit en 2.1.2, nous avons écrit durant notre thèse un équivalent du système INTEX en logiciel libre, que nous avons baptisé Unitex. Nous avons profité de cette réécriture pour intégrer de nouvelles fonctionnalités, que nous présenterons dans ce chapitre. Tout d'abord, nous montrerons les méthodes que nous avons utilisées pour gérer les langues sans séparateur, à travers l'exemple du thaï. Nous montrerons que le principe de segmentation des textes que nous avions adopté dès AGLAE, est très bien adapté à ce type de problème. Nous décrirons ensuite la solution que nous avons apportée au problème des mésoclises en portugais. Enfin, nous présenterons une étude sur l'analyse des mots composés obtenus par soudure de mots simples dans certaines langues germaniques. Nous décrirons la méthode que nous avons suivie pour résoudre ce problème en norvégien.

3.1 Traitement du thaï

Certaines langues s'écrivent sans séparateur entre les mots. C'est notamment le cas du thaï et du chinois. De fait, il n'est pas facile de savoir où se trouvent les limites des mots, alors que cela est trivial en première approximation dans les langues utilisant des séparateurs, comme l'anglais ou le français. Cette situation a de nombreuses conséquences sur le traitement automatique de ces langues. Nous allons montrer, à travers l'exemple du thaï, comment nous avons adapté les opérations usuelles pour gérer ce phénomène. Pour parvenir à ces résultats, nous avons collaboré avec Krit Kosawat qui a étudié l'analyse lexicale du thaï. On trouvera dans [48] une description appronfondie des particularités de cette langue. Nous terminerons en présentant un aspect typique du thaï et les problèmes qu'il soulève lorsque l'on souhaite trier du texte.

3.1.1 Segmentation des textes

Le problème principal lorsque l'on veut analyser automatiquement des textes en thaï, est l'absence de limites de mots explicites. Considérons le texte présenté sur la figure 3.1. Les

deux lignes correspondent aux phrases suivantes :

Il est vrai que nous vivons avec la technologie depuis l'âge de pierre.
Sans la technologie, nous ne serions pas très éloignés des animaux.

จริงหรอกครับที่เราอยู่กับเทคโนโลยีมาตั้งแต่สมัยหิน
ขาดเสียซึ่งเทคโนโลยีเราก็คงมีชีวิตไม่ต่างจากสัตว์มากนัก

FIG. 3.1 – Exemple de texte thaï

Le découpage correct de ce texte en mots est indiqué sur la figure 3.2 ; les limites de mots sont symbolisées par des points.

จริง . หรอก . ครับ . ที่ . เรา . อยู่ . กับ . เทคโนโลยี . มา . ตั้งแต่ . สมัย . หิน
ขาด . เสีย . ซึ่ง . เทคโนโลยี . เรา . ก็ . คง . มี . ชีวิต . ไม่ . ต่าง . จาก . สัตว์ . มาก . นัก

FIG. 3.2 – Texte de la figure 3.1 découpé en mots

Toutefois, il n'est pas facile d'obtenir automatiquement ce résultat, car il existe plusieurs façons de découper ce texte en mots. Par exemple, le début de ce texte peut se découper en :

จริง . หร . อก

FIG. 3.3 – Découpage erroné du début du texte de la figure 3.1

Ne pouvant pas décider uniquement avec un dictionnaire quel découpage il faut prendre, nous sommes dans l'obligation de préserver les ambiguïtés. Pour cela, nous prenons comme unité de traitement le caractère au lieu du mot. Ainsi, chaque token ne sera constitué que d'un caractère. En conséquence, tous les mots de plus d'un caractère seront considérés formellement par le système comme des mots composés. Bien sûr, cette définition formelle des mots composés ne correspond pas à la définition linguistique. Cependant, en considérant comme mots composés les mots de plus d'un caractère, nous pouvons gérer les ambiguïtés de découpage de la même manière que nous gérons les ambiguïtés dues aux mots composés dans les langues à séparateurs. Par exemple, la phrase

Luc recouvre une pomme de terre glaise

possède un découpage ambigu à cause des mots composés *pomme de terre* et *terre glaise*. On exprime cette ambiguïté en prenant le mot simple comme unité, et en autorisant des combinaisons de ces unités. Le principe est le même pour le thaï, la seule différence étant que l'unité est le caractère au lieu du mot simple.

Pour effectuer ce découpage, il nous a suffi en pratique de modifier le programme de segmentation. Étant donné que cette étape sert de base à toutes les autres opérations, qui prennent en compte des suites de tokens, il n'y avait quasiment rien d'autre à modifier dans nos programmes pour que l'on puisse travailler en thaï. Nous en concluons que ce principe de découpage explicite en unités est bien adapté au traitement automatique des langues, puisqu'il permet de gérer aisément et de façon quasi générique des langues aux fonctionnements aussi différents.

3.1.2 Application des dictionnaires

Une fois le texte segmenté, l'application des dictionnaires s'effectue de la même manière que pour les langues à séparateurs : on parcourt la suite de tokens représentant le texte, et toute séquence de tokens qui correspond à une entrée dans le dictionnaire est stockée dans les dictionnaires du texte. Si la séquence est constituée de plus d'un token, l'entrée est considérée comme un mot composé ; sinon, elle est considéré comme un mot simple. Avec un découpage caractère par caractère, tous les mots de plus d'une lettre sont donc stockés dans le dictionnaire des mots composés du texte, comme le montre la figure 3.4.

Cette façon de procéder a pour conséquence d'introduire dans les dictionnaires du texte de nombreuses entrées qui correspondent à de mauvais découpages du texte. À titre de comparaison, cela reviendrait en français à retenir toutes les entrées apparaissant dans les analyses du mot *carrément*, qui sont présentées sur la figure 3.5. Il y a donc un grand nombre d'ambiguïtés à lever.

Dans [48], Krit Kosawat décrit plusieurs procédés destinés à réduire le bruit ainsi causé. Il opère en 3 niveaux :

1. regroupement de caractères inséparables en segments ;
2. regroupement de segments inséparables en syllabes ;
3. regroupement de syllabes inséparables en mots.

Ces 3 niveaux correspondent tous à des opérations sur le texte linéaire.

3.1.3 Automate des phrases

Cependant, nous avons constaté que parmi toutes les analyses d'une portion de texte, représentée par des chemins dans un automate, il y en avait peu qui couvrent de larges portions de textes sans contenir de séquences non trouvées dans le dictionnaire. Pour illustrer ce phénomène, considérons la phrase

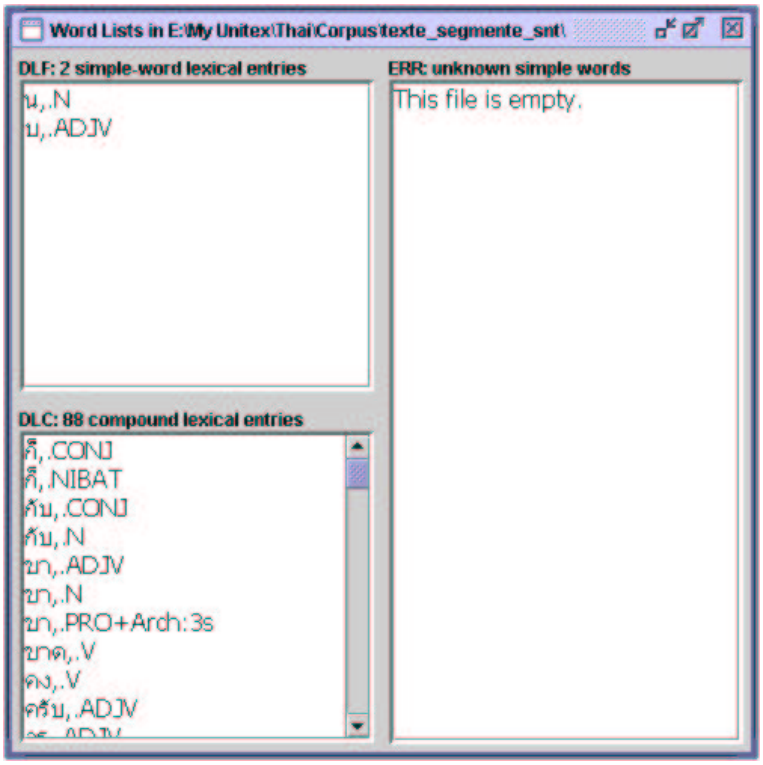

FIG. 3.4 – Dictionnaires du texte de la figure 3.1

Il part aujourd'hui.

Les découpages possibles de cette phrase sont indiqués sur la figure 3.6. On peut y voir que la séquence *aujourd'hui* peut s'analyser soit en un mot composé, soit en un mot inconnu (*aujourd*), suivi d'une apostrophe et du mot *hui*, qui est une forme du verbe *huir*. Si l'on ne souhaite garder que les analyses sans mot inconnu, on va donc écarter l'interprétation en plusieurs mots, et ainsi, ne plus prendre en compte l'entrée de dictionnaire correspondant à *hui*.

Nous avons appliqué ce principe au thaï. Pour cela, nous construisons l'automate des phrases du texte que nous voulons traiter. En pratique, nous représentons cet automate sous la forme de plusieurs automates correspondant chacun à une portion de texte. Étant donné que le découpage des phrases en thaï est très compliqué, nous opérons sur des portions de textes qui sont plutôt de la taille d'un paragraphe. Le principe de la construction de ces automates est de produire toutes les combinaisons possibles d'entrées du dictionnaire ; lorsqu'un mot est inconnu, il apparaît sans étiquette (voir [65] pour plus de détails sur la construction de l'automate des phrases).

La figure 3.8 montre les premières transitions de l'automate généré pour la première phrase du texte de la figure 3.1. On peut voir que plusieurs chemins passent par des mots inconnus,

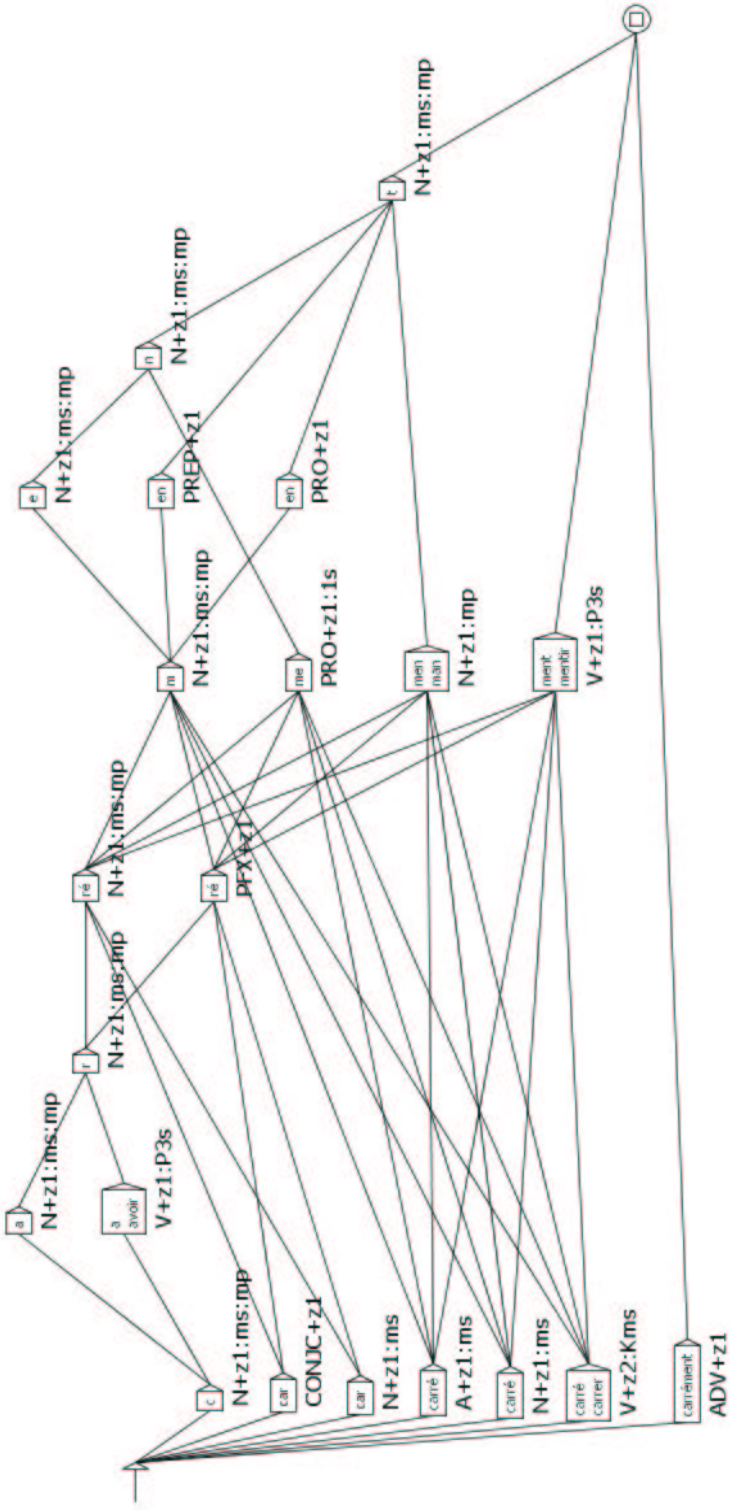

FIG. 3.5 – Découpage du mot *carrément*

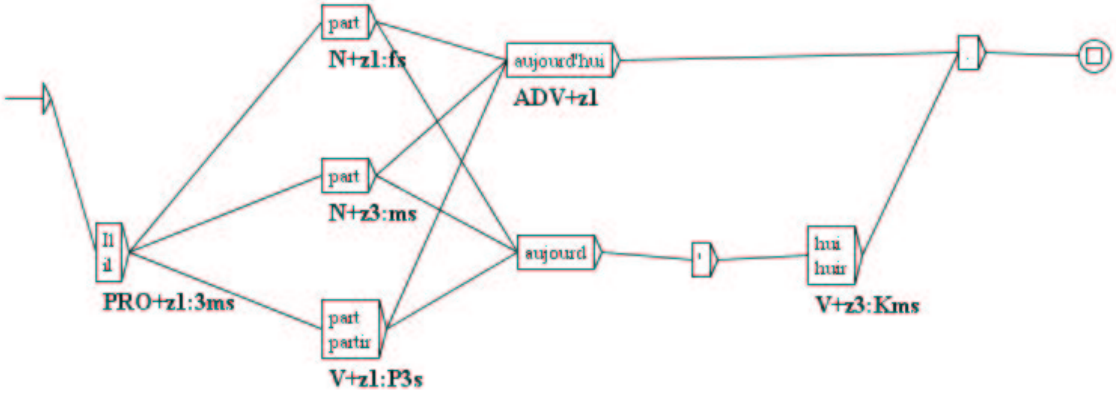

FIG. 3.6 – Découpages possibles de la phrase *Il part aujourd'hui.*

symbolisés par des boîtes sous lesquelles il n'y a pas d'information. Afin de réduire la proportion de chemins sans mot inconnu, nous avons ajouté une option au programme construisant l'automate, pour qu'il effectue une sélection des chemins selon le principe suivant : en cas de concurrence entre plusieurs chemins, on ne garde que le ou les chemins possédant le plus petit nombre de mots inconnus.

La figure 3.9 montre les premières transitions de l'automate que l'on obtient pour notre phrase d'exemple. On peut constater que le nombre de chemins a fortement diminué par rapport à l'automate de la figure 3.8. Le nombre de découpages possibles a également fortement baissé, et est maintenant du même ordre que pour des phrases en français. On peut donc en conclure que notre critère est efficace pour remédier aux ambiguïtés de segmentation.

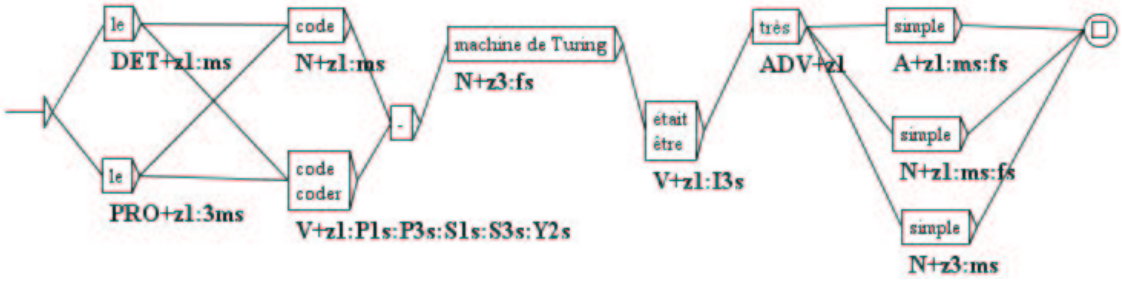

FIG. 3.7 – Exemple de découpage incorrect

Les seuls cas où ce critère produit des résultats incorrects sont ceux où le texte contient des noms propres absents du dictionnaire, et qui chevauchent des entrées de dictionnaires. Ainsi, pour une phrase telle que

le code-machine de Turing était très simple

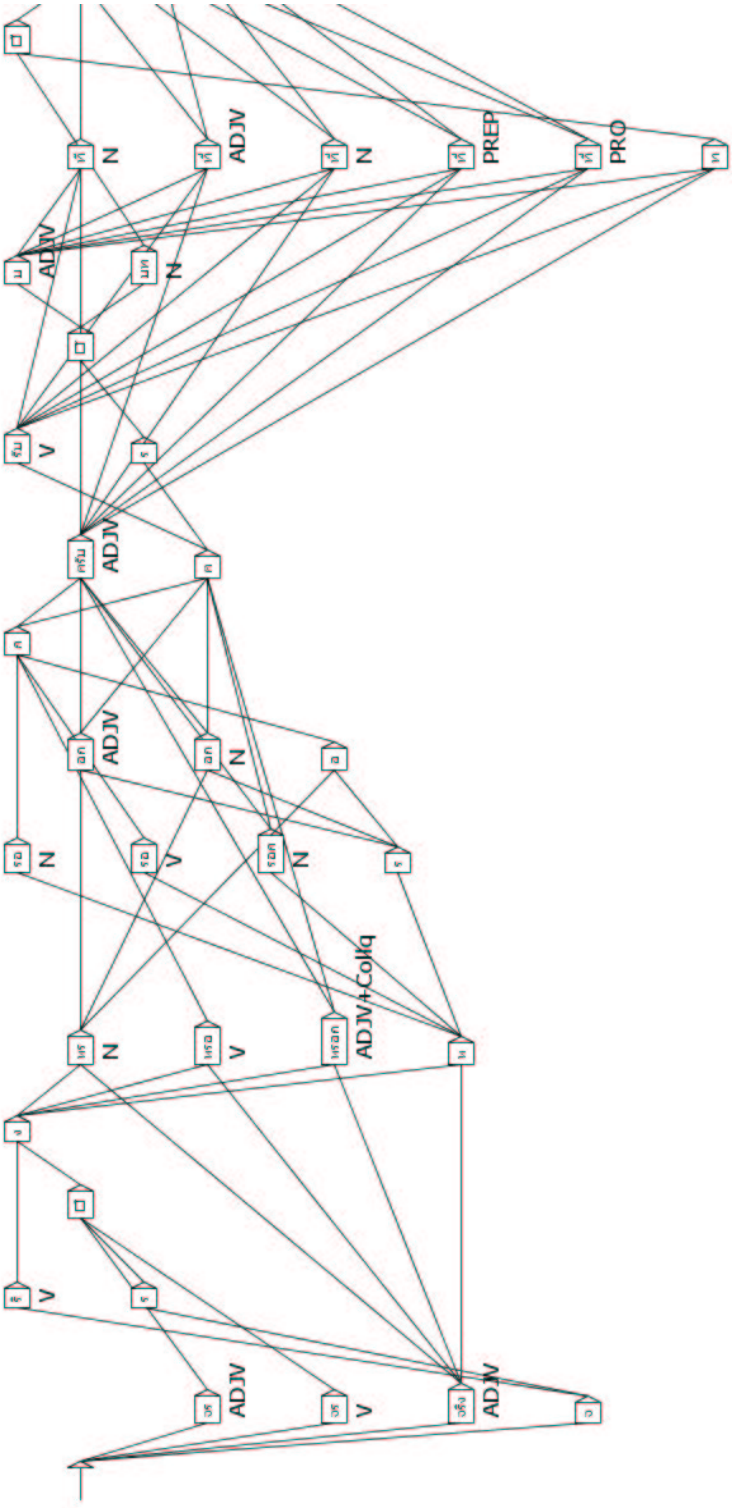

FIG. 3.8 – Découpages possibles de la première phrase de la figure 3.1

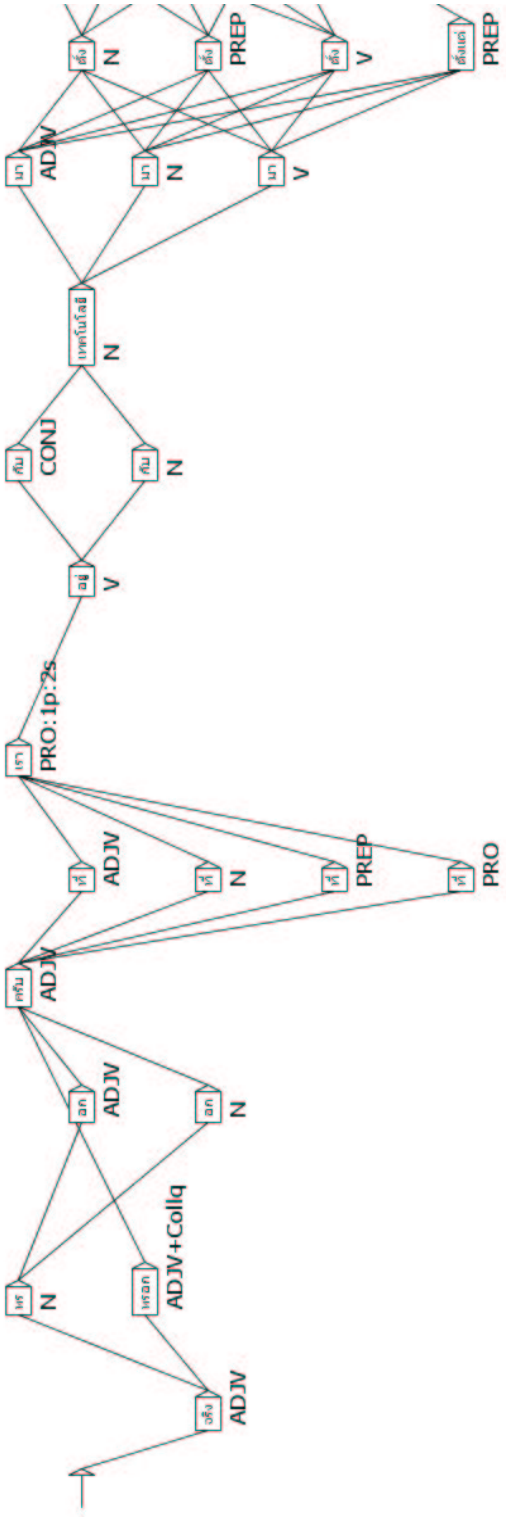

FIG. 3.9 – Automate de phrase contenant un nombre minimum de mots inconnus

l'interprétation correcte (le code-machine adopté par Turing) sera éliminée si *Turing* n'est pas dans le dictionnaire alors que *machine de Turing* y est, et le seul découpage qui sera retenu est celui présenté sur la figure 3.7.

Ces cas d'erreur sont cependant très rares. De plus, ils pourraient être gérés correctement en améliorant la couverture lexicale des dictionnaires de noms propres, de façon à éviter que des mots comme *Turing* soient considérés comme des mots inconnus. Notre critère nous semble donc robuste malgré sa simplicité, et ce, pour toutes les langues que nous avons testées, soit : thaï, français, anglais, portugais, et grec. En effet, notre critère permet d'éliminer automatiquement les mauvais chemins dans des automates tels que celui de la figure 3.6. Lors de la construction de l'automate des phrases d'un texte dans Unitex, ce principe de nettoyage est donc appliqué par défaut (option "Clean Text FST" sur la fenêtre de la figure 3.10).

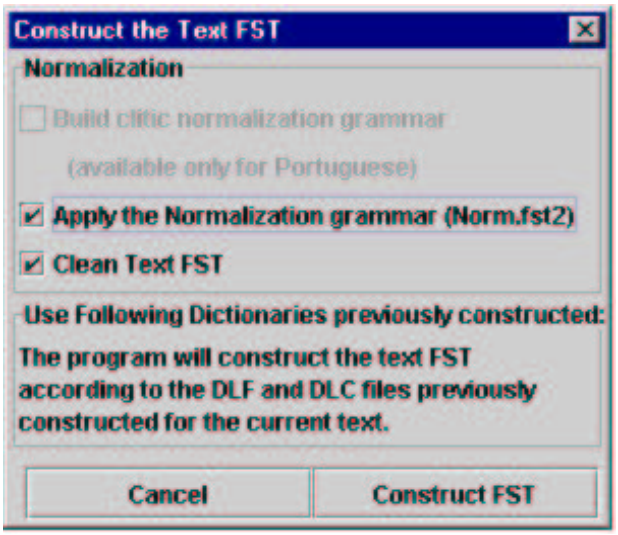

FIG. 3.10 – Paramétrage de la construction de l'automate des phrases d'un texte

3.1.4 Tri

L'ordre de tri naturel pour un locuteur thaïlandais est différent de celui utilisé dans les langues européennes. Pour le comprendre, il est nécessaire de connaître la structure des mots thaï. Un mot est composé de syllabes possédant toutes le même schéma de construction. Voici l'ordre dans lequel les éléments d'une syllabe sont codés dans les fichiers informatiques (les éléments entre crochets sont facultatifs) :

[Va]Ci[Vs][D][Vp][Cf]

Va : voyelle antéposée
Ci : consonne initiale
Vs : voyelle suscrite ou souscrite
D : signe diacritique
Vp : voyelle postposée
Cf : consonne finale

Pour un locuteur thaïlandais, le tri doit s'effectuer d'abord selon la consonne initiale, puis, si elle existe, sur la voyelle antéposée, et enfin, sur le reste éventuel de la syllabe. De plus, les signes diacritiques doivent être ignorés si le reste du texte permet de faire une distinction. Il s'agit là du même phénomène que celui qui affecte les accents en français. Par exemple, les accents sont ignorés lorsque l'on souhaite comparer les mots *manger* et *mangés*, et ce sont les contextes *r* et *s* qui permettent de décider de l'ordre. En revanche, si l'on compare les mots *pêche* et *pèche*, il est nécessaire de prendre les accents en compte car le contexte ne permet pas de trancher.

Pour pouvoir mettre ce principe en pratique, il faut donc être capable de localiser les voyelles antéposées dans un texte. Or, puisque les limites de mots, et a fortiori celles des syllabes, sont a priori inconnues, cette localisation pourrait s'avérer problématique. Fort heureusement, il n'en est rien car les voyelles antéposées ne sont pas ambiguës avec d'autres caractères. Il suffit donc d'en faire la liste (voir figure 3.11).

FIG. 3.11 – Les 5 voyelles antéposées du thaï

Les signes diacritiques n'étant pas non plus ambigus, nous pouvons donc écrire un algorithme de tri prenant en entrée un fichier texte, et triant les lignes qu'il contient suivant les règles énoncées précédemment.

Le principe de cette méthode est d'opérer un tri sur une version modifiée des chaînes de caractères. Pour cela, on utilise un arbre lexicographique dans lequel on associe à chaque chaîne modifiée sa chaîne d'origine. On parcourt ensuite cet arbre en profondeur, en sauvegardant dans le fichier de sortie les chaînes associées aux noeuds. Lorsque plusieurs chaînes sont associées à un noeud, cela signifie qu'elles sont identiques aux caractères diacritiques près. Dans ce cas, on les distingue en les comparant sans les modifier. En effet, l'ordre d'apparition des caractères diacritiques dans le codage Unicode correspond à l'ordre à utiliser pour le tri.

```
    Tri d'un fichier texte thaï :
0   Pour chaque ligne s
1       Créer une copie s' de s dans laquelle on a retiré les signes diacritiques
2       et inversé chaque voyelle antéposée avec le caractère qui la suit
3       Insérer s' dans un arbre lexicographique trié d'après l'ordre d'apparition
4       des caractères dans le codage Unicode
5       Ajouter s aux informations contenues dans le noeud final ainsi atteint
6   Fin Pour
9   Explorer l'arbre en profondeur
10  Fin.

    Explorer en profondeur un noeud N :
0   Si N est terminal
1   Alors écrire, s'il y en a, dans le fichier de sortie les chaînes associées à N
2   en les triant selon le codage Unicode des caractères
3   Fin Si
4   Pour chaque transition sortant de N
5       Explorer le noeud d'arrivée de la transition
6   Fin Pour
7   Fin.
```

Les règles de tri du thaï posent cependant un problème. En effet, puisque l'on doit prendre en compte la permutation de chaque voyelle antéposée avec la consonne initiale qui la suit, cela suppose que l'on sache où se situe cette consonne par rapport à la voyelle. Or, on a besoin de trier du texte inversé de droite à gauche, lors de la construction de concordances. En effet, si l'on souhaite trier des occurrences, on inverse le contexte gauche. Considérons les séquences suivantes :

abc def aaa
aba def ghi
aba def aaa

Si elles apparaissent dans une concordance que l'on souhaite trier selon la colonne du milieu, puis selon le contexte gauche, elles seront transformées en :

def cba aaa
def aba ghi
def aba aaa

On trie alors le fichier, ce qui donne :

def aba aaa
def aba ghi
def cba aaa

puis on reforme les colonnes d'origine en inversant celle qui correspond au contexte gauche :

aba def aaa
aba def ghi
abc def aaa

Lors de la construction de concordances, on est amené à trier des lignes de texte dans lesquelles se mélangent des séquences normales avec des séquences inversées. La consonne à permuter se trouve tantôt à gauche tantôt à droite de la voyelle antéposée, comme le montre le tableau 3.1.

	Séquence avant permutation des Va et des Ci	Séquence après permutation des Va et des Ci
Texte non inversé	Va Ci Vs	Ci Va Vs
Texte inversé	Vs Ci Va	Vs Va Ci

TAB. 3.1 – Permutations des Va et des Ci dans du texte inversé ou non

Étant donné qu'une consonne initiale peut apparaître à la fin d'un mot, il est impossible de déterminer avec quelle consonne on doit permuter la voyelle antéposée si l'on ne sait pas si le texte a été inversé.

Malheureusement, il est impossible de déterminer en fonction de son contenu si une portion de texte est inversée ou non. Nous avons cependant choisi de toujours permuter les voyelles antéposées avec les consonnes qui les suivent. Ainsi, notre programme effectuera des permutations erronées lorsqu'il traitera du texte inversé. Dans la pratique, ces erreurs semblent toutefois tolérables pour un locuteur thaïlandais, car en général, on trie d'abord suivant la séquence reconnue. Nous n'avons donc pas cherché à spécialiser la construction de concordances pour le thaï, afin de résoudre de façon exacte ce problème, ce qui nous a permis de préserver la généricité de nos programmes. L'autre solution consistait en effet à marquer les portions de texte inversées.

La méthode de tri que nous avons adoptée, basée sur l'emploi d'un premier tri opérant sur des chaînes modifiées, semble générique, car de nombreuses langues possèdent des contraintes similaires. Il est ainsi fréquent de considérer certains caractères comme équivalents lorsque le contexte permet de trancher, comme c'est souvent le cas pour les caractères diacritiques. Cette méthode permet également de tester des contraintes plus complexes sur les chaînes à trier, comme par exemple le fait de considérer la séquence *ch* comme un seul caractère en espagnol.

3.2 Les mésoclises en portugais

3.2.1 Objectif

Certaines langues présentent des phénomènes d'enclise, parfois accompagnées de rattachement des enclitiques au mot accentué. C'est par exemple le cas en espagnol, où les pronoms se collent au verbe à l'impératif :

damelo = donne-le moi

En portuguais, on rencontre un autre phénomène, la mésoclise, qui correspond à l'insertion d'un pronom clitique à l'intérieur d'un verbe. Le verbe est alors séparé en un radical et un suffixe. Par exemple, la séquence

dir-me-ão (ils me diront)

correspond à la forme verbale complète *dirão*, combinée avec le pronom *me*. Ce phénomène pose des problèmes lorsque l'on veut effectuer des recherches dans des textes, du fait qu'un verbe n'est pas nécessairement composé d'une séquence contiguë de lettres. Dans la mesure où les formes complètes telles que *dirão me* constituent des fautes de langue, il n'est pas possible d'effectuer une normalisation dans les textes. Nous avons donc choisi de travailler sur l'automate du texte. Notre but est d'insérer parallèlement aux chemins avec mésoclise, des chemins décrivant les formes complètes. De cette manière, les deux formes peuvent coexister et être utilisées l'une ou l'autre en fonction des besoins de l'utilisateur. Pour cela, nous avons décidé d'exploiter le principe de grammaire de normalisation qui existait déjà dans Unitex. Notre méthode consiste à générer, pour un texte donné, une grammaire de normalisation traitant les mésoclises contenues dans le texte. Nous allons donc montrer comment nous repérons les mésoclises, et de quelle façon nous générons des grammaires de normalisation.

3.2.2 Repérages des séquences

Les mésoclises n'interviennent que sur des verbes au futur ou au conditionnel. Les radicaux des verbes sont codés dans les dictionnaires électroniques du portuguais au moyen du code flexionnel `R`. Les pronoms clitiques pouvant intervenir dans les mésoclises sont codés par la séquence `PRO+Pes`. Les suffixes portant les marques de temps sont réguliers, et peuvent donc être listés. Grâce à ces informations, il est possible de repérer les mésoclises au moyen d'une recherche de motifs.

Pour cela, nous avons construit une grammaire décrivant les séquences recherchées. Cette grammaire est présentée sur la figure 3.12. On peut y voir que l'on recherche un radical, suivi d'un ou deux pronoms clitiques et d'un suffixe, ces différents éléments devant être séparés par des tirets. Cette grammaire contient des sorties destinées à être insérées dans le texte reconnu. La sortie `[FuturConditional]` est une marque permettant de vérifier que les occurrences ont été trouvées par cette grammaire. Ainsi, lorsque l'on chargera une liste d'occurrences pour construire une grammaire de normalisation, cela permettra de vérifier qu'il s'agit bien d'une liste prévue pour cette utilisation. Les sorties associées aux suffixes indiquent à quels temps, nombre et personne ils correspondent. Par exemple, la sortie `[:C1s:C4s:C3s]` indique que le suffixe *ia* peut correspondre au conditionnel, à la première ou troisième personne du singulier, ainsi qu'à la personne correspondant au *vous* de politesse singulier. Cette information nous permettra de savoir pour quels temps, nombre et personne il faudra générer les formes verbales complètes.

Cette grammaire est appliquée au texte à l'aide du programme `Locate` d'Unitex, en mode *MERGE*. Cela signifie que l'on obtiendra dans la liste des occurrences trouvées, les séquences

reconnues dans lesquelles auront été insérées les sorties. Par exemple, si l'on considère le texte suivant :

As coisas di-las-íeis recolhidas e cismáticas (Les choses semblaient pensives et recueillies)

la grammaire reconnaîtra la séquence *di-las-íeis* (*vous les auriez dites*), ce qui donnera le fichier d'occurrences suivant :

```
#M
4 8 [FuturConditional]di-las-íeis[:C2p]
```

La première ligne est un en-tête indiquant qu'il s'agit d'une liste d'occurrences obtenue en mode *MERGE*. La deuxième ligne indique qu'une occurrence a été trouvée entre les tokens 4 et 8. Le texte qui suit correspond à la séquence reconnue dans laquelle ont été insérées les sorties. Ce fichier contient donc toutes les informations dont nous avons besoin pour construire une grammaire de normalisation.

3.2.3 Génération d'une grammaire de normalisation

Une fois les mésoclises repérées, nous pouvons construire une grammaire qui va insérer dans l'automate du texte de nouveaux chemins correspondant aux formes complètes. Cette grammaire sera donc un transducteur prenant en entrée les mésoclises, et produisant en sortie des suites d'étiquettes décrivant les formes complètes. Pour cela, nous devons effectuer deux opérations : reconstituer la forme verbale complète et normaliser les pronoms clitiques.

La première étape consiste à retrouver une forme fléchie à partir de la forme canonique, obtenue à partir du radical, et d'un code de flexion. Nous allons donc devoir consulter les dictionnaires. Or, le codage de nos dictionnaires par automate nous permet de retrouver une ligne de dictionnaire à partir d'une forme fléchie, mais pas à partir d'une forme canonique. Nous pouvons donc retrouver la ou les forme(s) canonique(s) associée(s) à un radical, mais en revanche, nous ne pouvons pas obtenir une forme fléchie donnée à partir d'une forme canonique et d'un code flexionnel.

Nous avons résolu ce problème en adaptant les dictionnaires de façon à pouvoir effectuer des recherches à partir de formes canoniques au lieu de formes fléchies. Pour cela, nous avons utilisé une astuce très simple, qui consiste à inverser les formes fléchies et canoniques dans les dictionnaires. Nous avons simplement ajouté une option au programme de compression de dictionnaires, pour qu'il effectue ces inversions. À l'aide de cette nouvelle fonctionnalité, nous avons donc engendré un dictionnaire inversé, comprimé sous forme d'automate. Nous pouvons maintenant rechercher dans ce dictionnaire les entrées correspondant aux formes canoniques obtenues précédemment, et ne retenir que celles qui correspondent aux codes de flexion qui nous intéressent. Il ne reste plus qu'à reconstituer des étiquettes valides pour l'automate du texte en réinversant les formes fléchies et canoniques.

Si l'on reprend l'occurrence de l'exemple précédent, on obtiendrait les étapes suivantes :

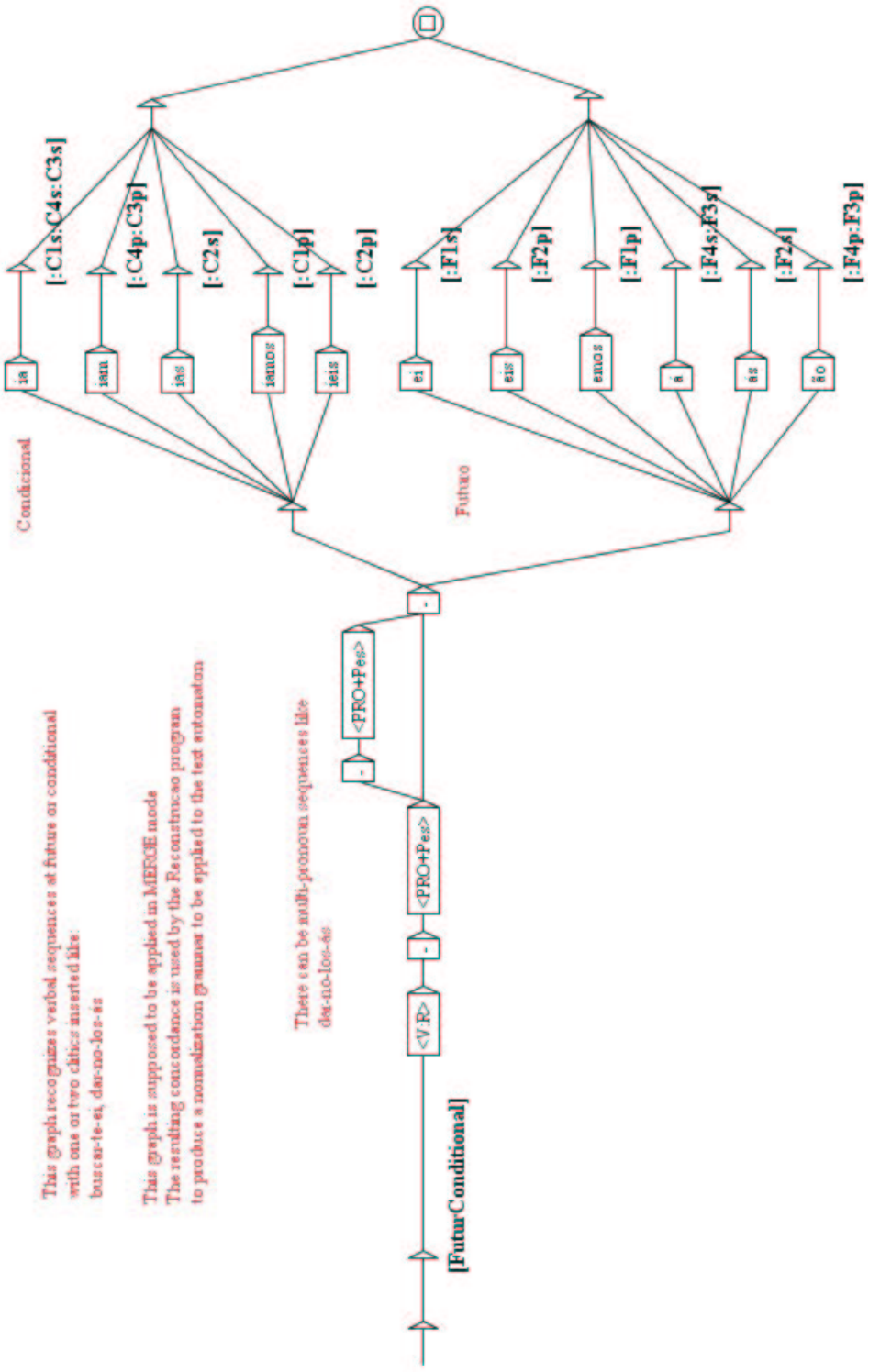

FIG. 3.12 – Graphe *V-Pro-Suf*

1. séquence repérée : `[FuturConditional]di-las-íeis[:C2p]` (*vous les auriez dites*)
2. recherche de l'infinitif du verbe : `dizer` (*dire*)
3. extraction des entrées du dictionnaire inversé ayant `dizer` dans le champ réservé à la forme fléchie :

```
dizer,diríeis.V:C2p
dizer,diríamos.V:C1p
dizer,dirão.V:F4p:F3p
dizer,dirás.V:F2s
dizer,dirá.V:F4s:F3s
dizer,dirias.V:C2s
dizer,diriam.V:C4p:C3p
dizer,diria.V:C1s:C4s:C3s
dizer,diremos.V:F1p
dizer,direis.V:F2p
dizer,direi.V:F1s
```

4. sélection de l'entrée correspondant au code de flexion `[:C2p]` : `dizer,diríeis.V:C2p`
5. construction de l'étiquette correspondante : `{diríeis,dizer.V:C2p}`

Cette première étape accomplie, il reste à produire les formes normalisées des pronoms clitiques. Cette tâche est aisée, car les pronoms se réécrivent de façon indépendante des verbes avec lesquels ils étaient combinés. Nous avons donc pu écrire une grammaire de réécriture des pronoms, qui est présentée sur la figure 3.13.

Pour des raisons de présentation, nous avons séparé de cette grammaire les pronoms clitiques qui se réécrivent en plusieurs pronoms. Ces pronoms sont décrits par la grammaire *PronounsIn2parts*, présentée sur la figure 3.15. Certaines règles de réécriture portant sur le clitique *nos* et les double-clitiques *no-(lo+la+los+las)* sont utilisées dans d'autres grammaires. C'est pourquoi ces pronoms sont traités dans une grammaire distincte, *NormalizeNosPronouns*, présentée sur la figure 3.14.

Comme on peut le voir, certains pronoms sont ambigus. Dans ce cas, on prend en compte toutes les possibilités. Cela se traduit par la génération de plusieurs séquences d'étiquettes. Par exemple, comme le clitique *se* est ambigu, la mésoclise *dir-se-iam* sera normalisée par les deux séquences suivantes :

```
{diriam,dizer.V:C4p:C3p} {se,eu.PRO+Pes:R4ms:R4fs:R4mp:R4fp}
{diriam,dizer.V:C4p:C3p} {se,se.PRO+Pes:N3s}
```

Les séquences d'étiquettes ainsi engendrées sont codées dans la grammaire de normalisation, comme des sorties associées aux mésoclises. À titre d'exemple, la figure 3.16 montre un extrait de la grammaire de normalisation générée pour le roman *Os Pobres*, de *Raul Brandão*.

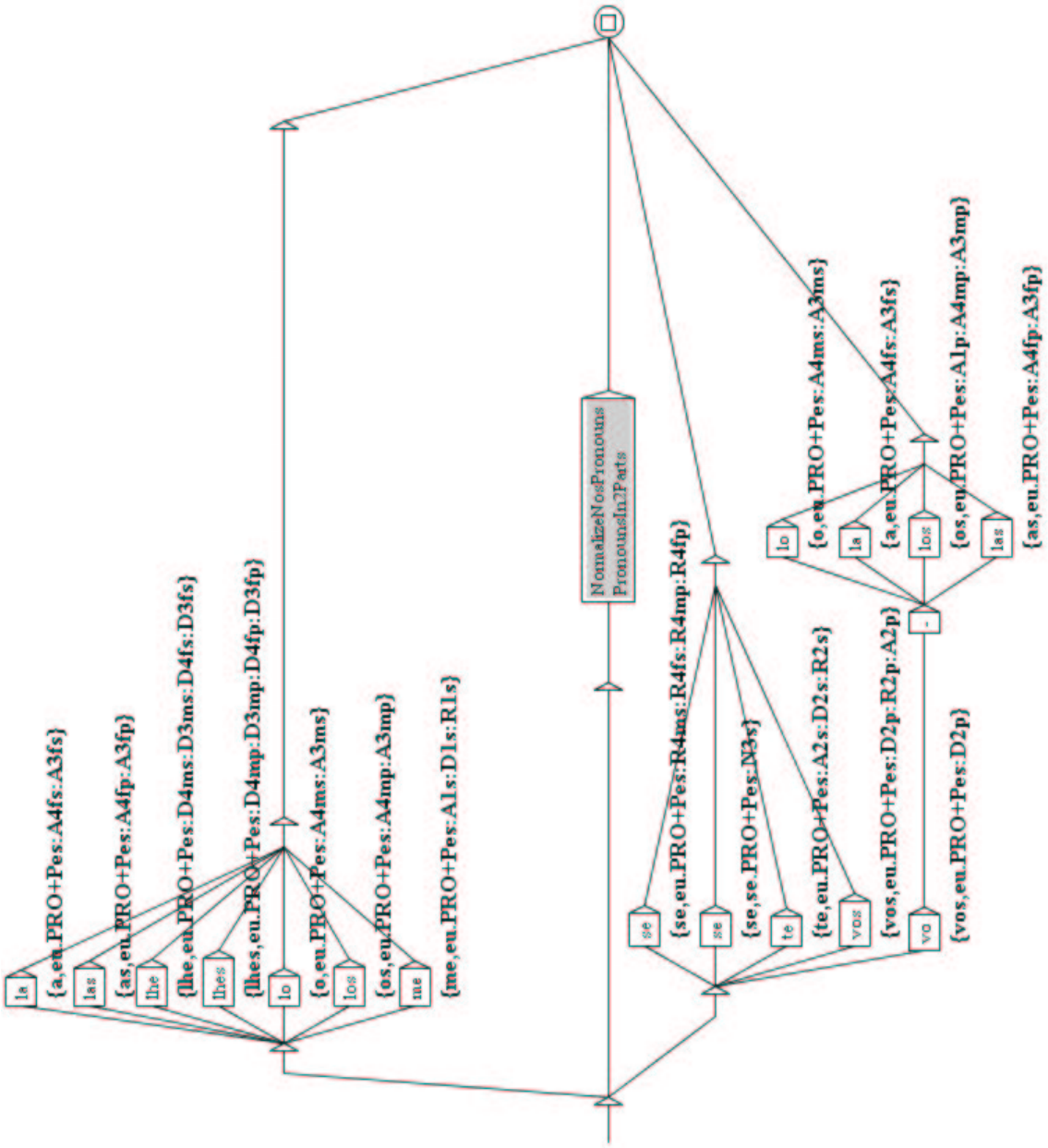

FIG. 3.13 – Graphe *NormalizePronouns*

3.2.4 Normalisation dans l'automate du texte

Une fois la grammaire de normalisation générée, il ne reste plus qu'à construire l'automate du texte en tenant compte de cette grammaire. Pour construire l'automate normalisé d'une phrase, on procède en deux étapes. Tout d'abord, on construit l'automate de la phrase, tel que nous l'avons présenté en 3.1.3. Ensuite, on applique la grammaire de normalisation à la

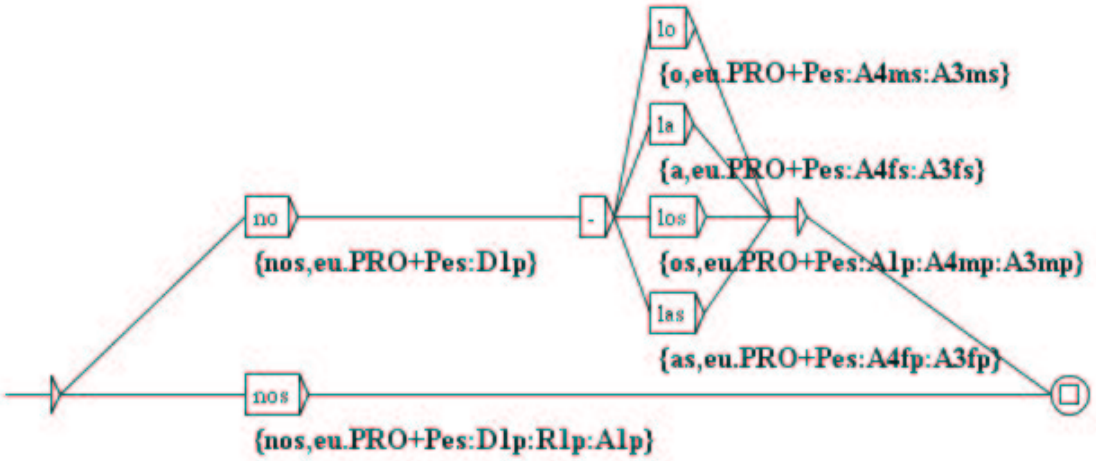

FIG. 3.14 – Graphe *NormalizeNosPronouns*

phrase : pour chaque séquence reconnue, on insère dans l'automate un ou plusieurs chemins, concurrents avec les chemins décrivant la mésoclise, dont les transitions portent les étiquettes contenues dans les sorties associées à la séquence reconnue.

Considérons le texte suivant :

Dir-se-ia no entanto que a vida redobrava ; (on aurait dit, cependant, que la vie s'exacerbait ;)

L'automate généré pour cette séquence est présenté sur la figure 3.17. L'application de la grammaire de normalisation va permettre de reconnaître la séquence *Dir-se-ia*. Cette mésoclise est normalisée par les deux séquences d'étiquettes suivantes :

```
{diria,dizer.V:C1s:C4s:C3s} {se,eu.PRO+Pes:R4ms:R4fs:R4mp:R4fp}
{diria,dizer.V:C1s:C4s:C3s} {se,se.PRO+Pes:N3s}
```

Le programme va donc insérer deux chemins dans l'automate de la phrase, concurrents avec ceux qui correspondent à la séquence *Dir-se-ia*. Le résultat de cette opération est l'automate de la figure 3.18.

En conclusion, nous avons mis au point une méthode permettant de prendre en compte les formes sous-jacentes des mésoclises, en les insérant dans l'automate du texte comme des alternatives aux séquences d'origine. Pour cela, nous avons utilisé les outils existants pour localiser les séquences concernées, puis générer une grammaire de normalisation pour ces formes. Nous aurions également pu résoudre ce problème en construisant la grammaire globale de toutes les règles de normalisation. Cependant, nous n'avons pas retenu cette solution, et ce pour trois raisons. Tout d'abord, le programme de construction de l'automate du texte n'était pas prévu pour fonctionner sur de très grosses grammaires de normalisation. Ensuite, le fait

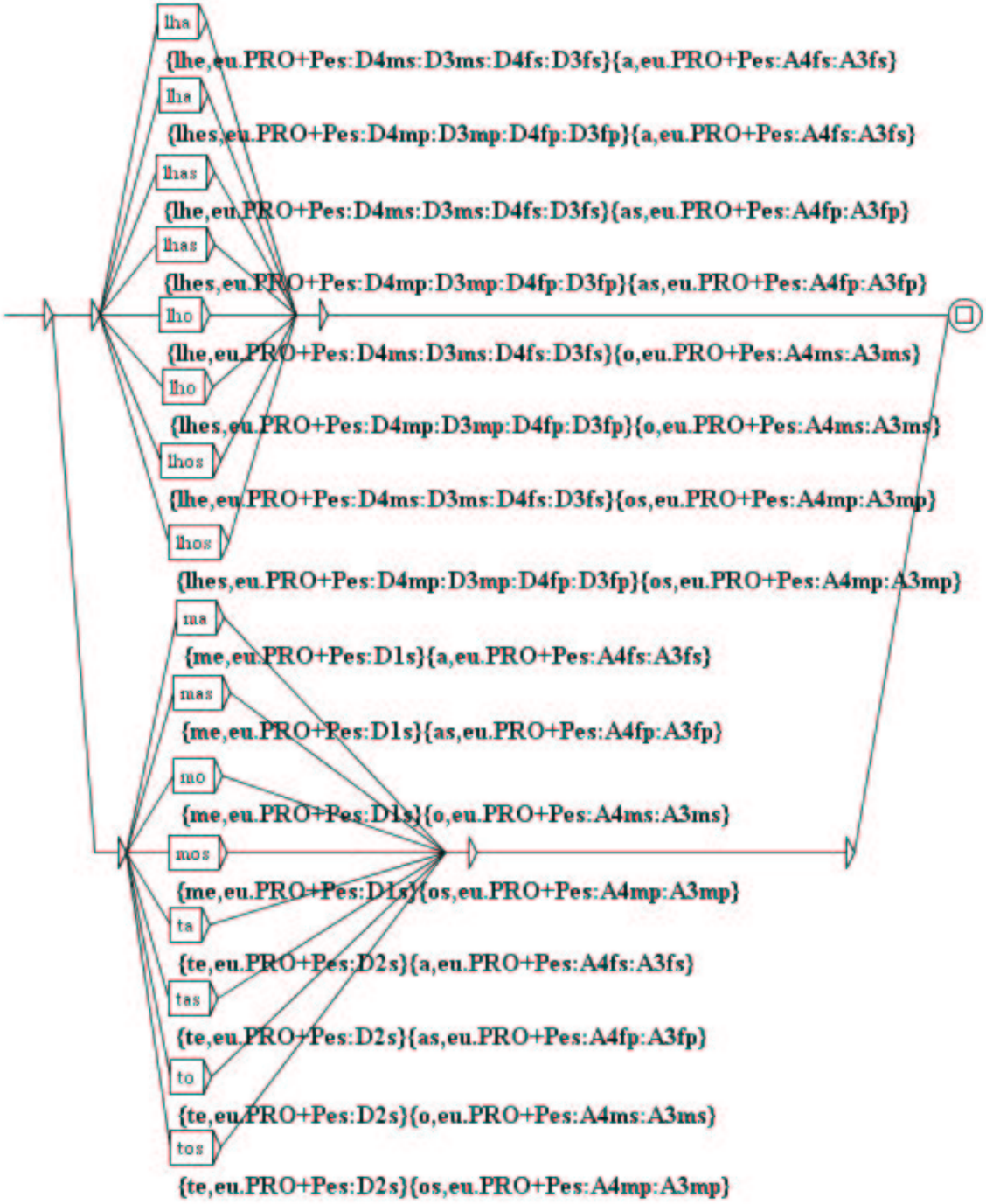

FIG. 3.15 – Graphe *PronounsIn2parts*

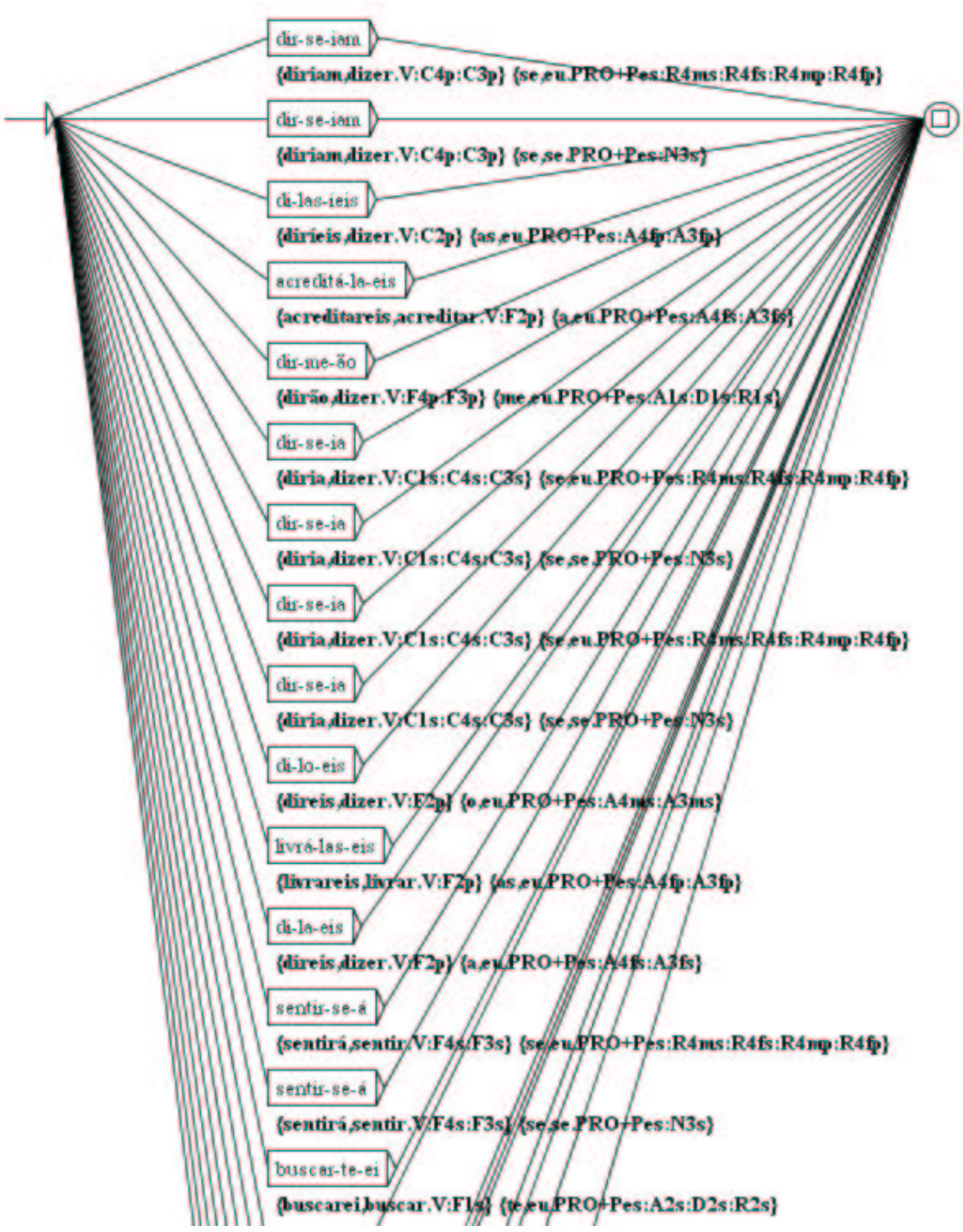

FIG. 3.16 – Extrait de la grammaire de normalisation générée pour le roman *Os Pobres*

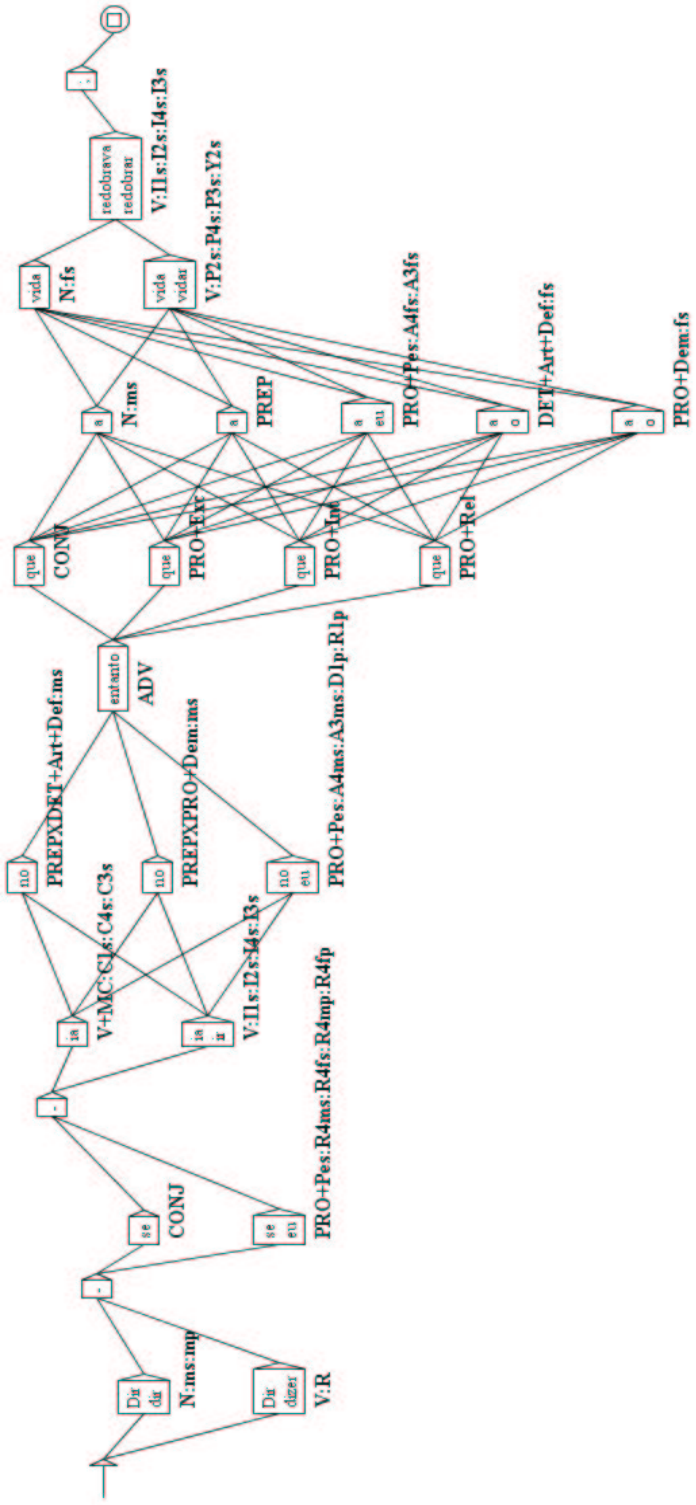

FIG. 3.17 – Automate de la phrase *Dir-se-ia no entanto que a vida redobrava ;*

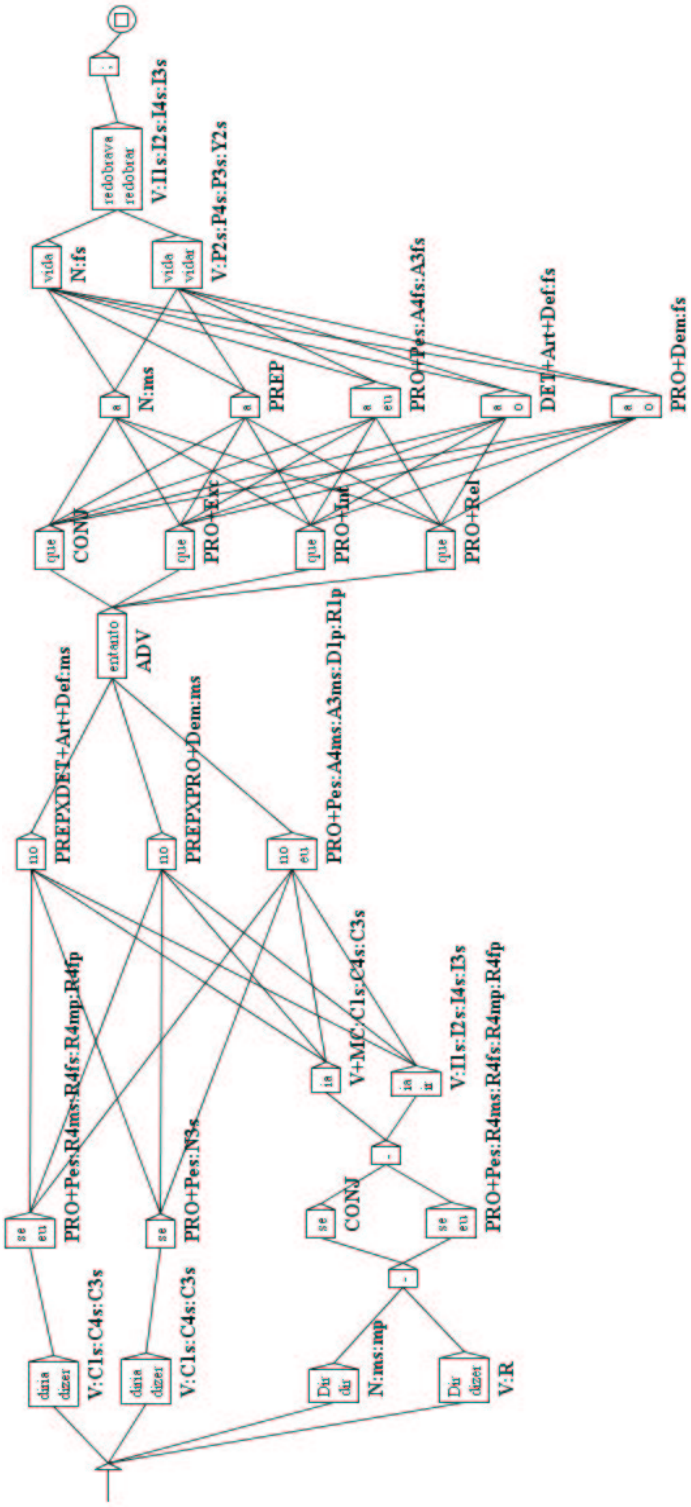

FIG. 3.18 – Automate normalisé de la phrase *Dir-se-ia no entanto que a vida redobrava ;*

de générer dynamiquement la grammaire permet de prendre en compte les dictionnaires dont dispose l'utilisateur, ce qui lui permet d'intégrer des verbes absents des dictionnaires généraux, comme par exemple des verbes techniques. Enfin, ce procédé de normalisation pourra sans doute s'appliquer à d'autres langues et/ou phénomènes. Il nous a donc semblé utile de fournir un outil de construction plutôt que des données qui seraient figées pour un phénomène précis. Le programme ainsi élaboré, nommé `Reconstrucao`, est intégré à Unitex.

3.3 Les mots polylexicaux en norvégien

Le norvégien est une langue germanique dans laquelle il existe des mots obtenus par la soudure de plusieurs mots simples. Nous reprendrons le terme *mots polylexicaux*, défini dans [33] pour les séquences non soudées. Il y a 3 classes de mots en norvégien :

(a) mots simples ou composés soudés : *hund* (chien), *vakthund* (chien de garde)
(b) mots composés non soudés : *røde hunder* ("rouges chiens" = rubéole)
(c) mots polylexicaux soudés : *artikkelforfatteren* (l'auteur de l'article)

Les types (a) et (b) sont faciles à traiter, puisqu'il suffit de les lister dans des dictionnaires de mots simples et de mots composés. Nous allons présenter une méthode permettant de traiter les mots du type (c), que nous avons développée en collaboration avec Harald Ulland.

3.3.1 Discussion linguistique

3.3.1.1 Mots traités

Parmi les mots constitués de plusieurs mots soudés, certains doivent être intégrés dans les dictionnaires. Il s'agit de combinaisons de mots figées, équivalentes à des mots composés, et dont le sens n'est pas compositionnel. Bae Sun-Mee a utilisé le terme de *noms compacts* pour ce type de mots ([5], [6]). Voici quelques exemples de tels mots :

dukketeater	"marionnette-théâtre"	théâtre de marionnettes
sultestreik	"faim-grève"	grève de la faim
kaffekopp	"café-tasse"	tasse à café
ishockey	"glace-hockey"	hockey sur glace

Une fois ces mots intégrés aux dictionnaires, leur reconnaissance ne posera plus de problème. La difficulté survient pour les mots polylexicaux qui correspondent à des syntagmes libres, comme :

artikkelforfatteren	"article-l'auteur"	l'auteur de l'article
arkitekturidsskrift	"architecture-revue"	revue d'architecture

Étant donné que ces constructions sont libres, il est irréaliste de les lister dans des dictionnaires. Il faut donc effectuer une analyse morphologique de ces mots. Nous allons donc nous intéresser à l'analyse de ce type de mots poylexicaux.

3.3.1.2 Morphèmes de soudure

Les mots polylexicaux ne sont pas toujours obtenus par simple concaténation. Il peut y avoir des morphèmes entre les constituants, que nous appellerons *morphèmes de soudure* (voir [50] pour une discussion sur les morphèmes de soudure de l'allemand). Là où il y a simple concaténation, on parlera de *morphème de soudure zéro (<Ø>)*. Si la soudure se fait par une lettre de transition, il s'agit le plus souvent de *s*, parfois de *e*. Mais la soudure peut également se faire par la suppression d'une lettre, lorsque l'on a une double consonne finale devant la même consonne, initiale du mot suivant. Cette suppression sera représentée par le symbole *<L>*. Voici des exemples illustrant ces cas de figure [1] :

augustkveld	*august <Ø> kveld*	"août-soir"	soir d'août
krigsfare	*krig s fare*	"guerre-danger"	danger de guerre
guttesport	*gutt e sport*	"garçon-sport"	sport de garçon
traffikkø	*traffikk <L> kø*	"circulation-queue"	queue de circulation

3.3.1.3 Morphologie flexionnelle des noms polylexicaux

La tête syntaxique est toujours le nom qui se trouve le plus à droite. Les informations flexionnelles d'un mot polylexical sont toujours portées par cette tête. Les traits portés par les noms peuvent être les suivants :

le genre :	masculin (m), féminin (f) ou neutre (n)
le nombre :	singulier (s) ou pluriel (p)
la définitude :	défini (d) ou indéfini (i)
le cas :	nominatif (a) ou génitif (g)

Il y donc $3 \times 2 \times 2 \times 2 = 24$ combinaisons de traits possibles :

```
<N:msia>
<N:msig>
<N:msda>
<N:msdg>
```
etc.

Les constituants gauches d'un nom polylexical doivent être des noms de forme non marquée, c'est-à-dire que ces noms doivent posséder les traits singulier, indéfini et nominatif. Étant donné qu'il n'y a pas de restriction sur le genre, les constituants se classent donc en 3 catégories :

```
<N:msia>
<N:fsia>
<N:nsia>
```

Lorsqu'il y a plusieurs constituants gauches, il y a une soudure entre chacun d'eux :

[1] Cette liste de morphèmes de soudure n'est pas exhaustive ; il existe quelques autres morphèmes, plus rares. Nous y reviendrons dans nos conclusions.

arbeidstidsloven	*arbeid s tid s loven*	"travail-durée-la loi"	la loi sur la durée du travail

On peut décrire la composition par la formule suivante[2] :

<N1> S <N2> S <N3>

Dans la plupart des cas, il serait plus exact de tenir compte de la hiérarchisation des constituants :

ou bien : *(<N1> S <N2>) S <N3>*
ou bien : *<N1> S (<N2> S <N3>)*

Si *(<N1> S <N2>)* ou *(<N2> S <N3>)* est figé et représente déjà une entrée de dictionnaire, on a donc une combinaison entre un mot polylexical et un mot simple. Cette analyse est alors concurrente avec une combinaison de trois mots simples. En pratique, les tests ont montré que lorsqu'il y a concurrence entre une bipartition et une tripartition, la bipartition représente toujours la bonne solution. Notons que l'on peut rencontrer des compositions de plus de trois éléments, mais cela reste assez rare.

3.3.1.4 Noms polylexicaux avec des constituants adjectivaux ou verbaux

Si la plupart des noms polylexicaux ont des constituants gauches qui sont eux aussi des noms, il est également possible de trouver des adjectifs et des verbes à cette position. En voici quelques exemples :

storindustri	"grande-industrie"	grande industrie
ståplass	"être debout-place"	place debout
spilleforbud	"jouer-interdiction"	interdiction de jouer

Les adjectifs et les verbes doivent apparaître, comme les noms, dans leur forme non marquée, qui est l'infinitif pour le verbe, et l'indéfini singulier au degré positif pour l'adjectif. Nous noterons repectivement `<A:sio>` et `<V:W>` les constituants adjectivaux et verbaux. De façon plus rare, on peut rencontrer des noms possédant des constituants gauches adverbiaux, notés `<ADV>`.

3.3.1.5 Adjectifs et verbes polylexicaux

Jusqu'à présent, nous avons traités uniquement les noms polylexicaux, c'est-à-dire les cas où la tête est un nom. Toutefois, il existe aussi des adjectifs et des verbes polylexicaux. De même que pour les noms, c'est la tête qui porte les informations flexionnelles, et les constituants gauches doivent être de forme non marquée. Voici quelques exemples de tels mots :

[2]Nous utilisons le symbole S pour désigner une soudure, qui peut être s, e, $<\emptyset>$ ou $<L>$.

blyinnfattet	"plomb-enchâssé"	serti de plomb
dødskul	"mort-cool"	supercool [3]
kaloririke	"calorie-riches"	riches en calories
tabubelegge	"tabou-couvrir"	couvrir de tabous

Nous avons maintenant une description quasi-exhaustive des procédés de création de mots polylexicaux. Il y a probablement quelques cas résiduels de combinaisons n'entrant pas dans cette description. Cependant, nous pensons qu'il serait préférable de les intégrer dans les dictionnaires plutôt que de chercher à les analyser, si leur nombre est très restreint (si c'est le cas ces mots sont d'ailleurs peut-être figés). En effet, en ajoutant de nouvelles règles de composition, nous risquons d'introduire un grand nombre d'analyses erronées.

3.3.2 Traitement informatique

Concrètement, l'analyse des mots polylexicaux d'un texte s'effectue en deux étapes :

- extraction des candidats ;
- analyse des mots obtenus.

La première opération est effectuée au moyen d'une consultation de dictionnaires : tout mot du texte qui n'apparaîtra ni comme mot simple, ni comme partie de mot composé, sera considéré comme mot inconnu. Les mots inconnus se répartissent en trois grandes catégories :

- mots mal orthographiés ;
- noms propres ;
- mots corrects absents des dictionnaires.

Dans le cas du norvégien, les mots polylexicaux appartiennent à la dernière catégorie, car ce sont des mots valides. En effet, si les mots constitués de plusieurs mots soudés et dont le sens n'est pas compositionnel doivent naturellement être recensés dans les dictionnaires, les mots polylexicaux libres ne peuvent pas être listés, car leur nombre est potentiellement infini. Ces mots vont donc apparaître comme mots inconnus après consultation des dictionnaires. Naturellement, cette méthode d'analyse des mots inconnus n'est valable que si l'on suppose que les dictionnaires utilisés soient exhaustifs. De plus, nous devons supposer que si un mot polylexical est ambigu avec une entrée de dictionnaire, alors l'analyse correspondant au mot polylexical doit être également présente dans le dictionnaire, faute de quoi, nous aurions du silence.

La seconde étape consiste à rechercher pour chacun des mots inconnus s'il est possible de l'analyser comme un mot polylexical. Si tel est le cas, les analyses produites sont stockées comme de nouvelles entrées dans le dictionnaire des mots simples du texte [4].

Nous commencerons par présenter les règles de composition des mots que nous avons prises en compte, puis nous présenterons les principales structures de données que nous avons utilisées. Enfin, nous présenterons en détail l'algorithme d'analyse des mots que nous avons utilisé.

[3] Le nom *død* (=mort) est ici un intensifieur.

[4] Rappelons que, formellement, ces mots sont considérés comme des mots simples, car ce sont des suites contiguës de lettres.

3.3.2.1 Règles de composition des mots polylexicaux

Nous ne cherchons pas ici à effectuer une analyse morphologique complète des mots inconnus. Ainsi, nous ne traitons pas les cas de dérivation classiques, qui sont connus et maîtrisés ([17]), et nous ne nous pencherons que sur l'analyse des mots polylexicaux. Pour ces mots, l'analyse morphologique se réduit à sa plus simple expression, puisque la concaténation est quasiment la seule opération nécessaire à leur formation. Il nous a donc semblé inutile de mettre en œuvre un formalisme complet tel que celui présenté dans ([47]). Puisque les règles de composition sont simples, la vraie difficulté consiste à écarter les analyses incorrectes. De même que dans la méthode utilisée pour le suédois présentée dans ([82]), nous avons dissocié la génération des analyses potentielles et l'élimination des analyses incorrectes en deux étapes distinctes.

Nous allons maintenant présenter les diverses contraintes que nous prenons en compte pour effectuer l'analyse.

Contraintes de catégories et contraintes flexionnelles

Les composants d'un mot polylexical sont soumis à des contraintes qui diffèrent selon qu'il s'agisse ou non de la tête du mot. Les mots qui peuvent apparaître en position de tête (mots les plus à droite) peuvent être des noms, des adjectifs ou des verbes. Ces mots ne sont pas soumis à des contraintes flexionnelles particulières, et donnent leurs caractéristiques grammaticales et flexionnelles au mot polylexical.

Les constituants gauches peuvent être :

- des noms singuliers, indéfinis et neutres (notés `<N:sia>`) ;
- des adjectifs singuliers, indéfinis et de degré positif (notés `<A:sio>`) ;
- des verbes à l'infinitif (notés `<V:W>`) ;
- des adverbes (notés `<ADV>`).

Nous avons considéré que toutes les combinaisons d'éléments vérifiant ces contraintes étaient possibles (nom-nom, adjectif-nom, nom-nom-nom, etc).

Longueur des mots

Les lettres sont souvent codées comme des mots en raison de l'utilisation que l'on peut en faire dans des phrases telles que :

Jean n'arrive pas à écrire les k correctement.
Le professeur lui a mis un B.

Afin d'éviter que chaque mot inconnu soit décomposé comme une suite de mots triviaux, nous avons décidé de ne pas prendre en considération les mots d'une seule lettre.

Mots interdits

Certains mots courts vérifiant les contraintes des mots tête introduisent des erreurs. Il s'agit de noms relativement rares ou stylistiquement marqués :

ar	are
ende	bout
ert	pois
te	thé
ve	bien-être
vis	manière

Les trois premiers noms présentent de surcroît une homonymie avec des suffixes flexionnels :

ar	= pluriel de noms masculins
ende	= participe présent
ert	= participe passé des verbes en *-ere*

Afin d'éviter que ces mots n'introduisent des analyses erronées, ils ne sont pas considérés comme des mots tête valides.

Règles de soudure

Nous avons pris en compte les trois types de soudure présentés précédemment :

- soudure par le mot vide : obtenue en concaténant simplement les mots

  `arbeiderstrøk` = `arbeider,.N:msia` / `strøk,.N:nsia:npia`[5]

- soudure avec insertion d'une lettre : obtenue en insérant un *e* ou un *s* entre les mots

  `landesorg` = `land,.N:nsia:npia` / `e` / `sorg,.N:fsia:msia`
  `ansiktsløse` = `ansikt,.N:nsia:npia` / `s` / `løse,.A:msdo`

- soudure avec suppression d'une lettre : lorsqu'un mot se termine par une double consonne et que le mot suivant commence par cette consonne

  `kupplaner` = `kupp,.N:nsia:npia` / `planer,plan.N:mpia`

Heuristiques de levée d'ambiguïtés

Lorsqu'il est possible de décomposer un mot de plusieurs façons, la préférence est donnée aux décompositions dans lesquelles le nombre de constituants est minimal. Par exemple, pour le mot *adferdsproblemer* (problèmes de conduite) qui peut se décomposer de deux façons :

```
adferd,.N:msia / s / problemer,problem.N:npia
ad,.ADV / ferd,.N:fsia:msia / s / problemer,problem.N:npia
```

la préférence sera donnée à la première analyse, car elle n'est constituée que de deux mots (on ne compte pas le *s* de soudure).

[5] Quand les formes fléchie et canonique sont identiques, on peut omettre la forme canonique. Ainsi, la ligne `vraiment,.ADV` est équivalente à `vraiment,vraiment.ADV`. La syntaxe des entrées de dictionnaire est présentée dans la section 3.3.2.2.

En cas de concurrence entre plusieurs analyses de même longueur, la préférence est donnée à celles où le mot tête est un nom. Ainsi, pour le mot *aksjonærutbytte* (bénéfice d'actionnaire) qui admet les huit analyses suivantes :

```
aksjonær,.N:msia / utbytte,.V:W
aksjonær,.N:msia / utbytte,.N:nsia
aksjonær,.N:msia / ut,.ADV / bytte,.V:W
aksjonær,.N:msia / ut,.ADV / bytte,.N:nsia
aks,.N:nsia:npia / jo,.ADV / nær,.A:msio:fsio / utbytte,.V:W
aks,.N:nsia:npia / jo,.ADV / nær,.A:msio:fsio / utbytte,.N:nsia
aks,.N:nsia:npia / jo,.ADV / nær,.A:msio:fsio / ut,.ADV / bytte,.V:W
aks,.N:nsia:npia / jo,.ADV / nær,.A:msio:fsio / ut,.ADV / bytte,.N:nsia
```

on donne la préférence à la seconde, car des deux analyses les plus courtes, c'est la seule qui ait un nom comme mot tête.

3.3.2.2 Structure du dictionnaire

L'analyse d'un mot inconnu s'effectue en testant s'il peut se décomposer en une combinaison de mots simples[6]. Pour cela, nous utilisons un dictionnaire de type DELAF ([73]). Ce dictionnaire contient des entrées de la forme suivante :

`aftenvandringer,aftenvandring.N:fpia:mpia` (promenades du soir)

Rappelons que les entrées sont composées d'une forme fléchie (`aftenvandringer`), d'une forme canonique (`aftenvandring`), d'une catégorie grammaticale (`N` pour nom) et d'informations flexionnelles (`fpia:mpia` pour "féminin pluriel indéfini et nominatif" ou "masculin pluriel indéfini et nominatif").

Afin de pouvoir consulter efficacement ce dictionnaire, nous le représentons sous la forme d'un automate de formes fléchies et d'une liste de codes permettant de reconstruire les entrées complètes du dictionnaire à partir des formes fléchies de l'automate. La représentation par automate est en effet très adaptée à la représentation de lexiques ([3], [54], [67], [69]). Ces deux objets constituent une forme comprimée du dictionnaire.

Par exemple, si l'on considère le dictionnaire formé des lignes suivantes :

`aften,aften.N:msia`	(soir)
`alder,alder.N:msia`	(âge)
`blad,blad.N:nsia:npia`	(journal)
`pensjon,pensjon.N:msia`	(retraite)

on obtient l'automate de la figure 3.19 :

[6]Nous prenons ici la définition formelle du mot simple, c'est à dire une suite contiguë de lettres. Cette définition est différente du mot simple linguistique, car nous considérons ainsi comme simples les mots tels que *bryllupsdag* (jour du mariage).

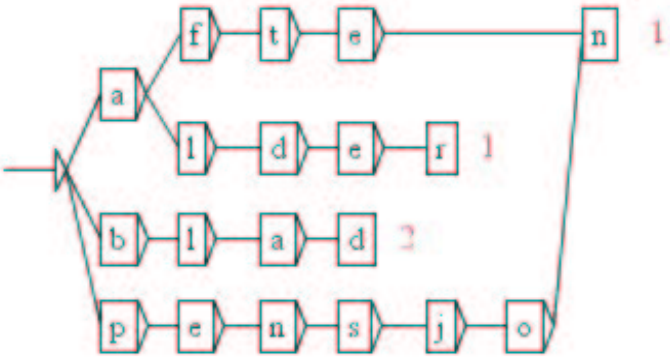

FIG. 3.19 – Exemple d'automate de formes fléchies

Les nombres à droite des boîtes terminales font référence aux codes à utiliser pour reconstruire les entrées complètes. Ici, les nombres 1 et 2 correspondent respectivement aux codes "`.N:msia`" et "`.N:nsia:msia`".

L'automate produit est déterministe et minimal. Il est stocké dans un fichier `.BIN`. La liste des codes permettant de reconstruire les entrées complètes est stockée dans un fichier `.INF` (voir [65] pour plus de détails sur la méthode de compression des dictionnaires).

Cette représentation offre l'avantage de pouvoir parcourir le dictionnaire de façon quasiment déterministe[7]. Il y a également un avantage à regrouper tous les codes dans un fichier `.INF` : on peut facilement pré-calculer pour chacun de ces codes s'il correspond à un mot pouvant apparaître dans un mot polylexical, et si oui, s'il peut apparaître en tête et/ou en facteur gauche. Pour cela, nous utilisons deux tableaux booléens indexés par les numéros des codes du fichier `.INF`. Le premier nous sert à marquer les mots pouvant apparaître comme facteur gauche, c'est à dire les mots vérifiant au moins un des motifs `<N:sia>`, `<A:sio>`, `<V:W>` ou `<ADV>`. Le second tableau sert à marquer les mots pouvant apparaître comme mots tête, c'est à dire les mots vérifiant au moins un des motifs `<N>`, `<A>` ou `<V>`.

Une fois ces pré-calculs effectués, nous pouvons analyser les mots inconnus du texte.

3.3.2.3 Analyse des mots inconnus

L'algorithme que nous allons présenter décrit l'analyse d'un mot inconnu par exploration simultanée de ce mot et de l'automate du dictionnaire. Cette fonction est appelée sur chaque mot inconnu. Au retour de cet appel, si la liste L n'est pas vide, on retire le mot de la liste des mots inconnus et on génère dans le dictionnaire du texte les entrées correspondant aux analyses que l'on conserve après application des heuristiques de préférence.

Cas de non-analyse

[7] Le parcours peut ne pas être totalement déterministe à cause des variantes de casse. En effet, si l'on cherche à analyser le mot *Aftenblad*, on parcourra à la fois les chemins commençant par *a* et par *A*.

Il y a deux cas de non-analyse. Le premier est dû à une incompatibilité avec une contrainte qui provoque donc le rejet d'une analyse potentielle. Ainsi, le mot *arbeidastrøk* ne peut se décomposer avec *arbeida* comme facteur gauche, car ce mot renvoie aux trois entrées de dictionnaire suivantes :

```
arbeida,arbeide.V:I
arbeida,arbeide.V:K
arbeida,arbeid.N:npda
```

Comme aucune de ces entrées ne vérifie une des contraintes `<N:sia>`, `<A:sio>`, `<V:W>` ou `<ADV>`, ce mot ne peut pas être un facteur gauche de *arbeidastrøk*.

Le second cas de non-analyse correspond à une impossibilité de découper le mot inconnu en une séquence de mots présents dans le dictionnaire. Ainsi, le mot *arbeidastrøk* ne peut pas se décomposer en *arbeid* et *astrøk*, car bien que *arbeid* soit un facteur gauche valide, il n'existe aucune façon de découper *astrøk* en une séquence de mots du dictionnaire. En fait, il n'existe aucune analyse pour ce mot, qui est un mot mal orthographié.

Entrées de dictionnaire produites

Lorsqu'une analyse conforme aux contraintes est détectée, une entrée de dictionnaire est générée puis ajoutée au dictionnaire du texte. En fonction des besoins, les informations fournies par l'analyse pourraient également être utilisées pour décomposer le mot et étiqueter ses composants. Une telle représentation pourrait être mise en parallèle de la première dans un automate représentant les ambiguïtés d'étiquetage. La figure 3.20 montre un exemple de ce que l'on pourrait obtenir pour une phrase contenant le mot *bryllupsdagen* [8].

Pour générer les entrées de dictionnaire, le programme normalise la casse du mot inconnu en fonction de la casse de ses composants dans le dictionnaire. Ainsi, le mot inconnu *Aftenblad* sera normalisé en *aftenblad* car il est formé à partir des deux lignes de dictionnaire suivantes, où les formes fléchies sont tout en minuscules :

```
aften,.N:msia
blad,.N:nsia:npia
```

Une fois que l'on dispose de cette forme fléchie normalisée, il faut encore générer la forme canonique de l'entrée ainsi que ses codes grammaticaux et flexionnels. Étant donné que les mots polylexicaux héritent des propriétés de leur mot tête, la construction de l'entrée s'effectue de manière très simple : on prend la ligne de dictionnaire du mot tête et l'on en préfixe les formes fléchie et canonique par les facteurs gauches du mot inconnu, dont la casse a été préalablement normalisée. Par exemple, le mot *Minnesfesttradisjonen* (la tradition de la fête du souvenir) se décompose en :

[8] Cette capture d'écran montre un automate de phrase dans lequel la décomposition en *bryllup* et *dagen* a été insérée manuellement en parallèle de l'entrée générée automatiquement pour *bryllupsdagen*.

```
     Explorer (POS, MOT, N, C, D, L) :
          POS : position dans le mot inconnu
          MOT : mot inconnu à traiter
          N : nœud courant dans l'automate du dictionnaire
          C : constituant courant
          D : décomposition courante
          L : liste des décompositions pour le mot inconnu
          On désignera par R le nœud racine du dictionnaire
0    Si N est terminal Alors
1         Si POS = longueur(MOT) Alors
2              Si C est un mot tête valide Alors
3                   Ajouter les analyses valides de C à D
4                   Ajouter les décompositions ainsi obtenues à L
5                   Retourner
6              Sinon Retourner
7              Fin Si
8         Sinon
9              Si C est un facteur gauche valide Alors
10                  Ajouter les analyses valides de C à D
11                  /* soudure directe */
12                  Explorer (POS+1, MOT, R, mot vide, D, L)
13                  /* soudure avec insertion de e ou s */
14                  Si MOT[POS+1] = 'e' ou 's' Alors
15                       Ajouter e ou s à D
16                       Explorer (POS+2, MOT, R, mot vide, D, L)
17                  Fin Si
18                  /* soudure avec suppression */
19                  Si C se termine par une double consonne Alors
20                       Explorer (POS, MOT, R, mot vide, D, L)
21                  Fin Si
22             Fin Si
23        Fin Si
24   Fin Si
25   Si POS = longueur(MOT) Alors Retourner
26   Sinon
27        Pour chaque transition T sortant de N
28             Si lettre de T = MOT[POS] Alors
29                  Ajouter cette lettre à C
30                  Explorer (POS+1, MOT, nœeud pointé par T, C, D, L)
31             Fin Si
32        Fin Pour
33   Fin Si
34   Fin.
```

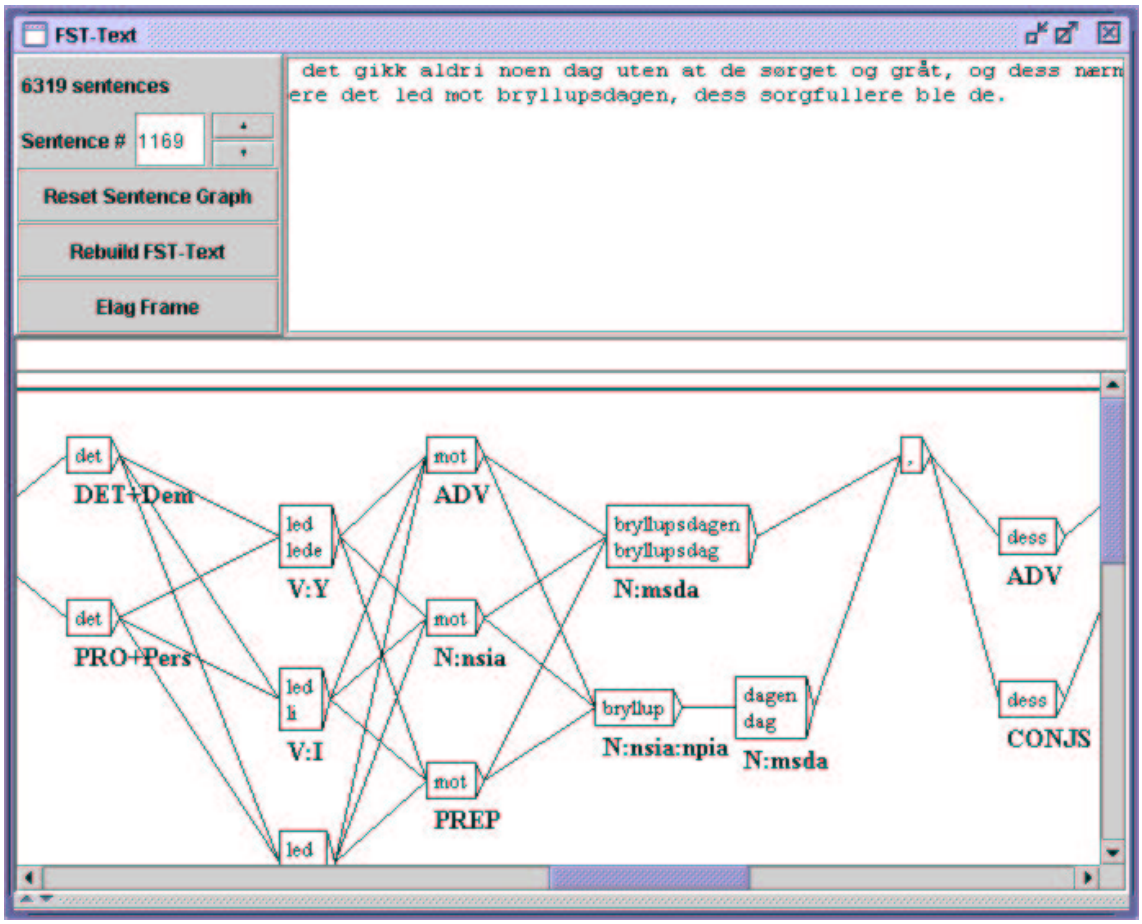

FIG. 3.20 – Exemple d'automate de phrase modifié

```
minnes,.V:W:P / fest,.N:msia / tradisjonen,tradisjon.N:msda
```

Pour produire l'entrée de dictionnaire correspondant à cette analyse, on prend donc l'entrée `tradisjonen,tradisjon.N:msda` que l'on transforme en lui ajoutant le préfixe `minnesfest`, ce qui donne l'entrée :

```
minnesfesttradisjonen,minnesfesttradisjon.N:msda
```

On obtient ainsi la forme canonique et les informations grammaticales et flexionnelles en une seule opération. Pour chaque entrée ainsi produite, on stocke une description de l'analyse sous forme de chaîne de caractères, la ligne de dictionnaire engendrée, le nombre d'éléments qui la composent ainsi qu'une valeur booléenne indiquant si le mot analysé est un nom. Une fois que toutes les analyses d'un mot ont été produites, ces informations sont utilisées pour déterminer lesquelles seront effectivement prises en compte[9]. Les critères de sélection sont alors le nombre

[9]Nous avons choisi de générer toutes les analyses et de les trier ensuite afin de préserver la généricité du procédé, ce qui nous permettra d'adapter aisément le programme à d'autres langues, comme l'allemand.

de composants ainsi que la préférence accordée aux analyses dans lesquelles le mot tête est un nom. Pour chacune des analyses conservées, on sauvegarde la ligne de dictionnaire et la description de l'analyse dans les fichiers de sortie. Si un mot possède au moins une analyse, on le retire de la liste des mots inconnus.

3.3.2.4 Le programme

Le programme qui effectue l'analyse des mots polylexicaux se nomme `PolyLex`, et a été incorporé à Unitex. L'analyse des mots inconnus y est intégrée dans les opérations de prétraitement des corpus, de sorte que, lorsque l'on ouvre un texte, l'analyse des mots composés soudés est automatiquement effectuée à l'issue de l'application des dictionnaires. Les mots ainsi analysés sont intégrés au dictionnaire de mots simples du texte, ce qui permet une exploitation immédiate des résultats.

3.3.3 Discussion sur les résultats expérimentaux

Pour tester le programme `PolyLex`, nous avons utilisé les ressources suivantes :

- des dictionnaires électroniques du norvégien du type DELAF [10] ;
- Unitex ;
- le moteur de recherche GlossaNet [11].

Nous avons fait des tests sur deux corpus de journaux, dont l'un est un ensemble d'articles du quotidien *Dagbladet* et l'autre consiste en une liste de mots, obtenue à l'aide de GlossaNet en utilisant comme corpus l'édition web du quotidien *Aftenposten*. Nous avons ainsi obtenu une liste de 831 mots inconnus, sur laquelle nous avons appliqué le programme `PolyLex`.

3.3.3.1 Examen des mots polylexicaux trouvés

Le programme a identifié 425 mots polylexicaux, répartis comme suit :

- 384 suites ont été étiquetées comme noms (`<N>`) ;
- 19 suites ont été étiquetées comme adjectifs (`<A>`) ;
- 19 suites ont été étiquetées comme verbes (`<V>`) ;
- 3 suites ont été étiquetées comme grammaticalement ambiguës (`<A>` et `<V>`).

Suites étiquetées comme des noms

La majeure partie des analyses fournies par `PolyLex` correspondent à des noms polylexicaux corrects. Toutefois, on constate quelques erreurs, dont la première est le bruit. En effet, certains mots inconnus sont analysés à tort, parmi lesquels on rencontre beaucoup de noms propres. Une première source d'erreurs est que beaucoup de noms propres norvégiens ont une éthymologie correspondant à deux noms soudés. C'est notamment le cas du nom du premier

[10] Élaborés à l'Université de Bergen par Harald Ulland en coopération avec Tove Jacobsen et Annette Jørgensen.

[11] Présenté dans [25] et accessible à l'adresse http://glossa.fltr.ucl.ac.be

ministre actuel, *Bondevik*, qui s'analyse comme la soudure de *bonde* (paysan), et *vik* (anse). Les autres erreurs sont dues à des noms propres étrangers. Par exemple, le nom propre *Ballantines*, qui désigne une marque de whisky, est analysé comme la concaténation d'un nom (*ball*), d'un adverbe (*an*) et d'un nom (*tines*). Au total, nous avons rencontré 33 erreurs dues à des noms propres.

D'autres mots causent également du bruit. Il s'agit de mots étrangers (3 cas), de néologismes (6 cas) et de fautes d'orthographe (1 cas). Voici les trois analyses incorrectes dues à des mots anglais :

```
angry,angry.N:nsia
dressed,dressed.N:msia
life,life.N:msia
life,life.N:nsia:npia
```

Ces analyses ont été obtenues à partir des mots norvégiens *an*, *gry*, *dress*, *ed*, *li* et *fe*, dont le dernier est ambigu (fée ou bétail). Les problèmes dus aux mots étrangers, néologismes et fautes d'orthographe, ne sont pas typiques de notre programme. Leur résolution reste un problème ouvert.

Parmi les autres erreurs constatées, 2 séquences ont été incorrectement analysées comme des noms, alors qu'il s'agissait d'adjectifs polylexicaux. Notre heuristique donnant la préférence aux noms a provoqué deux erreurs. L'erreur pour le premier mot est due à ce que l'interprétation nom est absurde. Le second problème vient de ce qu'il y a une ambiguïté réelle entre deux interpréations polylexicales : l'une nominale, et l'autre adjectivale. Notre programme a donc commis une erreur en supprimant l'interprétation adjectivale.

Un autre type d'erreur est que, si le mot est bien un nom, le lemme qui lui a été associé est incorrect. Il s'agit de cas où le nom tête est ambigu. Ainsi, parmi les 2 analyses

`badelaken,badelake.N:msda` (saumure de lit)
`badelaken,badelaken.N:nsia` (drap de lit)

seule la seconde est correcte. Il s'agit donc d'un problème d'ambiguïté très général, qui ne concerne pas notre programme en particulier.

Le dernier cas d'erreur correspond à des découpages incorrects :

```
timeglassand,timeglassand.N:fsia:msia
timeglassand,timeglassand.N:msia
vulkansand,vulkansand.N:fsia:msia
vulkansand,vulkansand.N:msia
```

Les découpages qui ont été faits sont les suivants :

`timeglass`	/	<Ø>	/	`and`	(canard de sablier)
`timeglass`	/	<L>	/	`sand`	(sable de sablier)
`vulkan`	/	`s`	/	`and`	(canard de volcan)
`vulkan`	/	<Ø>	/	`sand`	(sable de volcan)

Ces ambiguïtés interviennent dans tous les cas où un couple de noms ne se distingue que par la présence d'un *s* initial. Certaines de ces ambiguïtés, telles que *canard de sablier*, ne peuvent être levées que sémantiquement. D'autres pourraient être levées à l'aide de contraintes formelles, comme c'est le cas pour *canard de volcan*, car le nom *vulkan* ne peut pas être suivi par une soudure en *s*.

Suites étiquetées comme des verbes et des adjectifs

Nous avons également rencontré 19 suites étiquetées comme adjectifs polylexicaux et 19 autres étiquetées comme verbes. Parmi les adjectifs, on trouve 10 erreurs, réparties en 5 noms propres, 2 mots anglais et 3 noms analysés comme des adjectifs. Ces 3 dernières erreurs auraient été évitées si les dictionnaires étaient exhaustifs. Les analyses restantes sont correctes, même si un mot (*urutinerte*) était en réalité un adjectif simple qui était absent du dictionnaire.

Les analyses donnant des verbes polylexicaux sont pratiquement toutes des erreurs. Plus de la moitié des séquences sont analysées comme des participes passés, alors qu'il s'agit en réalité d'adjectifs. Ce problème pourrait être réglé en intégrant au programme une règle indiquant qu'un participe passé devient un adjectif lorsqu'il est en position de tête dans un mot polylexical. Pour cela, il sera nécessaire de vérifier au préalable la validité de cette règle en étudiant ce type de mots polylexicaux. Les autres analyses sont des erreurs qui concernent des formes étiquetées comme des verbes à l'impératif, au présent et à l'infinitif.

3.3.3.2 Examen des mots inconnus résiduels

Après l'application de notre programme sur notre corpus de tests, 406 mots sont toujours considérés comme des mots inconnus. L'étude de cette liste nous a permis d'évaluer le silence à 21 mots (20 noms et 1 adjectif), qui auraient du être analysés comme des mots polylexicaux. Parmi ces cas, 17 sont dus à des lacunes des dictionnaires. Ainsi, les mots suivants n'ont pas été analysés parce que les noms *chiffon*, *makeup* et *teller* étaient absents des dictionnaires :

chiffontopp
makeupartister
pengetellerne

Les 4 mots restants étaient :

gamletreneren
todagerskort
oslolufta
stromboliblått

Le premier cas est dû à ce qu'un adjectif défini (*gamle*) peut exceptionnellement apparaître à une position normalement réservée à des adjectifs indéfinis.

Le second est dû au schéma de production `<DET+Num><N:p><N>` (déterminant numéral, nom pluriel, nom) que nous n'avions pas pris en compte. Cette construction est d'ailleurs productive :

treværelsesleilighet	(appartement de trois pièces)
firemannstelt	(tente pour quatre personnes)
åttedagersfrist	(délai de huit jours)

Enfin, les deux derniers mots non analysés montrent que des noms propres, tels que *Oslo* et *Stromboli*, peuvent apparaître comme constituants (mais pas comme tête) de noms et d'adjectifs polylexicaux. Selon l'orthographe officielle, la majuscule initiale du nom propre est alors transformée en minuscule.

3.3.4 Conclusion

Après avoir effectué des tests, nous sommes arrivés à la conclusion que le programme `PolyLex` donnait de bons résultats pour les noms (87,6% de précision, 95% de rappel), des résultats moyens pour les adjectifs, et de mauvais résultats pour les verbes.

Nous pourrions améliorer la précision de l'analyse des noms en évitant d'analyser les mots inconnus commençant par une majuscule. Nous éviterions ainsi de décomposer abusivement des noms propres, qui sont à la source de la plupart des erreurs, mais il y aurait cependant un léger silence. Une bonne solution consisterait à appliquer une procédure de détection des noms propres, afin de ne travailler que sur les mots toujours inconnus après cette phase.

Une autre façon d'améliorer la précision de l'analyse serait d'expliciter les soudures pouvant intervenir après chaque mot [12]. Nous avons vu à travers l'exemple de *vulkansand*, qu'une analyse incorrecte (*canard de volcan)* aurait pu être évitée si l'on avait codé l'impossibilité pour le mot *vulkan* d'être suivi par une soudure en *s*. Toutefois, ce codage pour tous les constituants gauches potentiels en norvégien représente une masse de travail énorme [13]. Il faudrait donc faire une estimation du gain que l'on pourrait espérer, avant d'entreprendre une telle tâche.

L'analyse des adjectifs pourra sans doute être améliorée si l'on prend en compte la possibilité de voir apparaître des participes passés comme tête d'adjectifs polylexicaux. En revanche, l'analyse des verbes polylexicaux semble pour l'instant irréalisable, faute d'une étude poussée de leur morphologie.

[12] Il semble en effet que le morphème de soudure dépende du mot qui le précède.

[13] Ce travail a été effectué pour l'allemand au CIS (`http://www.cis.uni-muenchen.de`). Nous avons d'ailleurs commencé à utiliser ces codes pour adapter notre programme à l'allemand.

Conclusion

Notre objectif était de trouver un modèle permettant de décrire exhaustivement les schémas de phrases simples du français, qui nous permette d'accumuler des données sur une longue période de temps. Pour cela, nous avons choisi d'utiliser des descriptions par grammaires locales. Nous avons montré comment l'on pouvait engendrer ces grammaires à partir des tables de lexique-grammaire de façon simple et générique. Pour cela, nous avons étendu le principe des graphes patrons introduit par Roche, en travaillant à un niveau au-dessus des tables de lexique-grammaire, au moyen d'une super-table et de graphes génériques. Nous avons ainsi mis au point un modèle suffisamment simple pour permettre une maintenance efficace et coopérative des données. Le fait que l'on puisse décrire de telles constructions au moyen de grammaires locales montre que la distinction entre la recherche de motifs et l'analyse syntaxique n'est en réalité qu'une question d'échelle de la description.

Toutefois, nous avons pu voir que cette volonté de simplicité de description avait pour conséquence d'engendrer des masses de données si considérables que la question même de savoir si l'on peut manipuler de telles données se pose. Nous avons étudié ce problème en examinant diverses méthodes pour accélérer les calculs. Nous avons tout d'abord mis au point un programme de recherche de motifs qui met à profit une représentation judicieuse des textes pour optimiser les comparaisons entre les grammaires et les textes. Nous avons également montré que l'on pouvait accélérer les calculs en transformant les grammaires, soit grâce à une extension de l'algorithme de mise en forme normale de Greibach aux transducteurs algébriques étendus, soit en construisant une approximation d'un transducteur à états finis. De plus, nous avons montré que l'on pouvait appliquer les différentes grammaires en plusieurs passes, de façon à additionner les temps d'application des grammaires au lieu de les multiplier. Enfin, nous avons établi une méthode simple, basée sur une sélection des grammaires par le vocabulaire du texte, qui nous garantit qu'il sera possible d'appliquer les grammaires qui auront été accumulées, indépendamment de la taille de la base de données.

Toutes ces expériences sont autant de pistes de recherche, qui nous permettent de penser que l'on pourra exploiter les quantités de données que laissent prévoir nos premières estimations. Nous pensons donc que l'accumulation des grammaires décrivant toutes les phrases simples du français est envisageable. Pour y parvenir, il sera nécessaire d'affiner les descriptions linguistiques, en codant de nouvelles propriétés dans les tables de lexique-grammaire, et en décrivant chaque type de structure syntaxique au moyen de graphes génériques. Du point de vue informatique, il sera nécessaire d'adapter les outils progressivement, en fonction des données disponibles. Dans cette optique, la mise au point d'un système en logiciel libre nous

permet de garantir la maintenance des outils. Nous pensons donc que toutes les conditions sont réunies pour pouvoir entreprendre la description exhaustive des phrases simples du français par grammaires locales, ce qui nous permettra d'obtenir un analyseur syntaxique d'une finesse inégalée.

Bibliographie

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers. Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] Jürgen Albert, Dora Giammarresi, and Derick Wood. Extended context-free grammars and normal form algorithms. In J.-M. Champarnaud, D. Maurel, and D. Ziadi, editors, *Automata Implementation*, Rouen, France, 1998. Third International Workshop on Implementing Automata, WIA'98.

[3] A. W. Appel and G. J. Jacobson. The world's fastest scrabble program. *Communications of the ACM*, 31(5) :572–578, 1988.

[4] D. Appelt, J. Hobbs, D. Israel, and M. Tyson. Fastus : A Finite-State processor for information extraction from real-word text. In *IJCAI'93*, 1993.

[5] Sun-Mee Bae. Construction of an electronic dictionary for compound nouns in korean. *Studies in Lexicography*, 11(1) :151–164, 2001.

[6] Sun-Mee Bae. *Le dictionnaire électronique des séquences nominales figées en coréen et de leurs formes fléchies. Méthodes et applications*, 2002. Thèse de doctorat. Université de Marne-la-Vallée.

[7] Hava Bat-Zeev Shyldkrot. Analyse sémantique d'une forme passive complémentaire : se laisser. *Langages*, 135, 1999. Paris : Larousse.

[8] Jean-Paul Boons, Alain Guillet, and Christian Leclère. La structure des phrases simples en français : classes de constructions transitives. Technical report, LADL, Paris, 1976.

[9] Jean-Paul Boons, Alain Guillet, and Christian Leclère. *La structure des phrases simples en français : constructions intransitives*. Droz, Genève, 1976.

[10] Anne Brüggemann-Klein and Derick Wood. On predictive parsing and extended context-free grammars. In J.-M. Champarnaud and D. Maurel, editors, *Seventh International Conference on Implementation and Application of Automata*. University of Tours, 2002.

[11] Folker Caroli. Les verbes transitifs à complément de lieu en allemand. *Lingvisticæ Investigationes*, 8(2) :225–267, 1984. Amsterdam-Philadelphia : John Benjamins Publishing Company.

[12] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In MIT Press, editor, *Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI*, 1997.

[13] Eugene CHARNIAK. Statistical techniques for natural language parsing. *AI Magazine*, 1997.

[14] Fu-Dong CHIOU, David CHIANG, and Martha PALMER. Facilitating treebank annotation using a statistical parser. In *Proceedings of Human Language Technology Conference*, 2001.

[15] Noam CHOMSKY. *Syntactic Structures.* Mouton, La Haye, 1957.

[16] Noam CHOMSKY. *The Minimalist Program.* MIT Press, 1995.

[17] David CLÉMENCEAU. Finite-state morphology : Inflections and derivations in a single framework using dictionaries and rules. In Emmanuel ROCHE and Yves SCHABES, editors, *Finite-state language processing.* The MIT Press, 1997.

[18] Michael COLLINS. A new statistical parser based on bigram lexical dependencies. In *Proceedings of ACL 96*, 1996.

[19] Michael COLLINS, Jan HAJIC, Lance RAMSHAW, and Christoph TILLMANN. A statistical parser for Czech. In *Proceedings of ACL 99*, 1999.

[20] Matthieu CONSTANT. *Grammaires locales pour l'analyse automatique de textes : méthodes de construction et outils de gestion*, 2003. Thèse de doctorat. Université de Marne-la-Vallée.

[21] Terry COPECK, Sylvain DELISLE, and Stan SZPAKOWICZ. Parsing and case analysis in tanka. In *Proceedings of the 14th International Conference on Computational Linguistics - COLING-92*, 1992.

[22] Ann COPESTAKE and Dan FLICKINGER. An open source grammar development environment and broad-coverage english grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000), Athens, Greece*, 2000.

[23] Anibale ELIA. *Le verbe italien. Les complétives dans les phrases à un complément.* Schena/Nizet, Fasano/Paris, 1984.

[24] Anibale ELIA. *Lessico-grammatica dei verbi italiani a completiva. Tavole e indice generale.* Liguori, Napoli, 1984.

[25] Cédrick FAIRON. *Structures non-connexes. Grammaires des incises en français : description linguistique et outils informatiques*, 2000. Thèse de doctorat. Université Paris 7.

[26] Nathalie FRIBURGER. *Reconnaissance automatique des noms propres : application à la classification automatique de textes journalistiques*, 2002. Thèse de doctorat. Université de Tours.

[27] David GAATONE. *Le passif en français.* Duculot, Paris-Bruxelles, 1998.

[28] Jean H. GALLIER, Salvatore LA TORRE, and Supratik MUKHOPADHYAY. Deterministic Finite Automata with Recursive Calls and DPDA's, 2002. Trouvé à l'adresse `http://www.cis.upenn.edu/~jean/recdfa.ps`.

[29] Jacqueline GIRY-SCHNEIDER. *Les prédicats nominaux en français. Les phrases simples à verbe support.* Droz, Genève-Paris, 1987.

[30] Sheila GREIBACH. A new normal form theorem for context-free phrase structure grammars. *Journal of ACM*, 12 :42–52, 1965.

[31] Maurice GREVISSE. *Le bon usage*. J. Duculot, Gembloux, 1955. Sixième édition.

[32] Gaston GROSS. Classes d'objets et descriptions des verbes. *Langages*, 115, 1994. Paris : Larousse.

[33] Gaston GROSS. *Les expressions figées en français*. Ophrys, Paris, 1996.

[34] Maurice GROSS. *Méthodes en syntaxe*. Hermann, Paris, 1975.

[35] Maurice GROSS. On the failure of generative grammar. *Language*, 55(4) :859–885, 1979.

[36] Maurice GROSS. *Grammaire transformationnelle du français. 3 - Syntaxe de l'adverbe*. ASSTRIL, Paris, 1986.

[37] Maurice GROSS. Un nouvel agent en *par*. *Langages*, 109, 1993. Paris : Larousse.

[38] Maurice GROSS. Lemmatization of compound tenses in english. *Lingvisticæ Investigationes*, 22, 1998. Amsterdam-Philadelphia : John Benjamins Publishing Company.

[39] Maurice GROSS. Sur la définition d'auxiliaire du verbe. *Langages*, 135, 1999. Paris : Larousse.

[40] Maurice GROSS. Les ambiguïtés. *Lingvisticæ Investigationes*, 24(1), 2001. Amsterdam-Philadelphia : John Benjamins Publishing Company.

[41] Maurice GROSS and Jean SENELLART. Nouvelles bases statistiques pour les mots du français. In *Actes des JADT*. Université de Nice, 1998.

[42] XTAG Research Group. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

[43] Alain GUILLET and Christian LECLÈRE. *La structure des phrases simples en français : les constructions transitives locatives*. Droz, Genève, 1992.

[44] Zellig S. HARRIS. Transformational theory. *Language*, 41(3) :363–401, 1965.

[45] Zellig S. HARRIS. *Mathematical structures of language*. Wiley, New York, 1968.

[46] Zellig S. HARRIS. *A theory of language and information : a mathematical approach*. Clarendon press, Oxford, 1991.

[47] F. KARLSSON, A. ANTTILA, J. HEIKKILA, and A. VOUTILAINEN. *Constraint Grammar : a Language Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin,New-York, 1995.

[48] Krit KOSAWAT. *Méthodes de segmentation et d'analyse automatique de textes thaï*, 2003. Thèse de doctorat. Université de Marne-la-Vallée.

[49] Jacques LABELLE. *Etude de constructions avec opérateur avoir (nominalisations et extensions)*, 1974. Thèse de doctorat. Université Paris 8.

[50] Stefan LANGER. Zur morphologie und semantik von nominalkomposita. In *Tagungsband KONVENS 98. Zur Kodierung der Kompositionsformen (Fugenelemente) für Nomina im CISLEX*, 1988.

[51] Eric LAPORTE and Anne MONCEAUX. Elimination of lexical ambiguities by grammars : The ELAG system. *Lingvisticæ Investigationes*, 22, 1998. Amsterdam-Philadelphia : John Benjamins Publishing Company.

[52] Christian LECLÈRE. Datifs syntaxiques et datif éthique. *Méthodes en grammaire française*, pages 73–96, 1976. Paris : Klincksieck.

[53] Christian LECLÈRE. Organisation du lexique-grammaire des verbes du français. In B. Courtois and M. Silberztein, editors, *Langue Française*, volume 87, Les dictionnaires électroniques du français. Larousse, 1990.

[54] Cláudio LUCCHESI and Tomasz KOWALTOWSKI. Applications of finite automata representing large vocabularies. *Software - Practice and Experience*, 23(1) :15–30, 1993.

[55] Elisabete MARQUES RANCHHOD. Remarks on the complementation of aspectual verbs. *Lingvisticæ Investigationes Supplementa 24*, Syntaxe, Lexique et Lexique-Grammaire. Hommage à Maurice Gross, (à paraître, 2003). Amsterdam-Philadelphia : John Benjamins Publishing Company.

[56] Yvette Yannick MATHIEU. Quelques passifs avec agent obligatoire. *Langages*, 109, 1993. Paris : Larousse.

[57] Annie MEUNIER. *Nominalisation d'adjectifs par verbes supports*, 1981. Thèse de doctorat. Université Paris 7.

[58] Dong-Ho PAK. *Lexique-grammaire comparé français-coréen. Syntaxe des constructions complétives.* PhD thesis, UQAM, Montréal, 1996.

[59] Soun-Nam PARK. *La construction des verbes neutres en coréen*, 1996. Thèse de doctorat. Université Paris 7.

[60] Sébastien PAUMIER. A time-efficient token representation for parsers. In proceedings of Workshop on Finite-State Methods in Natural Language Processing at EACL 2003.

[61] Sébastien PAUMIER. Nouvelles méthodes pour la recherche d'expressions dans de grands corpus. In Anne Dister, editor, *Revue Informatique et Statistique dans les Sciences Humaines*, volume Actes des 3èmes Journées INTEX, pages 289–295, 2000.

[62] Sébastien PAUMIER. *Recherche d'expressions dans de grands corpus : le système* AGLAE, 2000. Mémoire de DEA. Université de Marne-la-Vallée.

[63] Sébastien PAUMIER. Some remarks on the application of a lexicon-grammar. In *Lingvisticæ Investigationes*, volume 24, Amsterdam-Philadelphia, 2001. John Benjamins Publishing Company.

[64] Sébastien PAUMIER. Some remarks on the application of a lexicon-grammar. Online Proceedings of the 4th Intex workshop, 2001. Téléchargeable à l'adresse `http ://www.nyu.edu/pages/linguistics/intex/downloads/SebastienPaumier.pdf`.

[65] Sébastien PAUMIER. Manuel d'utilisation du logiciel UNITEX. Téléchargeable à l'adresse `http://www-igm.univ-mlv.fr/~unitex/manuelunitex.ps`, 2002.

[66] Roger-Bruno RABENNILAINA. *Le verbe malgache.* AUPELF-UREF et Université Paris 13, Paris, 1991.

[67] Dominique REVUZ. *Dictionnaires et lexiques : méthodes et algorithmes*, 1990. Thèse de doctorat. Université Paris 7.

[68] Emmanuel ROCHE. *Analyse syntaxique transformationnelle du français par transducteurs et lexique-grammaire*, 1993. Thèse de doctorat. Université Paris 7.

[69] Emmanuel ROCHE and Yves SCHABES, editors. *Finite-state language processing.* The MIT Press, 1997.

[70] Morris SALKOFF. Bees are swarming in the garden. *Language*, 59(2) :288–346, 1983.

[71] Jean SENELLART. *Outils de reconnaissance d'expressions linguistiques complexes dans des grands corpus*, 1999. Thèse de doctorat. Université Paris 7.

[72] Jean SENELLART. Reconnaissance automatique des entrées du lexique-grammaire des phrases figées. In B. Lamiroy, editor, *Le lexique-grammaire*, number 37 in Travaux de linguistique, pages 109–121. Duculot, Bruxelles, 1999.

[73] Max SILBERZTEIN. *Dictionnaires électroniques et analyse automatique de textes : Le système INTEX*. Masson, Paris, 1993.

[74] Max SILBERZTEIN. INTEX : a corpus processing system. In *COLING 94 Proceedings*, Kyoto, 1994.

[75] Max SILBERZTEIN. Manuel d'utilisation d'INTEX version 4.12. Téléchargeable à l'adresse `http://grelis.univ-fcomte.fr/intex/downloads/Manuel.pdf`, 1999.

[76] Carlos SUBIRATS-RÜGGEBERG. *Sentential complementation in Spanish. A lexico-grammatical study of three classes of verbs.* John Benjamins, Amsterdam/Philadelphia, 1987.

[77] Min TANG, Xiaoqiang LUO, and Salim ROUKOS. Active learning for statistical natural language parsing. In *Proceedings of the Association for Computational Linguistics 40th Anniversary Meeting*, 2002.

[78] Caroline TICE and Susan L. GRAHAM. Key instructions : Solving the code location problem for optimized code. Technical Report 164, Compaq System Research Center, 2000.

[79] Thomas TREIG. Complétives en allemand. classification. Technical Report 7, LADL, 1977.

[80] Lidia VARGA. Classification syntaxique des verbes de mouvement en hongrois dans l'optique d'un traitement automatique. In F. Kiefer, G. Kiss, and J. Pajzs, editors, *Papers in Computational Lexicography (COMPLEX)*, pages 257–265, Budapest, Research Institute for Linguistics, Hungarian Academy of Sciences, 1996.

[81] Jacques VERGNE. Analyse syntaxique automatique de langues : du combinatoire au calculatoire. `http://users.info.unicaen.fr/~jvergne/#TALN 2001`.

[82] Atro VOUTILAINEN. Tagging and parsing with rules : The case of swedish. *Lingvisticæ Investigationes*, 24(1) :43–66, 2001. Amsterdam-Philadelphia : John Benjamins Publishing Company.

[83] Atro VOUTILAINEN and Juha HEIKKILÄ. An english constraint grammar (EngCG) : a surface-syntactic parser of english. In Fries, Tottie, and Schneider, editors, *Creating and using English language corpora*. Dept. General Linguistics, University of Helsinki, 1994.

Index