Université Denis Diderot – Paris 7

UFR d'Informatique **École doctorale de Mathématiques de Paris Centre**

Analyse automatique du français : lexiques, formalismes, analyseurs

THÈSE

présentée et soutenue publiquement le 7 avril 2006

pour l'obtention du titre de

Docteur de l'université Paris 7 Denis Diderot

(spécialité informatique)

par

Benoît Sagot

Composition du jury

Président :	Sylvain Kahane	Professeur des Universités (Paris 10)
Rapporteurs :	Philippe Blache	Directeur de Recherche au CNRS
	Gérard Huet	Directeur de Recherche à l'INRIA, Membre de l'Académie des Sciences
Examinateur :	John Carroll	Reader à l'Université de Sussex
Directeur de thèse :	Laurence Danlos	Professeur des Universités (Paris 7) Membre de l'Institut Universitaire de France
Co-directeur de thèse :	Éric de La Clergerie	Chargé de Recherche à l'INRIA
Invité :	Pierre Boullier	Directeur de Recherche à l'INRIA

INRIA Rocquencourt – Projet ATOLL

Remerciements

Je voudrais tout d'abord remercier Laurence Danlos d'avoir dirigé cette thèse. C'est à elle que je dois nombre de remarques pertinentes sur mon travail, un accueil chaleureux au sein de TALaNa, mais également quelques soirées fort sympathiques.

Je remercie également Éric de La Clergerie, qui m'a accueilli à l'INRIA au sein du projet Atoll et qui a co-encadré ma thèse. C'est grâce à lui qu'Atoll a su mener à bien des projets ambitieux, comme la fameuse campagne EASy, et nombreux sont par conséquents les travaux que j'ai effectué en collaboration ou en interaction avec lui, toujours de façon intéressante et constructive.

Les commentaires, les remarques et les conseils de mes deux rapporteurs, Philippe Blache et Gérard Huet, m'ont permis d'améliorer le manuscrit originel. Qu'ils soient ici remerciés d'avoir accepté de consacrer de leur temps à cette tâche. Merci également aux autres membres de mon jury, John Carroll et Sylvain Kahane, d'avoir accepté cette responsabilité. Sylvain, en particulier, m'a guidé dans les inévitables errements des premiers temps de ma thèse, et les discussions que j'ai eues avec lui par la suite ont toujours été très enrichissantes.

Je voudrais exprimer une immense gratitude envers Pierre Boullier. C'est probablement avec lui que j'ai le plus collaboré tout au long de ces années. Ses recherches sont à l'origine même de certaines parties de cette thèse, et nombreux sont les travaux que nous avons menés ensemble. Ainsi, je me souviendrai encore longtemps des longues journées passées derrière son écran à démêler avec amusement et détermination les arcanes de notre analyseur, espérant produire des résultats raisonnables dans les temps impartis par la campagne EASy. Travailler avec Pierre, c'est faire l'expérience d'une alliance précieuse entre intelligence, pertinence, humilité et gentillesse. Je lui dois beaucoup, et lui en suis très reconnaissant.

J'adresse des remerciements tout aussi chaleureux à Lionel Clément. À bien des égards, il est pour moi l'exemple à suivre dans notre domaine. Il est l'incarnation de ce qu'il est convenu d'appeler la double compétence, mais également un modèle d'humilité, de sérieux et de rigueur. C'est lui qui m'a initié à la linguistique, et je le compte désormais parmi mes amis.

Tous les autres membres du projet Atoll ont également contribué, d'une façon ou d'une autre, à ce que ces années se passent aussi bien qu'elles se sont passées. Bernard Lang, dont les travaux et les idées contribuent à donner une vision au projet Atoll. Philippe Deschamp, tout à la fois informaticien et lexicographe. Guillaume Rousse, qui m'a aidé d'innombrables fois dans le développement de nombreux outils. François Barthélémy, pour nos discussions sur la morphologie du turc ou du slovaque. Et tous les autres membres qui ont été membres permanents du projet à un moment ou à un autre. Tous ceux qui ont effectué un stage au sein d'Atoll au cours de ces années m'ont également beaucoup apporté, chacun pour des raisons différentes, et en particulier José Manuel Aguirre-Ruiz, Angélique Pochon, Tania Samoussina, Mehdi Ben Hmida, Alexandra Mounier, Milagros Fernandez Gavilanes. Merci enfin à nos assistantes successives, Josy Baron, Emanuelle Grousset et Nadia Mesrar, pour leur aide et leur souriant dévouement.

À TALaNa également, j'ai eu la chance de discuter et de collaborer avec des chercheurs et des étudiants hors du commun, comme Pascal Amsili, Alexandra Volanschi, François Toussenel, Laurence Delort et tous les autres doctorants et stagiaires. Une mention spéciale pour Alexis Nasr, depuis près d'un an en délégation au sein d'Atoll, et grâce à qui de nouvelles perspectives ont pu commencer à être mises en œuvre. J'espère vivement que notre collaboration se poursuivra activement dans l'avenir.

Je voudrais également remercier tous les autres chercheurs et doctorants que j'ai rencontrés et avec qui j'ai discuté voire travaillé. Ils ont tous contribué à me faire découvrir d'autres aspects de notre domaine ou d'autres points de vue. Merci en particulier à Susanne Alt, Benoit Crabbé, Claire Gardent, Kim Gerdes, Marie-Laure Guénot, Éric Laporte, Yannick Parmentier, Christian Rétoré, Djamé Seddah, à tous les autres membres des projets LexSynt et Mosaïque, et à tous ceux que j'ai oublié.

C'est avec émotion que je remercie aussi tous mes amis et toute ma famille. Ils se souviendront des périodes d'enthousiasme, mais aussi des moments difficile que, pour une raison ou une autre, j'ai eu à traverser pendant ces années. Ils m'ont toujours été d'un indéfectible secours, et bien que je ne sois pas sûr ce document sera pour eux d'une lecture passionnante, c'est en grande partie grâce à eux qu'il a pu voir le jour.

Pour finir, je dédie cette thèse à Katka.

Table des matières

III Utilisation des informations lexicales pour l'analyse automatique

Table des figures

10 Analyse non-linéaire : les Méta-RCG **191**

11 Mise en œuvre de systèmes d'analyse complets **221**

12 Évaluation de systèmes d'analyse syntaxique **241**

Liste des tableaux

Introduction

1 Informatique et linguistique

Depuis l'arrivée des premiers ordinateurs, une des applications importantes en a été le traitement automatisé des langues naturelles. Cette entreprise, toujours en construction, a connu tantôt des périodes fastes et tantôt des remises en question en profondeur. Elle continue toutefois à poursuivre sa quête, la mécanisation du langage. Mais la masse de données textuelles produites et accessibles a récemment connu une augmentation considérable. Parallèlement, la puissance des ordinateurs utilisés quotidiennement a poursuivi sa croissance exponentielle. Ces deux évolutions ont remis au goût du jour les applications linguistiques de l'informatique.

L'intérêt majeur de ce domaine de recherches est d'être à la croisée de nombreuses disciplines qui doivent toutes interagir étroitement, comme la linguistique, l'algorithmique, l'informatique et les mathématiques. Inévitablement, tout chercheur apporte une contribution teintée par celle de ces disciplines pour laquelle il a reçu une formation plus approfondie. Pour notre part, il s'agit de l'informatique. Toutefois, tout travail pertinent ne peut que conduire à une connaissance approfondie des autres domaines. C'est ce qui en fait la richesse.

Différentes approches coexistent aujourd'hui dans le domaine du traitement automatique des langues (TAL). Elles diffèrent non par leur pertinence, mais par leurs objectifs et les moyens qu'elles emploient. Traditionnellement, on distingue les travaux en analyse des travaux en génération. Les premiers se concentrent sur la modélisation de la structure et/ou du sens d'énoncés pris en entrée, alors que les seconds ont pour but la production d'énoncés à partir de représentations abstraites de leur sens. Les travaux présentés ici, bien que pouvant avoir un impact sur certains travaux en génération, se concentrent sur l'analyse.

Les démarches les plus directement applicatives dans le domaine de l'analyse reposent le plus souvent sur des techniques stochastiques appliquées à de grandes masses de données textuelles. Ces approches statistiques permettent l'obtention de résultats convenables à relativement peu de frais, avec une bonne robustesse face aux innombrables surprises que réserve tout échantillon de données linguistiques, mais sans que l'on puisse en attendre la possibilité d'extraire et d'utiliser des informations complexes.

Indépendamment de ces techniques, de nombreux travaux de linguistes ont pour objectif la formalisation d'idées linguistiques théoriques : c'est la linguistique formelle. Cette approche, qui vise à faire de la linguistique une science positive, se préoccupe principalement de l'adéquation des formalisations proposées à une modélisation linguistiquement satisfaisante et scientifiquement pertinente des données linguistiques.

Entre ces deux approches se logent de nombreux travaux applicatifs qui cherchent à utiliser informatiquement les modèles proposés par la linguistique formelle, sans tourner le dos aux approches stochastiques. La difficulté de l'exercice est multiple. Tout d'abord, ces modèles ne sont pas toujours conçus à des fins d'implémentation. La pertinence linguistique d'une modélisation formelle ne saurait en effet se mesurer aux propriétés algorithmiques des programmes permettant son implémentation. Quant aux modèles explicitement conçus pour être exploitables informatiquement, ils ne sont pas toujours linguistiquement satisfaisants. Ensuite, il est bien plus difficile de développer des outils reposant sur de tels modèles que de mettre en œuvre des traitements statistiques. Cette difficulté ne peut être surmontée qu'au prix de connaissances très pointues en algorithmique de l'analyse syntaxique. Enfin, tout traitement à grande échelle de données linguistiques nécessite une description aussi complète que possible de la langue considérée (grammaire et lexique), et pas seulement une analyse très pointue de quelques phénomènes syntaxiques bien particuliers.

1.1 Lexiques et grammaires

Le traitement automatique d'une langue repose sur l'étroite interaction entre une grammaire et un lexique, qui représentent tous deux, dans un formalisme donné, l'ensemble des informations dont on dispose. On entend par *lexique* l'ensemble des informations spécifiques à chaque unité lexicale de la langue (pour faire simple, à chaque mot), et par *grammaire* les informations articulatoires permettant de décrire de façon générale la façon dont se combinent les unités lexicales pour former des énoncés (suites linéaires d'unités lexicales) ayant une structure et véhiculant de manière explicite ou implicite un sens.

Chaque formalisme linguistique peut interpréter à sa manière ces définitions génériques. En particulier, un certain nombre de formalismes grammaticaux, dits *lexicalisés*, réduisent à sa plus simple expression la partie grammaticale : les informations articulatoires sont décrites dans le lexique sous forme de structures combinables, et la grammaire à proprement parler se réduit à la définition de quelques opérateurs de combinaison de ces structures lexicales. C'est le cas de formalismes aussi différents que les Lexicalized Tree-Adjoining Grammars (LTAG) ou le lexique-grammaire de Maurice Gross.

D'autres formalismes répartissent les tâches de manière plus équilibrée entre le lexique et la (ou les) grammaires. C'est en particulier le cas des Lexical-Functional Grammars (LFG), des grammaires de contraintes ou des formalismes issus de la Théorie Sens-Texte d'Igor Meľčuk (GUST). Le formalisme se doit alors de décrire la façon dont interagissent lexique et grammaire. Naturellement, l'étendue des phénomènes couverts par la grammaire dicte la richesse informationnelle dont on a besoin dans le lexique. A l'inverse, un lexique moins riche en informations ne pourra être utilisé convenablement que par des grammaires n'attendant pas plus d'informations que celles qui sont disponibles.

1.2 Efficacité et robustesse

Le développement d'outils pour l'analyse automatique repose donc sur de tels formalismes. Il s'agit toutefois d'un domaine de recherches en soi, dont l'objectif premier est l'optimisation des outils ainsi construits. En effet, la complexité des formalismes et des grammaires, ainsi que l'ambiguïté (souvent insoupçonnée) des mécanismes mis en jeu, rendent non-triviale la recherche de l'efficacité.

De plus, une autre problématique a une importance cruciale : la robustesse. En effet, il n'y a pas nécessairement en linguistique de frontière entre les énoncés corrects et les énoncés incorrects. De nombreux formalismes reposent pourtant sur une telle dichotomie. Il est alors nécessaire de disposer de moyens pour construire autant d'informations que possible non seulement à partir d'énoncés corrects, mais également d'énoncés incorrects. Ce n'est qu'à ce prix que l'on peut véritablement envisager des traitements linguistiques automatisés à grande échelle et dans un environnement réel.

1.3 Syntaxe et sémantique

Aujourd'hui, la majorité des systèmes d'analyse automatisée d'énoncés en langue naturelle ne produit d'analyse que syntaxique. C'est-à-dire que le résultat de l'analyse d'une phrase consiste en la représentation de la structure grammaticale de la phrase dans le formalisme choisi et selon une grammaire écrite dans ce formalisme. Les formalismes utilisés par ce type de système sont dits *formalismes syntaxiques*, puisqu'ils permettent le développement de grammaires qui modélisent la syntaxe d'une langue naturelle. Ils introduisent parfois quelques éléments de sémantique, c'est-à-dire ayant trait au sens des énoncés, dans la mesure où ils sont inévitables pour modéliser correctement la syntaxe.

Cependant, la syntaxe prise indépendamment de la sémantique ne permet pas toujours de résoudre les ambiguïtés qui peuvent apparaître lors de l'analyse, et ne permet évidemment pas la construction de représentations du sens d'un énoncé. C'est la raison pour laquelle de nombreux

travaux récents, poursuivant ainsi un certain nombre de travaux plus anciens, cherchent à intégrer des informations sémantiques plus riches dans les modélisations linguistiques et dans les implémentations de ces modélisations.

La prise en compte d'informations sémantiques est cependant loin d'être immédiate. Elle nécessite des lexiques enrichis d'informations sémantiques et des grammaires mêlant, simultanément ou successivement, syntaxe et sémantique. Par conséquent, elle passe par le développement de formalismes capables de représenter de tels lexiques et de telles grammaires. Il peut s'agir de formalismes qui étendent des formalismes syntaxiques, ou de formalismes développés spécifiquement. Mais dans tous les cas, ces formalismes sont plus complexes que les formalismes purement syntaxiques, et nécessitent à la fois une assise linguistique plus profonde et des techniques informatiques et algorithmiques plus élaborées si l'on souhaite les mettre en œuvre dans des applications efficaces. De plus, les contraintes sémantiques sont souvent bien moins rigides que les contraintes syntaxiques : la problématique de la robustesse prend ici une importance cruciale.

2 Présentation du travail

2.1 Objectifs

C'est dans ce cadre que se situe ce travail de thèse. L'objectif principal en est de comprendre certaines limites des approches actuelles, et de les dépasser en réconcilliant au mieux pertinence linguistique, efficacité opérationnelle et robustesse des outils. En particulier, nous avons cherché à comprendre comment des informations syntaxiques et sémantiques, présentes principalement dans le lexique mais également dans la grammaire, peuvent être utilisées dans les analyseurs tout en manipulant des représentations linguistiquement pertinentes.

Un tel objectif, très large, concerne donc tout à la fois

- les problématiques liées à la représentation, la structuration et l'acquisition d'un lexique qui contient à la fois des informations morphologiques, syntaxiques et sémantiques,
- le choix de formalismes existants ou le développement de nouveaux formalismes, qui puissent permettre, en conservant des propriétés algorithmiques raisonnables, de représenter une grammaire traitant la syntaxe, voire également la sémantique,
- le développement effectif de systèmes complets d'analyse, et par conséquent de lexiques, de grammaires, d'analyseurs, et de tous les outils permettant ce développement.

2.2 Résultats principaux

Cette thèse tente de répondre à ces trois problématiques. Au niveau des formalismes et des analyseurs, nous avons suivi deux pistes, qui ont toutes deux un même objectif à moyen terme mais reposent sur deux compromis différents. La première piste consiste en l'exploration d'une approche relativement originale de l'analyse et de la formalisation linguistique, que l'on peut caractériser par le terme d'*analyse non-linéaire*. Nous montrons en effet que des raisons à la fois linguistiques, algorithmiques et mathématiques[1] conduisent à l'utilisation de formalismes non-linéaires[2]. Malgré cela, les formalismes syntaxiques ou syntaxico-sémantiques utilisés communément combinent le plus souvent un squelette *linéaire* et des *décorations logiques* se combinant par unification par-dessus les analyses produites par le squelette. Nous montrons les inconvénients de ces architectures à deux niveaux, en mettant en avant au contraire les avantages des formalismes non-linéaires, qui peuvent se passer de ces décorations.

En particulier, nous introduisons un nouveau formalisme linguistique non-linéaire qui permet à la fois la construction d'analyseurs efficaces (polynomiaux) et la représentation de façon satisfaisante de faits linguistiques concernant au moins la syntaxe, la sémantique et les relations de discours. Les analyses obtenues sont alors suffisemment riches pour exprimer de façon satisfaisante tous ces faits linguistiques, en sorte que les analyses en constituants, en dépendances syntaxiques, en boîtes topologiques ou en sémantique prédicative peuvent être obtenues par simple projection de l'analyse complète. Ce formalisme de description repose sur un formalisme d'implémentation, les Grammaires à Concaténation d'Intervalles (Range Concatenation Grammars, RCG), introduit par Pierre Boullier. Ce dernier ayant développé un analyseur pour ces grammaires, la construction d'analyseurs pour notre formalisme de description (appelé Méta-RCG) n'a nécessité que le développement d'un compilateur traduisant une Méta-RCG en RCG. Pour montrer la pertinence de notre approche, nous avons développé une grammaire du français à moyenne couverture dans le formalisme Méta-RCG, ainsi qu'un lexique syntaxico-sémantique jouet couvrant un corpus de test particulier.

L'autre piste que nous avons suivie pour l'analyse automatique consiste à utiliser un formalisme déjà bien connu, celui des Grammaires Lexicales-Fonctionnelles (LFG), pour développer un analyseur, SXLFG, qui allie grande efficacité, robustesse, et techniques de désambiguïsation heuristiques. À l'aide d'une grammaire LFG du français à large couverture, ainsi que d'une chaîne de traitement pré-syntaxique que nous avons développée[3], nous sommes en me-

[1] Au sens où la théorie des grammaires et des langages formels est une branche des mathématiques, aux frontières de l'informatique et de la logique.

[2] Par *non-linéaires*, nous voulons dire qu'il est possible de faire interagir plusieurs contraintes sur un même mot de la phrase : les mots sont « consommables » plusieurs fois. Le parallèle est direct avec les logiques non-linéaires, qui, contrairement aux logiques linéaires, peuvent utiliser plus d'une fois un même axiome.

[3] Et qui est utilisée en amont d'autres analyseurs.

sure d'effectuer l'analyse profonde et non-probabiliste de corpus de plusieurs millions de mots en quelques heures[4]. De plus, les techniques de robustesse mises en œuvre nous permettent de dépasser la dichotomie, habituelle avec des formalismes génératifs comme LFG[5], entre énoncés analysés et énoncés rejetés. À terme — et ce travail est actuellement en cours — nous avons pour ambition de combiner les avantages de la non-linéarité des Méta-RCG avec l'extrême efficacité et les techniques de robustesse de notre analyseur LFG.

Les travaux sur les analyseurs ne sont toutefois possibles qu'à la condition de disposer de ressources lexicales adaptées et riches d'informations. C'est pourquoi nous avons également mené différents types de travaux sur la représentation et l'acquisition d'informations lexicales. Nous avons mis en place une architecture hiérarchique permettant de représenter un lexique morphologique et syntaxique à large couverture de manière compacte et facile à maintenir, ainsi que diverses techniques permettant d'automatiser autant que faire se peut la constitution d'un tel lexique à partir de corpus bruts. En particulier, nous avons développé des méthodes originales d'acquisition automatique de lexiques morphologiques à partir de corpus bruts, des méthodes d'acquisition automatique d'informations syntaxiques élémentaires à partir de corpus étiquetés, et des méthodes de fouille d'erreurs dans les résultats d'analyses syntaxiques pour identifier des manques et des erreurs dans les ressources utilisées, dont les lexiques, et pour acquérir des informations quantitatives destinées à court terme à compléter nos techniques de guidage et de désambiguïsation. Le résultat de ces travaux est le Le*fff* (Lexique des formes fléchies du français), lexique morphologique et syntaxique disponible librement et utilisé (entre autres) par nos analyseurs.

Pour résumer ce qui précède, nous nous proposons de développer, de défendre et d'illustrer les trois principes suivants :

1. Il est possible d'effectuer efficacement l'analyse non-générativiste d'énoncés avec des formalismes génératifs justifiés linguistiquement.
2. La place des probabilités pour la modélisation des langues est dans le classement d'hypothèses obtenues par des mécanismes symboliques : apprentissage automatique (avec le cas échéant validation manuelle), désambiguïsation, guidage de rattrapage d'erreurs, etc.
3. Il est pertinent et possible de coupler des contraintes morphologiques, syntaxiques et de sémantique lexicale, voire au-delà, à la fois dans les lexiques, les grammaires et les analyseurs.

[4]Nous avons par exemple participé à la campagne EASy d'évalution des anlayseurs syntaxiques, quoiqu'à un stade peu avancé du développement de SXLFG et de la grammaire du français associée.

[5]Par *génératifs* nous entendons ici des formalismes définissant un ensemble d'énoncés comme étant valides, à l'exclusion stricte des autres, et qui permettent en théorie la génération de tous ces énoncés valides.

2.3 Déroulement

La première partie de ce document dresse un état des lieux bref et partiel des trois grandes notions qui sont aux fondements de la recherche en traitement automatique des langues : les formalismes (chapitre 1), les lexiques (chapitre 2) et les analyseurs (chapitre 3).

La seconde partie peut être considérée comme un préliminaire à ce qui suit, puisqu'elle traite des problématiques liées à la représentation et l'acquisition d'informations lexicales, techniques utilisées entre autres dans le développement du Le*fff*. Le chapitre 4 présente un formalisme de description de la morphologie (flexionnelle et dérivationnelle) que nous avons développé. Le chapitre 5 décrit le format intensionnel que nous avons défini pour le développement du Le*fff*, qui repose sur une architecture par héritage de propriétés syntaxiques atomiques, ainsi que le format extensionnel, utilisé par les analyseurs, obtenu par compilation. Le chapitre 6 décrit la méthode que nous avons développée pour l'apprentissage automatique de lexiques morphologiques (avec des informations tant flexionnelles que dérivationnelles) à partir de corpus bruts, ainsi qu'une extension de cette méthode permettant l'apprentissage automatique d'étiqueteurs morphologiques, également à partir de corpus bruts. Enfin, le chapitre 7 présente deux types de travaux que nous avons effectués pour faciliter le développement de la partie syntaxique du Le*fff*. D'une part nous présentons une méthode de fouille d'erreurs sur les résultats de l'analyse syntaxique de corpus volumineux, qui permet de détecter les manques et les erreurs dans les informations lexicales (y compris syntaxiques), mais aussi dans la chaîne SXPipe et dans les grammaires. D'autre part nous proposons une méthodologie simple pour l'acquisition rapide d'informations sémantiques élémentaires, comme les noms prédicatifs.

La troisième partie est centrée sur l'analyse automatique, et en particulier l'anayse du français. Le chapitre 8 présente tout d'abord notre vision sur les problématiques mises en œuvre dans le développement de formalismes linguistiques et d'analyseurs. Nous montrons en particulier comment l'introduction de la robustesse permet de ne pas ignorer la différence entre langue normée et langue performée, comment il est possible de faire de l'analyse automatique non-générativiste à partir de grammaires développées dans des formalismes génératifs, comment le développement de formalismes et d'analyseurs doit rechercher un compromis entre efficacité et pertinence linguistique, comment mettre en œuvre les différentes techniques de désambiguïsation, et pourquoi il est souhaitable de savoir prendre en compte des informations sémantiques. Nous présentons alors, comme indiqué plus haut, deux analyseurs qui implémentent de façon partielle mais complémentaire nos idées sur l'analyse. Tout d'abord, nous présentons notre analyseur LFG, nommé SXLFG (chapitre 9). Puis nous présentons au chapitre 10 l'approche non-linéaire pour la formalisation de la langue et l'analyse automatique. Nous montrons tout d'abord les fondements théoriques, à la fois linguistiques et algorithmiques, qui justifient l'introduction de la non-linéarité pour pouvoir prendre en compte de manière satisfaisante les faits

linguistiques relevant, entre autres, de la sémantique pendant l'analyse d'un énoncé. Elle compare les propriétés des formalismes non-linéaires à ceux des formalismes habituels, et définit notre formalisme, les Méta-RCG. Une grammaire du français écrite en Méta-RCG et le lexique syntaxico-sémantique associé sont alors présentés. Le chapitre 11 décrit alors les outils que nous avons développés pour la construction de systèmes d'analyse complets, et en particulier notre chaîne de traitements pré-syntaxiques SXPipe. Enfin, le chapitre 12 propose diverses expériences permettant l'évaluation de nos systèmes d'analyse, tant en termes d'efficacité et de couverture qu'en termes de précision.

Nous terminons par une conclusion générale, qui tente de tirer les principaux enseignements de nos travaux, et présente les pistes que nous souhaitons poursuivre à l'avenir.

Première partie

Formalismes, lexiques et analyseurs : état de l’art

Chapitre 1

Formalismes linguistiques

Note : tout au long de ce document, certains paragraphes sont munis d'un trait vertical dans la marge. Ceci signifie que le paragraphe correspondant est relativement formel. Dans la plupart des cas, et sauf lorsque nous ne l'avons pas jugé indispensable, une partie formelle est précédé d'une partie informelle qui la résume de façon plus lisible.

1 Structures, langages et analyseurs

1.1 Structures

1.1.1 Graphes dirigés, graphes dirigés acycliques (DAG), arbres

Soit un ensemble E. Un *graphe dirigé*[1] d'éléments de E est la donnée d'un couple $(F, \mathcal{A})$, où F est un sous-ensemble de E et où $\mathcal{A}$ est un sous-ensemble du produit cartésien $F \times F$. Les éléments de $\mathcal{A}$ sont appelés *arcs* du graphe, et les éléments de F en sont les *sommets*. Si $(s_1, s_2) \in \mathcal{A}$ est un arc, s_1 est dit sa *source* et s_2 sa *cible*. On a l'habitude de représenter un graphe dans un plan en représentant chaque sommet (élément de F) par un point et chaque arc (élément de $\mathcal{A}$) par une flèche allant de sa source à sa cible.

Soit un graphe $(F, \mathcal{A})$ d'éléments de E, et (s_1, s_2) un couple d'éléments de F. L'ensemble $\mathcal{A}$ définit canoniquement une relation binaire $\mathcal{R}_\mathcal{A}$ entre éléments de F, telle que $s_1 \mathcal{R}_\mathcal{A} s_2$ si et seulement si $(s_1, s_2) \in \mathcal{A}$. On dit que alors que s_1 est *relié* à s_2 (mais pas nécessairement que s_2 est relié à s_1). On appelle *parcours* dans le graphe une suite finie $s_1, \ldots, s_n$ d'au moins deux éléments telle que tout élément de cette suite est relié à l'élément suivant. On dit qu'un tel parcours va *de* s_1 *à* s_n. On appelle *parcours fermé* (ou *circuit*) dans un graphe dirigé un

[1] On notera qu'il y a une distinction entre les notions de graphe dirigé et de graphe orienté. C'est bien de celle de graphe dirigé que nous avons besoin.

parcours dont le premier élément est égal au dernier. Par ailleurs, ceux des sommets qui ne sont la source d'aucun arc sont appelés les *feuilles* du graphe, les autres sont les *nœuds internes*.

On appelle *graphe non-orienté* d'éléments de E la donnée d'un couple $(F, \mathcal{A})$, où F est un sous-ensemble de E et où $\mathcal{A}$ est un sous-ensemble des parties de F qui ont un ou deux éléments. On peut associer de façon canonique à tout graphe dirigé G un graphe non-orienté G', en conservant les mêmes sommets et en prenant pour arcs de G' d'une part l'ensemble des singletons $\{s\}$ tels que (s, s) est un arc de G, et d'autre part les ensembles $\{s_1, s_2\}$, où $s_1 \neq s_2$, tels que (s_1, s_2) ou (s_2, s_1) est un arc de G. Si $\{s_1, s_2\}$, où $s_1 \neq s_2$, est un arc de G', on dit que s_1 et s_2 sont *reliés* (c'est dans ce cas une relation symétrique). La notion de parcours est trivialement adaptée au cas des graphes non-orientés. Un graphe non-orienté est dit *connexe* si et seulement si pour tout couple (s_1, s_2) d'éléments de F il existe au moins un parcours qui va de s_1 à s_2. Un graphe dirigé est dit connexe si et seulement si le graphe non-orienté qui lui est canoniquement associé est lui-même connexe.

Par souci de simplification, on appelle souvent du même nom des éléments distincts de E, pour peu qu'on considère qu'ils partagent une certaine propriété en commun. Ainsi, $A \rightarrow A$ peut représenter un graphe à deux sommets distincts, bien que ceux-ci soient tous deux appelés A.

On appelle *graphe dirigé acyclique* (ou *DAG*, d'après le nom anglais *directed acyclic graph*) un graphe qui ne comprend aucun parcours fermé. La relation $\mathcal{R}_\mathcal{A}$ définit alors un ordre partiel. On peut représenter un DAG en représentant les arcs non pas par des flèches mais par des traits, pour peu qu'à l'ordre partiel défini sur les sommets par $\mathcal{R}_\mathcal{A}$ corresponde un ordre partiel (géométrique) entre les points représentant ces sommets (typiquement de haut en bas ou de gauche à droite).

On appelle *arbre* un DAG tel que tous les sommets sont la cible d'exactement un arc, sauf exactement un sommet qui n'est la cible d'aucun arc, appelé *racine*. C'est donc un DAG connexe.

1.1.2 Vocabulaire, chaîne, structure

On appelle *vocabulaire* (ou *alphabet*) un ensemble fini de symboles[2]. Une *chaîne* sur un certain vocabulaire est une séquence (ou liste, ou suite) finie de symboles qui sont des éléments de ce vocabulaire[3]. Une chaîne étant une liste, on dispose sur les chaînes d'opérations telles que :

[2]Le terme *symbole* désigne une entité atomique abstraite que l'on ne peut définir, tout comme les notions de *point* ou de *droite* ne peuvent être définies. Des exemples de symboles fréquemment utilisés sont les caractères (en théorie des langages) mais aussi les phonèmes ou les mots (en modélisation des langues).

[3]Pour le lecteur familier de ce type de notions, l'ensemble des chaînes d'un vocabulaire, muni de l'opération de concaténation définie plus bas, forme un monoïde libre dont le mot vide est l'élément neutre.

- la longueur d'une chaîne w, notée $|w|$: c'est le nombre d'éléments dans la chaîne ;
- l'accès à un élément d'une chaîne en fonction de sa *position* i : c'est le i-ième élément de la chaîne, noté w_i ;
- l'accès à une sous-chaîne en fonction de son *intervalle* noté $i..j$: c'est la sous-chaîne, notée $w_{i..j}$ formée des éléments $i+1$ à j inclus ;
- la concaténation de deux chaînes w_1 et w_2, notée $w_1\ w_2$ (ou w_1w_2 si cela ne prête pas à confusion[4]) ;
- l'itération k fois d'une chaîne w, notée w^k, que l'on peut définir récursivement de la façon suivante : $w^1 = w$ et $w^k = w\ w^{k-1}$,
- l'égalité de deux chaînes.

Par convention, on note ε la chaîne vide (de longueur nulle), on note par son unique symbole une chaîne de longueur 1, et on note les chaînes de longueur supérieure comme la concaténation de chaînes de longueur 1. On note $\mathcal{V}^*$ l'ensemble des chaînes de $\mathcal{V}$ (qui est donc l'ensemble des listes d'éléments de $\mathcal{V}$). Par exemple, si l'on choisit un vocabulaire $\mathcal{V} = \{a, b\}$, alors ε, a et $abab$ (ou $a\ b\ a\ b$) sont des éléments de $\mathcal{V}^*$. On dit, par abus de langage, qu'une chaîne sur un vocabulaire $\mathcal{V}$ (et donc un élément de $\mathcal{V}^*$), ou même simplement une chaîne dont tous ses symboles sont dans $\mathcal{V}$, est une chaîne de $\mathcal{V}$.

Dans la suite, nous appellerons *structure* sur E un graphe dirigé d'éléments de E ou une chaîne sur E considéré comme un vocabulaire. C'est donc un raccourci qui suppose dans certains cas que l'on ait défini une opération de concaténation.

1.2 Éléments d'une théorie des langages

Cette section tente de définir ce que sont les langages, les grammaires, les formalismes, les grammaires et les formalismes de réécriture, les structures de dérivation, les structures dérivées et d'autres notions nécessaires à leur définition.

Nous commençons par une présentation rapide et informelle, avant de tenter une définition formelle qui place dans un même cadre différents types de formalismes de réécriture, qu'il s'agisse de formalismes de réécriture de chaînes (grammaires non-contextuelles), d'arbres (grammaires d'adjonction d'arbres), ou de prédicats (grammaires à concaténation d'intervalle), sans contexte ou avec contexte (grammaires d'adjonction d'arbres multi-composants, grammaires contextuelles). Cette partie formelle est aride, mais elle est utile dans la mesure où les termes qui y sont définis seront utilisés de façon intensive dans ce document. Le lecteur familier de ces notions ou que les présentations formelles rebutent est donc invité à ne lire que la présentation informelle, puis à passer à la section suivante.

[4] Voire à l'aide d'un caractère autre qu'un espace pour noter la concaténation si l'espace est utilisé à d'autres fins.

1.2.1 Présentation informelle

Si l'on se donne un *vocabulaire*, c'est-à-dire un ensemble de symboles, ou *mots*, on appelle *langage* un sous-ensemble de toutes les suites de mots de ce vocabulaire (les *phrases*) que l'on peut construire. Par exemple, si le vocabulaire est l'ensemble $\{a, b\}$, alors $\{a^n b^n \mid n \geq 0\}$ est un langage, qui comprend toutes les phrases formées d'un nombre quelconque de fois le mot a puis du même nombre de fois le mot b. Ainsi, $aabb$ fait partie de ce langage, mais pas abb.

Pour définir un langage, on peut en donner une définition ensembliste, qu'elle soit intensionnelle, comme dans l'exemple précédent, ou extensionnelle, si le langage est fini, en donnant les unes après les autres toutes les phrases du langage. Mais le plus souvent ce n'est pas possible en pratique. On a alors recours à ce qu'on appelle des *grammaires de réécriture*, qui vont avoir un double rôle : d'une part elles vont permettre de définir un langage donné de façon concise, et d'autre part elles vont conférer à chaque phrase du langage une certaine structure, une certaine organisation.

Une grammaire de réécriture est un système formel qui manipule des structures de la façon suivante. On se donne tout d'abord un certain nombre de structures, dites *structures élémentaires*, associées à chaque mot du vocabulaire. Dans le cas général, ces structures sont des graphes dirigés (en réalité, ce sont souvent des arbres) dont les nœuds peuvent être étiquetés par des prédicats logiques ou simplement des constantes. Ces étiquettes sont appelés *composants*. On se donne également des règles de combinaison de structures. On parle de grammaires de *réécriture* car ces règles de combinaison de structures indiquent comment, dans une structure donnée, on peut (ou on doit) remplacer certains éléments par d'autres structures pour construire une nouvelle structure plus riche. Cette opération de remplacement s'appelle la *réécriture*, et les règles de combinaison sont des *règles de réécriture*. Supposons alors que l'on ait une suite de mots de notre vocabulaire. À quelle condition est-ce une phrase correcte, c'est-à-dire un élément du langage défini par notre grammaire ? Pour le savoir, on essaye de construire une structure complète qui relie d'une part une ancre spéciale, l'*axiome*, et d'autres part des structures élémentaires associées aux mots concernés, en respectant les règles de réécriture. Pour que cette construction, appelée *dérivation*, soit un succès, la structure obtenue, appelée *structure dérivée*, doit donc s'ancrer sur un *axiome*. La suite d'opérations effectuées pour construire la structure dérivée étant une suite de réécritures, on peut construire un graphe dirigé à partir de ces opérations, en reliant pour une opération donnée la structure élémentaire dans laquelle on effectue la réécriture à la structure élémentaire utilisée par cette réécriture. On obtient ainsi un graphe dirigé qui représente l'historique de la dérivation, appelé *graphe de dérivation*.

On appelle alors *reconnaissance* l'opération consistant à déterminer, pour une phrase donnée, si elle est dans le langage défini par une grammaire de réécriture. On appelle *analyse syntaxique*, ou plus simplement *analyse*, l'opération consistant à construire, pour une phrase

donnée, toutes les structures dérivées possibles d'une phrase (s'il y en a), ce qui a pour effet, en principe, de construire également toutes les structures de dérivation.

De façon légèrement plus précise une structure dérivée est correcte si on peut la construire à partir d'un axiome en appliquant les unes après les autres des règles de réécriture, jusqu'à un résultat final qui soit correct. Les résultats intermédiaires successifs sont appelés *proto-phrases*, et la dernière proto-phrase est correcte si elle vérifie une certaine fonction booléenne, dite fonction de correction, que nous noterons $Corr$. Cette proto-phrase finale n'a aucune raison particulière d'être la structure dérivée, et ceci n'est le cas que lorsque les opérations de réécriture ne font que rajouter de l'information sans jamais en retirer.

Une règle de réécriture a la forme suivante : $G \rightarrow D$, où G est appelé *partie gauche* de la règle, et D *partie droite*. À partir d'une proto-phrase donnée, on peut appliquer une telle règle si les composants spécifiés dans G (constantes ou prédicats logiques) se retrouvent dans la proto-phrase, à une relation d'équivalence près notée $\mathcal{R}$ (et qui est non-triviale seulement si les composants sont des prédicats). L'application de la règle se fait alors en remplaçant ce ou ces composants dans la proto-phrase d'entrée par ceux spécifiés dans D. S'il y a plusieurs composants dans G, ceci suppose que l'on a un moyen de savoir qui remplacer par quoi, et implique parfois certaines conditions sur la localisation des composants concernés dans la proto-phrase.

L'ensemble formé de $\mathcal{R}$, de $Corr$, ainsi que des ensembles de symboles permettant la définition des composants (constantes ou prédicats logiques, ainsi que les mots du vocabulaire) forme ce que l'on appelle un *formalisme de réécriture*. Nous voyons un formalisme (et en particulier un formalisme de réécriture) comme une fonction qui à une grammaire associe le langage qu'elle définit, pour peu que la grammaire appartienne à son domaine de définition (c'est une grammaire valide du formalisme).

1.2.2 Langage, grammaire, formalisme

Si $\mathcal{V}$ est un vocabulaire, nous appellerons *langage* sur $\mathcal{V}$ un ensemble fini ou infini de structures de $\mathcal{V}$. Si ces structures sont des chaînes, un langage est donc un sous-ensemble de $\mathcal{V}^*$, et nous parlerons alors de *langage de chaînes*. Un langage infini ne peut pas être défini par extension (en listant tous ses éléments), et ne peut donc être défini qu'en intension. Une telle définition en intension d'un langage $\mathcal{L}$ nécessite deux composants distincts, un formalisme et une grammaire, définis comme suit. On appelle *formalisme* une fonction $\mathcal{F}$ dont le domaine de définition $\mathcal{G}$ est un langage appelé *langage grammatical* et dont l'ensemble d'arrivée est l'ensemble des langages d'un certain vocabulaire $\mathcal{V}$.

On appelle *grammaire* un élément G de $\mathcal{G}$. Par abus de langage, on dit qu'une grammaire G qui appartient au domaine de définition $\mathcal{G}$ d'un formalisme $\mathcal{F}$ *appartient* à ce formalisme. On dit par ailleurs que la grammaire G *décrit* (ou *définit*) un langage $\mathcal{L}$ dans le formalisme

$\mathcal{F}$ si et seulement si $\mathcal{F}(G) = \mathcal{L}$. Par abus de langage, on note souvent $\mathcal{L}_{\mathcal{F}}(G)$ voir $\mathcal{L}(G)$ ce langage $\mathcal{L}$. Enfin, et toujours par abus de langage, si une grammaire G appartient au domaine de définition d'un formalisme $\mathcal{F}$, on dira que cette grammaire est une grammaire *de* $\mathcal{F}$, voire qu'elle *appartient* au formalisme $\mathcal{F}$.

On appelle *fonction de reconnaissance* la fonction d'appartenance d'une chaîne à un langage. On appelle *reconnaisseur* d'un langage un algorithme qui implémente d'une façon ou d'une autre sa fonction de reconnaissance[5]. On appelle *constructeur de reconnaisseurs* pour un formalisme $\mathcal{F}$ un algorithme qui construit un reconnaisseur à partir d'une grammaire dans ce formalisme.

1.2.3 Grammaires et formalismes de réécriture

La définition que nous allons donner des formalismes et des grammaires de réécriture essaye d'abstraire ce qui fait l'essence de ce qu'on appelle la *réécriture*. Nous avons fait en sorte, en particulier, que ces définitions conviennent à différents types de formalismes, qu'ils s'agisse de formalismes de réécriture de chaînes[6], d'arbres[7] ou de prédicats logiques[8].

On appelle *grammaire de réécriture* sur un vocabulaire $\mathcal{V}$ une grammaire G qui est un ensemble non vide de *règles de réécriture* sur $\mathcal{V}$. Une règle de réécriture sur $\mathcal{V}$ est un couple formé d'un $2n$-uplet $(G_1, \ldots, G_n, D_1, \ldots, D_n)$ de structures sur $\mathcal{V}$ (chaînes ou graphes dirigés d'éléments de $\mathcal{V}$) avec $n \geq 1$. Si les structures sont des chaînes de constantes, on parle de *grammaire de chaînes*. Si ce sont des chaînes de prédicats, on parle de *grammaire de prédicats*. Si ce sont des graphes, on parle de *grammaires de graphes* (ou de *grammaires d'arbres* si les graphes concernés sont des arbres). Une telle règle est représentée sous la forme $G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n$ (on suppose que les symboles $\rightarrow$ et la virgule ne sont pas des éléments de $\mathcal{V}$). $G_1, \ldots, G_n$ est appelé sa *partie gauche*, $D_1, \ldots, D_n$ sa *partie droite*. Chaque G_i est appelé *composant de partie gauche* de la règle, chaque D_j *composant de partie droite*. Les composants de partie gauche et de partie droite sont appelés *composants* de la règle. On supposera que les composants peuvent représenter des constantes ou des prédicats logiques, définis comme suit. On dit qu'un composant dans une règle d'une grammaire de graphes est constitué d'un seul *élément* (le composant lui-même), mais dans le cas d'une grammaire de chaînes ou de prédicats on dira que le composant a autant d'*éléments* qu'il y a de composants dans la chaîne.

Le vocabulaire $\mathcal{V}$ d'une grammaire de réécriture est partitionné en quatre sous-ensembles :

[5] On notera que, dans le cas général, rien ne garantit, pour un formalisme et une grammaire donnés, qu'un reconnaisseur puisse exister.

[6] Grammaires non-contextuelles (voir plus bas), mais en fait toutes les classes de grammaires de la hiérarchie de Chomsky.

[7] Comme les TAG

[8] Comme les RCG ou les sLMG.

- l'ensemble des *symboles terminaux* (ou *terminaux*), noté $T_{\mathcal{V}}$, dont on verra qu'il s'agit du vocabulaire du langage défini par G dans le formalisme $\mathcal{F}$,
- l'ensemble non-vide des *variables*, noté $V_{\mathcal{V}}$,
- l'ensemble des *constantes*, noté $C_{\mathcal{V}}$,
- l'ensemble des *méta-symboles*, noté $M_{\mathcal{V}}$,

Les variables ne sont utilisées que si les composants des règles de réécriture sont des prédicats logiques, et alors $M_{\mathcal{V}}$ comporte des symboles permettant de séparer un nom de prédicat de ses arguments, éventuellement de dénoter une frontière entre arguments, voire des opérateurs sur les variables. Si l'ensemble des variables est vide, on a l'habitude de désigner une constante par le vocable de *symbole non-terminal*, ou simplement *non-terminal*.

Soit une chaîne w de vocabulaire $T_{\mathcal{V}}$. Comme dit plus haut, on note $w_{i..j}$ la sous-chaîne de w correspondant à l'intervalle $i..j$. On appelle *intervalle w-instancié* le triplet $(i, j, w_{i..j})$ correspondant, et $\mathcal{I}^w$ l'ensemble des intervalles w-instanciés. On dit que sous-chaîne $w_{i..j}$ de w est la sous-chaîne *couverte* par l'intervalle w-instancié $(i, j, w_{i..j})$.

Soient C un k-uplet de structures de $\mathcal{V}$ ou une règle de réécriture d'une grammaire sur $\mathcal{V}$. Soit w une chaîne de $T_{\mathcal{V}}$. On appelle *w-instanciation* de C la structure de $\mathcal{V}_{inst} \cup \mathcal{I}^w$ obtenue à partir de C en y remplaçant toutes les occurrences d'une même variable (élément de $V_{\mathcal{V}}$) par un même intervalle instancié et chaque occurrence d'un symbole terminal t ($t \in T_{\mathcal{V}}$) par un intervalle instancié quelconque couvrant la sous-chaîne t (c'est-à-dire de la forme (i, j, t). Par exemple, si $ababab$ et X sont des chaînes respectivement de $T_{\mathcal{V}}$ et $V_{\mathcal{V}}$, alors $(2, 6, abab)$ est une $ababab$-instanciation de X. On appelle *structure w-instanciée* de $\mathcal{V}$ la w-instanciation d'une structure de $\mathcal{V}$.

Soit G une grammaire de réécriture, et $G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n$ une règle de G, et soit $(G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n)^w$ une w-instanciation de cette règle. La w-instanciation de modifiant pas les symboles spéciaux que sont $\rightarrow$ et l'espace, on peut voir cette règle comme étant de la forme $G_1^w, \ldots, G_n^w \rightarrow D_1^w, \ldots, D_n^w$, où chaque G_i^w (respectivement D_j^w) est la w-instanciation du composant G_i (respectivement D_i) qui correspond à la w-instanciation de la règle[9].

Soit G une grammaire de réécriture sur $\mathcal{V}$, et $G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n$ une règle de G. Soit C^w une w-instanciation d'une structure de $\mathcal{V}$ pour une certaine chaîne w de $T_{\mathcal{V}}$. Soit enfin une relation d'équivalence $\mathcal{R}$ sur les structures w-instanciées de $\mathcal{V}$. On dit alors que $G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n$ *est $(w, \mathcal{R})$-applicable à C^w* si et seulement si on peut choisir :

- des sous-structures (w-instanciées) disjointes de C^w, nommées $C_1^w, \ldots, C_n^w$,
- une w-instanciation $G_1^w, \ldots, G_n^w \rightarrow D_1^w, \ldots, D_n^w$ de la règle de réécriture

[9] Bien sûr, procéder séparément à la w-instanciation de chaque composant de la règle ne conduit pas nécessairement à une w-instanciation de la règle : pour w-instancier une règle, il faut faire suivre toutes les occurrences d'une même variable (élément de $V_{\mathcal{V}}$) d'un même intervalle instancié, ce qui n'est pas nécessairement obtenu par w-instanciations indépendantes des composants.

tels que $\forall l \mid 1 \leq l \leq n,\ C_l^w \mathcal{R} G_l^w$. On définit dans ce cas le résultat de cette application particulière de la règle à C^w comme étant le résultat, noté $\tilde{C}^w$, du remplacement de chaque C_l par D_l^w. Naturellement, si les composants sont des graphes (et donc en particulier des arbres), ceci suppose qu'on dispose de règles sur la façon dont ces remplacements modifient les arcs dont la source ou la cible sont dans C_l^w. On dit alors les trois choses suivantes :

- C_l^w a été $(\mathcal{R}, w)$-*dérivé* en G_l^w par la règle $R = G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n$ appliquée à C aux points $C_1^w, \ldots, C_n^w$; on note cela $C^w \underset{\mathcal{R},R,w}{\Longrightarrow} \tilde{C}^w$;
- $C_1^w, \ldots, C_n^w$ a été $(\mathcal{R}, w)$-*réécrit* en $D_1^w, \ldots, D_n^w$ dans C^w par la règle

$$R = G_1, \ldots, G_n \rightarrow D_1, \ldots, D_n.$$

Cette opération, dite *opération de réécriture*, remplace donc des éléments d'une structure d'entrée par d'autres, qui sont associés aux premiers via une règle de réécriture. A partir d'une structure de départ, et par applications successives de règles, on obtient donc des structures différentes.

Pour $l \in [1..n]$, on définit le *lieu* de la réécriture de G_l^w en D_l^w comme étant l'élément du composant instancié de partie droite d'une règle précédemment appliquée, ou la structure dont on est parti initialement, telle que G_l^w est une sous-structure de cet élément (sinon, le lieu ne peut être défini). Cette définition peut s'appliquer dès lors que C_l^w est une constante ou un prédicat et non une sous-structure plus complexe (car dans ce cas, la sous-structure peut être issue de plusieurs éléments distincts). En notant $L(C_l^w)$ cet élément instancié, on définit alors une relation de filiation comme suit : on dit que, pour chaque l, chaque D_{lk}^w $(1 \leq k \leq K_l)$ est un *fils* de $L(C_l^w)$; on note cela $L(C_l^w) \underset{\mathcal{R},R,w}{\longmapsto} D_{lk}^w$.

Soit une grammaire de réécriture G. Pour une relation d'équivalence $\mathcal{R}$ telle que ci-dessus, on appelle *relation de* $(\mathcal{R}, w)$-*dérivation* la relation binaire entre structures instanciées, notée $\underset{\mathcal{R},G,w}{\Longrightarrow}$ ou simplement $\underset{G,w}{\Longrightarrow}$, qui est l'union sur toutes les règles de réécriture R appartenant à G des relations $\underset{\mathcal{R},R,w}{\Longrightarrow}$. Enfin, on définit une autre relation binaire entre structures instanciées, notée $\overset{+}{\underset{\mathcal{R},G,w}{\Longrightarrow}}$ ou simplement $\overset{+}{\underset{G,w}{\Longrightarrow}}$, qui est la fermeture transitive (non-réflexive) de $\underset{\mathcal{R},G,w}{\Longrightarrow}$. Si deux structures instanciées S_1 et S_2 sont reliés par $\overset{+}{\underset{\mathcal{R},G,w}{\Longrightarrow}}$, on appelle $(\mathcal{R}, w)$-*dérivation* ou simplement *dérivation* une suite finie de structures instanciées reliés par $\underset{G,w}{\Longrightarrow}$ dont le premier est S_1 et le dernier est S_2. On dit alors que S_2 est $(\mathcal{R}, w)$-*dérivable* ou simplement *dérivable* de S_1.

On appelle *formalisme de réécriture* un formalisme $\mathcal{F}$ défini sur un ensemble de grammaires de réécriture sur un vocabulaire $\mathcal{V}$ et qui associe à chacune de ces grammaires un langage comme suit. Soit une grammaire G dans $\mathcal{F}$ (c'est-à-dire un élément de son domaine de définition). Tout d'abord, $\mathcal{F}$ effectue la partition de $\mathcal{V}$ en quatre sous-ensembles tels que définis plus haut. Ensuite, $\mathcal{F}$ identifie un ensemble non vide $\mathcal{S}$ de parties gauches de règles de réécri-

ture de G, appelées *axiomes*. Enfin, $\mathcal{F}$ est définie à partir d'une relation $\mathcal{R}$ telle que définie précédemment de la façon suivante : une chaîne w de $T_\mathcal{V}$ est un élément du langage $\mathcal{L} = \mathcal{F}(G)$ si et seulement si on peut trouver un axiome S tel qu'il existe une w-instanciation D^w d'une structure D de $\mathcal{V}$ qui est $(\mathcal{R}, w)$-dérivable d'une w-instanciation de S et qui satisfait un critère dit de *correction* vérifié par une fonction booléenne $Corr$ définie sur les structures instanciées ($Corr(D^w) = \mathsf{TRUE}$). Une telle structure[10] D^w est appelée *structure dérivée* (ou *structure d'analyse*) de w par la grammaire G.

Pour définir un formalisme, il suffit par conséquent de fournir la partition du vocabulaire, le domaine de définition de $\mathcal{F}$, le moyen d'identifier les axiomes d'une grammaire, la relation $\mathcal{R}$ et la fonction booléenne $Corr$.

Lorsqu'une grammaire de réécriture manipule des structures qui sont des arbres ou des graphes dont les nœuds sont des constantes, on parle de *grammaires de réécriture d'arbres* ou de *grammaires de réécriture de graphes*. Lorsqu'une grammaire de réécriture manipule des chaînes dont les éléments sont des prédicats, on parle de *grammaires de réécriture logiques*.

1.2.4 Graphes de dérivation et analyse

Soit un formalisme de réécriture $\mathcal{F}$ reposant sur une relation $\mathcal{R}$. Soit une grammaire de réécriture G appartenant à $\mathcal{F}$ et formée de règles de réécriture sur un vocabulaire $\mathcal{V}$. Une chaîne w de $T_\mathcal{V}$ est donc dans $\mathcal{L}$ si et seulement si il existe au moins une $(\mathcal{R}, W, w)$-dérivation reliant un axiome w-instancié $S^w = C_0^w$ à un élément $D^w = C_N^w$ de $\mathcal{V}$ via les règles $R_0, \ldots, R_{N-1}$, avec $Corr(D^w) = \mathsf{TRUE}$. Si l'on utilise les relations de dérivation spécifiques à chaque règle de réécriture, cette dérivation peut s'écrire comme suit (en n'indiquant ni $\mathcal{R}$ ni W ni w pour alléger les notations) :

$$S_0^w \underset{R_O}{\Longrightarrow} S_1^w \underset{R_1}{\Longrightarrow} \ldots \underset{R_{N-1}}{\Longrightarrow} S_N^w.$$

On dit que les structures S_i^w sont des *proto-phrases* (ou *formes sentencielles*).

On appelle *graphe de w-dérivation* le graphe orienté[11] de structures instanciées reliées dans la dérivation par une des relations $\underset{\mathcal{R},W,R,w}{\longmapsto}$ (il y en a autant que de règles R dans G). Le sous-graphe correspondant seulement à un intervalle strict $S_i^w \underset{R_i}{\Longrightarrow} \ldots \underset{R_{j-1}}{\Longrightarrow} S_j^w$ de la dérivation est appelé *sous-graphe de w-dérivation*.

On appellera *analyse* l'opération consistant à chercher *tous* les graphes de dérivation d'une chaîne w (et donc les structures dérivées). S'il n'y en a aucun, la chaîne w n'est pas dans $\mathcal{L}$,

[10] Naturellement, il est fort rare qu'il n'existe qu'une seule structure dérivée pour une chaîne w par une grammaire G qui la reconnaît

[11] Puisque nous l'avons défini comme un graphe sur $\mathcal{V}_{inst}^*$, nous ne permettons pas à une même chaîne de correspondre à deux nœuds différents. On ne peut donc se contenter, dans le cas général, d'arbres de dérivation. Nous verrons plus tard que le cas des Grammaires à Concaténation d'Intervalle (RCG) illustre ceci.

et on dit que l'analyse a *échoué*. S'il y en a exactement un, la chaîne est dans $\mathcal{L}$, et on dit que l'analyse est *non-ambiguë*. S'il y en a plus d'un, la chaîne est également dans $\mathcal{L}$, et, s'ils correspondent à des structures dérivées différentes, on dit que l'analyse est ambiguë (par abus de langage, on dira également que la chaîne est ambiguë pour cette grammaire).

On appelle *analyseur*, voire *parseur* du langage (par transposition de l'anglais *parser*), un algorithme qui réalise l'opération d'analyse. L'analyse est donc (pour nous) une procédure distincte (et plus complexe) que la reconnaissance : avec notre définition de ce qu'est un analyseur[12], on peut construire un reconnaisseur à partir d'un analyseur en répondant `VRAI` si et seulement si il existe au moins une analyse à la chaîne donnée en entrée. On appelle *constructeurs d'analyseurs* pour un formalisme $\mathcal{F}$ un algorithme qui prend en entrée une grammaire G appartenant à $\mathcal{F}$ et qui donne en sortie l'analyseur correspondant. Un constructeur d'analyseurs est donc un algorithme totalement distinct des analyseurs qu'il produit, et il convient de ne pas confondre l'un et l'autre[13].

1.2.5 Équivalences faible et forte

Soient deux formalismes $\mathcal{F}_1$ et $\mathcal{F}_2$ et deux grammaires G_1 et G_2 qui sont des grammaires respectivement de $\mathcal{F}_1$ et $\mathcal{F}_2$. On dit que ces grammaires, dans leurs formalismes respectifs, sont *faiblement équivalentes* si et seulement si elles définissent le même langage, c'est-à-dire si et seulement si $\mathcal{F}_1(G_1) = \mathcal{F}_2(G_2)$. Si, pour toute grammaire G_1 de $\mathcal{F}_1$, il existe une grammaire G_2 de $\mathcal{F}_2$ qui lui est faiblement équivalente, et si la réciproque est vraie, alors on dit que les formalismes $\mathcal{F}_1$ et $\mathcal{F}_2$ sont faiblement équivalents.

Nous ne donnerons pas de définition formelle de l'*équivalence forte* entre deux grammaires. Ceci nécessiterait des développements qui s'éloigneraient trop de nos préoccupations. De façon informelle, supposons que $\mathcal{F}_1$ et $\mathcal{F}_2$ sont faiblement équivalents, et que G_1 et G_2, qui sont respectivement des grammaires du premier et du second, sont faiblement équivalentes : elles y définissent donc le même langage $\mathcal{L}$. On dira que ces deux grammaires sont fortement équivalentes si l'on peut obtenir une structure dérivée selon G_2 d'une chaîne w de $\mathcal{L}$ à partir d'une structure dérivée selon G_1 et à l'aide d'une méthodologie compositionnelle, réversible et indépendante de w. Si, pour toute grammaire G_1 de $\mathcal{F}_1$, il existe une grammaire G_2 de $\mathcal{F}_2$ qui lui est fortement équivalente, et si la réciproque est vraie, alors on dit que les formalismes $\mathcal{F}_1$ et $\mathcal{F}_2$ sont fortement équivalents.

[12] Notons que nous n'avons défini ce que nous appelons « analyseur » que dans le cas des grammaires et des formalismes de réécriture.

[13] Le constructeur d'analyseurs commence bien sûr par effectuer l'analyse de la grammaire qu'il reçoit en entrée. Mais cette analyse utilise un analyseur du langage grammatical (c'est-à-dire du domaine de définition du formalisme), mais bien évidemment pas l'analyseur du langage décrit par la grammaire, puisqu'il a pour but de produire ledit analyseur.

1.2.6 Propriétés de clôture

Un langage étant un ensemble, on peut définir sur les langages des opérateurs comme l'union, l'intersection ou le complémentaire. Par ailleurs, on définit la concaténation de deux langages comme étant le langage constitué de chaînes décomposables en une chaîne du premier langage et une chaîne du second. Enfin, la t-substitution d'un langage par un autre est l'opération consistant à remplacer chaque chaîne du premier langage par les chaînes obtenues en remplaçant toutes les occurrences du terminal t dans la chaîne de départ par un même mot (quelconque) du second langage. Une substitution est une succession de t_i-substitutions.

Soit un formalisme dont deux grammaires G_1 et G_2 définissent deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$. On dit que le formalisme est *clos* par une certaine opération si le langage $\mathcal{L}$ obtenu par l'application de l'opération à $\mathcal{L}_1$ et $\mathcal{L}_2$ (ou à l'un d'entre eux si l'opération est unaire) est définissable par une certaine grammaire G dans le même formalisme.

2 Complexité et expressivité d'un formalisme

2.1 Complexité d'un formalisme

L'utilisation de formalismes pour des grammaires linguistiques a généralement pour objectif de permettre la construction des analyseurs ou des générateurs associés. Dans cette thèse, nous nous limitons aux analyseurs. Par conséquent, comme pour tout programme, la question de la *complexité* algorithmique de ces analyseurs se pose.

2.1.1 Complexité algorithmique

La complexité d'un algorithme est une notion complexe. De façon informelle, elle repose sur la quantité de temps, d'espace ou de toute autre ressource nécessaire à l'exécution de cet algorithme en fonction de la taille de ce qu'on lui donne en entrée (ce qui suppose une définition de ce qu'est la taille d'une entrée valide du programme). On peut donc définir différentes complexités, d'une part selon la ressource dont on mesure l'utilisation, et d'autre part selon que l'on se place dans le pire des cas, dans le cas moyen, ou dans tout autre cas pertinent. Il est d'usage d'indiquer les complexités par leur comportement asymptotique quand la longueur de l'entrée, souvent notée n si c'est un entier, tend vers l'infini. Pour dire par exemple qu'une fonction $f(n)$ « grandit à une vitesse inférieure ou égale » à celle d'une autre fonction $g(n)$, on utilise la nota-

tion suivante : $f(n) = O(g(n))$ (prononcé « $f(n)$ est un grand O de $g(n)$ »)[14]. Formellement[15], cela signifie qu'il existe un réel positif M et un entier n_0 tel que $\forall n \geq n_0,\ |f(n)| \leq M|g(n)|$. On parlera par exemple de complexité linéaire si l'on est en $O(n)$, de complexité cubique si l'on est en $O(n^3)$, de complexité polynomiale si l'on est en $O(n^k)$ pour un certain réel positif k, de complexité exponentielle si l'on est en $O(a^n)$ pour un certain réel positif a, et ainsi de suite. Dans certains cas, on indique aussi le comportement vis-à-vis de paramètres de l'algorithme. Si par exemple on dispose d'un algorithme de paramètre p, on pourra dire que sa complexité est en $O(p^2n^6)$.

2.1.2 Minimiser la complexité : graphes partagés d'analyse

Pour tenter de limiter au maximum la complexité des analyseurs, un certain nombre de techniques existent. Elles peuvent se regrouper en trois grandes familles : les techniques de partage, les techniques de représentation compacte de l'information, et les techniques d'évaluation paresseuse. Nous reviendrons sur ces techniques au chapitre 3, consacré aux analyseurs et aux techniques algorithmiques et informatiques qu'ils utilisent. Toutefois, nous allons présenter dès maintenant un aperçu de ce qu'est le *partage*, pour pouvoir introduire la notion de *structures partagées de dérivations*.

La construction des structures d'analyse d'une phrase d'entrée selon une grammaire (et un formalisme) peut être vue comme une succession de choix : à chaque étape du processus, on peut en général appliquer plusieurs règles de réécriture, chacune pouvant même être appliquée de différentes façons. Il est alors raisonnable, lorsqu'on arrive devant une telle alternative, de ne pas dupliquer le sous-graphe de dérivation déjà obtenu en autant de sous-graphes qu'il y a de choix possibles, multipliant ainsi inutilement les structures. A l'inverse, si deux analyses différentes conduisent, à un endroit particulier de leur sous-graphe d'analyse, à deux nœuds égaux, il semble souhaitable de ne stocker qu'une seule fois en mémoire ce nœud et le ou les sous-graphes dont elle est la source. La solution est de représenter l'ensemble des graphes de dérivation sous la forme d'un graphe *et-ou*, dont les nœuds *et* sont des règles instanciées ayant pour fils ses composants de partie droite instanciés, un nœud *ou* étant donc un composant instancié qui a pour fils les différentes règles instanciées qui effectuent une réécriture sur lui. Un tel graphe est appelé *graphe partagé d'analyses*, ou, si les graphes d'analyses sont des arbres, *forêt partagée d'analyse*. Nous reviendrons sur ces notions dans le chapitre 3.

[14] Le lecteur averti aura remarqué que la théorie de la complexité utilise la notation $O(g(n))$ là où elle aurait parfois envie de dire $\Theta(g(n))$. Dans le cas général, toutefois, on ne dispose pas de preuve que le comportement algorithmique donné est le meilleur possible. L'usage est donc, même lorsqu'on dispose d'une telle preuve, de se cantonner à la notation $O(g(n))$.

[15] Nous donnons une définition pour des fonctions définies sur les entiers. C'est évidemment un cas particulier du cas général où elles sont définies sur les réels.

2.1.3 Complexité en temps

L'expérience montre qu'un analyseur correctement écrit (c'est-à-dire, entre autres, qui utilise le partage de calculs et de structures et produit un graphe partagé d'analyses) est plus handicapé par les contraintes de *temps* que par les contraintes d'espace (au moins pour les formalismes utilisés couramment en linguistique computationnelle). Par ailleurs, la complexité dans le pire des cas est généralement la moins délicate à déterminer *a priori*, bien que la complexité moyenne ou médiane soit parfois plus pertinente d'un point de vue pratique. Pour ces raisons, nous utiliserons le raccourci suivant : nous appellerons *complexité* d'un formalisme la complexité en temps et dans le pire des cas d'un algorithme raisonnable[16] d'analyse en fonction de la longueur de ce qu'on lui donne en entrée[17]. Pour dire les choses autrement, la complexité d'un formalisme indique la façon dont le temps d'analyse d'une phrase de longueur[18] n est susceptible de croître en fonction de n (et éventuellement en fonction de propriétés de la grammaire utilisée). Certains formalismes permettent l'écriture de grammaires pour lesquelles il n'existe pas d'analyseur qui donne toujours une réponse (positive ou négative) en un temps fini que l'on peut borner par une fonction de la longueur de la phrase d'entrée. La complexité de ces formalismes ne peut donc pas être définie. Nous dirons que ces formalismes sont *indécidables*.

2.2 Expressivité d'un formalisme

L'*expressivité* d'un formalisme peut être définie comme l'ensemble des langages définis par les grammaires de ce formalisme. Pour reprendre les concepts définis dans la section précédente, un formalisme est une application dont l'image est appelée son expressivité.

Un moyen de distinguer deux formalismes par leur expressivité consiste donc à exhiber des langages, ou des classes de langages, définis par des grammaires d'un des formalismes mais par aucune grammaire de l'autre. On utilise souvent les familles de langages suivants, paramétrées par un entier $k \geq 1$ et définis sur un alphabet $\{a_1, \ldots, a_k, b_1, \ldots, b_k\}$:

[16] Par « algorithme raisonnable » nous voulons parler d'un algorithme aussi efficace que possible qui peut être implémenté dans un programme informatique utilisable dans la pratique. C'est donc une notion floue. L'idée est par exemple d'exclure les algorithmes d'analyse pour les grammaires non-contextuelles qui sont en dessous de $O(n^3)$, simplement parce qu'en pratique ils sont bien moins efficaces que les algorithmes en $O(n^3)$. Ainsi, nous dirons (et rappelons que nous avons qualifié de « raccourci » notre définition de la complexité d'un formalisme) que la complexité des grammaires non-contextuelles est $O(n^3)$

[17] Nous considérons qu'un analyseur rend toujours une réponse, éventuellement vide dans le cas où il n'y a aucune analyse.

[18] Ce qui nécessite une définition de la longueur d'une phrase. Tant qu'une phrase est une chaîne de n symboles (ou mots), la définition est simple. Nous verrons, par exemple au chapitre 9 sur l'analyse LFG, que l'on peut être face à des notions plus complexes de phrase.

– l'accord multiple d'ordre k, correspondant au langage

$$\mathcal{M}_k = \{a_1^n \dots a_k^n \mid n \geq 1\},$$

– le scrambling d'ordre k, correspondant au langage

$$\mathcal{S}_k = \{a_1^{n_1} \dots a_k^{n_k} b_1^{n_1} \dots b_k^{n_k} \mid \forall i \in [1..k],\ n_i \geq 1\},$$

– la copie d'ordre k (ou k-copie), correspondant au langage

$$\mathcal{C}_k = \{w^k \mid w \in \{a_1, b_1\}^*\}.$$

On note souvent un langage par un raccourci consistant à ne pas quantifier les variables entières libres, en sous-entendant systématiquement un quantifieur universel sur les entiers positifs (la présence ou non de zéro étant laissée au bon sens du lecteur). On parlera ainsi du langage $a_1^n \dots a_k^n$ ou du langage $a_1^{n_1} \dots a_k^{n_k} b_1^{n_1} \dots b_k^{n_k}$.

Il y a un lien entre expressivité et complexité d'un formalisme. De façon générale, plus un formalisme est expressif, plus sa complexité est grande. Mais ce lien n'a rien d'automatique, et on peut exhiber des formalismes différents qui ont la même complexité mais une expressivité différente.

3 Panorama des formalismes les plus répandus

Les formalismes utilisés pour la linguistique peuvent être répartis en deux grandes familles. Il y a tout d'abord un ensemble de formalismes de réécriture, dont certains ont été développés explicitement pour la modélisation des langues et dont d'autres ont eu dès le départ une portée plus large.

Mais certains formalismes parmi les plus utilisés ne sont pas de simples formalismes de réécriture, mais ce que nous appellerons des *formalismes à décorations* ou *formalismes à deux niveaux*. Un formalisme à décorations repose sur un formalisme de réécriture, qui est son *formalisme squelette*, et dont les grammaires sont dites *grammaires squelettes*. Mais une grammaire d'un formalisme à décorations associe à chaque règle de réécriture de ce squelette un ensemble de contraintes définies sur des structures appelées *décorations*. Ces décorations sont des structures associées, pour une chaîne w du langage, à chaque nœud du graphe de dérivation de w. Pour les formalismes non-probabilistes (dont nous parlerons plus loin), il s'agit le plus souvent

de *structures de traits typés* combinées par des opérations d'*unification*[19]. Selon la nature des décorations, la complexité d'un formalisme à décorations peut être égale à celle de son formalisme squelette, comme elle peut lui être très supérieure.

Une autre distinction existe, orthogonale à celle qui sépare les formalismes de réécriture des formalismes à décorations. On distingue en effet les formalismes *lexicalisés* des autres formalismes. Les formalismes (complètement) lexicalisés sont ceux dont toutes les règles de réécriture (celles du formalisme squelette s'il s'agit d'un formalisme à décorations) comportent au moins un symbole terminal en partie droite, qui est appelé l'*ancre* de la règle. L'avantage de ces formalismes est qu'il est plus facile d'implémenter des analyseurs, puisqu'une chaîne d'entrée donnée sélectionne les règles qui peuvent servir à son analyse (seules les règles ancrées par des terminaux présents dans la chaîne d'entrée sont utiles), ainsi que le nombre de règles à appliquer (exactement égal à la longueur de la chaîne d'entrée). Pour cette raison, les formalismes lexicalisés se rencontrent fréquemment pour modéliser les langues. Mais ils ne permettent pas nécessairement une souplesse suffisante dans les descriptions et les généralisations que l'on peut vouloir mettre en place.

Cette section est un survol de l'éventail des formalismes les plus cités dans la littérature de la modélisation des langues. Nous commencerons par les formalismes de réécriture, avant de citer certains formalismes à décorations. La figure 1 récapitule le positionnement des formalismes cités en termes de complexité du squelette et des décorations. Les sections suivantes présenteront plus en détail certains d'entre eux, en commençant par deux formalismes de réécriture (CFG et RCG), en poursuivant par certains formalismes à décorations spécifiquement développés pour la modélisation des langues, puis en terminant sur l'utilisation de décorations pour probabiliser les formalismes.

3.1 Formalismes de réécriture

3.1.1 Hiérarchie de Chomsky et formalismes faiblement sensibles au contexte

Les formalismes les plus simples sont ceux qui appartiennent à ce qu'on appelle la *hiérarchie de Chomsky*. Il s'agit de formalismes dont les grammaires sont des grammaires de chaînes, où les seuls symboles sont des constantes (appelées non-terminaux) ou des terminaux, où l'axiome est unique, et où la réécriture est définie de la façon la plus simple possible : les règles de réécriture n'ont qu'un composant de chaque côté du signe $\rightarrow$, chaque composant étant une chaîne de symboles terminaux et non-terminaux, et la relation $\mathcal{R}$ est l'identité. Enfin, la fonction de correction $Corr$ rend TRUE si et seulement si son argument est une chaîne de sym-

[19] Faute de place, nous ne définirons formellement ni les structures de traits ni les opérations d'unification. De telles définitions sont répandues dans la littérature.

boles terminaux instanciés adjacents. Autrement dit, une dérivation passe de proto-phrase en proto-phrase en remplaçant une suite de symboles par la partie droite d'une règle dont cette suite est la partie gauche.

3.1.1.1 Hiérarchie de Chomsky

Chomsky distingue quatre familles de grammaires de ce type (et donc de langages), chacune étant strictement incluse dans la précédente :

Grammaires de type 0 ou grammaires générales : aucune restriction. Un langage est défini par une grammaire générale si et seulement s'il est reconnu par une machine de Turing (un automate à deux piles), mais un mot n'appartenant pas au langage pourra faire boucler indéfiniment la machine. Les grammaires générales forment donc un formalisme indécidable. Les langages définis par ces grammaires sont les langages récursivement énumérables. Ils forment un ensemble clos par intersection, concaténation, substitution, mais pas par complémentation.

Grammaires de type 1 ou grammaires contextuelles : la partie droite d'une règle doit avoir au moins autant de symboles que la partie gauche. Les langages ainsi définis sont des langages récursifs (mais pas tous). Un langage est contextuel s'il est reconnu par une machine de Turing non déterministe dont la mémoire est borné par une fonction linéaire de la longueur de la chaîne d'entrée. La complexité est donc exponentielle. Les propriétés de clôture sont les mêmes que pour le type 0, à ceci près qu'on ignore ce qu'il en est de la clôture par complémentation.

Grammaires de type 2 ou grammaires non-contextuelles (*Context-Free Grammars, CFG*) : les parties gauches de règles sont constituées exactement d'un seul non-terminal, ce qui permet de définir trivialement la relation de filiation (alors qu'on ne peut la définir pour les types précédents). Un langage est non-contextuel s'il est reconnu par un automate à pile éventuellement non déterministe. La complexité est polynomiale. On connaît des algorithmes d'analyse efficaces en $O(n^3)$ et des algorithmes difficiles à implémenter efficacement en $O(n^p)$ avec $2 < p < 3$. Les langages ainsi définis forment un ensemble clos par concaténation et substitution, mais pas par intersection ni par complémentation. La section 1.4.1 étudie plus en détail ces grammaires.

Grammaires de type 3 ou grammaires rationnelles (ou régulières) (GR) : les règles doivent être de la forme $X \rightarrow a$ ou $X \rightarrow a\ Y$ (ou $X \rightarrow Y\ a$), où a est un terminal et Y un non terminal. Un langage est rationnel (ou régulier) s'il est reconnu par un automate fini. Ces langages forment un ensemble clos par intersection, concaténation, substitution et complémentation.

3.1.1.2 Formalismes faiblement sensibles au contexte (MCS)

Depuis Schieber (1985), on sait que les langues sont au-delà de l'expressivité des grammaires non-contextuelles. Toutefois, les grammaires contextuelles sont difficiles à utiliser dans la pratique en raison de leur complexité. C'est ce qui a motivé la définition de la notion de formalismes *faiblement sensibles au contexte* (*Mildly Context-Sensitive*, ou *MCS*) par Joshi (1985) afin d'essayer de trouver un niveau de complexité intermédiaire. Un formalisme est dit faiblement sensible au contexte s'il vérifie les 4 conditions suivantes :

MCS1. il étend strictement les grammaires non-contextuelles,

MCS2. l'analyse est de complexité polynomiale,

MCS3. son expressivité lui permet de décrire des langages modélisant des versions faibles de dépendance à longue distance et d'accord multiple (typiquement les langages $\mathcal{M}_4 = \{a^n b^n c^n\}$, $\mathcal{S}_2 = \{a^n b^m c^n d^m\}$ ou *2-copie*, c'est-à-dire $\mathcal{C}_2 = \{ww \mid w \in \{a, b\}^*\}$.)

MCS4. il satisfait la propriété de Croissance Constante (*Constant Growth Property*, *CGP*)[20].

Un certain nombre des formalismes que nous allons décrire maintenant sont des formalismes MCS. Actuellement, deux classes de complexité regroupent tous les formalismes MCS usuels. La classe MCS la plus faible est celle des Grammaires d'Adjonction d'Arbres (TAG, voir ci-dessous) et des formalismes faiblement équivalents, analysables en $O(n^6)$. La classe MCS la plus forte est celle des Systèmes de Réécriture Linéaires Non-Contextuels (LCFRS, voir ci-dessous) et des formalismes faiblement équivalents, analysables en $O(n^k)$, où k dépend de la grammaire. Cette classe ne couvre toutefois pas la classe de tous les langages analysables en temps polynomial, appelée *PTIME*, puisqu'il existe de tels langages ne vérifiant pas la Propriété de Croissance Constante. En réalité, *PTIME* est couverte par une autre classe de formalismes, dont nous aurons l'occasion d'étudier un exemple en détails, le formalisme des Grammaires à Concaténation d'Intervalles (RCG, voir ci-dessous).

3.1.2 Formalismes de réécriture d'arbres

La famille de formalismes constituée autour des Grammaires d'Adjonction d'Arbres (*Tree Adjoining Grammars*, d'où l'acronyme que nous utiliserons : *TAG*) sont des grammaires de réécriture d'arbres : les parties droites des règles de réécriture sont composés d'un arbre, dit *arbre élémentaire*[21]. Les arbres élémentaires sont partitionnés en *arbres initiaux* et *arbres auxiliaires*. Ce sont des formalismes faiblement sensibles au contexte.

[20] De façon informelle, un langage vérifie la propriété de Croissance Constante si ses éléments (ses phrases) respectent la condition suivante. Si on classe ses phrases par longueur croissante, la différence entre les longueurs de deux phrases consécutives est bornée par une constante. Un formalisme vérifie cette propriété si les langages qu'il permet de définir la vérifient.

[21] En réalité, ces arbres élémentaires sont les parties droites de règles de réécriture dont la racine forme également la partie gauche.

Le formalisme des TAG, proposé par Joshi (1987), se caractérise comme suit. Une grammaire TAG est assimilée à un ensemble d'arbres élémentaires. Les nœuds internes de ces arbres sont des non-terminaux, et les nœuds feuilles des non-terminaux ou des terminaux. Chaque arbre auxiliaire comporte un nœud feuille spécial, appelé *pied*, qui a le même non-terminal que la racine.

Les règles de réécriture, en TAG, peuvent se définir comme des opérations de manipulation des arbres de la façon suivante. La combinaison d'un arbre avec un autre peut se faire selon deux opérations distinctes, selon la nature du second arbre. L'opération de *substitution* remplace une feuille d'un arbre par un arbre initial dont la racine a le même non-terminal. L'opération d'*adjonction* remplace un nœud N par un arbre auxiliaire complet dont la racine a le même non-terminal que N, en reliant le père du nœud N à la racine de l'arbre auxiliaire et en reliant le pied de l'arbre auxiliaire à tous les fils du nœud N.

Une variante des TAG, appelée *LTAG* (TAG lexicalisées, Group (1995)), impose par ailleurs à tout arbre de la grammaire d'avoir au moins une feuille étiquetée par un terminal. Comme indiqué précédemment, cette feuille s'appelle l'*ancre* de l'arbre, et d'éventuelles autres feuilles étiquetées par un terminal sont appelées *co-ancre*.

La complexité des TAG est $O(n^6)$ dans le cas général. Toutefois, si on impose aux arbres auxiliaires de n'être pas *englobants*, c'est-à-dire que leur nœud pied doit toujours être sur leur frontière droite ou sur leur frontière gauche, et que l'on interdit les adjonctions qui transforment en nœuds internes des nœuds de cette frontière, alors on obtient des grammaires *TIG* (*Tree Insertion Grammar*, *Grammaires d'Insertion d'Arbres*). Les TIG, dont la complexité est en $O(n^3)$, sont faiblement (mais pas fortement) équivalentes aux grammaires non-contextuelles.

Enfin, une extension des TAG, appelée *MC-TAG* (*Multi-Component TAGs*, TAG multi-composants) a été proposée par Joshi (1987). L'idée, comme son nom l'indique, est qu'une opération de manipulation d'arbres puisse effectuer simultanément plusieurs opérations, une partie droite de règle comportant plusieurs *composants* qui sont des arbres élémentaires TAG. Les MC-TAG standard sont équivalentes aux LCFRS et aux autres formalismes faiblement sensibles au contexte de complexité strictement supérieure aux TAG.

3.1.3 Formalismes de réécriture de prédicats logiques

De nombreux formalismes de réécriture de prédicats logiques (ou apparentés) ont été définis, le plus souvent comme extensions des grammaires non-contextuelles (CFG). En voici quelques-uns.

Les Grammaires de Têtes (*Head Grammars* ou *HG*, voir Pollard et Sag (1994)) étendent les CFG en manipulant des chaînes avec tête, qui peuvent être concaténées ou encapsulées les unes dans les autres. Elles sont faiblement équivalentes aux TAG.

Les Grammaires Catégorielles (*Categorial Grammars*, *CG*, voir Ajdukiewicz (1935); Bar-Hillel (1953); Lambek (1958)) sont faiblement équivalentes aux CFG. Elles ne manipulent pas des règles de grammaires mais des catégories fonctionnelles complexes qui se combinent par des règles d'application de fonctions.

Les Grammaires Catégorielles Combinatoires (*Combinatorial Categorial Grammars*, voir Steedman (1985)) sont une extension des Grammaires Catégorielles qui ajoutent des règles de combinaison de fonctions, étendant ainsi les CFG jusqu'à être faiblement équivalentes aux TAG.

Les Systèmes Linéaires Non-contextuels de Réécriture (*Linear Context-Free Rewriting Systems*, *LCFRS*, voir Vijay-Shanker *et al.* (1987)) sont des grammaires de prédicats dont les arguments sont des n-uplets de chaînes de caractères. Certaines contraintes caractéristiques des LCFRS les rendent faiblement sensibles au contexte (MCS).

Les Grammaires Non-Contextuelles Parallèles Multiples (*Parallel Multiple CFGs*, *PMCFG*, voir Seki *et al.* (1991)) sont similaires aux LCFRS, mais ne respectent pas la propriété de croissance constante, et ne sont donc pas faiblement sensibles au contexte (MCS). Elles ont toutefois une expressivité strictement inférieure à celle des deux formalismes suivants.

Enfin, les Grammaires à Concaténation d'Intervalles (*Range Concatenation Grammars*, *RCG*, voir entre autres Boullier (2004)) et les Grammaires simples à Mouvements de Littéraux (*simple Literal Movement Grammars*, *sLMG*, voir Groenink (1996)) sont des grammaires de prédicats dont les arguments sont respectivement des intervalles et des sous-chaînes de la chaîne d'entrée. Ces formalismes sont plus expressifs que tous ceux précédemment cités, et couvrent tout *PTIME*, c'est-à-dire qu'ils ont la plus grande expressivité possible si l'on veut rester avec une complexité polynomiale. Contrairement aux formalismes précédents, ils sont ne sont pas clos par substitution mais sont clos par intersection et complémentation.

3.2 Formalismes à décorations

La plupart des formalismes à décorations largement répandus ont pour squelette les grammaires non-contextuelles. Il s'agit des Grammaires à Clauses Définies (*Definite Clause Grammars*, *DCG*), des LFG (voir ci-dessous) et des HPSG (idem). Si l'on restreint les DCG d'une certaine manière[22], la complexité reste celle des grammaires non-contextuelles. Les fb-TAG (*feature-based TAG*), qui sont des TAG à décorations, ont une complexité qui dépend elle aussi de ce que l'on met dans ces décorations. Quant aux Méta-RCG, nous les présentons au chapitre 10.

[22] Si les attributs prennent des valeurs dans des domaines finis.

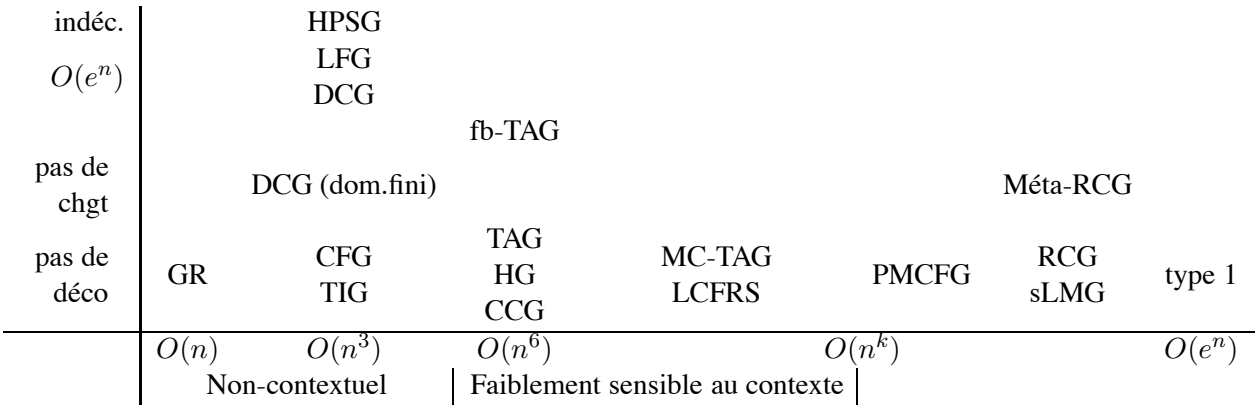


FIG. 1 – Complexité de quelques formalismes. En abcisse est indiquée la complexité du formalisme de réécriture squelette, ou du formalisme lui-même s'il n'est pas un formalisme à décorations. En ordonnée est indiquée soit la complexité du calcul des décorations soit le fait que ces décorations ne modifient pas la complexité par rapport à celle du squelette.

4 Deux formalismes de réécriture : CFG et RCG

Dans cette section, nous présentons plus en détail deux formalismes de réécriture que nous avons mis en œuvre dans nos travaux : les grammaires non-contextuelles et les Grammaires à Concaténation d'Intervalles. Pour chacun de ces formalismes, nous proposons à la fois une présentation informelle et une définition plus formelle.

4.1 Grammaires non-contextuelles (CFG)

Les grammaires non-contextuelles (en anglais *Context-Free Grammars*, d'où l'acronyme *CFG*) sont des grammaires de réécriture associées à un formalisme de réécriture que l'on peut définir informellement comme suit. Une grammaire est constituée de *productions* (ce sont les règles de réécriture de la grammaire) qui sont de la forme

$$A_0 \rightarrow A_1 \dots A_n,$$

où $n \geq 0$. Les A_i sont de deux types : on distingue les *symboles non-terminaux* des *symboles terminaux*. Parmi les symboles non-terminaux, il y a un symbole particulier qui est l'axiome. Une chaîne w de symboles terminaux fait partie du langage défini par une grammaire non-contextuelle si et seulement si, en partant de l'axiome de la grammaire, on peut répéter jusqu'à obtenir w l'opération consistant à remplacer un symbole non-terminal par la partie droite d'une production dont il est la partie gauche.

Considérons par exemple la grammaire non-contextuelle suivante, dont l'axiome est S.

$$\begin{aligned} \text{S} &\rightarrow \text{a S b} \\ \text{S} &\rightarrow \text{a b} \end{aligned}$$

Cette grammaire définit le langage $\{a^n b^n \mid n \geq 1\}$. En effet, si l'on veut montrer par exemple que $aaabbb$ appartient au langage défini par cette grammaire, on part de l'axiome S, que l'on peut remplacer (*réécrire*) aSb grâce à la première production. On peut à nouveau remplacer le non-terminal S par $a\ S\ b$, ce qui donne $aa\ S\ bb$. Puis on peut remplacer le non-terminal S ainsi obtenu par ab à l'aide de la seconde production, ce qui donne $aaabbb$, c'est-à-dire notre chaîne de départ.

Formellement, et pour reprendre les termes définis au paragraphe précédent, il nous suffit de fournir le mode de partition du vocabulaire, le domaine de définition du formalisme, le moyen d'identifier les axiomes d'une grammaire, la relation $\mathcal{R}$ et la fonction booléenne $Corr$.

Le formalisme des grammaires non-contextuelles, $\mathcal{F}_{CFG}$, est un formalisme de réécriture entièrement défini par ce qui suit :

- Le vocabulaire ne comporte pas de variables, et les constantes sont appelés symboles non-terminaux. Il ne comporte pas non plus de méta-symboles. Il est d'usage de distinguer les terminaux des non-terminaux par leur casse (un terminal commence par une minuscule, ou bien est entre guillemets, alors qu'un non terminal commence par une majuscule).
- Le formalisme est défini sur des grammaires composées de règles dont toutes les parties gauches comportent un et un seul symbole non-terminal.
- L'axiome (unique) est en général le symbole de partie gauche de la première règle, pour peu que l'on ait un ordre sur les règles.
- La relation $\mathcal{R}$ est l'identité.
- la fonction de correction $Corr_{CFG}$ rend `TRUE` si et seulement si son argument est une chaîne de symboles terminaux instanciés adjacents (aucun symbole non-terminal ne reste).

4.2 Grammaires à Concaténation d'Intervalles (RCG)

Les Grammaires à Concaténation d'Intervalles (en anglais *Range Concatenation Grammars*, d'où l'acronyme que nous utiliserons désormais, *RCG*) ont été introduites par Pierre Boullier (voir par exemple Boullier (2004)). Des applications en ont été proposées dans Boullier (2003a), afin de montrer à la fois le fonctionnement, la simplicité et l'expressivité de ce formalisme. Les RCG définissent une classe de langages, les Langages à Concaténation d'Intervalles (*Range Concatenation Languages, RCL*), qui couvrent exactement *PTIME*, la classe de tous les langages reconnaissables en temps polynomial déterministe. Ainsi, les RCG sont plus puissantes que n'importe quel formalisme MCS (faiblement sensible au contexte) comme les TAG, les

MC-TAG ou les LCFRS, tout en conservant des propriétés computationnelles raisonnables, l'analyse se faisant en temps polynomial par rapport à la longueur de la chaîne d'entrée.

L'apport des RCG est double :

- l'utilisation directe des RCG permet la modélisation des faits linguistiques de façon originale et prometteuse, sous la forme de prédicats sur des intervalles de la chaîne d'entrée ;
- on peut convertir en RCG la plupart des formalismes classiques (CFG, TAG, MC-TAG, LIG, LCFRS,...), ce qui permet l'obtention d'analyseurs pour ces formalismes ainsi que d'un cadre unifié permettant leur comparaison.

Dans un premier temps, nous allons présenter les RCG de façon informelle, pour expliciter leur fonctionnement et leur logique. Dans un second temps, nous donnerons une description formelle de ces grammaires.

4.2.1 Les RCG : présentation informelle

Les grammaires non-contextuelles reposent sur un principe fondamental qui n'est pas toujours explicité : lorsque l'on écrit $\mathrm{A} \rightarrow \mathrm{BC}$, les deux symboles non-terminaux B et C sont reliés simultanément par deux opérateurs :

- la *concaténation*, puisque l'on exprime que la portion de chaîne couverte par A est découpée en deux, l'une couverte par B et l'autre par C,
- la *conjonction*, puisque l'on exprime que A est vrai sur une certaine portion de la chaîne s'il en est ainsi de B *et* de C.

De façon duale, on peut dire qu'un symbole non-terminal, dans une grammaire non-contextuelle, désigne simultanément :

- d'une part un *prédicat* à un argument, que l'on peut gloser par *il est possible de construire un* A *sur un certain intervalle*,
- d'autre part l'*intervalle* en question de la chaîne d'entrée.

Du reste, cette superposition a une conséquence immédiate : la structure de dérivation et la structure dérivée sont deux arbres isomorphes, qui expriment donc simultanément une succession de découpages de la chaîne d'entrée et un arbre de preuve.

Cependant, rien ne semble interdire de décorréler ces deux opérations. Pour cela, il faut naturellement découpler les deux significations que recouvrent chaque non-terminal. On va donc appeler A, B et C les trois prédicats correspondant. On va appeler X l'intervalle couvert par B et Y l'intervalle couvert par C. Enfin, on note par un espace l'opération de concaténation entre intervalles, en sorte que l'intervalle couvert par A est $X\ Y$. La règle $\mathrm{A} \rightarrow \mathrm{BC}$ peut alors s'exprimer de la façon suivante : $\mathrm{A}(X\ Y) \rightarrow \mathrm{B}(X)\mathrm{C}(Y)$. C'est déjà une *clause* RCG. Pour exprimer de cette façon une règle lexicale telle que $\mathrm{V} \rightarrow \mathrm{v}$, où *v* est un terminal, on va utiliser le symbole v pour dénoter un intervalle de longueur 1 recouvrant le terminal *v*. Il suffit alors poser le « fait »

V(v), ce que l'on exprime par V(v) $\rightarrow \varepsilon$. Ici, ε signifie qu'il n'y a rien à vérifier pour garantir la vérité de V(v) : ce prédicat est donc toujours vrai dès lors qu'il est appliqué à un terminal v.

Ce découplage entre prédicats et intervalles permet immédiatement plusieurs avancées par rapport aux grammaires non-contextuelles. La première avancée est la possibilité d'introduire des prédicats à plusieurs arguments. Dans une grammaire non-contextuelle, c'est impossible, puisque le prédicat et son argument, l'intervalle sur lequel il s'applique, sont confondus : il ne peut donc y avoir qu'un seul argument à ce prédicat. Mais en RCG, on peut par exemple remplacer notre clause unique par les deux clauses suivantes :

$$\begin{aligned} \text{A}(X\ Y) &\rightarrow \text{BC}(X, Y) \\ \text{BC}(X, Y) &\rightarrow \text{B}(X)\ \text{C}(Y) \end{aligned}$$

Autrement dit, on rajoute une étape intermédiaire où le découpage de l'intervalle $X\ Y$ en deux intervalles, X et Y a été effectué, sans que les prédicats correspondant aux deux sous-intervalles n'aient été appelés. On peut gloser cette suite de clauses de la façon suivante : A est vrai pour l'intervalle $X\ Y$ si le prédicat à deux arguments BC est vrai pour le couple X,Y. Par ailleurs, BC est vrai sur un couple d'arguments si B est vrai pour son premier argument et C pour son deuxième. On constate donc que les intervalles sont des variables liées au sein d'une clause, et qu'il n'y a aucun rapport, de façon générale, entre le X d'une clause et celui d'une autre. Remplacer la deuxième clause par BC(Y, X) $\rightarrow$ B(Y)C(X) ne change absolument pas la grammaire.

Prenons un autre exemple. Nous allons définir un prédicat EQLEN à deux arguments qui est vrai si et seulement si ses deux arguments sont de même longueur. Supposons que l'on ait défini un prédicat à un seul argument, STRLEN_1, qui est vrai si et seulement si son argument est de longueur 1. Les deux clauses suivantes définissent alors EQLEN[23] (on remarquera que ε dénote un intervalle de longueur nulle lorsqu'il est argument d'un prédicat) :

$$\begin{aligned} \text{EQLEN}(\varepsilon, \varepsilon) &\rightarrow \varepsilon \\ \text{EQLEN}(T_1\ X, T_2\ Y) &\rightarrow \text{STRLEN}_1(T_1)\ \text{STRLEN}_1(T_2)\ \text{EQLEN}(X, Y) \end{aligned}$$

La première clause énonce que deux intervalles de longueur nulle (c'est ce que dénote ε en tant qu'argument de prédicat) sont de même longueur. La seconde clause énonce que l'on a aussi égalité de longueur si, en découpant chaque argument en deux intervalles dont le premier est de longueur 1 (prédicats STRLEN_1), les deux autres intervalles sont eux-mêmes d'égale longueur. D'un point de vue opérationnel, on peut considérer que l'on diminue de 1 à chaque étape la longueur de chaque argument, jusqu'à un succès si les deux arguments sont vides en même temps, et un échec sinon.

[23] Ce n'est évidemment pas la meilleure manière de faire. Un tel prédicat, très utile, gagne à être prédéfini dans l'analyseur, puisqu'il peut être calculé en temps constant de manière directe : en notant $i..j$ l'intervalle allant du $(i+1)$-ième symbole au j-ième symbole inclus, EQLEN($i..j$, $k..l$) est vrai si et seulement si $j - i = k - l$. Mais peu importe ici

La deuxième avancée est la *non-linéarité*, c'est-à-dire la possibilité d'utiliser plusieurs fois le même intervalle dans plusieurs prédicats de partie droite d'une même clause, voire dans plusieurs arguments d'un même prédicat (de partie droite ou de partie gauche). Autrement dit, la non-linéarité permet de « consommer » chaque terminal de la chaîne d'entrée plus d'une fois[24]. Les grammaires non-contextuelles sont linéaires, puisqu'un terminal ne peut avoir qu'un seul père dans l'arbre d'analyse : il n'est « consommé » qu'une fois. Mais les RCG peuvent être non-linéaires, ce qui permet d'obtenir très facilement une RCG définissant l'intersection de deux langages définis par des RCG séparées. Supposons par exemple que les prédicats S_1 et S_2 soient les axiomes de grammaires définissant deux langages, et que ces grammaires n'ont aucun nom de prédicat en commun (pour simplifier). Pour obtenir une grammaire définissant leur intersection, il suffit de concaténer les deux grammaires et d'utiliser comme axiome un nouveau prédicat S définit par la clause suivante : $\mathrm{S}(X) \rightarrow \mathrm{S}_1(X)\mathrm{S}_2(X)$ En effet, ceci signifie qu'une chaîne d'entrée est dans le langage intersection si elle est dans le premier langage *et* si elle est dans le second.

La non-linéarité permet de décrire très simplement des langages réputés complexes. Étudions par exemple le langage $\mathcal{L} = \{a^{2^p} \mid p \geq 0\}$, constitué de suites de a dont la longueur est une puissance de 2. On peut le définir de la façon suivante :

$$\begin{array}{lcl} \mathrm{S}(a) & \rightarrow & \varepsilon \\ \mathrm{S}(X\ Y) & \rightarrow & \textsc{EqLen}(X,\ Y)\ \mathrm{S}(X) \end{array}$$

Cette grammaire dit qu'une chaîne de symboles a est dans $\mathcal{L}$ si elle est de longueur 1, ou si on peut la découper en deux intervalles de même longueur dénotant des sous-chaînes qui sont aussi dans ce langage[25]. La non linéarité réside ici dans le fait que l'intervalle X est utilisé en partie droite à la fois comme premier argument de EQLEN et comme argument (unique) de S. D'un point de vue opérationnel, on peut dire qu'on coupe la chaîne en deux sous-chaînes de même longueur, en ne gardant que la première, et ce tant que c'est possible. Quand ce n'est plus possible, on a un succès si la chaîne obtenue est de longueur 1 (c'est donc la chaîne a), et échec si elle est de longueur (impaire !) supérieure à 1. Si l'on suppose (ce qui est le cas, voir la dernière note) que l'on peut vérifier EQLEN en temps constant, et que l'on a préalablement vérifié que la chaîne d'entrée n'est constituée que de symboles a, alors l'analyse se fait donc en temps logarithmique (donc sub-linéaire) en la longueur de la phrase. Pourtant, ce langage ne

[24]Cette notion de *(non-)linéarité* correspond exactement à la notion de (non-)linéarité en logique. C'est également celle dénotée par l'adjectif *linéaire* dans le nom des Systèmes de Réécriture Linéaires Non-Contextuels (*Linear Context-Free Rewriting Systems*, *LCFRS*, voir ci-dessus).

[25]On peut démontrer facilement par récurrence sur p que l'on définit bien $\mathcal{L}$ de cette façon. En effet, $a^{2^0} = a$ est reconnu par la première clause. Supposons que a^{2^p} soit reconnu par la grammaire. Alors $a^{2^{p+1}}$ est découpé en deux sous-chaînes de longueur égale par la seconde clause, qui sont donc toutes deux la sous-chaîne a^{2^p}. Par hypothèse, cette sous-chaîne est reconnue par la grammaire. Donc la seconde clause est vérifiée. Et aucune autre chaîne qu'une chaîne de la forme a^{2^p} ne peut être reconnue par cette grammaire.

respecte même pas la propriété de Croissance Constante (voir ci-dessus), qui est généralement prise pour être vérifiée par tout langage « facile » à analyser.

Nous verrons au chapitre 10 dans quelle mesure ces propriétés permettent également d'écrire de façon satisfaisante des grammaires linguistiques, c'est-à-dire des grammaires ayant pour but de modéliser une langue.

4.2.2 Les RCG : présentation formelle

La description formelle des RCG présentée ici suit largement Boullier (2004).

4.2.2.1 RCG positives

Une *RCG positive* est un quintuplet $G = (N, T, V, P, S)$ dans lequel :
- N est un ensemble fini de *noms de prédicats*,
- T est un ensemble fini de *symboles terminaux*,
- V est un ensemble fini de *symboles de variables* tel que $T \cap V = \emptyset$,
- $\mathrm{S} \in N$ est l'*axiome*,
- P est un ensemble fini de *clauses*, qui sont définies ci-dessous.

Une clause C a la forme $\psi_0 \rightarrow \psi_1 \ldots \psi_j \ldots \psi_m$, où $m \geq 0$ et chaque ψ_j est un *prédicat* de la forme $\mathrm{A}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_p)$, où $p \geq 1$ est son *arité*, $\mathrm{A} \in N$, et chaque α_i est un *argument* de A. Chaque argument α_i de A est de la forme $X_1 \ldots X_l \ldots X_q$, où chaque X_l est dans $V \cup T$. La *partie gauche* de C est ψ_0, sa *partie droite* is $\psi_1 \ldots \psi_j \ldots \psi_m$. Les prédicats de partie droite forment un ensemble de prédicats, ce qui signifie que l'ordre n'importe pas et que dupliquer un prédicat est inutile. Par la suite, nous dénoterons souvent, par abus de langage, un prédicat par son nom, parlant ainsi du *prédicat* A. L'arité de l'axiome doit être 1, et l'arité d'un prédicat A est fixée tout au long de la grammaire. Nous appelons A-clause une clause dont le prédicat de partie gauche est A. On définit l'arité d'une A-clause comme étant l'arité de A, et l'arité de la grammaire comme l'arité maximale de ses prédicats. Une RCG d'arité k est une k-RCG.

La définition d'un langage par une RCG repose sur la notion d'intervalle de la chaîne d'entrée. Soit une chaîne $w = a_1 \ldots a_n$ de symboles terminaux ($w \in T^*$). Chaque couple d'entiers (i, j) tel que $0 \leq i \leq j \leq n$ est appelé un *intervalle* de w et est dénoté par $\langle i..j \rangle_w$ ou par $i..j$ lorsque w peut être omis, i et j étant respectivement la *borne inférieure* et la *borne supérieure* de l'intervalle, et $j - i$ sa *longueur*. Si $i = j$, l'intervalle est de longueur nulle, on parle d'intervalle *vide*. Deux intervalles sont égaux si leurs bornes inférieures et leurs bornes supérieures sont égales[26]. Un intervalle $\langle i..j \rangle_w$ correspond à une sous-chaîne de w, à savoir $a_{i+1} \ldots a_j$. La

[26]Par conséquent, si $i_1 \neq i_2$, les intervalles $i_1..i_1$ et $i_2..i_2$, quoique tous deux vides, ne sont pas égaux.

concaténation de deux intervalles $i..j$ et $k..l$ est définie si et seulement si $j = k$, et c'est alors l'intervalle $i..l$.

Les symboles de variables et les symboles terminaux dénotent des intervalles. Un symbole terminal `t` dénote un intervalle de longueur 1 correspondant à une sous-chaîne de w qui est le symbole *t*. La *concaténation X Y* de deux symboles de variables ou terminaux *X* et *Y* (*X*,*Y*$\in V \cup T$) dénote l'intervalle résultant de la concaténation des deux intervalles dénotés respectivement par *X* et *Y*. Il est par conséquent défini si et seulement si ces intervalles peuvent être concaténés. On dénote souvent (par abus de langage) par *intervalle X*, où *X* est un symbole de variable, l'intervalle dénoté par *X*.

Soit $w \in T^*$. On appelle *prédicat w-instancié*, ou simplement *prédicat instancié*, un prédicat dans lequel tous les symboles de variables et terminaux ont été remplacés par des intervalles de w, et où tous les intervalles d'un même argument d'un même prédicat ont été remplacés par l'intervalle dénotant leur concaténation. Si cela est possible, le prédicat est alors dit *instanciable par* w. Par exemple, si la longueur de w est 3 ($w = a_1 a_2 a_3$), une instanciation possible pour le prédicat A(*X Y*, a_3, a_2 *Y*) est A(*0..3*, *2..3*, *1..3*). On définit de la même façon les *clauses instanciées*. Les arguments d'un prédicat peuvent bien sûr être remplacés par des intervalles qui sont disjoints ou se superposent, entre autres car la même variable peut intervenir dans plusieurs arguments de plusieurs prédicats (ce n'est en général pas vrai en revanche pour les variables d'un même argument[27]). C'est là qu'apparaît la *non-linéarité* des RCG, et qui permet d'exprimer différents points de vues ou propriétés sur un même intervalle de la chaîne d'entrée. Comme nous le verrons, c'est un des intérêts des RCG pour le développement de grammaires linguistiques. De façon plus générale, c'est en raison de cette non-linéarité que les RCG ont la puissance d'expression nécessaire pour couvrir tout *PTIME*.

Pour une RCG donnée G et une chaîne d'entrée w, on définit une relation binaire *dérive*, dénotée par $\underset{G,w}{\Longrightarrow}$, et qui a pour opérandes des ensembles de prédicats instanciés, de la façon suivante. Soit $\Gamma_1\ \gamma\ \Gamma_2$ la partie droite instanciée d'une clause (et donc un ensemble de prédicats instanciés), où γ est également la partie gauche instanciée d'une clause $\gamma \rightarrow \Gamma$. On a alors $\Gamma_1\ \gamma\ \Gamma_2 \underset{G,w}{\Longrightarrow} \Gamma_1\ \Gamma\ \Gamma_2$. Une chaîne $w \in T^*$ de longueur n est reconnue par la grammaire G si et seulement si un ensemble vide de prédicats peut être dérivé du prédicat instancié S($0..n$) (S étant l'axiome), c'est-à-dire si et seulement si S($0..n$) $\overset{+}{\underset{G,w}{\Longrightarrow}} \varepsilon$, la relation binaire $\overset{+}{\underset{G,w}{\Longrightarrow}}$ étant la clôture transitive de $\underset{G,w}{\Longrightarrow}$.

Par ailleurs, on appelle *linéaire* une RCG dans laquelle une variable est présente au plus une fois en partie droite de cette règle et au plus une fois en partie gauche de cette règle. Enfin, on dit

[27] En réalité, seul un intervalle vide peut être concaténé avec lui-même, par exemple. Et un argument de type *X Y X* ne peut être instancié qu'en un intervalle vide, puisque les deux concaténations doivent être définies.

qu'une RCG est *combinatoire* si tous les arguments de prédicats de partie droite sont constitués d'au plus une variable.

4.2.2.2 RCG négatives

Les RCG positives couvrent *PTIME*. Cependant, l'ensemble des langages qui peuvent être reconnus par une RCG positive est clos par complémentation. Pour cette raison, il est possible, pour des raisons pratiques mais sans augmenter le pouvoir d'expression du formalisme, d'introduire des prédicats négatifs.

On appelle donc *prédicat négatif* un prédicat noté soit par une barre sur le prédicat ($\overline{\mathrm{A}(\ldots)}$), soit en le faisant précéder du symbole ! (!A(...)). La négation ainsi introduite est une *négation par échec* : l'ensemble vide (de prédicats instancié) peut être dérivé d'un prédicat négatif si et seulement si il ne peut pas être dérivé de sa contrepartie positive.

On appelle RCG négative une RCG dont au moins une clause contient dans sa partie droite au moins un prédicat négatif. Une RCG négative est dite *cohérente* si, pour tout $w \in T^*$ il n'y a aucun prédicat w-instancié A(...) tel que l'ensemble vide (de prédicats instancié) peut être dérivé à la fois de A(...) et de !A(...). Comme indiqué précédemment, le pouvoir d'expression des RCG négatives est le même que celui des RCG positives : la polynomialité est préservée.

4.2.2.3 Propriétés de clôture

Les RCL (les langages définis par les RCG, voir plus haut) sont clos par union, concaténation, itération de Kleene, mais également par intersection et complémentation. Les grammaires qui reconnaissent les langages opérandes n'ont même pas besoin d'être modifiées[28]. Une ou deux clauses supplémentaires suffisent. De façon informelle, et en appelant S_1 et S_2 les axiomes des grammaires reconnaissant respectivement les langages L_1 et L_2, on peut obtenir les différentes clôtures en ajoutant les clauses suivantes, S étant l'axiome du langage résultant[29] :

[28] Ceci n'est vrai que si les grammaires opérandes n'ont aucun nom de prédicat en commun. Sinon, il faut faire en sorte d'éviter les conflits de noms.

[29] Comme on peut le voir pour l'itération de Kleene, la notation ε recouvre de façon quelque peu ambiguë deux choses différentes, comme on peut le voir dans la dernière clause de la grammaire donnée en exemple : elle peut dénoter d'une part une chaîne vide, c'est-à-dire un élément de T^*, ou un ensemble vide de prédicats.

Union	$S(X) \rightarrow S_1(X)$ $S(X) \rightarrow S_2(X)$
Concatenation	$S(X\ Y) \rightarrow S_1(X)S_2(Y)$
Intersection	$S(X) \rightarrow S_1(X)S_2(X)$
Itération de Kleene	$S(\varepsilon) \rightarrow \varepsilon$ $S(X\ Y) \rightarrow S_1(X)S(Y)$
Complémentation	$S(X) \rightarrow\ !S_1(X)$

En revanche, les RCG ne sont *pas* clos par substitution. En effet, rappelons le résultat de Ginsburg (1975) : un formalisme qui étend les grammaires non-contextuelles tout en ayant une expressivité strictement inférieure à celle des langages de type 0 (ce qui est évidemment souhaitable pour modéliser les langues de façon exploitable par les traitements automatiques) ne peut pas être clos à la fois par homomorphisme et par intersection.

4.2.2.4 Complexité de l'analyse

Les RCL peuvent être reconnus et analysés en temps polynomial en la longueur de la chaîne d'entrée. Plus précisément, soit $|G|$ la *taille* d'une k-RCG, définie comme étant la somme du nombre de prédicats de partie droite toutes clauses confondues, et l le nombre maximum de prédicats apparaissant dans la partie droite d'une clause. Boullier (2004) donne un algorithme, repris au chapitre suivant, permettant d'effectuer l'analyse en $O(|G|n^{2k(1+l)})$ (où n est la longueur de la chaîne d'entrée).

De plus, Boullier a développé un analyseur RCG très efficace, qui a déjà été utilisé pour construire des analyseurs TAG et MC-TAG moyennant une phase de conversion appropriée des grammaires. Les résultats en sont excellent (Barthélemy *et al.*, 2001).

5 Formalismes linguistiques

Comme nous l'avons indiqué plus haut, les formalismes linguistiques les plus utilisés actuellement sont souvent des formalismes à décorations, qui ont pour squelette des grammaires non-contextuelles ou des TAG. D'autres formalismes existent, comme les Grammaires de Propriétés ou d'autres types de Grammaires de Contraintes. Ces formalismes sont très riches, parfois très élégants linguistiquement, et peuvent dans certains cas induire des analyseurs raisonnablement efficaces. Toutefois, nous avons orienté nos travaux sur les formalismes de réécriture, et nous allons présenter certains d'entre eux, en mettant l'accent sur un formalisme particulier qui a reçu une attention particulière de notre part, le formalisme LFG. Nous présenterons également rapidement les Grammaires de Propriétés, parce qu'elle permettent, de façon très élégante et in-

téressante, d'éviter des inconvénients des formalismes à décorations que nous avons également cherché à surmonter. Mais avant cela, nous allons faire un tour d'horizon de la multiplicité des formalismes linguistiques existant[30].

5.1 Multiplicité des formalismes linguistiques

Depuis quelques dizaines d'années, un grand nombre de formalismes ont été proposés pour modéliser les langues — parfois pour modéliser une langue particulière. Plus que de simples formalismes, ces formalismes linguistiques cherchent, au moins pour certains, à proposer une véritable *théorie linguistique*, assortie d'un *appareillage formel* permettant la formalisation de cette théorie. Ce sont ces deux aspects, complémentaires, que nous appelons *formalisme linguistique*. Certains ont donné naissance à des travaux concernant de nombreuses langues. Certains ont donné naissance, pour certaines langues plus souvent étudiées que d'autres, à des grammaires et des ressources lexicales à large couverture. Certains ont bénéficié du développement d'analyseurs utilisant des techniques algorithmiques et informatiques sophistiquées. Mais nombreux sont les formalismes linguistiques qui restent relativement confidentiels, et aucun formalisme linguistique n'a réussi à s'imposer nettement.

Cette multiplicité est à la fois une richesse, un handicap et un défi. Une richesse, tout d'abord, car elle permet la multiplicité des points de vue sur les phénomènes linguistiques, et en particulier sur la variabilité entre les langues, telle langue étant mieux modélisée par un certain formalisme, telle autre langue par un autre formalisme. Un handicap, ensuite, car cette multiplicité disperse les travaux de modélisation et de développement de ressources linguistiques, travaux pourtant coûteux en soi. Un défi, enfin, car c'est par l'étude des raisons pour lesquelles aucun formalisme n'a su s'imposer qu'on peut envisager de faire avancer les choses, en identifiant les forces et les faiblesses de chaque formalisme, mais aussi leurs points communs et leurs différences.

Car les points communs ne manquent pas, et sont le reflet de ce qu'ils cherchent tous à modéliser la même faculté langagière[31]. Tous manipulent donc, de façon plus ou moins élaborée, approchée, indirecte ou explicite, des faits linguistiques tels que les faits morphologiques, les rapports syntagmatiques, les places topologiques, les dépendances syntaxiques (ou fonctions syntaxiques), les dépendances sémantiques (relations prédicat-arguments), et parfois même les relations de discours, les relations anaphoriques, et d'autres types de faits linguistiques.

[30] Pour une présentation plus détaillée de certains de ces formalismes, on pourra se référer entre autres à Abeillé (1993).

[31] En effet, nous pensons qu'un formalisme linguistique doit avoir pour ambition la capacité à décrire toutes les langues. Ce qui ne veut pas dire qu'il est inutile d'utiliser des formalismes linguistiques qui ne sont appropriés que pour certaines catégories de langues (langues à ordre très contraint, par exemple), en particulier dans un but opérationnel.

On distingue souvent deux grandes familles de formalismes linguistiques : les formalismes dits *génératifs*, et ceux qui ne le sont pas[32]. Un formalisme est génératif dès lors qu'il privilégie le point de vue de la production d'énoncés selon une grammaire : il y a frontière nette entre énoncés corrects (générables par la grammaire) et énoncés incorrects (les autres). C'est le cas des formalismes de réécriture et des formalismes à décorations, décrits précédemment. Un formalisme est non-génératif s'il privilégie l'extraction d'un maximum d'informations à partir d'un énoncé donné. On peut, si on le souhaite, définir une notion de grammaticalité, et distinguer des phrases pour lesquelles cette extraction d'informations a « complètement » réussi (c'est le cas dans le formalisme des Grammaires de Propriétés, Blache (2001)). Mais ce n'est ni nécessaire ni central.

Nous verrons au chapitre 8 que ce point de vue nous semble simpliste, et qu'il est possible d'aller au-delà de cette contradiction apparente. Il se trouve que nous sommes partis de formalismes génératifs, dont en particulier les Grammaires lexicales-fonctionnelles (LFG, voir chapitre 9) et les RCG. Nous décrivons ci-dessous très brièvement les LFG et quelques autres formalismes (ou familles de formalismes). Le formalisme des Méta-RCG, que nous avons développé, est étudié au chapitre 10.

5.2 Grammaires lexicales-fonctionnelles (LFG)

Le formalisme des grammaires lexicales-fonctionnelles (*Lexical-Functional Grammars*, d'où l'acronyme que nous utiliserons désormais, *LFG*) est un formalisme à décorations dont le formalisme squelette est celui des grammaires non-contextuelles, et dont les décorations sont appelées *structures fonctionnelles* (Kaplan et Bresnan, 1982). Historiquement, la définition du modèle par Kaplan et Bresnan procède d'une critique des grammaires génératives transformationnelles, en vogue à l'époque, mais dont les trois principaux points faibles étaient leur complexité excessive, leur difficulté à rendre compte des langues à ordre plus libre, et leur inadéquation aux expériences psycholinguistiques censées au contraire plaider en leur faveur.

L'idée sous-jacente à LFG est qu'il n'y a pas nécessairement superposition des fonctions grammaticales avec les positions syntaxiques, pour au moins deux raisons :

1. les fonctions grammaticales induisent un graphe de dépendances syntaxiques qui n'est pas nécessairement un arbre, alors que les positions syntaxiques induisent une structure d'arbre,
2. une même position syntaxique peut être remplie par la réalisation de diverses fonctions syntaxiques.

[32] L'école de Cambridge et ses *grammaires génératives transformationnelles* est à l'origine du terme. Mais de nombreux formalismes qui se sont définis en réaction contre les grammaires génératives transformationnelles sont néanmoins des formalismes génératifs.

Les fonctions grammaticales sont de ce fait des fondamentaux de la théorie LFG, au même titre que les constituants. Les *constituants* sont donc gérés par le squelette syntaxique non-contextuel[33], et les fonctions grammaticales sont gérées par des *équations fonctionnelles* qui reposent sur l'unification. Une analyse LFG est donc d'une part un arbre de constituants, appelé *structure de constituants* ou *c-structure*, et d'autre part une *structure fonctionnelle*, ou *f-structure*, qui est une structure de traits dont nous allons maintenant montrer le contenu et la construction.

Soit une règle du squelette, par exemple $\mathrm{P} \rightarrow \mathrm{SNV}$. À chaque non-terminal, de partie droite comme de partie gauche, est associée une f-structure (si P est l'axiome, sa f-structure est la f-structure complète ; les autres non-terminaux n'ont qu'une f-structure partielle). Les décorations associées à cette règle squelette définissent par unification la f-structure de P, notée $\uparrow$, en fonction de celle de SN et de celle de V. Pour cela, on note sous chaque non-terminal les équations où sa f-structure intervient, équations dans lesquelles cette f-structure est notée $\downarrow$. L'opérateur = note l'unification. Une façon raisonnable de décorer la règle squelette ci-dessus est donc par exemple :

$$\begin{array}{lcll} \mathrm{S} & \rightarrow & \mathrm{SN} & \mathrm{V} \\ & & (\uparrow \mathrm{subj}) = \downarrow & \uparrow = \downarrow \end{array}$$

Il s'avère pourtant que cette notation, qui positionne des règles sous chaque non-terminal, n'est pas toujours la plus pratique, bien qu'elle soit souvent très lisible. Nous utiliserons donc une autre notation, où toutes les équations sont les unes en dessous des autres, et où on identifie la f-structure du i-ième non-terminal de partie droite par $\downarrow i$. On notera donc la règle LFG précédente sous la forme suivante :

$$\begin{array}{lcl} \mathrm{S} & \rightarrow & \mathrm{SN\ V} \\ & & \uparrow \mathrm{subj} = \downarrow 1 \\ & & \uparrow = \downarrow 2 \end{array}$$

Certaines règles comportent seulement un mot en partie droite, ce sont les règles lexicales. En voici deux exemples :

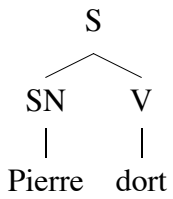
[33] Il est habituel de considérer que les grammaires non-contextuelles ne permettent pas l'optionnalité ou l'opérateur de Kleene sur les non-terminaux de partie droite, alors que le squelette syntaxique de LFG le permet. Ce n'est pourtant qu'une question de notations, puisqu'il y a équivalence forte entre les deux types de grammaires.

SN → Pierre
↑pred = 'Pierre'
↑gender = masc
↑number = sing
V → dort
↑pred = 'dormir<subj>'
↑number = sing
↑pers = 3
↑mode = indicative
↑tense = present

Les attributs utilisés dans ces règles sont standards, à l'exception de l'attribut *pred*, qui a une signification spéciale. La valeur d'un pred commence par un identifiant de l'unité sémantique véhiculée par la tête de la structure. Il est le cas échéant suivi d'un cadre de sous-catégorisation, constitué d'arguments syntaxiques séparés par des virgules. Différentes classes de réalisations possibles d'un même argument sont séparées par le signe | et un argument facultatif est mis entre parenthèses. Les classes de réalisations possibles d'arguments sont prises dans une liste close comprenant typiquement *obj* (syntagme nominal ou clitique), *vcomp* (infinitive objet), *scomp* (complétive objet), *acomp* (attribut adjectival), *ncomp* (attribut nominal), toutes les combinaisons possibles de ces classes de réalisations avec une préposition, notées *prep-classe* (par exemple *de-obj* ou *pour-acomp*, voire *à-scomp* pour *à ce que...*), ainsi que *subj*, et *adjunct* (modifieurs). La classe *adjunct* est spéciale en ce sens qu'elle va correspondre dans la f-structure à un attribut dont les valeurs possibles ne sont pas des f-structures mais des listes (éventuellement vides) de f-structures. Enfin, les arguments syntaxiques correspondant à des arguments sémantiques sont mis à l'intérieur de chevrons, les autres sont donnés après (voir un exemple plus bas).

Les règles données jusqu'ici permettent d'analyser la phrase *Jean dort*, le résultat étant le suivant :

$$\begin{bmatrix} \text{pred} = \textbf{'dormir <subj>'} \\ \text{subj} = \begin{bmatrix} \text{pred} = \textbf{'Pierre'} \\ \text{gender} = \text{masc} \\ \text{number} = \text{sing} \end{bmatrix} \\ \text{number} = \text{sing} \\ \text{person} = 3 \\ \text{mode} = \text{indicative} \\ \text{tense} = \text{present} \end{bmatrix}$$



Cette analyse est correcte car la f-structure obtenue vérifie les trois principes suivants :

Unicité : un même attribut ne peut apparaître qu'une seule fois dans une même sous-f-structure, et deux réalisations d'un même argument sous-catégorisé par le *pred* d'une sous-f-structure donnée ne peuvent pas coexister dans cette sous-f-structure,

Cohérence : tout attribut qui est une réalisation d'argument doit être sous-catégorisé par le *pred* de la sous-f-structure courante,

Complétude : tout argument sous-catégorisé par le *pred* d'une sous-f-structure doit avoir une réalisation qui est présente dans cette sous-f-structure.

Outre l'opérateur = qui dénote l'unification, le formalisme définit d'autres opérateurs permettant de vérifier l'existence ou l'absence d'un trait ou d'ajouter une structure dans une liste de structures (qui remplit un attribut *adjunct*). Un opérateur particulier permet de rendre obligatoire l'unification d'une structure avec une autre qui apporte une certaine valeur à un certain trait. Cet opérateur, noté $=_c$, est appelé *unification contrainte*. Pour illustrer cet opérateur, voici une entrée possible de la forme verbale *faut* (du verbe *falloir*) :

V → faut
↑pred = 'falloir<vcomp>subj'
↑subj form $=_c$ il
...

Cette règle ne pourra participer à la construction d'une structure complète que si la sous-f-structure associée au V est unifiée, à un moment donné de l'analyse, avec une autre structure qui lui confère le trait subj form = il. On note par ailleurs que l'argument *subj* ne correspondant pas à un argument sémantique, il est placé hors chevrons. C'est ainsi que l'on peut gérer l'impersonnel.

Par ailleurs, il peut s'avérer que deux attributs différents (éventuellement de deux sous-f-structures différentes) ont pour valeur la même structure. Cette valeur est alors partagée.

Enfin, on peut dénoter dans une équation non seulement un (sous-)attribut dans une structure mais également un ensemble de (sous-)attributs, chaque analyse devant choisir un des éléments de cet ensemble. Ceci se fait par l'utilisation d'expressions régulières sur les chemins. Par exemple, une équation comme ↑(vcomp|scomp)* vcomp = ↓1 signifie que l'on la f-structure associée au premier symbole de partie droite doit devenir la valeur de l'attribut vcomp de la f-structure associée au symbole de partie gauche ou d'une quelconque de ses sous-f-structures accessible en suivant un chemin ne passant que par des vcomp ou des scomp.

Le modèle proposé par LFG est séduisant à bien des égards, tant du point de vue de la modélisation de la langue que du point de vue de la lisibilité des grammaires. De notre point de vue, il a cependant trois inconvénients principaux. Tout d'abord, c'est un formalisme à décorations

dont la complexité importante (il est NP-complet) n'est pas triviale à gérer dans des analyseurs efficaces. Ensuite, les constituants sont nécessairement continus. Enfin, et c'est le principal problème à notre avis, il y a quasiment[34] identité entre dépendances syntaxiques et dépendances sémantiques.

5.3 Grammaires d'Adjonction d'Arbres avec structures de traits (fb-LTAG)

Les fb-LTAG (pour *feature-based Lexicalized TAGs*, souvent abrégé par abus de langage en *TAG*) sont des TAG lexicalisées (voir plus haut) décorées par des structures de traits typés (Abeillé *et al.*, 2002). Elles sont couramment utilisées car elles permettent l'obtention immédiate d'une structure prédicats-arguments presque correcte, pour peu que la grammaire suive trois principes linguistiques de bonne correction : le *principe de minimalité sémantique* (chaque arbre doit correspondre à une unité sémantique minimale), le *principe de lexicalisation* (chaque arbre doit être lexicalisé par un lexème sémantiquement non vide, son ancre principale), et le *principe de co-occurrence prédicat-arguments*. Ce dernier principe est responsable de la localisation et de la définition des dépendances sémantiques. Il force à considérer l'ancre principale d'un arbre comme un prédicat dont l'arité (sémantique) est fixée, en sorte que tous ses arguments (syntaxiques) sont obligatoires. Ce principe est mis en œuvre par l'opération de substitution. Ces trois principes permettent de considérer la structure de dérivation proposée par LTAG comme une structure proche d'une structure de dépendances syntaxico-sémantiques (Rambow et Joshi, 1994a).

Ces grammaires souffrent toutefois de trois problèmes principaux. Tout d'abord, l'interface syntaxe-sémantique est automatiquement constituée par l'appariement entre arbre dérivé et arbre de dérivation. Ceci est élégant, tant que cela fonctionne bien. Toutefois, on trouve facilement des exemples où l'arbre de dérivation n'encode pas correctement les dépendances syntaxiques. Mais c'est surtout le fait que les dépendances sémantiques forment un DAG, et non un arbre, qui induit une incomplétude profonde des TAG à cet égard. Divers travaux cherchent d'ailleurs à modifier l'arbre de dérivation (parfois en utilisant également des informations extraites de l'arbre dérivé, voir Seddah (2004)), pour reconstituer un graphe de dépendances sémantiques plus pertinent. Enfin, toute dépendance syntaxique induit une dépendance sémantique, puisqu'elle se traduit (par exemple) par une substitution. Pour cette raison, la dépendance syntaxique entre le sujet d'un verbe à montée (*Pierre semble dormir*) et ledit verbe n'est pas modélisée, car elle induirait une dépendance sémantique incorrecte.

[34] Les seuls points qui permettent d'entrevoir une distinction sont d'une part la notion d'arguments hors-chevrons dans les cadres de sous-catégorisation, et d'autre part l'ordre dans lequel les arguments sous-catégorisés sont répertoriés, qui peut éventuellement porter du sens.

Par ailleurs, la complexité du squelette TAG (sans les traits) est insuffisante pour encoder convenablement les structures syntaxiques complexes (pour l'allemand, par exemple, on pourra se reporter à Rambow et Joshi (1994b); Gerdes et Kahane (2001)). Nous reviendrons sur ce problème au chapitre 10.

Le troisième problème des TAG (au sens de fb-LTAG) est celui de la redondance pathologique qui caractérise ses grammaires : même en associant les mots à des listes de *schémas d'arbres* (arbres élémentaires dont l'ancre et certains traits sont sous-spécifiés), on a besoin de plusieurs milliers de schémas d'arbre pour construire une grammaire à moyenne couverture d'une langue comme le français (Abeillé *et al.*, 2002). Une solution possible à ce problème a été proposée par Candito (1999) : on peut décrire dans une couche plus abstraite, appelée *méta-grammaire*, les différents schémas d'arbres possibles sous la forme d'une hiérarchie de classes de propriétés linguistiques. Ces travaux, prolongés par d'autres (Clément et Kinyon, 2003; Crabbé *et al.*, 2004; Thomasset et Éric Villemonte de la Clergerie, 2005), ne semblent toutefois pas réhabiliter les TAG, mais bien au contraire montrer la faisabilité et la pertinence d'une description de la langue qui soit d'un niveau d'abstraction plus élevé et plus proche des concepts linguistiques. Malheureusement, les métagrammaires sont principalement utilisées pour générer des TAG, et non comme un formalisme linguistique à part entière. Des travaux à mi-chemin entre ces deux extrêmes sont toutefois en cours, pour utiliser au mieux lors de l'analyse les abstractions et factorisations présentes dans la métagrammaires mais absentes de la TAG telle qu'elle est générée habituellement (Thomasset et Éric Villemonte de la Clergerie, 2005).

5.4 Grammaires Syntagmatiques guidées par les Têtes (HPSG)

Les grammaires du modèle HPSG (*Head-driven Phrase Structure Grammars*, voir Pollard et Sag (1994)) reposent entièrement sur un graphe d'héritage multiple dont les entrées lexicales sont les feuilles, auxquelles sont associées des structures de traits typées appelées *descriptions*. Ces structures de traits, combinées entre elles via des structures de combinaison exprimées dans la même syntaxe que les descriptions elles-mêmes.

Descriptions et structures de combinaison répartissent l'information linguistique en trois ou quatre attributs principaux qui correspondent à peu près à un niveau sémantique, un niveau syntaxique (dépendances), parfois un niveau topologique (ordre des mots), et enfin un niveau phonologique.

Ce formalisme est très intéressant en ce qu'il tente d'unifier dans une même description des informations linguistiques de natures variées. Mais il souffre, à notre avis, de trois défauts majeurs. Tout d'abord, sa complexité est bien trop importante (dans le cas général c'est un formalisme indécidable), ce qui a pour manifestation que l'encodage en HPSG de structures

relativement élémentaires se fait souvent au prix d'une perte de l'accès aux propriétés simples de ces structures (on se reportera par exemple à Kahane (2006) pour une discussion de ce type à propos des structures d'arbres à bulles). Ensuite, les motivations de la répartition en trois ou quatre niveaux sont peu claires et assez floues (la preuve en est par exemple la variabilité du nombre de niveaux). Enfin, la structure en constituants que l'on obtient est induite par le processus d'analyse plutôt que par la grammaire elle-même, ce qui est loin d'être pleinement satisfaisant.

5.5 Grammaires d'Unification Polarisées (GUP)

Les grammaires d'unification polarisées (GUP), introduites par Kahane (2004), est un formalisme générique de combinaison de DAG où les combinaisons sont contrôlées par une polarisation des nœuds (la notion de polarité a été introduite par Nasr (1995) et développée en profondeur par *Perrier02GI*). Il peut être vu comme un appareil formel permettant d'encoder divers formalismes, dont certains, développés depuis plusieurs dizaines d'années, sont regroupés sous l'appellation *Grammaires de dépendances*. Les GUP permettent la formalisation ou l'encodage des Grammaires d'Unification Sens-Texte (GUST) présentées dans Kahane (2005), qui sont une variante formalisée de la Théorie Sens-Texte d'Igor Mel'čuk. Elles sont une extension de formalismes tels que celui de Nasr (1995), et peuvent représenter très facilement les Grammaires d'Interactions de Perrier (2000) (pour lesquelles il existe une grammaire du français à large couverture et un analyseur syntaxique, LEOPAR).

Il nous semble qu'à l'heure actuelle les GUP soient un formalisme agréable pour la visualisation et la compréhension d'autres formalismes dans lesquelles on les représente. Toutefois, il ne s'agit finalement « que » d'un formalisme d'implémentation. En tant que tel, il serait souhaitable que propriétés formelles soient bien connues, que les restrictions permettant d'en rendre l'analyse polynomiale soient bien définies, et qu'il existe des analyseurs efficaces pour GUP. Ce n'est pas le cas actuellement (Kahane, 2004).

5.6 Grammaires de Propriétés

Le formalisme des Grammaires de Propriétés (GP), développé par Ph. Blache (Blache, 2001), est très différent des formalismes présentés jusqu'ici : ce n'est ni un formalisme de réécriture, ni un formalisme à décorations. Il s'agit d'un formalisme reposant sur les *contraintes*, c'est-à-dire que l'on définit la structure et la correction d'une phrase par la satisfaction sur cette phrase d'un certain nombre de contraintes. Comme indiqué plus haut, ce n'est pas un formalisme génératif.

Les GP définissent donc un certain nombre de contraintes, qui relient deux catégories (ou plus) entre elles, le terme « catégorie » étant entendu ici au sens de syntagme. Six types de contraintes sont définies : les contraintes de linéarité, de dépendance, d'obligation, d'exclusion, d'implication et d'unicité. Par conséquent, la structure en constituants n'a pas le statut privilégié qu'elle a dans la plupart des autres formalismes. De plus, on peut demander à un analyseur de ne vérifier que certaines contraintes, obtenant ainsi des analyses de granularités différentes. Enfin, la non-générativité est liée à ce que l'on ne cherche pas à imposer à une phrase de vérifier *toutes* les contraintes définies dans la grammaires, mais que l'on cherche la ou les analyses maximisant le nombre de contraintes vérifiées : le but d'une analyse est d'extraire le maximum d'informations d'une phrase, même dans le cas où elle ne satisfait pas toutes les contraintes. Nous reviendrons sur ces problématiques au chapitre 8.

6 Grammaires probabilistes

L'analyse d'une phrase à l'aide d'une grammaire de réécriture conduit généralement, en cas de succès, à plus d'une analyse. Cette multiplicité des analyses réside dans le fait qu'à chaque pas de dérivation, plusieurs choix sont possibles. Le paradigme des grammaires probabilistes utilise les décorations, présentées précédemment, pour associer à chacun de ces choix une probabilité qui, dans le cas général, dépend des choix déjà effectués, c'est-à-dire en fonction de l'*historique*[35], et obtenir ainsi *in fine* une probabilité pour l'analyse complète. D'où le nom de *grammaires fondées sur l'historique* donné à ce paradigme (Black *et al.*, 1992). L'objectif est d'éliminer alors toutes les analyses qui ne sont pas parmi les n plus probables (on parle de n-*best*).

Formellement, supposons d'abord que l'on sache restreindre les chaînes de dérivation possibles en sorte qu'il y ait une bijection entre chaînes de dérivation acceptées et graphes d'analyses[36]. Soit alors une chaîne de dérivation acceptée (qui aboutit ou non à une chaîne finale correcte) :

$$P_w^0 \underset{R_0,w}{\Longrightarrow} \dots \underset{R_{i-1},w}{\Longrightarrow} P_w^i.$$

[35]Ce qui est à la fois général et réducteur. En effet, il est donc impossible dans ce cadre de faire dépendre la probabilité d'une sous-analyse de l'existence d'une autre sous-analyse (ayant certaines probabilités) pour un intervalle différent. C'est pourtant à l'aide d'heuristiques mettant en œuvre de telles dépendances que nous avons construit un chunker aux résultats très satisfaisants (voir chapitre 12).

[36]Dans le cas de grammaires non-contextuelles, on peut par exemple se restreindre aux dérivations gauches, c'est-à-dire celles où à chaque étape le symbole réécrit est le plus à gauche dans la proto-phrase courante.

La probabilité qu'on associe à cette chaîne de dérivation est :

$$P(P_w^0, \ldots, P_w^i) = \prod_{k=0}^{i-1} P(R_k \mid P_w^0, \ldots, P_w^k).$$

En général, on considère qu'il est impossible et pas véritablement souhaitable d'utiliser tout l'historique : outre la complexité d'un tel modèle (exponentielle en la longueur de la phrase même sur un squelette non-contextuel), cela suppose que l'on dispose de moyens d'apprendre une distribution de probabilités d'une granularité telle qu'elle nécessiterait un corpus d'apprentissage colossal. Par conséquent, il est d'usage de regrouper les historiques en classes d'équivalences (Nasr, 2005) et de faire dépendre la probabilité non pas de l'historique $P_w^0, \ldots, P_w^k$ mais de sa classe d'équivalence $\overline{P_w^0, \ldots, P_w^k}$:

$$P(P_w^0, \ldots, P_w^i) = \prod_{k=0}^{i-1} P(R_k \mid \overline{P_w^0, \ldots, P_w^k}).$$

Un tel modèle permet différentes utilisations :

– choisir une ou plusieurs sous-analyses les plus probables en cours d'analyse, et ne continuer à essayer de construire une analyse qu'à l'aide de ces sous-analyses,
– choisir une ou plusieurs analyses les plus probables parmi les analyses de toute une phrase,
– choisir, à un moment donné de la dérivation, la règle la plus probable à essayer.

De nombreux travaux récents, et – il faut bien le dire – à la mode, cherchent à modéliser les langues par des grammaires de réécriture aussi simples que les grammaires non-contextuelles, en décorant ces dernières de modèles probabilistes reposant sur de tels principes. Le formalisme obtenu en regroupant trivialement les historiques en une seule classe d'équivalence est appelé Grammaires Non-Contextuelles Probabilistes (*Probabilistic Context-Free Grammars*, *PCFG*). Dans ce cas, il n'est donc pas tenu compte de l'historique :

$$P_{PCFG}(P_w^0, \ldots, P_w^i) = \prod_{k=0}^{i-1} P_{PCFG}(R_k).$$

Si les résultats obtenus avec des PCFG semblent bons, c'est toutefois au prix de trois sacrifices, à notre sens douloureux, qui sont faits sur l'autel du pragmatisme :

– l'apprentissage de ces modèles nécessite des corpus annotés manuellement, qui sont coûteux à développer et n'existent pratiquement que pour l'anglais à une échelle suffisante,

– les structures complexes des langues, qui ne sont pas exprimables par des grammaires non-contextuelles, ne sont que simulées via le modèle probabiliste, c'est-à-dire d'une façon linguistiquement non pertinente,
– de tels modèles ont un pouvoir d'explication et d'explicitation des mécanismes à l'œuvre dans les langues tout à fait limité.

Ces remarques ne sont pas à comprendre comme un rejet de l'introduction de probabilités dans les formalismes linguistiques. Bien au contraire, nous pensons que c'est là une des prochaines étapes importantes de nos travaux. Mais ceci doit se faire à deux conditions :

– être capable de se passer des corpus annotés manuellement, et savoir apprendre des modèles à partir de corpus bruts,
– utiliser comme formalismes squelette des formalismes linguistiquement adaptés, c'est-à-dire décorer par des modèles probabilistes des grammaires plus puissantes que les grammaires non-contextuelles.

De telles approches sont du reste déjà en cours de développement (on pourra par exemple se reporter à Nasr (2005)).

Chapitre 2

Lexiques pour le traitement automatique : modèles et ressources

1 Lexiques et traitements pré-syntaxiques

Les grammaires modélisant les langues ont de nombreuses spécificités par rapport aux autres grammaires, comme celles décrivant les langages de programmation : de façon informelle, le nombre de mots (ou de formes) que l'on doit pouvoir traiter est considérable. De plus, il n'est pas immédiat d'identifier quels sont les mots (ou les formes) qui constituent une phrase donnée. C'est la raison pour laquelle ce ne sont pas les phrases brutes qui sont prises en entrée des grammaires, mais des structures plus complexes, construites grâce à divers outils qui utilisent des recueils de données associant à chaque mot de la langue un certain nombre d'informations. De tels recueils de données sont appelées des *lexiques*.

Ce chapitre commence par des définitions générales concernant les lexiques et les entités qui les composent. Puis nous donnons quelques rappels sur la modélisation de la morphologie et des lexiques dits morphologiques, puis sur les lexiques dits syntaxiques (des définitions de ces deux types de lexiques sont proposées plus bas). Nous finissons par un bref panorama des principaux lexiques disponibles pour le français.

1.1 Formes, tokens, mots et lemmes

Il nous faut introduire quelques définitions. Nous appelons *forme* (ou *forme fléchie*) une unité graphique linguistique atomique, c'est-à-dire une unité qui est élémentaire du point de vue syntaxique (elle est syntaxiquement inanalysable, même si elle peut l'être d'un autre point de

vue, en particulier du point de vue morphologique[1]). On peut placer les formes aux feuilles d'un arbre dont les différents niveaux, en partant des formes, sont les lemmes, les classes syntaxiques et les catégories.

Nous appelons *token* les séquences de caractères présents dans le corpus et séparés par des espaces ou par certaines autres marques typographiques (ponctuation,...) qui varient selon la langue (et qui sont malheureusement parfois ambiguës). Nous appelons *mot* une forme simple (c'est-à-dire une forme, ou le résultat de la fusion de plusieurs formes, qui constitue tout ou partie d'un token) ou un composant de forme composée (c'est-à-dire une forme qui se répartit sur plusieurs tokens). Ainsi, *a priori* comporte deux mots mais constitue une seule forme, *aux* est un seul token et un seul mot mais deux formes, et *donne-moi* est un seul token, mais deux mots et deux formes. Une forme composée de plusieurs mots est appelée *multi-mot* (ainsi *pomme de terre*). Un mot composé de plusieurs formes est une *agglutinée* (ainsi *aux* ou *duquel*).

Les lemmes (ou lexies) sont des classes d'équivalence de formes perçues comme partageant un même « sens » (un même signifié), à certaines variations près[2]. La définition de ces « variations » constitue des *informations morphologiques*, le plus souvent structurées en *traits morphologiques* (comme le genre, le nombre, la personne, le temps, le mode, l'aspect,...). Nous emploierons donc parfois le terme de *forme non-décorée* à la place du terme *forme*, par opposition au terme de *forme semi-décorée*, qui dénote une forme (non-décorée) assortie d'informations morphologiques, et à celui de *forme décorée*, qui dénote une forme semi-décorée assortie de son lemme. Une forme décorée est donc équivalente à un triplet *(forme,lemme, étiquette morphologique)*.

Par exemple, les formes *mangions* et *mangeraient* sont deux formes d'un même lemme, que l'on peut représenter, suivant l'usage, par l'identifiant *manger*. Les informations morphologiques associées aux formes peuvent être respectivement représentées par *[verbe, indicatif, imparfait, 1^e personne, pluriel]* et *[verbe, conditionnel, présent, 3^e personne, pluriel]*. La forme semi-décorée associée à la forme non-décorée *mangions* est alors *(mangions, [verbe, indicatif, imparfait, 1^e personne, pluriel])*, et sa forme décorée est *(mangions, manger, [verbe, indicatif, imparfait, 1^e personne, pluriel])*.

[1]Ceci suppose que l'on dispose d'une distinction précise entre morphologie et syntaxe. C'est pourtant loin d'être le cas. En effet, même à l'écrit il est difficile de définir une forme autrement qu'en des termes vagues faisant appel à des notions floues comme l'atomicité syntaxique et/ou sémantique. Ce qui n'aide pas nécessairement à décider combien de formes il y a dans *ceux-ci* (cf. *ceci* ou *ce ...-ci*), *attrape-nigaud*, *aux*, *pomme de terre*, ou *connu comme le loup blanc*. Nous reviendrons sur ce problème au chapitre 4. Quoi qu'il en soit, la définition de ce qu'est une forme relève donc en partie d'une convention.

[2]En toute rigueur, il faudrait ne pas confondre la forme (non-décorée) avec sa graphie, c'est-à-dire avec la suite de caractère qui la constitue. C'est en effet un abus de langage, puisque nous venons de définir un lemme comme une classe d'équivalence : une forme ne peut appartenir qu'à une seule classe, alors qu'une même graphie peut correspondre à différentes formes de différents lemmes. Mais nous nous autoriserons à faire usage de cet abus de langage.

Tokens	Mots	DAG de formes (non-décorées)
ereurs	erreurs	erreurs
aujourd'hui	aujourd'hui	aujourd'hui
l'idée	l' idée	l' idée
a priori	a priori	a_priori
pomme de terre	pomme de terre	(pomme de terre | pomme_de_terre)
duquel	duquel	de lequel
grace au	grâce au	(grâce à | grâce_à) le
à l'image de	à l' image de	(à_l'_image_de | à l' image de)
à l'instar de	à l' instar de	à_l'_instar_de
donne-m'en	donne-m'en	donne -m' en
a-t-elle	a -t-elle	a -t-elle
12 345,6	_NUMBER	_NUMBER
www.inria.fr	_URL	_URL

TAB. 1 – Quelques exemples du passage de tokens aux mots puis aux formes.

Quelques exemples de ces différences sont données au tableau 1.

1.2 Du texte brut à l'entrée des analyseurs

1.2.1 Quelle entrée pour les analyseurs ?

Au niveau des grammaires, la pléthore de formes possibles peut être traitée par différentes approches, dont les suivantes.

1. On peut utiliser directement les tokens comme symboles terminaux de la grammaire.
2. On peut remplacer la chaîne de tokens par un DAG de mots (ou une chaîne de mots, si on utilise un modèle probabiliste ou heuristique pour choisir un seul chemin parmi le DAG complet). Les mots sont alors utilisés comme symboles terminaux de la grammaire.
3. On peut alors remplacer le DAG de mots par un DAG de formes non décorées (même remarque que précédemment). Ces formes non décorées sont alors utilisés comme symboles terminaux de la grammaire.
4. On peut alors remplacer le DAG de formes par un DAG de formes décorées (même remarque que précédemment), c'est-à-dire (en oubliant la forme non-décorée sous-jacente) un DAG de lemmes décorés d'informations morphologiques. Les lemmes sont alors utilisés comme symboles terminaux de la grammaire. Les traits morphologiques sont quant à eux traités soit dans le squelette syntaxique de la grammaire, s'il en est capable (grammaires de prédicats clos par intersection), soit dans des décorations.

5. On peut alors remplacer les lemmes (ou les formes) par leur(s) classe(s) syntaxique(s) décoré(s) des informations morphologiques (dont les lemmes). Les classes syntaxiques sont alors utilisés comme symboles terminaux de la grammaire.

6. On peut également remplacer les lemmes par leur(s) catégorie(s) décorée(s) d'informations morphologiques (dont les lemmes) et d'informations syntaxiques. Ces informations syntaxiques sont également traitées soit dans le squelette syntaxique de la grammaire, soit dans des décorations.

Toutes ces approches sont compatibles avec la lexicalisation de la grammaire, mais les symboles terminaux ne représentent pas la même chose dans chaque cas.

On appelle *lexique morphologique* la ressource linguistique associant à chaque forme son lemme et ses traits morphologiques, et *analyseur morphologique* un outil qui calcule ces mêmes informations (avec le risque de se tromper) lorsque la forme étudiée est absente du lexique morphologique.

1.2.2 Traitements pré-syntaxiques

L'opération consistant à transformer un texte brut en une chaîne (voire un DAG) de mots est dénotée par l'anglicisme *tokenisation*[3]. Nous appellerons *DAGage* l'opération consistant à construire, à partir des tokens, un DAG (ou une chaîne) de formes non décorées. On notera qu'il n'est pas nécessaire que les mots correspondent précisément aux tokens. Outre une éventuelle correction orthographique, il est par exemple raisonnable d'associer au token *123* le mot *_NUMBER*, ou encore à un token inconnu un mot spécial modélisant les mots inconnus, comme *_uw*. Nous appellerons abusivement *entités nommées* toutes les formes « artificielles » de ce type. Enfin, l'opération consistant à construire une chaîne (ou un DAG) de formes (semi-)décorées à partir des tokens ou des formes non-décorées et à l'aide d'un lexique et/ou d'un analyseur morphologique est appelée *étiquetage morphologique* (ou *morphosyntaxique*)[4]. Tous ces traitements forment ce que nous appellerons le *traitement pré-syntaxique*, qui est effectué par une *chaîne de traitement pré-syntaxique*. Nous avons développé une telle chaîne, nommée SxPipe, décrite au chapitre 11 (mais qui n'est pas un étiqueteur morphosyntaxique).

[3] Il y a donc confusion dans les termes : la tokenisation n'est *pas* la construction d'un DAG de tokens, mais la construction d'un DAG de mots à partir d'une suite de tokens. Par ailleurs, nous réservons le terme *segmentation* au découpage en phrases, c'est-à-dire par définition en chaînes de formes perçues comme devant être reconnues par (l'axiome d')une grammaire.

[4] Cette opération est très couramment utilisée, en particulier parce qu'elle permet de diminuer sensiblement l'ambiguïté au niveau des analyseurs. Mais elle induit de nombreux problèmes. Or nos systèmes d'analyse sont suffisamment efficaces pour se passer de cette phase préliminaire de réduction de l'ambiguïté. Ils n'utilisent donc pas d'étiqueteur morphosyntaxique (ils utilisent évidemment une chaîne de traitement pré-syntaxique, laquelle transforme les corpus bruts en DAG de formes non-décorées). Nous reviendrons sur ces questions au chapitre 8.

1.2.3 Lexiques et analyseurs

Après ces traitements pré-syntaxiques, et en préliminaire à l'analyse proprement dite, un analyseur syntaxique va en général chercher dans les lexiques l'ensemble des informations disponibles pour chaque élément du DAG (ou de la chaîne) qu'il reçoit en entrée. Cette opération, consistant à décorer tokens, mots ou formes, est effectuée par ce qu'on appelle le *lexeur*. L'analyseur travaille alors sur le résultat du lexeur. Les différents types de remplacements définis plus haut sont donc mis en œuvre par la combinaison du traitement pré-syntaxique et du lexeur.

1.2.4 Quelques exemples : LTAG, LFG, Méta-RCG

En général, les grammaires LTAG prennent en entrée des chaînes (ou des DAG) de classes syntaxiques décorées d'informations morphologiques. Les grammaires LFG prennent généralement en entrée des chaînes (ou des DAG) de mots. Toutefois, nous verrons au chapitre 9, où nous décrivons notre analyseur LFG nommé SXLFG, que le lexeur de ce dernier transforme un DAG de mots en un DAG de catégories décorées d'informations morphologiques et syntaxiques, évitant ainsi à la grammaire de comporter plusieurs centaines de milliers de symboles terminaux distincts. Le chapitre 10 présente un formalisme capable de prendre en compte des informations syntaxiques et sémantiques de façon efficace, tout en prenant en entrée une suite de mots.

1.3 Lexiques morphologiques et lexiques syntaxiques

Les informations morphologiques sont présentes dans le lexique morphologique ou calculées par un analyseur morphologique, mais ce n'est pas le cas des informations syntaxiques ou de l'association entre lemmes (ou formes) et classes syntaxiques. Ces données (ou le moyen de les calculer) constituent ce que l'on appelle un *lexique syntaxique*. Si de plus les informations ou les classes considérées ne sont pas purement syntaxiques, mais relèvent également de la sémantique (lexicale), on parle alors de *lexique syntaxico-sémantique*.

2 Ressources morphologiques

2.1 Lexiques, analyseurs et descriptions morphologiques

Comme indiqué précédemment, on peut classer en deux familles les moyens d'associer à un mot toutes les formes qui lui correspondent, c'est-à-dire tous les couples formés d'un lemme et d'étiquettes morphologiques : les *lexiques morphologiques* et les *analyseurs morphologiques*.

Ces deux familles sont complémentaires : les lexiques morphologiques permettent un accès immédiat au résultat, ainsi qu'une précision optimale (dans les limites de la qualité des lexiques utilisés), alors que les analyseurs morphologiques permettent le traitement de mécanismes superposés très nombreux, ainsi qu'un rappel optimal (dans les limites de la couverture de l'analyseur). Pour cette raison, les langues flexionnelles sont plutôt traitées à l'aide de lexiques morphologiques (mais les analyseurs morphologiques sont très utiles pour traiter par exemple des phénomènes productifs de dérivation morphologique), mais les langues agglutinantes (turc, coréen) font un usage plus intensif des analyseurs morphologiques (un lexique morphologique du turc, pour pouvoir éviter le recours à un analyseur morphologique, devrait par exemple associer plusieurs milliers de formes à chaque lemme verbal).

Cependant, le développement d'un lexique morphologique pour une langue donnée ne se fait pas directement par la description manuelle des formes associées à chaque lemme : en effet, les langues font généralement usage de motifs en nombre raisonnable pour construire les formes d'un même lemme, voire pour construire un lemme à partir d'un autre. Ces motifs, appelés *paradigmes morphologiques*, permettent la construction d'un lexique morphologique à partir des lemmes pour peu que ces derniers soient représentés comme des couples *(identifiant de lemme, identifiant de paradigme morphologique)*[5]. Si l'on sait utiliser ces paradigmes morphologiques en sens inverse, on peut les utiliser également dans les analyseurs morphologiques, pour identifier les formes possibles associées à un mot.

Pour développer un lexique morphologique et/ou un analyseur morphologique d'une langue, il est donc utile de disposer d'une *description morphologique* de cette langue, qui décrit les paradigmes morphologiques dans un formalisme qui permette la construction d'un outil permettant de passer des lemmes aux formes (*conjugueur*) et d'un outil permettant de passer des formes aux lemmes (*lemmatiseur* ou *dé-conjugueur*).

2.2 Morphologie à deux niveaux

Depuis le début des années 80, la représentation et l'utilisation de descriptions morphologiques des langues se fait dans une des variantes ou des évolutions du paradigme de la *morphologie à deux niveaux*, introduit initialement par Koskenniemi (1983).

Nous ne donnerons ici qu'un aperçu très rapide de ce qu'est la morphologie à deux niveaux. L'idée est d'associer une forme de surface à une forme profonde (ce sont les deux niveaux) par l'intermédiaire de diverses contraintes sur ces deux formes, qui doivent répondre à trois critères :

[5] Nous reviendrons sur ces questions plus en détails au chapitre 4.

- elles procèdent par association d'un caractère d'un niveau à un caractère d'un autre niveau,
- elles sont appliquées en parallèle (et non en cascade),
- elles peuvent faire allusion aux contextes qui entourent le lieu où elles ont un effet,
- elles ont accès au lexique.

Ces contraintes sont exprimées sous une forme régulière, et toutes les contraintes sont compilées en un transducteur à états finis. La raison pour laquelle le formalisme de la morphologie à deux niveaux repose sur des contraintes appliquées en parallèle (et non en cascade) est opérationnelle (Karttunen, 2006). Il en est de même de la nécessité que les contraintes apparient un caractère d'un niveau à exactement un caractère de l'autre niveau, ce qui induit la nécessité pour le moins artificielle d'introduire des caractères destinés à être supprimés, représentés par `0`. On introduit de plus un caractère représentant les frontières entre morphèmes, le caractère `+`.

Prenons un exemple. Soit le verbe anglais *spy* (*espionner*). À la troisième personne du singulier du présent, la forme profonde est obtenue par concaténation de `spy` et de la terminaison `s` précédée du caractère `+`. Il se trouve que `spy+s` ne permet pas une mise en correspondance avec une forme de surface qui respecte des règles raisonnables décrivant la morphologie de l'anglais. En revanche, `spy0+s` peut être mis en correspondance avec `spie0s` qui représente bien *spies*, à l'aide des règles suivantes :

```
y :i <=> _ 0 :e
0 :e <=> Cy : _ +s :
```

La première indique qu'un `y` profond est associé à un `i` de surface si immédiatement à droite du couple de positions qui suit le couple (`y`,`i`) dans les deux chaînes (couple de positions représenté par _) se trouve un `0` au niveau profond et un `e` au niveau de surface, c'est-à-dire un *e* épenthétique. La seconde règle, où `C` désigne une quelconque consonne, indique qu'un *e* épenthétique est inséré en surface si le `0` qui lui correspond est encadré par une consonne et un `y` à gauche, et une frontière de morphèmes suivie d'un `s` à droite.

De nombreux travaux ont fait évoluer ce modèle de diverses façons. Les plus intéressants sont ceux qui se placent dans le paradigme de la Théorie de l'Optimalité. Ils mettent en œuvre des contraintes qui ne sont plus nécessairement inviolables (contrairement à ce qui se passe dans le modèle initial), qui sont pondérées, et qui sont en principe universelles (seules les pondérations font la différence entre plusieurs modèles et donc plusieurs langues). C'est donc une généralisation, très intéressante sur le fond, de la notion d'application en parallèle des contraintes.

Cependant, et bien que certains travaux utilisent la Théorie de l'Optimalité pour mettre en avant les avantages du parallélisme dans l'application des contraintes, certains autres travaux remettent au goût du jour une certaine forme de successivité. De fait, rappelons qu'historiquement, ce sont des raisons opérationnelles qui ont favorisé l'émergence du parallélisme, mais que

les linguistes ont mis très longtemps à accepter ce point de vue : il semble qu'il y ait peu d'arguments linguistiques en sa faveur, et qu'au contraire il soit plus naturel de décrire la morphologie à l'aide de règles successives.

Mais dans tous les cas, et tant qu'on se restreint à l'utilisation de règles régulières (ce qui est presque toujours le cas), il n'y a pas de différence en termes d'expressivité. C'est donc le compromis entre efficacité et facilité de développement qui devrait ici être la règle. L'intérêt de l'utilisation de règles régulières garantit malgré tout une bonne efficacité dans tous les cas, et permet l'utilisation de ces règles à la fois dans le sens de l'analyse morphologique (lemmatisation) et dans le sens de la génération (flexion/conjugaison). Des outils tels que `xfst` (développé chez Xerox), `fsm` (développé chez AT&T), ou `Zen` (développé par Gérard Huet) permettent la construction effective de lemmatiseurs et de conjugueurs efficaces sous la forme de transducteurs.

Il est fait parfois usage, malgré tout, de *décorations* au dessus des règles régulières à deux niveaux (voir par exemple Namer et Schmidt (1995), qui propose une description de la morphologie du français). Ces décorations, si elles permettent une description plus lisible, et peut-être plus naturelle et plus compacte, induisent des effets négatifs en termes d'efficacité. De plus, rien n'indique qu'un tel point de vue soit indispensable. Par exemple, utiliser des mécanismes spécifiques faisant usage de traits (dans les décorations) pour décrire l'alternance *acteur/actrice* peut être évité en indiquant qu'*acteur* se décline selon un modèle dont les terminaisons sont *teur/teurs/trice/trices*, modèle utilisé également par *recteur* ou *facteur*. Si l'on peut également utiliser des décorations pour induire la suppression d'un caractère dans certains cas, on peut tout aussi bien utiliser un caractère spécial qui tantôt devient ledit caractère et tantôt s'efface. Un tel point de vue systématise l'utilisation de ce que Namer et Schmidt (1995) appelle l'*approche diacritique*.

2.3 Morphologie flexionnelle et morphologie dérivationnelle

Lorsque l'on parle de morphologie, on pense souvent en premier lieu à la *morphologie flexionnelle*, dispositif permettant de passer d'un lemme à ses formes (ou inversement) à l'aide de la classe morphologique du lemme. C'est toutefois oublier un dispositif au moins aussi important, appelé *morphologie dérivationnelle* (ou *morphologie constructionnelle*), qui permet d'obtenir (directement ou non) un lemme à partir d'un ou plusieurs autres lemmes, le principal d'entre eux étant appelé la *base*, en respectant d'autres types de règles qui font partie de la classe morphologique de la base.

C'est la morphologie dérivationnelle qui permet de faire le lien entre *ouvrir* et *boîte* d'une part et *ouvre-boîte* d'autre part, ou encore entre *constituer* et *anticonstitutionnellement*.

L'intérêt de la morphologie dérivationnelle, par rapport à la morphologie flexionnelle, est au moins triple (on se référera par exemple à Dal *et al.* (2005)) :

- elle permet l'analyse morphologique de formes appartenant à des lemmes inconnus du lexique mais qui sont *dérivés* de lemme(s) connu(s) (et permet donc d'éviter un traitement de ces formes en tant que mots inconnus dans les traitements ultérieurs comme l'analyse syntaxique),
- les lemmes dérivés sont souvent très pertinents dans les domaines de spécialité, en particulier pour structuration de terminologies,
- les relations de dérivation morphologique entre lemmes permettent des généralisations pertinentes pour améliorer des tâches telles que la fouille de textes et la recherche d'informations.

Pour le français, ce sont les travaux autour de F. Namer qui sont le plus aboutis quant aux rapports entre morphologie dérivationnelle et traitement automatique des langues. Dans Dal *et al.* (2005) qui décrit le projet MorTAL, les auteurs indiquent en effet que cinq suffixes et trois préfixes fréquents ont été étudiés, constituant ainsi une base de 6 000 entrées lexicales dérivées. Les travaux de P. Zweigenbaum et de ses collaborateurs montrent l'intérêt de l'utilisation de la morphologie dérivationnelle dans l'extraction et la structuration de terminologies pour des domaines spécialisés, en s'aidant d'une ontologie du domaine (voir par exemple Zweigenbaum *et al.* (2003)).

3 Lexiques syntaxiques et sémantiques

Les lexiques syntaxiques (et *a fortiori* sémantiques) sont une ressource autrement plus complexe, et donc plus coûteuse à développer, que les lexiques morphologiques. Les informations qu'associe un lexique syntaxique à une entrée sont principalement de trois types : des informations sur les possibilités topologiques (l'ordre des mots), des informations sur la compatibilité avec certaines constructions syntaxiques, et surtout des informations dites de *sous-catégorisation*[6], qui indiquent les arguments syntaxiques possibles et/ou obligatoires, ainsi que leurs possibles réalisations. Quant aux informations sémantiques, elles sont censées formaliser tout ou partie du sens de l'entrée lexicale. Le lien entre les deux est direct : si le sens de l'entrée est de nature prédicative, ce prédicat (sémantique) a des arguments (sémantiques) qui sont le plus souvent réalisés sous la forme d'arguments syntaxiques. Inversement, la plupart des arguments syntaxiques sont la réalisation d'arguments sémantiques. Mais ni l'une ni l'autre de ces implications n'est systématique : un argument sémantique mais non syntaxique est appelé

[6]Le terme de *régime* semble plus adapté, mais *sous-catégorisation*, calque de l'américain *subcategorization*, est de très loin le plus employé dans la communauté.

argument interne, et un argument syntaxique mais pas sémantique est dit *argument surfacique* (par exemple le sujet des verbes impersonnels).

En laissant de côté les formalismes linguistiques majeurs (comme LFG ou HPSG), qui permettent évidemment la représentation d'informations syntaxiques (et parfois sémantiques), et en ne prenant pas en compte les représentations semi-formalisées (proches des dictionnaires grand public), on peut citer six cadres importants de représentation des informations lexicales syntaxiques et/ou sémantiques, que nous allons survoler successivement.

3.1 Le lexique-grammaire

Le lexique-grammaire est un paradigme de description et de représentation de données syntaxiques développé autour de Maurice Gross au LADL à partir de deux idées maîtresses. La première est que la séparation entre lexique et grammaire est arbitraire et néfaste. À ce titre, il est un précurseur des recherches ultérieures autour de la notion de lexicalisation des grammaires. La seconde est que la recherche de points communs entre entrées lexicales est souvent vaine, chaque entrée ayant des propriétés spécifiques. Le lexique-grammaire met en œuvre des idées remontant à la théorie des transformations de Harris. Un lexique-grammaire pour une langue donnée se présente sous la forme d'un recueil de *tables* regroupant un grand nombre d'entrées. Chaque table attribue à chacune d'entre elles un certain nombre de propriétés élémentaires parmi un ensemble de propriétés spécifique à la table. Ces propriétés élémentaires, qui ne concernent que les arguments (à l'exclusion des circonstants), sont définies dans un langage pseudo-formel. Les tables sont donc des tableaux à double entrée, avec les entrées lexicales en ordonnée (qui distinguent les différents sens d'un même lemme), et les propriétés élémentaires en abscisse. À l'intersection d'une ligne et d'une colonne, un symbole + indique que l'entrée a la propriété concernée, un symbole - indique qu'elle ne l'a pas, et un symbole ~ indique que l'information n'est pas encore renseignée. La systématicité de la méthode et des critères a pour but de permettre l'obtention de données identiques par différentes équipes ou individus, ce qui permet un développement coopératif des lexiques-grammaires.

Cette approche repose sur des fondements solides de linguistique descriptive. Un certain nombre de travaux cherchent à exploiter les tables développées à la main depuis de nombreuses années pour construire des analyseurs ou des lexiques syntaxiques qui allient couverture et précision. Toutefois, la façon dont l'information syntaxique est encodée dans les tables n'est pas facilement exploitable directement à ces fins. De plus, il n'y a pratiquement aucune prise en compte de la sémantique lexicale. Enfin, les tables du français ne sont pas (au moins aujourd'hui) toutes disponibles librement.

3.2 La lexicologie explicative et combinatoire

Le paradigme de la lexicologie explicative et combinatoire fait partie de la Théorie Sens-Texte (TST) d'Igor Melčuk. L'objectif de cette approche est une description formelle de la langue selon une approche sémantique. Le principe sous-jacent est qu'un dictionnaire doit fournir les informations permettant de construire tous les énoncés corrects exprimant toute pensée dans n'importe quel contexte : c'est une approche orientée génération. Ces informations, qui doivent être formalisées, doivent donc regrouper les facettes sémantiques, syntaxiques, collocationnelles et morphologiques de chaque entrée. En particulier, les liens entre entrées ont une importance capitale, et sont modélisés par des *fonctions lexicales*.

La richesse des informations prises en compte par la lexicologie explicative et combinatoire est un avantage mais également un inconvénient, car elle rend très laborieux le développement d'une ressource lexicale à grande échelle, et nécessite des formalismes très expressifs pour pouvoir être exploitée pleinement. De telles ressources et de tels formalismes sont en cours de développement, mais on est loin de pouvoir les utiliser de façon opérationnelle.

3.3 L'approche pronominale

L'approche pronominale (Blanche-Benveniste *et al.*, 1984; Eynde et Mertens, 2003) repose sur l'idée suivante : les constructions à compléments pronominaux (clitiques ou non, y compris les interrogatifs et d'autres types de pronoms) sont plus pertinentes que les compléments non-pronominaux pour l'identification et la description des arguments syntaxiques. En effet, l'utilisation de paradigmes pronominaux permet de distinguer des compléments nominaux en apparence similaires (prépositions identiques, formes identiques telles que le « il » personnel ou impersonnel,...) et d'identifier des restrictions de sélection (caractère animé ou non, relatif obligatoirement pluriel ou non,...). Cette approche, qui repose donc sur des critères précis, induit une typologie des compléments plus riche que la typologie habituelle (sujet, objet direct, objet datif, objet génitif, complément de quantité, locatif, délocatif, temps, manière, et complément prépositionnel à préposition fixe). Elle est complétée par des indications sur les reformulations possibles (passivation classique, passivation par *se*,...).

C'est une approche très intéressante, très pertinente d'un point de vue linguistique, mais qui n'a pas été utilisée (à notre connaissance) dans des analyseurs à grande échelle. Nous verrons à la fin de ce chapitre qu'elle est toutefois au cœur du lexique *Proton* des verbes français.

3.4 FrameNet

FrameNet est un projet conduit à l'université de Berkeley qui a pour but de décrire la combinatoire syntaxique et sémantique de tous les lemmes sous tous leurs sens, à l'aide de l'annotation assistée par l'ordinateur de corpus d'exemples. Le résultat est un recueil d'unités lexicales, qui sont des couples *(lemme,sens)*. Chaque sens d'un lemme polysémique constitue un *cadre sémantique* (*semantic frame*), qui est une structure conceptuelle décrivant un type particulier d'événement, de situation ou d'objet ainsi que les actants et les propriétés concernées. Un cadre sémantique peut être associé à plusieurs unités lexicales, qui *évoquent* ce cadre. Les actants ou propriétés sont décrites par des triplets *(élément de cadre, fonction grammaticale, type de constituant)*. Par exemple, un des actants de l'unité lexicale *eat* (*manger*) sera décrit par le triplet *(Nourriture, Objet direct, Syntagme nominal)*. Le développement se fait cadre par cadre, et non lemme par lemme. Enfin, les cadres sont placés aux sommets d'un graphe de relations. Pour le moment, seul l'anglais a été étudié en profondeur, bien que des projets pour d'autres langues existent, dont le français.

3.5 Le Lexique Génératif

Le lexique génératif est un paradigme développé par James Pustejovky (Pustejovsky, 1995). L'objectif est d'associer à chaque entrée lexicale une description sémantique structurée à partir de relations lexicales et de propriétés sémantiques. Chaque élément de la polysémie d'une entrée (chaque sens, donc) est défini par un quadruplet d'attributs : une structure argumentale, une structure événementielle, une structure de qualia, et une structure d'héritage. La structure argumentale attribue à chaque élément de la polysémie un concept un ou plusieurs concepts hypéronymiques (ainsi, les deux sens de *journal* seront associés l'un à *organisation* et l'autre à *informations.objet-physique*, où le point est un *et* logique). La structure événementielle indique les événements possibles associés à l'entrée lexicale (pour une maison, l'un d'entre eux peut être sa destruction). La structure de qualia indique successivement ce qui distingue cette entrée des autres (qualia formelle), ce qui relie l'objet représenté par l'entrée à ses constituants (qualia constitutive), ce à quoi sert généralement cet objet (qualia télique) et la façon dont il est créé (qualia agentive). Enfin, la structure d'héritage définit les mécanismes d'héritage entre entrées lexicales, qui permet à une entrée d'hériter l'un ou l'autre de ses qualias de ceux d'une autre entrée.

Ce modèle est très intéressant, en particulier par la structuration des informations sémantiques et l'utilisation de hiérarchies d'héritage multiple, mais souffre de trois principaux inconvénients :

– la complexité du développement d'un lexique génératif à grande échelle,

– le caractère relativement arbitraire de la liste de qualias (voire parfois de la séparation entre qualias) : rien ne prouve que ces quatre qualias sont suffisants,
– le fait que le lexique génératif se restreint à la seule sémantique lexicale, sans formaliser les aspects syntaxiques,
– le manque de robustesse du modèle, tant vis-à-vis des corpus que des linguistes, qui rend délicat d'envisager la mise en œuvre de campagnes d'évaluation.

3.6 WordNet

WordNet est une ressource lexicale pour l'anglais développée à l'université de Princetown (Miller *et al.*, 1990). Elle se présente sous la forme d'une collection de *synsets* (*synonym sets*), représentés par des ensembles de mots qui ont un sens en commun (relation de synonymie). C'est donc une ontologie. Les synsets sont reliés par des relations dont les principales sont :

– hypéronymie : relation entre un sens et un sens plus général (*véhicule* est un hypéronyme de *automobile*),
– hyponymie : relation inverse de l'hyperonymie,
– méronymie : relie un sens à ses parties constitutives (*moteur d'automobile* est un méronyme de *automobile*),
– holonymie : relation inverse de la méronymie.

Un des problèmes majeurs de WordNet et de ses avatars pour d'autres langues est la granularité et la couverture très variable d'un endroit à un autre de l'ontologie. Dans le WordNet anglais, les noms sont ainsi bien mieux organisés selon la relation d'hypéronymie que ne le sont les verbes ou, pire encore, les adjectifs et les adverbes. Même au sein des noms, certains synsets sont reliés à une longue chaîne de synsets de plus en plus généraux, d'autres sont en revanche très proches des synsets généraux. L'autre problème majeur est le caractère normatif de WordNet. Un nombre très important de travaux ont fait un usage intensif de WordNet, au point qu'on est en droit de se demander si ces travaux étudient véritablement les *sens* constitutifs d'une langue (typiquement l'anglais) ou les *synsets* constitutifs de WordNet. Enfin, des problèmes de droits font que très peu de travaux utilisent les versions de WordNet développées pour les langues d'Europe occidentale autres que l'anglais (français compris).

4 Acquisition d'informations lexicales

Le développement de ressources lexicales est une tâche de très grande ampleur, compte tenu de la double complexité de telles ressources :

– la richesse et la variabilité des informations à représenter,

– le nombre très important d'entrées (lexèmes, constructions, synsets,...) à décrire.

Pour cette raison, divers travaux se sont penchés sur la possibilité d'automatiser tout ou partie du processus de description de certaines informations lexicales. Bien que la majorité des travaux de ce type concernent l'acquisition automatique de terminologie (Daille, 2000) ou de collocations (Dunning, 1993), nous ne présenterons pas ces travaux ici dans la mesure où nous n'avons travaillé que de façon anecdotique sur ces questions. En revanche, deux types de travaux nous intéressent plus particulièrement : l'acquisition automatique de lexiques morphologiques et l'acquisition automatique de cadres de sous-catégorisation verbaux.

4.1 Acquisition automatique de lexiques morphologiques

Tout application réelle de traitement automatique rencontre le problème de la gestion des mots inconnus (c'est-à-dire des mots qui ne sont pas présents dans les ressources lexicales utilisées[7]). On peut envisager deux grandes familles de techniques pour pallier à de tels problèmes, qui ne sont pas exclusives l'une de l'autre[8] :

– extraire dynamiquement le maximum d'informations d'un mot inconnu trouvé dans une phrase donnée, en exploitant les caractéristiques du mot lui même et/ou le contexte au sein duquel il apparaît,
– extraire statiquement d'un corpus entier le maximum d'entrées lexicales manquantes correspondant à des mots inconnus, en mettant en œuvre des techniques globales tirant parti, entre autres, des indications de fréquence et de parenté morphologique entre mots attestés dans le corpus.

La première solution a l'avantage de la robustesse. Elle peut permettre, par exemple, la lemmatisation de mots inconnus en tirant parti d'informations morphosyntaxiques fournies par un étiqueteur probabiliste (on pourra se reporter à Erjavec et Džeroski (2004) pour une application au slovène). La lemmatisation est en effet un pré-traitement important pour des tâches telles que l'extraction d'informations ou l'acquisition de terminologie.

Cependant, cette solution a l'inconvénient majeur de ne pas tirer parti de la disponibilité d'un corpus complet et des informations fréquentielles correspondantes pour résoudre un certain nombre d'ambiguïtés. C'est la raison pour laquelle, sur des langues comme le français dont la morphologie est moins riche et plus ambiguë que celle du slovène, elle a été appliquée principalement pour l'analyse de dérivés morphologiques à partir d'un lexique conséquent

[7]Nous laissons de côté le problème des mots composés, dont l'acquisition automatique a fait également l'objet de nombreux travaux (voir par exemple, pour une comparaison de ces méthodes, Schone et Jurafsky (2001a).

[8]Nous laissons de côté les travaux d'acquisition automatique de la morphologie elle-même, c'est-à-dire des paradigmes, et d'acquisition de relations morphologiques non-décorées entre lexèmes (Schone et Jurafsky, 2001b). En effet, aucune de ces techniques n'est suffisante pour la constitution d'un lexique morphologique.

de lexèmes simples (Tanguy et Hathout, 2002), parfois même en s'aidant d'une ontologie du domaine (Grabar et Zweigenbaum, 1999).

L'apprentissage de lexèmes simples, et en particulier l'apprentissage *ex nihilo* de lexiques morphologiques, n'a été que peu étudié. Outre nos travaux, décrits au chapitre 4, on peut citer toutefois les travaux d'Oliver et ses collaborateurs ont mené des travaux comparables (Oliver *et al.*, 2003; Oliver et Tadić, 2004) pour acquérir ou enrichir des lexiques de verbes, noms et adjectifs pour le croate et le russe. Toutefois, les techniques utilisées par ces auteurs restent simples : pas de prise en compte des indications de fréquence, pas de modèle probabiliste pour la résolution des ambiguïtés de lemmatisation et pas de prise en compte de la morphologie dérivationnelle, pourtant responsable de nombreux mots inconnus. Les ambiguïtés de lemmatisation sont levées soit par une heuristique simple (comptage du pourcentage de formes dans le paradigme flexionnel qui sont attestées dans le corpus), soit à l'aide de requêtes sur Internet pour chercher celles des formes des différentes paradigmes possibles qui ne sont pas attestées dans le corpus. Cette dernière technique, qui met en œuvre le paradigme de l'Internet vu comme un corpus, est un frein au déploiement de la méthode à grande échelle, en raison du nombre très important de requêtes nécessaire.

4.2 Acquisition automatique de cadres de sous-catégorisation

Un certain nombre de travaux décrivent des techniques pour acquérir automatiquement des cadres de sous-catégorisation verbaux. Mis à part des travaux préliminaires menés par Brent (Brent, 1991), la première expérience à grande échelle d'acquisition de cadres de sous-catégorisation est celle décrite par Manning (1993) pour l'anglais. Ce dernier énumère 19 cadres possibles, dont il suppose qu'elles sont suffisantes. Il utilise un analyseur très simple qui lui permet d'attribuer à chaque occurrence d'un verbe l'un de ces 19 cadres. Enfin, il utilise un filtre statistique, qui repose sur la loi binomiale, pour déterminer celles des structures attribuées au moins une fois à un verbe qui sont effectivement pertinentes.

Si les résultats de cette expérience sont prometteurs, ils souffrent de la simplicité de l'analyseur utilisé et du très petit nombre de cadres considérées. Dans Briscoe et Carroll (1997), les auteurs utilisent précisément un analyseur plus sophistiqué ainsi qu'un nombre de cadres bien plus grand. Diverses améliorations du système ont été étudiées, telles que celles proposées dans Korhonen *et al.* (2000), y compris la prise en compte du résultat d'une phase préalable de désambiguïsation des sens lexicaux (*word sense disambiguation*), comme dans Korhonen et Preiss (2003).

Pour le français, les travaux les plus avancés sur ce sujet sont ceux menés autour de D. Bourigault (Bourigault et Frérot, 2005). Toutefois, ils ne concernent que les syntagmes nominaux prépositionnels sous-catégorisés par les verbes.

5 Ressources lexicales pour le français

5.1 Lexiques morphologiques

Au fil des années, un certain nombre de ressources lexicales morphologiques pour le français ont été développées, et en particulier celle développée dans le cas du projet MulText (Ide et Véronis, 1994), ou celle disponible auprès de l'ABU (ABU, 1999).

Plus récemment, l'ATILF (autour de S. Salmon-Alt) a développé un lexique ouvert des formes fléchies du français, Morphalou (Romary *et al.*, 2004). Les données initiales de Morphalou proviennent du TLFnome, la nomenclature du Trésor de la Langue Française (Dendien et Pierrel, 2003) qui a fourni environ 540 000 formes fléchies, appartenant à plus de 68 000 lemmes. Il est en accès libre à des fins de recherche et d'enseignement, bien que la licence ne soit pas. Le maintien et la mise à jour du lexique sont assurés par l'ATILF.

5.2 Lexiques syntaxiques et sémantiques

Contrairement à ce qui se passe pour les lexiques morphologiques, il n'existe pas, à l'heure actuelle, de lexique syntaxique (et encore moins sémantique) qui combine à la fois large couverture, bonne précision, et adéquation à l'utilisation dans des systèmes de traitement automatique. C'est l'objectif du projet LexSynt que de tenter de remédier à cette situation.

En particulier, il n'existe pas de ressource de taille significative pour le français qui mette en œuvre les modèles que sont le Lexique Génératif et FrameNet. Quant à WordNet, il en existe une réalisation pour le français, en tant que composante d'EuroWordNet (Vossen *et al.*, 1997), mais elle est de bien moins grande envergure que le WordNet pour l'anglais, et surtout elle n'est plus maintenue et n'est pas distribuée librement.

5.2.1 Ressources à large couverture mais pas directement utilisables pour le TAL

Les principales ressources syntaxiques qui associent large couverture et bonne précision sont les tables du lexique-grammaire et le Trésor de la Langue Française Informatisé (TLFi).

Le lexique-grammaire du français, initié au LADL par Maurice Gross au début des années 1970 et maintenant développé par l'IGM (Institut d'électronique et d'informatique Gaspard-Monge, Université de Marne-la-Vallée, sous la responsabilité d'Éric Laporte) constitue l'une

des plus importantes ressources lexicales avec de riches informations syntaxiques. Néanmoins ce lexique vient seulement d'être rendu public, qui plus est partiellement, et il n'a jamais été interfacé avec les grammaires développées depuis les années 1980 et notamment les grammaires basées sur l'unification.

Une autre ressource publique de grande ampleur est le Trésor de la Langue Française Informatisé (TLFi), développé par le laboratoire ATILF (Analyse et Traitement Informatique de la Langue Française, dir. Jean-Marie Pierrel). Ce dictionnaire, bien que très structuré, a été d'abord conçu comme un dictionnaire grand public (pour consultation humaine) et ne peut donc constituer en l'état une composante d'un modèle linguistique du français. Il est néanmoins possible d'en extraire automatiquement ou semi-automatiquement une grande quantité d'informations précieuses pour l'élaboration d'un modèle formel. Un projet de collaboration entre l'ATILF et LED (Langue et Dialogue, LORIA) sur ce thème est déjà en place.

5.2.2 Ressources à couverture réduite

Une autre ressource remarquable pour le français est le Dictionnaire Explicatif et Combinatoire initié par Igor Mel'čuk à la fin des années 1970 et maintenant développé sous format électronique à l'OLST (Observatoire de Linguistique Sens-Texte, Université de Montréal, sous la responsabilité d'Alain Polguère). Ce dictionnaire formalisé et conçu pour être intégré à un modèle complet de la langue (dans le cadre de la théorie Sens-Texte) sert de référence pour de nombreux travaux théoriques sur les lexiques syntaxiques et sémantiques. Sa couverture reste néanmoins partielle et son interfaçage avec une grammaire n'est pas achevé.

Les chercheurs travaillant sur l'approche pronominale ont par ailleurs développé à l'Université catholique de Leuven une ressource lexicale sur les verbes du français. Cette ressource, le lexique de valence Proton, est une ressource de grande valeur, que l'on peut interroger librement sur Internet, mais qui n'est pas distribuée en tant que telle.

5.2.3 Ressources imparfaites mais orientées TAL

Certains systèmes de traitement automatique utilisent également des ressources à large couverture, mais dont la richesse et la précision sont bien inférieures à celles des ressources comme les tables du lexique-grammaire ou le TLFi. Elles ont toutefois l'avantage d'être exploitables (et exploitées) dans des systèmes réels. Leur développement n'est pas le fait de lexicographes, mais plutôt celui de spécialistes du TAL, parfois secondés par des techniques d'acquisition automatique d'informations lexicales telles que celles évoquées au paragraphe précédent.

Outre le Le*fff* (Lexique des Formes Fléchies du Français) sur lequel nous reviendrons à plusieurs reprises au long de ce document, on peut citer les ressources utilisées pour et grâce

à l'analyseur Syntex de D. Bourigault. On notera également que des travaux en cours (autour de Claire Gardent) visent à extraire un lexique syntaxique orienté TAL à partir des tables du lexique-grammaire[9] (projet SynLex, Gardent *et al.* (2005)).

[9]Pour le moment, à partir de celles des tables qui sont disponibles.

Chapitre 3

Analyseurs

Ce chapitre est consacré aux techniques algorithmiques et opérationnelles permettant le développement d'analyseurs efficaces. Avant de poursuivre, rappelons une distinction fondamentale : un *constructeur d'analyseurs* est un système qui, à partir d'une grammaire dans un formalisme donné, construit un *analyseur*. Ce dernier est un système qui, à partir d'une phrase, construit la ou les analyses de cette phrase selon la grammaire dont il est issu — si elle(s) existe(nt). Autrement dit, les analyseurs qui reposent sur des grammaires ne sont pas développés manuellement, mais sont produits par des constructeurs d'analyseurs à partir de grammaires. À partir d'une même grammaire, deux constructeurs d'analyseurs différents peuvent donc induire des analyseurs d'efficacité très différentes. Inversement, un même constructeur d'analyseurs peut produire des analyseurs d'efficacité très différentes à partir de deux grammaires distinctes.

Nous ne décrirons dans ce chapitre que ce qui sera utile par la suite :

- deux algorithmes d'analyse :
  - l'algorithme d'Earley pour l'analyse non-contextuelle, et divers types d'optimisations,
  - l'algorithme de Pierre Boullier pour l'analyse RCG,
- deux constructeurs d'analyseurs :
  - le système SYNTAX, sur lequel nous illustrerons un certain nombre de techniques avancées d'implémentation,
  - le système DyALog, dont nous ne ferons qu'un aperçu car nous ne l'avons pas directement utilisé dans nos travaux, mais qui a servi a construire un analyseur, FRMG, dont nous étudions et utilisons les résultats sur divers corpus.

Ces algorithmes et ces constructeurs d'analyseurs ont un point commun qui sont au fondement de leur efficacité algorithmique et opérationnelle : l'analyse tabulaire, c'est-à-dire le partage de calcul et de structures.

Ce ne sont pas les seules approches suivies aujourd'hui pour le développement d'analyseurs syntaxiques. En effet, à côté ou en plus des techniques d'analyse tabulaire et de guidage,

certains auteurs mettent en œuvre des méthodes incrémentales (le plus souvent des cascades d'automates). Ces méthodes permettent une rapide analyse, mais avec quatre types de défauts : les analyses sont souvent surfaciques, elles sont souvent déterministes, l'absence d'interactions entre les différents automates limite la puissance d'expression de façon problématique tout en induisant parfois des problèmes de cohérence, et les descriptions de la langue qui sont utilisées sont souvent de peu d'intérêt d'un point de vue linguistique. Les méthodes incrémentales ne sont pas sans rappeler les techniques de guidage, mais ces dernières sont non-déterministes : elles préservent l'ambiguïté.

Nous ne parlerons pas non plus du traitement des décorations. Le cas des décorations fonctionnelles est traité au chapitre 9 sur l'analyseur SXLFG, et celui des décorations représentant un modèle probabiliste n'a pas été abordé suffisamment au cours de nos recherches[1] pour que nous jugions nécessaire d'en parler. C'est pourtant un domaine qui fait l'objet de nombreux travaux, et sur lequel nous comptons travailler dans un avenir très proche. Enfin, le cas du formalisme Méta-RCG (chapitre 10) est à part, puisque les décorations sont converties en contraintes purement syntaxiques : l'analyse Méta-RCG est une analyse RCG (voir ci-dessous) à l'aide d'une grammaire RCG générée automatiquement à partir de la grammaire Méta-RCG.

Bien qu'utilisé dans le système FRMG, nous n'étudions pas non plus l'analyse du formalisme TAG. Rappelons simplement ici qu'une TIG peut être convertie en CFG, et une TAG générale en RCG. On peut alors utiliser ces formalismes comme formalismes d'implémentation, en utilisant les analyseurs développés pour eux.

Enfin, nous n'abordons pas ici les techniques d'analyse utilisées pour des formalismes qui ne sont pas des formalismes de réécriture (Grammaires de Contraintes, Grammaires de Propriétés), puisque nous n'avons pas travaillé avec de tels formalismes.

1 Analyse non-contextuelle par l'algorithme d'Earley

L'algorithme d'Earley, qui porte le nom de son auteur, a été décrit pour la première fois par Earley (1970). Il se range, aux côtés des algorithmes CYK et GLR, parmi les algorithmes qui font usage de la notion de partage (de calculs et de structures) et qui construisent toutes les analyses possibles d'une phrase (et pas seulement une de ces analyses). Ce sont donc des algorithmes non-déterministes qui font usage d'idées venues du domaine de la programmation dynamique.

[1] Plus exactement, celles décrites dans ce document.

1.1 L'algorithme d'Earley

Considérons une grammaire non-contextuelle ainsi qu'une chaîne d'entrée de longueur n notée $a_1 \dots a_n$. L'analyse par l'algorithme d'Earley consiste à parcourir la chaîne linéairement, tant que c'est possible, et à faire toutes sortes de calculs entre chaque mot de cette chaîne. Il se déroule donc en $n+1$ étapes, numérotées de 0 (étape d'initialisation) à n (étape finale). L'analyse réussit si on a réussi à atteindre la n-ième étape, et si le résultat des calculs que l'on y fait est positif.

L'algorithme d'Earley fait usage de ce que l'on appelle des *items Earley*, ou plus simplement des *items*. Un item est entièrement défini par une production de la grammaire, un entier i, appelé *indice de début*, tel que $0 \leq i \leq n$, et un identifiant de position dans la partie droite de la règle, que l'on représente par un *point*. On représente un item sous la forme $[i]\ A \to \alpha \bullet \beta$, où $0 \leq i \leq n$. Un item privé de son indice de début (c'est-à-dire une production munie d'un identifiant de position dans sa partie droite) est appelé une *production pointée*. Une production pointée est donc de la forme $A \to \alpha \bullet \beta$.

Supposons que l'on soit à la j-ième étape (et donc à la position j de la chaîne d'entrée). Un item est dit *j-valide* (ou *valide* s'il n'y a pas d'ambiguïté) s'il représente une production dont on est en train d'essayer de faire dériver une sous-chaîne $a_i \dots a_k$ ($0 \leq i \leq j \leq k \leq n$) de la chaîne d'entrée. Puisque l'on a vu que l'on avance de gauche à droite progressivement, une telle production a sa partie droite partagée entre une partie déjà reconnue (la sous-chaîne $a_i \dots a_j$) puis une partie qu'il faut encore reconnaître (la sous-chaîne $a_{j+1} \dots a_k$). L'une ou l'autre de ces parties peut être vide, voir les deux si la partie droite est vide.

On définit la *table* $T[j]$ comme l'ensemble des items j-valides. Un item de la table $T[j]$ est donc valide s'il représente une situation[2] où l'on est à la j-ième étape (on a reconnu les j premiers symboles de la chaîne d'entrée, c'est-à-dire le *préfixe* $a_1 \dots\ a_j$), et où l'on est en train d'essayer de construire un non-terminal A couvrant une sous-chaîne commençant par a_{i+1} à l'aide de la règle $A \to \alpha\ \beta$. Le point symbolise donc la position courante dans la règle, qui correspond à la position courante dans la chaîne d'entrée, à savoir j. Si α est vide (l'item est de la forme $[i]\ A \to \bullet\ \beta$), c'est que l'on n'a encore rien reconnu : la reconnaissance de A à partir de la position i n'a pas encore commencé). Si β est vide (l'item est de la forme $[i]\ A \to \alpha\bullet$), c'est que la reconnaissance est un succès : on a réussi à construire un non-terminal A couvrant la sous-chaîne $a_i \dots a_j$.

Supposons que S soit l'axiome de la grammaire. Les items de la forme $[0]\ S \to \bullet\ \alpha$, où $S \to \alpha$ est une production, sont valides, et forment la table $T[0]$: ils représentent la situation (étape 0) où l'on n'a encore rien reconnu, mais où l'on cherche à reconnaître l'axiome à partir

[2] En toute rigueur, un tel item est valide si on peut dériver une proto-phrase $a_1 \dots a_i\ A\ \gamma$ et une proto-phrase $a_1 \dots a_i\ a_{i+1} \dots a_j\ \beta\ \gamma$.

du début de la chaîne d'entrée. Une analyse Earley réussit si on réussit à construire, à partir de ces items de départ, un item n-valide de la forme $[0]S \rightarrow \alpha\bullet$, où $S \rightarrow \alpha$ est une production.

Le but de l'algorithme est donc de calculer en $n+1$ étapes tous les items valides et seulement les items valides. La j-ième étape va donc être chargée de construire la table $T[j]$ en construisant tous les items j-valides, et seulement ceux-ci.

Chacune des étapes exécute trois types d'opérations, nommées LECTURE, PRÉDICTION et COMPLÉTION (en anglais, respectivement SCANNER, PREDICTOR, COMPLETOR). À l'exception de l'étape 0, qui commence par la construction des items de départ comme dit ci-dessus (construction de la table $T[0]$), la j-ième étape commence par les opérations de LECTURE. Le but de ces opérations est de faire avancer les points dans les items de la table précédente où le point est juste avant un terminal qui est précisément a_j. Autrement dit, pour tout item de $T[j-1]$ de la forme $[i]\ A \rightarrow \alpha \bullet a_j\ \beta$ on ajoute dans $T[j]$ l'item $[i]\ A \rightarrow \alpha\ a_j \bullet \beta$. S'il n'y a pas d'item de cette forme, l'analyse échoue. Les items créés par cette phase de LECTURE sont appelés *items noyau*. La construction du non-terminal A correspondant a donc avancé d'un terminal. Dans une règle donnée, cette avancée peut avoir trois conséquences :

1. Soit β commence par un terminal, et la phase LECTURE de l'étape $j+1$ s'en occupera plus tard. On ne fait donc rien pour le moment.

2. Soit β commence par un non terminal B. Dans ce cas, il faut lancer (« prédire ») la reconnaissance du non-terminal B à partir de la position courante j. On rajoute donc dans $T[j]$, pour toutes les règles $B \rightarrow \delta$ ayant B en partie gauche, un item $[j]\ B \rightarrow \bullet\ \delta$. L'ajout d'un tel item est une opération dite de PRÉDICTION.

3. Soit β est vide. On a donc terminé et réussi la reconnaissance de A entre les positions i et j. On va donc chercher tous les items qui avaient lancé la reconnaissance de A à partir de la position i, c'est-à-dire les items de la table $T[i]$ de la forme $[k]\ A \rightarrow \delta \bullet\ A\ \gamma$, et on va ajouter dans $T[j]$ pour chacun d'entre eux l'item $[k]\ A \rightarrow \delta\ A\ \bullet \gamma$ qui prend acte de ce que l'on a reconnu A entre i et j. L'ajout d'un tel item est une opération dite de COMPLÉTION.

L'exécution d'une PRÉDICTION ou d'une COMPLÉTION ajoute un item dans la table courante (la table $T[j]$). Ce nouvel item peut lui-même permettre de nouvelles PRÉDICTIONS et de nouvelles COMPLÉTIONS. Celles-ci sont exécutées, et ainsi de suite jusqu'à ce que la table courante soit stable (et donc complète). Toutes ces opérations peuvent avoir pour effet de faire avancer le point dans des règles, si les non-terminaux que l'on franchit peuvent dériver dans le vide. Mais elles ne font pas avancer le point dans la chaîne d'entrée : on reste à la position j, et on ne remplit que la table $T[j]$.

On peut se convaincre facilement (quitte à lire une preuve dans la littérature) que l'on ne construit ainsi que des items valides destinés à la table $T[j]$, mais qu'on les construit tous.

Une fois que l'on ne peut plus construire de nouvel item pour la table $T[j]$, on passe à l'étape $j+1$. L'analyse réussit si on arrive de cette façon à exécuter les n étapes et si on obtient une table $T[n]$ comprenant des items de la forme $[0]S \rightarrow \alpha\bullet$, où $S \rightarrow \alpha$ est une production.

1.2 Optimisations de l'algorithme d'Earley

L'algorithme d'Earley peut être optimisé de différentes façons. Outre le guidage, qui fait l'objet de la section suivante, l'implémentation de Pierre Boullier dans le système SYNTAX fait appel aux deux optimisations suivantes[3], que nous détaillons ci-dessous :

- les productions pointées des items qui ne diffèrent que par leur indice de début sont partagées ;
- toutes les opérations de PRÉDICTION sont faites statiquement (une fois pour toutes au moment de la compilation de la grammaire), ainsi que toutes les opérations de COMPLÉTION qui peuvent l'être (ce qui est le cas lorsque des non-terminaux dérivent dans la chaîne vide).

La première optimisation permet d'éviter de stocker plusieurs fois p items ne différant que par leur indice de début, $i_1, \ldots, i_p$. Les items sont donc généralisés en sorte que l'indice de début puisse être un ensemble. Un item généralisé est alors de la forme $[i_1, \ldots, i_p]\ A \rightarrow \alpha \bullet \beta\ [j]$. Rajouter un item dans une table consiste donc soit, si l'item généralisé correspondant existe déjà, à rajouter un élément à son ensemble d'indices de début s'il n'y est pas déjà, soit, dans le cas contraire, à le créer effectivement.

La seconde optimisation consiste donc à préparer à l'avance, au moment de la construction de l'analyseur (on parle de calcul *statique*), un ensemble de structures contenant pour chaque non-terminal A l'ensemble des productions pointées auxquelles on peut arriver à partir des A-productions et en faisant uniquement des PRÉDICTIONS et des COMPLÉTIONS[4] (et donc en ne changeant ni de position dans la chaîne d'entrée ni de table). Après la phase de LECTURE qui crée des items noyau, l'étape j consiste en les opérations suivantes. Pour tous les items de la table courante $T[j]$ où le point est devant un non-terminal, on rajoute dans $T[j]$ l'ensemble des productions pointées accessibles associées à ce terminal, en les complétant par l'indice de début approprié pour en faire de véritables items, et sauf si ces items sont déjà dans $T[j]$ (on appelle cette opération la PRÉDICTION-COMPLÉTION SEMI-STATIQUE). On crée alors de

[3] D'autres optimisations, comme le partage de parties communes d'items, ont été implémentées également, mais il s'est avéré que leur coût à l'exécution était supérieur au bénéfice qu'elles apportaient. Elles ne sont donc pas utilisées en pratique.

[4] Ces COMPLÉTIONS statiques ne peuvent avoir lieu que sur des items qui font dériver la partie gauche de leur production pointée dans le vide.

nouveaux items noyau à partir des items noyau déjà obtenus en faisant franchir au point le non-terminal A devant lequel il se trouve si A dérive dans la chaîne vide (c'est-à-dire si la PRÉDICTION-COMPLÉTION SEMI-STATIQUE a ajouté dans $T[j]$ au moins un item de la forme $[j]A \rightarrow \alpha \bullet$). Puis on effectue les COMPLÉTIONS possibles, et on recommence jusqu'à ce que la table courante soit stable (et donc complète).

1.3 Guidage

L'idée générale mise en œuvre dans les techniques de guidage est la suivante. Dans un processus non-déterministe, certaines étapes, qui sont la source du non-déterminisme, consistent en l'exploration de plusieurs possibilités. Par exemple, l'étape de PRÉDICTION de l'algorithme d'Earley est non-déterministe, puisque l'on peut, à partir d'un seul item, créer plusieurs items mutuellement exclusifs (ils ne pourront servir dans la même analyse).

Parmi ces possibilités que l'on explore, certaines conduisent à un échec. On peut donc souhaiter disposer d'un mécanisme qui permette d'éviter au maximum l'exploration de telles possibilités. On appelle *oracle* un mécanisme indiquant toutes les possibilités conduisant à un succès et seulement celles-ci, et *guide* un mécanisme indiquant toutes les possibilités conduisant à un succès, mais également des possibilités qui conduisent à un échec. Un *oracle* est donc un *guide parfait* (Boullier, 2003b).

Dans le cas d'un analyseur, cette idée se met en œuvre de la façon suivante. Tout d'abord, dans une première phase de l'analyse, un guide est construit par un *analyseur guidant*. Ensuite, dans une seconde phase, un *analyseur guidé* fait l'analyse proprement dite, en se servant du guide construit à l'étape précédente.

Pour guider un analyseur Earley, on peut donc construire un guide qui spécifie, au moment où l'on veut faire une PRÉDICTION, quelles sont les possibilités (c'est-à-dire les items) qui valent le coup d'être essayés, afin de ne pas s'embarrasser d'items qui ne serviront pas à construire une analyse complète de la chaîne d'entrée.

On peut concevoir différents types de guides. Boullier (2003b) en compare trois, qui sont implémentés dans SYNTAX. Le premier type de guide consiste à ne construire d'items qu'à partir de productions dont tous les terminaux de partie droite sont présents dans la chaîne d'entrée[5]. Le second type de guide est similaire au premier, sauf que l'on se restreint aux terminaux présents dans la chaîne d'entrée plus à droite que la position courante. Le troisième type de guide repose sur un sur-langage régulier du langage non-contextuel : on ne construit un item que s'il correspond à une analyse de la chaîne d'entrée par un automate fini (coin-gauche) reconnaissant ce sur-langage régulier. On notera que l'analyseur guidant produit par ce dernier type

[5]En réalité, on peut être plus fin que cela à propos des productions ne comportant pas de terminal en partie droite, moyennant des calculs statiques appropriés.

de guide peut être vu comme un super-étiqueteur non-déterministe non-probabiliste construit automatiquement à partir de la grammaire (Boullier, 2003c).

Les expériences décrites par Boullier (2003b) montrent l'efficacité de ces techniques de guidage pour l'analyse Earley. Sur une grammaire non-contextuelle formant un sur-langage de la grammaire TAG de l'anglais développée pour le système XTAG, les trois types de guides donnent des résultats comparables : il y a équilibre entre la précision du guide (la proportion de possibilités conduisant à un succès par rapport aux possibilités proposées) et le temps passé à le construire et à l'appliquer. Le guide le plus rapide mais le moins précis (39%) est le guide du premier type. Le guide le plus précis (78%) mais le moins rapide est le guide régulier[6]. Les performances de l'analyseur complet sont fortement améliorées grâce à ces guides.

1.4 Extensions

Nous reviendrons au chapitre 9 sur deux extensions apportées au cours de ce travail de thèse à l'implémentation de l'algorithme Earley dans SYNTAX. Il s'agit en particulier de la possibilité de prendre non pas une chaîne (de symboles) mais un DAG (de symboles) en entrée, ainsi que de l'application dans le cas non-déterministe de techniques de rattrapage d'erreurs.

2 Analyse RCG

Nous rappelons dans cette section l'algorithme d'analyse RCG proposé en détails dans Boullier (2004). L'idée générale est en réalité très simple : on construit une fonction qui à un prédicat instancié associe un booléen indiquant si ce prédicat est vrai (il peut se réécrire comme la liste vide de prédicats instancié) ou non. Pour cela, il construit toutes les clauses instanciées possibles ayant pour partie gauche son argument, et pour chacune d'entre elles il fait un *et* sur les résultats des appels correspondant aux prédicats instanciées de partie droite, et fait un *ou* entre tous ces résultats. La procédure d'analyse suit donc exactement la grammaire. Naturellement, les résultats de cette fonction sont mémorisés, en sorte que l'on ne refait jamais deux fois le même calcul.

Une description en pseudo-code en est donnée dans l'algorithme 1 pour les RCG positives. Le cas des RCG négatives s'en déduit trivialement (aux problèmes de cycles près) en remplaçant pour les appels négatifs la ligne $résultat-local=appel-predicat$(A_i(\vec{\rho_i}))$ par la ligne $résultat-local= !appel-predicat$(A_i(\vec{\rho_i}))$.

[6]Naturellement, un oracle aurait une précision de 100%. À titre de comparaison, la phase de PRÉDICTION sans guide a une précision de 8%).

La complexité de cet algorithme réside en grande partie dans les deux boucles **Pour**, qui parcourent un ensemble important de possibilités. C'est la raison pour laquelle on peut mettre en place un mécanisme de guidage, comme cela a été présenté précédemment pour l'analyse non-contextuelle. C'est ce qui est présenté dans Barthélemy *et al.* (2001). Mais des techniques plus directes permettent également de réduire l'espace de recherche. En particulier, on peut par des calculs statiques déterminer dans certains cas des longueurs minimales et/ou maximales aux instanciations possibles de certains arguments de certains prédicats. C'est ce qui est fait dans l'analyseur RCG du système SYNTAX.

[On suppose que l'axiome est S, et que la chaîne d'entrée est de longueur n. La fonction appel-prédicat essaye de reconnaître le prédicat instancié donné en argument.]
[Une liste d'intervalles (liste de couples d'entiers) est noté sous la forme d'un vecteur $\vec{\rho}$]

Appeler appel-predicat($S(0..n)$)
Fonction appel-predicat($A_0(\vec{\rho}_0)$: **prédicat instancié**) : **booléen**
  Si (Si \$mémo($(A_0(\vec{\rho}_0)$ est déjà positionné) **Alors**
    Rendre sa valeur
  Fin Si
  \$mémo($(A_0(\vec{\rho}_0) = 0$ *[pour détecter les récursions]*
  \$résultat= **faux**
  Pour chaque A_0-clause **faire**
    Pour chaque instanciation possible $A_1(\vec{\rho}_1) \ldots A_m(\vec{\rho}_m)$ de cette clause **faire**
      \$résultat-local= **vrai** *[pour le cas où la partie droite est vide]*
      Pour i **de** 1 **à** m **faire**
        \$résultat-local=appel-predicat($A_i(\vec{\rho}_i)$)
        Si (\$résultat-local= **faux**) **Alors**
          Break
        Fin Si
      Fin Pour
      \$résultat=\$résultat **OU** \$résultat-local
    Fin Pour
  Fin Pour
  Stocker \$résultat dans \$mémo($A_0(\vec{\rho}_0)$)
  Rendre \$résultat
Fin

Algorithme 1: Algorithme d'analyse RCG.

3 Construction d'analyseurs efficaces : le système SYNTAX

3.1 Le système SYNTAX

Le système SYNTAX, développé par Pierre Boullier (avec la participation, entre autres, de Philippe Deschamp), est un environnement complet pour la génération d'analyseurs syntaxiques. Historiquement, SYNTAX a d'abord eu pour principale application la construction d'analyseurs pour les langages de programmation, dans le domaine de la compilation (C, Ada, etc...). C'était donc principalement un générateur d'analyseurs non-contextuels déterministes, à l'instar des programmes unix `lex` et `yacc`. La force de SYNTAX par rapport à ces programmes était alors triple :

- une plus grande variété et une plus grande puissance d'expression dans les mécanismes sémantiques utilisables pendant ou après l'analyse,
- des mécanismes performants de rattrapage d'erreurs, (Boullier et Jourdan, 1987),
- la génération d'analyseurs compacts et performants (plus performants que `yacc`).

C'est la version 3.9 de SYNTAX qui est l'aboutissement des recherches dans le domaine de l'analyse déterministe. Il existe un manuel d'utilisation et de mise en œuvre de la version 3.5 de SYNTAX (Boullier et Deschamp, 2004).

Depuis une dizaine d'années, SYNTAX s'est tourné vers des applications de traitement des langues. Ceci a conduit à quatre types de développements :

- la possibilité de traiter (de façon déterministe) une plus grande classe de langages non-contextuels, et en particulier les langages RLR[7] ; ceci a donné lieu à la version 4 de SYNTAX ;
- l'analyse non-contextuelle non-déterministe, avec des générateurs d'analyseurs à la Earley et à la GLR ; ceci a donné lieu à la version 5 de SYNTAX ;
- l'analyse contextuelle (non-déterministe), avec un générateur d'analyseurs RCG ;
- l'analyse guidée (pour les grammaires non-contextuelles comme pour les RCG) ; ce point et le précédent ont donné lieu à la version 6 de SYNTAX.

Les versions 4 à 6 ne sont pas distribuées actuellement[8], alors que la version 3 était disponible gratuitement à des fins de recherche. SYNTAX est développé en C.

SYNTAX ne produit pas directement des analyseurs à partir des grammaires. Il produit un ensemble de données en langage C qui doivent être compilées et liées (*linked*) avec différents

[7] Un langage non-contextuel est dit RLR si, pour tout conflit dans l'automate LR(0), on peut trouver une grammaire régulière avec laquelle on peut analyser ce qui reste de la chaîne d'entrée jusqu'à atteindre un point où l'on peut décider comment résoudre le conflit (c'est donc une forme de *lookahead* non-borné).

[8] La raison invoquée est que SYNTAX est en perpétuel développement, et que par conséquent il est très difficile de stabiliser une version. Mais toute personne désirant utiliser SYNTAX peut contacter Pierre Boullier.

modules généraux. Ces données sont des *tables*, c'est-à-dire des tableaux C initialisés, et des fonctions.

Le cœur de SYNTAX est constitué principalement de trois types de fichiers :

- un certain nombre de constructeurs qui construisent les tables et les fonctions spécifiques à un analyseur à partir de la grammaire, et qui définissent entre autres la grammaire elle-même, les données pour la construction du lexeur, celles pour la construction de l'analyseur syntaxique et celles pour le traitement des erreurs ;
- les modules contenant les fonctions génériques qui forment le cœur de chaque analyseur,
- un nombre important de modules utilitaires, qui sont une des causes de l'efficacité de SYNTAX, et qui permettent une gestion très performantes des chaînes de caractères, des tables de hachage, des tableaux associatifs, des vecteurs de bits et des ensembles, etc...

Enfin, SYNTAX est *bootstrappé* : la compilation des constructeurs se fait à l'aide de SYNTAX lui-même (et donc desdits constructeurs).

3.2 Principes pour l'analyse efficace

Le développement de constructeurs d'analyseurs syntaxiques nécessite une forte compétence en algorithmique, afin de contrôler au mieux la complexité en temps (et en espace) des analyseurs que l'on produit. Mais il est illusoire de penser que c'est suffisant. Un même algorithme (guidage et autres optimisations algorithmiques compris), selon la façon dont il est implémenté, pourra induire des parseurs aux performances variant de plusieurs ordres de grandeur[9].

Ce qui fait la force de SYNTAX est en partie dans cette composante que l'on pourrait qualifier de technologique. Elle se traduit par l'optimisation des structures et des calculs selon trois grands principes :

- utiliser des représentations compactes des données,
- calculer un maximum de choses une fois pour toutes,
- retarder au maximum l'évaluation.

4 Aperçu du système DyALog

Développé à l'origine pour apporter à la Programmation Logique la notion de partage de calculs par l'intégration bas-niveau des techniques de tabulation utilisées en analyse tabulaire,

[9] L'algorithme d'analyse pour les RCG a été implémenté par d'autres chercheurs que Pierre Boullier, donnant lieu à des temps d'analyse environ 1000 fois supérieurs (Boullier, communication personnelle).

le système DyALog (Villemonte de la Clergerie, 2005) a évolué vers la création d'analyseurs efficaces pour des formalismes à décorations.

Tout en préservant la puissance d'expression de la Programmation Logique, DyALog a été étendu pour faciliter le développement de grammaires pour divers formalismes, ainsi que la mise au point des analyseurs correspondant en vue d'augmenter leur efficacité. En particulier, DyALog remplace le mécanisme de copie de structures de la plupart des implémentations et extensions de Prolog par un mécanisme original de partage de structures. Le mécanisme d'indexation des termes logiques a été également étendu au-delà de la technique usuelle qui repose uniquement sur le nom du prédicat et la nature de son premier argument.

DyALog couvre de nombreux formalismes utilisés pour la modélisation et l'analyse des langues, comme les DCG, les BMG (Grammaires à Mouvement Borné, *Bound Movement Grammars*), TIG et TAG lexicalisées ou non, avec traits ou non, RCG positives avec traits ou non.

De nombreux opérateurs permettent de plus l'utilisation de grammaires factorisées, ce qui est inutile en particulier lorsque l'on construit automatiquement une grammaire à partir d'une description plus abstraite (méta-grammaire) qui porte en elle un grand nombre de factorisations.

DyALog permet l'utilisation de différentes stratégies d'analyse pour construire la forêt partagée de dérivation. Enfin, les analyseurs construits avec DyALog sont des analyseurs tout à fait efficaces qui reposent sur divers types d'automates à piles.

Deuxième partie

Structuration, représentation et acquisition des informations lexicales

Chapitre 4

Représentation de la morphologie flexionnelle et dérivationnelle

Comme nous l'avons vu au chapitre 2, les lexiques sont au cœur de l'interface entre textes et grammaires linguistiques. Il est donc indispensable de disposer de lexiques regroupant des informations lexicales aussi riches et nombreuses que possibles, tout en recherchant un équilibre entre facilité d'exploitation et facilité de développement. Pour atteindre ces objectifs, le développement de lexiques doit passer par l'étude de deux problématiques majeures :

– comment représenter les informations lexicales ?

– comment acquérir ces informations ?

Ce chapitre et le suivant traitent de cette première question, les deux chapitres suivants traitent de la seconde. À chaque fois, nous nous intéresserons d'abord aux informations morphologiques puis aux informations syntaxiques et sémantiques.

Dans ce chapitre, nous nous intéressons donc à la représentation des informations morphologiques. Ces informations sont très redondantes, et pour cette raison il est approprié de les factoriser sous la forme de *paradigmes*, au sens du chapitre 2. Ces paradigmes eux-mêmes, et c'est le sens de la démarche adoptée dans ce chapitre, peuvent être représentés de façon factorisée. Nous avons pour cela développé un formalisme de description de la morphologie, appelé MetaMorpho, qui permet la compilation d'une description morphologique d'une langue en des outils de flexion, de lemmatisation ambiguë et de génération de lemmes dérivés potentiels. Nous l'avons utilisé pour le français et pour le slovaque[1] (voir figure 1), à la fois pour la compilation en lexiques extensionnels de lexiques intensionnels à large couverture (en particulier notre

[1]L'intérêt du slovaque pour le développement d'un formalisme de description morphologique est qu'il s'agit d'une langue à la morphologie très riche, contrairement au français. En particulier, cette langue utilise de nombreux paradigmes de déclinaison, dont certains sont des variantes les uns des autres, là où le français ne fait plus usage des cas.

lexique du français, le Le*fff*) et pour l'acquisition automatique de lexiques morphologiques à partir de corpus bruts.

1 Pertinence, réversibilité, compacité et lisibilité

Le paradigme classique pour représenter un modèle de la morphologie d'une langue est le paradigme de la morphologie à deux niveaux (cf. chapitre 2). Toutefois les règles à deux niveaux sont fondamentalement unidirectionnelles. De plus, le développement de descriptions morphologiques dans ce formalisme est délicat, car le non-ordonnancement des règles induit des conflits entre règles qui sont difficiles à éviter, même s'ils sont détectables automatiquement (comme le fait le compilateur développé chez Xerox). À l'inverse, ranger les règles dans un certain ordre ne semble pas poser de problème intellectuel particulier, et semble en réalité une solution bien naturelle. C'est par exemple la solution la plus simple pour exprimer une règle générale assortie d'exceptions : on introduit en premier les cas particuliers, que l'on fait suivre d'un cas par défaut. Naturellement, aucune des deux solutions (avec ou sans ordonnancement) n'a un pouvoir expressif plus grand que l'autre : les deux points de vue sont formellement équivalents puisqu'il ne s'agit que de deux façons différentes de décomposer une opération régulière complexe en opérations régulières plus simples.

De façon plus générale, nous adoptons le point de vue selon lequel la description morphologique d'une langue doit présenter *a priori* un certain nombre de propriétés, qui sont la pertinence linguistique, la réversibilité, la compacité et la lisibilité. Examinons successivement ces propriétés.

Toute modélisation de la morphologie d'une langue devrait être pertinente linguistiquement, c'est-à-dire manipuler des objets ayant un sens linguistique. En effet la description de la morphologie d'une langue est un travail linguistique pour lequel il est peu satisfaisant, voire assez handicapant, d'avoir à manipuler des concepts totalement artificiels (ainsi le « `0` » de la morphologie à deux niveaux) ou des radicaux qui n'ont aucun sens ni en synchronie ni en diachronie. A l'inverse, les travaux (linguistiques) en morphologie (tels que Goffic (1997)) permettent d'asseoir des regroupements de paradigmes différents en apparence, mais identiques pour peu que l'on prenne en considération des transformations phonétiques diachroniques ou synchroniques. Pour prendre un exemple, nous verrons dans notre description de la morphologie du français qu'il est possible de n'attribuer qu'un seul radical au verbe *peindre* et de lui faire partager un même paradigme avec tous les verbes en *-endre*, *-battre*, *rompre*, *-eindre*, *-oindre*, *-aindre*, *-vaincre*, *-prendre*, *-mettre*, et certains verbes en *-seoir*. De plus, et contrairement à nombre de ressources morphologiques (cf. cependant des travaux tels que ceux de F. Namer cités au chapitre 2), une description morphologique ne devrait pas se cantonner à la description de la

La langue slovaque est une langue slave, et par conséquent indo-européenne, qui est la langue officielle de la République slovaque et qui est parlée par 5,5 à 6 millions de personnes, principalement dans ce pays. La langue la plus proche du slovaque est le tchèque. Ces deux langues ont coexisté pendant une longue période au sein de ce qui était la Tchécoslovaquie. Pour cette raison, mais aussi à cause de la proximité des deux langues, la plupart des Slovaques comprennent le tchèque, et les personnes désireuses d'apprendre « la » langue parlée en Tchécoslovaquie apprenaient le tchèque. De plus, le tchèque était souvent *de facto* la langue des élites praguoises, en particulier par son passé universitaire, littéraire et scientifique bien plus fourni que celui du slovaque. Une des conséquences de cet état de fait est que le slovaque est sous-représenté parmi les ouvrages et les études linguistiques, et en particulier parmi les manuels d'apprentissage des langues. À titre d'illustration, le slovaque est à notre connaissance la seule langue officielle d'un pays européen pour laquelle il n'existe aucune méthode d'apprentissage en français. Elle est pourtant parlée par bien plus de personnes que des langues comme le slovène, les langues baltes, le maltais, ou l'islandais. De plus, elle a reçu moins d'attention de la part des linguistes computationnels que les autres langues slaves telles que le tchèque ou le russe. Le seul projet d'importance concernant le slovaque dans notre domaine est le Corpus National Slovaque (Jazykovedný ústav Ľ. Štúra SAV, 2004), qui est une ressource de valeur. Toutefois, parce que nous pensons qu'il n'est pas nécessairement satisfaisant de ne disposer que d'une seule ressource linguistique pour une langue donnée, nous avons choisi de développer nos propres ressources indépendamment de ce Corpus. Cependant, nous avons l'intention de comparer nos propres résultats à cette ressource dans un avenir proche.
Le slovaque s'écrit à l'aide de l'alphabet latin supplémenté par quelques signes diacritiques. En particulier, l'accent aigu allonge une voyelle ou une liquide qui sert de tête de syllabe, et le symbole de mouillure ˇ transforme certaines consonnes dures en leur correspondant « mouillé » (ainsi, *ň* correspond au *gn* français). À quelques détails près, l'orthographe slovaque respecte systématiquement la prononciation : il y a quasiment correspondance biunivoque entre phonèmes et graphèmes (composés de un caractère, parfois deux).
Comme la plupart des autres langues slaves, et contrairement à l'anglais ou au français, le slovaque est une langue flexionnelle à morphologie casuelle. Ceci signifie que les noms et les adjectifs (entre autres) sont fléchis non seulement en fonction de leur genre et de leur nombre, mais aussi selon leur fonction grammaticale ou selon la préposition qui les gouvernent : c'est la notion de *cas*. La flexion d'un lemme variant selon son cas est appelée *déclinaison*. La plupart des formes fléchies d'un lemme sont obtenues par ajout d'un suffixe sur son radical selon sa classe morphologique, mais le radical lui-même peut être modifié. Ceci a lieu en particulier au génitif pluriel de certains noms féminins et neutres. Ainsi, *žena* (« femme »), dont le radical est *žen-*, a pour génitif pluriel la forme *žien*.

FIG. 1 – Quelques généralités sur la langue slovaque.

morphologie flexionnelle, en laissant de côté la morphologie dérivationnelle. Cette dernière permet en effet de nombreuses généralisations, tant au niveau morphologique qu'aux niveaux syntaxique et sémantique[2].

Un autre impératif est celui de la réversibilité théorique et pratique. À ce titre, la morphologie à deux niveaux est exemplaire sur le plan pratique (puisqu'elle se compile en transducteurs qui peuvent être utilisés dans les deux directions), mais sur le plan théorique la situation est moins claire, en particulier en termes de réversibilité de la description (comme évoqué plus haut). Il est pourtant indispensable de disposer d'une description morphologique permettant à la fois :

- la génération de lexiques, par exemple pour la transformation d'un lexique décrit de façon intensionnelle en lexique extensionnel, ou encore pour les tâches de génération automatique de textes,
- la lemmatisation ambiguë, par exemple pour l'apprentissage automatique de lexique ou la lemmatisation de mots inconnus.

Enfin, une représentation écrite par un linguiste se doit d'être à la fois compacte et lisible. La compacité est une nécessité pratique, mais également un objectif théorique : une représentation est d'autant plus explicatoire qu'elle abstrait les différences de surface, conduisant à une plus grande compacité du modèle. Toutefois, cette compacité ne doit pas être obtenue au détriment de la lisibilité. Comme toute ressource linguistique, une description morphologique doit pouvoir être amendée, corrigée ou complétée par d'autres linguistes que celui qui en est l'auteur initial (ou l'auteur du formalisme dans lequel elle est décrite).

C'est en respectant au mieux l'ensemble de ces principes que nous avons développé et utilisé le formalisme de description de la morphologie flexionnelle et dérivationnelle MetaMorpho, décrit dans la suite de ce chapitre et utilisé dans les suivants.

2 Tokens, formes, lemmes

Avant de poursuivre, il nous faut définir la signification que nous donnons à un certain nombre de termes, en reprenant certaines définitions données au chapitre 2. Faute d'un équivalent français adéquat, nous appelons *token* une suite de caractères issue d'un texte et pouvant éventuellement comporter des espaces. Les tokens sont par conséquent les observables du traitement automatique de textes.

[2]Ce qui ne veut pas dire que toutes les généralisations possibles doivent être faites. Il y a là un compromis à trouver entre complexité et compacité. Et dans tous les cas il faut veiller à ce que ces généralisations n'induisent aucune sur-génération.

Nous appelons *forme* une unité linguistique atomique, c'est-à-dire une unité qui est élémentaire du point de vue syntaxique (elle est syntaxiquement inanalysable, même si elle peut l'être d'un autre point de vue, en particulier du point de vue morphologique). Une forme peut être *décorée*, *semi-décorée* ou *non décorée*. Une *forme non décorée* (ou simplement *forme*, par abus de langage) est une suite de caractères. Une forme semi-décorée est une forme non décorée à laquelle on a associé un certain nombre d'informations, dites *informations morphologiques* et regroupées en une *étiquette morphosyntaxique*. C'est donc un couple *(forme non-décorée, étiquette morphosyntaxique)*. Cette étiquette est le plus souvent donnée sous la forme d'une structure de traits, comportant des attributs tels que la catégorie, le genre, le nombre, le cas, la personne, le temps, le mode, l'aspect, et d'autres. Nous verrons plus bas ce que nous appelons *forme décorée*.

Il n'y a pas d'égalité directe entre tokens et formes, même si c'est souvent le cas. En effet :

- il peut y avoir du « bruit » dans ce signal qu'est la suite de tokens :
  - un token peut représenter une forme de façon imparfaite (fautes d'orthographes),
  - un token peut représenter l'agglutination par erreur de plusieurs formes (*untoken* en lieu et place de *un token*),
  - plusieurs tokens peuvent représenter une seule forme scindée par erreur (*to ken* en lieu et place de *token*),
  - ces trois phénomènes peuvent se combiner ;
- même un signal « non bruité » n'induit pas une correspondance directe :
  - un token peut désigner plusieurs formes : c'est le cas des agglutinées (*aux* désigne la suite de formes *à le*$_{det}$, *donne-m'en* désigne *donne -m' en*),
  - plusieurs tokens peuvent désigner une seule forme : c'est le cas des multi-mots (*à l' instar de* désigne la forme unique *à_l'instar_de*,
  - ces deux phénomènes peuvent se coupler (*à l'instar du*),
  - on peut par ailleurs considérer, en première approximation, que les *entités nommées* sont également des formes représentées par plusieurs tokens (ainsi, on peut assimiler la suite de tokens *1, rue de la Paix, 75008 Paris* à une forme spéciale unique *_ADRESSE*).

Nous appelons *lemme* une classe d'équivalence de formes semi-décorées[3]. Une *forme décorée* est alors une forme semi-décorée à laquelle on adjoint un lemme.

Un lemme est donc un *ensemble*. Par conséquent, il peut être décrit en extension, c'est-à-dire par la donnée de tous ses éléments. Mais il peut également être décrit en intension, c'est-à-dire à l'aide d'une fonction qui associe à un identifiant de lemme un ensemble de formes semi-décorées. Une telle fonction est appelée *table de flexion* ou *paradigme flexionnel*. On appelle

[3]Rappelons ici l'abus de langage dont il est fait usage (voir chapitre 2) : nous confondons les formes et les graphies de ces formes.

classes morphologiques (ou *classes flexionnelles*) les classes formées par les lemmes que l'on peut définir à l'aide de la même table de flexion. Par conséquent, un lemme est intégralement défini par la donnée d'un identifiant et d'une classe morphologique : cette classe correspond à une table de flexion, qui est une fonction qui associe à l'identifiant du lemme touts ses éléments (qui sont des formes décorées). C'est la raison pour laquelle nous appellerons également *lemme*, par abus de langage, la donnée d'un couple *(identifiant de lemme, identifiant de classe morphologique)*. De même, nous appellerons *forme décorée* un quadruplet *(forme non-décorée, étiquette morphosyntaxique, identifiant de lemme, identifiant de classe morphologique)*. Enfin, nous dénoterons parfois une forme décorée par le terme de *forme fléchie*.

Certaines tables de flexion prennent en entrée comme identifiant une forme (non-décorée) particulière, jugée représentative selon des critères parfaitement arbitraires (comme l'infinitif pour les verbe en français). Certaines tables de flexions utilisent plutôt un *radical*, c'est-à-dire une suite de caractères qui peut ne pas être une forme du lemme concerné, mais qui est supposée représenter approximativement la plus grande partie commune à toutes les formes constituant le lemme. On pourrait imaginer d'autres types d'identifiants (racines trilitères dans les langues sémitiques…). Mais la distinction n'est que sémantique, et un lemme peut être dans tous les cas désigné et décrit de façon non-ambiguë par la donnée d'un couple *(identifiant, classe morphologique)*, pour peu que l'identifiant appartienne au domaine de définition de la table de flexion. Pour simplifier la présentation, nous désignerons l'identifiant par le terme *radical*, même s'il peut s'agir non pas d'un radical mais d'une forme choisie parmi les éléments du lemme.

Nous appelons *forme décorée* la donnée d'une forme semi-décorée et de son lemme. C'est par conséquent un triplet *(forme non-décorée, étiquette morphosyntaxique, lemme)*, ou, de façon équivalente, un quadruplet *(forme non-décorée, étiquette morphosyntaxique, radical, classe morphologique)*. L'opération transformant un texte en un treillis de formes (décorées, semi-décorées ou non décorées) via un treillis de tokens est appelé le *traitement pré-syntaxique*. Dans le cas particulier où le treillis est composé uniquement de formes décorées, on parle de *traitement morphosyntaxtique*. Ces traitements sont naturellement ambigus. On utilise parfois des modèles heuristiques ou probabilistes pour réduire ou supprimer cette ambiguïté.

On appelle *description morphologique* d'une langue la donnée d'un ensemble de tables de flexions. Celles-ci peuvent elles-même définies de façon extensionnelle, en indiquant successivement le moyen d'obtenir chacune des formes souhaitées, ou de façon plus compacte, à l'aide d'autres fonctions voire de foncteurs.

Enfin, on appelle *lexique morphologique* d'une langue la donnée d'un ensemble de lemmes, c'est-à-dire un ensemble de couples *(radical, classe morphologique)*. Par extension, on appelle également *lexique morphologique* le résultat de l'application des tables de flexions, c'est-à-dire

(en préservant le radical et la classe morphologique dans le résultat de cette opération), un ensemble de formes décorées.

3 Déclinaison et conjugaison

La façon la plus simple de représenter la flexion est celle de Bescherelle (1990), c'est-à-dire de manière pleinement extensionnelle et sans regroupements de paradigmes flexionnels. Toutefois, c'est une manière de faire qui ne favorise ni la pertinence linguistique, ni la compacité, ni la maintenabilité de la description morphologique. Un simple coup d'œil permet de constater que plusieurs tables sont extrêmement proches, et peuvent être regroupées en une seule table pour peu que l'on sache indiquer ici ou là des alternatives entre formes différentes, contraintes par un test sur le radical ou par une indication spécifique associée aux lemmes concernés[4]. Dans le premier cas, on parle de *contrainte locale*, c'est-à-dire au niveau de la forme. Par exemple, on pourra dire que le pluriel des adjectifs français en *-al* se fait en *-aux*, sauf si le radical est dans la liste des exceptions (*naval, banals, …*). Dans le second cas, on appelle *variante* de la table l'information supplémentaire qui doit être précisée pour choisir la bonne forme. Ainsi, parmi les noms communs inanimés du slovaque dont les radicaux se terminent par une consonne mouillés, on pourra distinguer (entre autres variantes) ceux qui ont un instrumental pluriel en *-mi* de ceux qui ont *-ami*. Naturellement, ces deux types de contraintes peuvent être couplées. De plus, pour contrôler la cohérence d'un lexique ou pour générer des lemmes hypothétiques pour une forme inconnue, une table peut contraindre elle-même les radicaux qui lui sont associés. On parle de *contrainte globale*.

3.1 Contraintes locales, contraintes globales, variantes, et classes de caractères

À titre d'exemple, considérons en slovaque la table de déclinaison des noms communs masculins animés dont le nominatif singulier ne prend pas la terminaison *-a*. Le lemme archétypal de cette table est généralement *chlap* (« garçon »). Dans notre description, cette table a pour nom `nc-mac` (pour « Noms Communs Masculins Animés dont le radical se termine généralement par une Consonne »). Cette table est munie d'une contrainte globale : le radical doit se terminer par une consonne ou par *-i*. Parmi les variantes de cette table, trois concernent le nominatif singulier et sont donc mutuellement exclusives : la variante `0` correspond aux lemmes dont le nominatif singulier reçoit une terminaison zéro, la variante `o` qui donne pour ce cas la

[4]Nous entendons par *lemme* un couple formé par un *radical* et une *classe morphologique*.

terminaison *-o*, et la variante `us` qui donne la terminaison *-us*. Les trois terminaisons correspondant à ces variantes concurrentes sont elles-même munies de contraintes locales : la terminaison zéro ne concerne que des radicaux se terminant par une consonne, la terminaison *-o* ne peut pas concerner des radicaux se terminant par *-tel'* ou *-an*, et la terminaison *-us* ne concerne que des radicaux se terminant par *i*, *j* ou une consonne dure. Par ailleurs, le cas du datif singulier illustre un autre phénomène : parfois, une même étiquette morphologique correspond pour un même lemme à plusieurs formes concurrentes. Ainsi, la terminaison par défaut pour cette table au datif singulier est *-ovi*. Une contrainte locale stipule cependant que le radical *boh* (« dieu ») ne reçoit pas cette terminaison. En revanche, une terminaison alternative *-u* ne concernant, grâce à une contrainte locale, que les radicaux *boh*, *duch*, *čert* et *pán*, est possible pour la même étiquette. Le fait que cette alternative ne soit pas liée à des variantes induit automatiquement pour *duch*, *čert* et *pán* deux formes en concurrences pour le datif singulier, l'une en *-ovi* et l'autre en *-u*.

Pour décrire les contraintes locales et globales, il est donc approprié de définir des classes de caractères (comme les consonnes dures du slovaque). Ces classes reçoivent un nom qui est un caractère unique, et pour une classe `c` l'expression `\c` représente une expression régulière reconnaissant un caractère de la classe `c`. Toutefois, ce n'est pas réellement le niveau du caractère qui est ici pertinent, mais un niveau intermédiaire entre le caractère et le phonème. C'est la raison pour laquelle on peut indiquer que certaines suites de caractères doivent être considérées comme un seul « caractère ». Il en est ainsi, entre autres, de *ch* en français, ou en slovaque de *ch*, *dz*, *dž* et des diphtongues bi-caractères (*ie*, *ia* et *iu*).

La figure 1 montre la façon dont sont représentées les flexions citées ci-dessus. Les éléments principaux sont `table` et `form`. La présence de plusieurs éléments `form` regroupés dans un élément `alt` indique que les variantes associées à ces formes sont mutuellement exclusives (ce qui ne veut pas nécessairement dire que ces formes sont elles-même mutuellement exclusives). L'attribut *rads* indique les contraintes globales positives sur les radicaux (dans un élément *table*) ou locales (dans un élément *form*), l'attribut *except* indique de la même façon les contraintes négatives sur les radicaux, et l'attribut *var* indique le nom de la variante. Par exemple, un lemme associé à la table `nc-mac` et à sa seule variante `us` se verra associé la classe morphologique `nc-mac :us`. L'ordre des variantes dans le nom d'une classe morphologique, s'il y en a plus d'une, n'est pas significatif. L'attribut `show` est présenté plus bas.

3.2 Modélisation des interactions entre radicaux et suffixes

Comme suggéré précédemment, une façon de factoriser l'information morphologique est de regrouper les paradigmes qui ne diffèrent qu'en surface, en raison de l'application de règles de

```
<table name="nc-mac" rads="..*\i">
      <alt>
        <form target="" tag="n-ma-s" rads="..*\c" var="0" show="ten #"/>
        <form target="o" tag="n-ma-s" rads="...*\c" except=".*(an|teľ)" var="o" show="ten #"/>
        <form target="us" tag="n-ma-s" rads="..*([ij]|\d)" var="us" show="ten #"/>
      </alt>
      <form target="a" tag="g-ma-s"/>
      <form target="ovi" tag="d-ma-s" except="boh"/>
      <form target="u" tag="d-ma-s" rads="(boh|duch|čert|pán)"/>
...
```

TAB. 1 – Extrait d'une table de déclinaison pour le slovaque.

collision à l'interface entre un radical et un affixe[5]. En français, par exemple, il n'y a en réalité qu'un seul paradigme pour les verbes dits du premier groupe. Pourtant, Bescherelle (1990) donne plusieurs paradigmes pour ces verbes. Mais ceux-ci ne diffèrent du paradigme de base (lorsqu'ils diffèrent) que par le résultat de l'application de règles simples, telles que l'adjonction d'un *e* aux radicaux en *-g* dès lors qu'on leur adjoint un suffixe qui commence par *o* ou *a* (ainsi *manger*, *(je) mange*, *(nous) mangeons*.

C'est la raison pour laquelle nous avons défini des règles dites de *collision*, décrites sous la forme d'expressions régulières ancrées sur une frontière entre radical et affixe. Pour répondre à certains besoins pratiques, ces règles peuvent être de deux types : les règles finales et les règles non finales. Elles sont ordonnées, et leur ordre d'application lors de la flexion d'un lemme est le suivant : tant qu'au moins une règle non finale peut être appliquée, elles sont appliquées dans l'ordre où elles apparaissent dans la description. Lorsqu'aucune règle non finale n'est appliquable, on applique (une seule fois) les règles finales dans leur ordre d'apparition. Dans ces règles, le symbole _ symbolise un endroit où se fait la transition entre radical et affixe, et on peut faire usage des classes de caractères.

Ainsi, en nommant `\o` la classe contenant les voyelles *o* et *a*, la règle permettant de traiter l'exemple ci-dessus couple le motif `g_\o`, du côté des radicaux et affixes, au motif `ge_\o`, du côté des formes fléchies (il est implicite que, dans un sens comme dans l'autre, le caractère reconnu par `\o` est laissé tel quel à l'emplacement où le motif cible contient également `\o`[6]).

Quelques exemples de telles règles de collision pour le slovaque sont donnés dans la table 2.

Le modèle ainsi défini peut être ramené à un ensemble d'opérations régulières, c'est-à-dire à une transduction. À ce titre, le pouvoir d'expression de notre formalisme est identique à celui

[5]En sanskrit, le phénomène représenté par ces règles s'appelle le *sandhi interne* (par opposition avec le *sandhi externe* qui est modélisé par des règles de collision à l'interface entre deux mots. Pour plus de détails sur le sandhi en sanskrit, on se reportera par exemple à Huet (2004). Par ailleurs, Gérard Huet préfère le terme *jonction* pour traduire *sandhi* (communication personnelle). Par habitude, nous nous en tiendrons à *collision*.

[6]S'il y en avait plusieurs, leur ordre serait préservé.

<fusion source="s_š" target="_š"/>	*comparatifs des adjectifs en* -sý *ou* -sy
<fusion source="s_ˇ" target="š_"/>	*mouillure de la dernière consonne du radical*
<fusion source="bc$" target="bec$"/>	*voyelle d'appui pour radicaux à complexe consonnantique en cas de terminaison zéro*
<fusion source="ä\c+_-" target="ia\c+_"/>	*allongement du* ä *en* ia *induit par la terminaison spéciale « - »*
<fusion source="ň_\r" target="n_\r"/>	*la mouillure n'est pas marquée devant* e, i, í *et les diphtongues* ie, ia, iu

TAB. 2 – Quelques règles de collision pour le slovaque .

de Kaplan et Kay (1994). Il permet toutefois une plus grande lisibilité de la description, en particulier par une plus grande factorisation de l'information à l'aide des règles de collision mais aussi des mécanismes d'héritage entre classes présentés ci-dessous.

3.3 Héritage entre classes morphologiques

Dans certains cas, il est plus économique de ne pas regrouper en une seule classe deux classes morphologiques qui partagent pourtant un sous-ensemble commun, car les règles de collision seraient inutilement lourdes. Notre formalisme dispose à cet effet d'un mécanisme d'héritage entre classes. De façon générale, on peut exprimer le fait qu'une classe hérite d'une autre, sauf éventuellement pour certaines étiquettes morphologiques. On peut indiquer les formes associées à certaines étiquettes, qui remplaceront les formes héritées.

Prenons un exemple. En français, le verbe *haïr* se conjugue comme n'importe quel verbe du deuxième groupe, à l'exception des formes singulier du présent de l'indicatif et de l'impératif (*hais*/*hait* et non *haïs*/*haït*). Mais dans la description de la classe des verbes du deuxième groupe, la forme en *-s* des première et deuxième personne du singulier du présent de l'indicatif est fusionnée avec celle du passé simple à l'aide d'une étiquette doublement ambiguë notée `PJ12s`, où `PJ` indique le présent ou le passé simple et `12` indique la première ou la deuxième personne, `s` indiquant le singulier. Par conséquent, la classe morphologique du verbe *haïr* sera décrite comme suit : on indique qu'elle hérite de la table `v-ir2` des verbes du deuxième groupe[7] à l'exception de la ou des formes associées à l'étiquette `PJ12s`, puis on (re)définit les formes du singulier du présent de l'indicatif et de l'impératif, mais aussi la forme de première et deuxième personne du singulier du passé simple. La figure 3 montre la façon dont est représentée cette table en XML : l'élément `like` indique l'héritage (l'attribut `except` indiquant la forme dont on veut bloquer l'héritage). On notera la présence du caractère `=` dans certains suffixes, qui

[7] Dans notre description, pour des raisons à la fois pratiques et diachroniques, les radicaux du deuxième groupe se terminent en *i* : nous ne coupons pas *finir* en *fin_ir* mais en *fini_r*.

```
<table name="v-haïr" source="r" rads="haï">
   <like name="v-ir2" except="PJ12s"/>
   <form target="=t" tag="P3s" show="#"/>      hait
   <form target="=s" tag="P12s"/>              hais
   <form target="=s" tag="Y2s"/>
   <form target="s" tag="J12s"/>               haïs
</table>
```

TAB. 3 – Une table de conjugaison pour le français avec héritage. On notera la présence de l'attribut `source` dans l'élément `table`, qui indique le suffixe qu'il faut ajouter au radical pour obtenir non pas une forme fléchie mais l'identifiant de lemme.

modélise le fait qu'il faut retirer le tréma sur le *ï* terminant le radical : une règle de collision spécifique associe `ï_=` à `i_`, permettant ainsi d'associer *haï_=t* à *hai_t*.

4 Morphologie dérivationnelle

Enfin, conformément à ce que nous avons évoqué plus haut, nous avons mis en place un mécanisme pour représenter les dérivations morphologiques. Contrairement aux formes fléchies d'un lemme, qui sont toutes des formes correctes, les lemmes dérivés qui lui sont associés par la description morphologique sont des lemmes possibles. Ainsi, on associera (entre autres) à la table de conjugaison des verbes français du premier groupe des dérivations en *-age* et en *-ement*, alors même que *passement* et *changeage* n'existent pas. De même que les formes fléchies, ces dérivations peuvent toutefois être conditionnées par des tests (positifs et/ou négatifs) sur le radical.

On indique donc pour chaque dérivation possible le ou les affixes à ajouter au radical d'origine pour obtenir le radical dérivé ainsi que la table selon laquelle se fléchit ce radical dérivé. Enfin, les patrons de collision sont appliqués à l'interface entre radical d'origine et affixes permettant d'obtenir le radical dérivé (ils sont naturellement appliqués aussi à l'interface entre le radical dérivé et les affixes qu'il reçoit lors de sa propre flexion).

La partie de la table des verbes français du premier groupe décrivant les lemmes dérivés potentiels est reproduite table 4.

```
<table name="v-er" rads="...*">
...
    <derivation target="ade" table="nc-2"/>      dérobade
    <derivation target="age" table="nc-2"/>      accrochage
    <derivation target="e" table="nc-2"/>        accroche
    <derivation target="ement" table="nc-2"/>    rangement
    <derivation target="tion" table="nc-2"/>     dilution, aliénation revendication
    <derivation target="cation" table="nc-2"/>   unification
    <derivation target="at" table="nc-teur"/>    examinateur, examinatrice
    <derivation target="cat" table="nc-teur"/>   unificateur
    <derivation target="ure" table="nc-2"/>      raclure
    <derivation target="oir" table="nc-4"/>      racloir
    <derivation target="et" table="nc-et"/>      jouet
    <derivation target="ette" table="nc-2"/>     sonnette
    <derivation target="isme" table="nc-2"/>     échangisme
    <derivation target="iste" table="nc-2"/>     échangiste
    <derivation target="" table="nc-eur"/>       semeur, semeuse
</table>
```

TAB. 4 – Lemmes dérivés potentiels pour les verbes français du premier groupe.

5 Compilation de ce formalisme en outils morphologiques

L'utilisation de ce formalisme morphologique se fait à l'aide d'un compilateur qui génère trois scripts et une librairie en langage *perl*[8]. Les scripts sont respectivement un script de flexion, un script de dérivation et un script de lemmatisation ambiguë. Les deux premiers prennent en entrée des lemmes (des couples *(radical, classe morphologique)*), le dernier prend en entrée une forme fléchie[9]. La librairie contient quelques fonctions génériques utilisées par les trois scripts, et en particulier une fonction permettant de vérifier qu'un lemme est cohérent (en particulier que le radical vérifie les tests globaux de sa classe morphologique et au moins un test local par étiquette).

Le script de flexion peut recevoir une option lui indiquant de ne pas donner toute la flexion du lemme donné en entrée, mais seulement un aperçu plus facilement contrôlable par un humain. C'est le rôle de l'attribut `show`. En effet, l'activation de cette option permet d'obtenir le résultat de la concaténation de toutes les valeurs des attributs `show` des formes qui en ont, où le symbole # a été remplacé par la forme fléchie correspondante.

[8]Cette solution est transitoire. Nous envisageons dans un avenir proche de compiler nos descriptions morphologiques sous la forme de transducteurs à l'aide de la boîte à outils *Zen* développée par Gérard Huet (voir Huet (2002)).

[9]Nos scripts, comme nos descriptions morphologiques, reposent sur l'encodage de caractères UTF-8 qui permet d'éviter tout problème lié à la présence dans la langue considérée de caractères accentués ou non-standard, voire de systèmes d'écriture différents.

name	rads	source	Exemples
adj-ique2	..*i	que	comique
adj-blanc	(blan|fran|se)	c	blanc
adj-eur	..*	eur	chanteur
adj-teur	..*[dt]	eur	agriculteur
adj-1	..*		super
adj-s3	(..*[iu][sz]|..*\vrs|absconsl.*clos|dispos)		gris
adj-er4	.*.	er	fermier
adj-f4	(.*.\v|ser)	f	vif
adj-2	(..*\E|..*\K|...*e)		rouge
adj-4	.*(\E|\K)		joli
adj-x3	..*u	x	fameux
adj-gros3	.*.[aio]s		gras
adj-al4	.*.a	l	normal
adj-l4	(.*ell.*eill.*all|nul|gentil)		éternel
adj-n4	.*..n		parisien
adj-t4	(.*etl.*ot|chat|chouchoul|favoril|rat|rigolo)		simplet
adj-gu4	.*.gu		aigu
adj-eur-eresse	..*	eur	vengeur

TAB. 5 – Classes morphologiques adjectivales définies dans notre description morphologique du français.

6 Application au français

Ce formalisme METAMORPHO nous a permis de remplacer la description morphologique du français dont nous disposions, qui définissait un nombre très important de classes morphologiques sans toutefois représenter la morphologie dérivationnelle. Notre nouvelle description morphologique, grâce aux propriétés de factorisation et d'héritage de METAMORPHO, et grâce aux règles de collision, ne comporte plus en effet que :

- 39 classes verbales (dont une seule pour les verbes du premier groupe plus une table pour le verbe *fiche*, et deux pour ceux du deuxième groupe),
- 29 classes nominales,
- 18 classes adjectivales (dont 13 héritent sans modification d'une table nominale),
- 1 « table » adverbiale.

Les tables 5, 6 et 7 montrent l'ensemble de ces classes morphologiques, ainsi que quelques lemmes traités par chacune d'entre elles.

Notre description utilise plus de « caractères »[10] qu'il n'y a de lettres dans l'alphabet slovaque, afin de modéliser des différences de comportement. Ainsi, outre les « lettres » standard

[10]Nous avons conscience de résoudre de manière pragmatique, mais pas toujours satisfaisante d'un point de vue théorique, le hiatus qu'il peut y avoir entre graphème et phonème. Toutefois, dans le cas du slovaque, cette différence est minime, puisqu'il y a quasiment correspondance biunivoque entre phonèmes et graphèmes, à condition de considérer comme nous le faisons certaines suites de deux caractères comme un seul graphème.

name	rads	source	Exemples
nc-1	..*		albinos
nc-1m	..*		tonus
nc-1f	..*		toux
nc-1fp	..*		mœurs
nc-1mp	..*		oripeaux
nc-2m	(..*\E|..*\K|...*e)		truc
nc-2f	(..*\E|..*\K|...*e)		chose
nc-l2m	..*a	l	cheval
nc-il2m	..*a	il	corail
nc-u2m	..*u		rideau
nc-man2m	..*m	an	barman
nc-y2m	..*	y	dandy
nc-y2f	..*	y	lady
nc-2	(..*\E|..*\K|...*e)		collègue
nc-x3	..*u	x	boiteux
nc-s3	(..*[ui][zs]|lours|pervers|reclus)		polonais
nc-eau4	.*..	eau	chameau
nc-n4	.*..n		chien
nc-f4	(.*.\v|ser)	f	captif
nc-er4	.*..	er	berger
nc-4	.*(\E|\S)		lapin
nc-4sse	.*\ce		ogre
nc-teur	..*[dt]	eur	narrateur
nc-eur	..*	eur	voleur
nc-eur-eresse	..*	eur	pécheur
nc-t4	(.*etl.*ot|chat|chouchou|favori|rat|rigolo)		blondinet
nc-l4	(.*ell.*eill.*all|nul|gentil)		officiel
nc-al4	.*.a	l	provincial
nc-blanc	(blan|fran|se)	c	blanc, franc, sec

TAB. 6 – Classes morphologiques nominales définies dans notre description morphologique du français.

name	rads	source	Exemples
v-er	...*	er	aimer, manger, payer, jeter, acheter, appeler, peler, courbaturer
v-fiche	.*fich	e	fiche
v-ir2	..*i	r	finir
v-haïr	<haï	r	haïr
v-maudire	<maudi	re	maudire
v-aller	(r|)all	er	aller
v-ir3	.*(bouill|dorm|serv|.\Tt|ten|ven|enfui|fui|oï|.*quér)	ir	bouillir, dormir, servir, tenir, ouïr, acquérir
v-re3	.*(nd|batt|rompl.*[aeo]iŋ|.*vainqu|	re	fendre, battre, rompre, peindre, joindre, feindre, asseoir (je m'asssieds)
	r ?asseyl|seyl|messeyl|rd)		
v-faillir	faill	ir	faillir
v-vêtir	.*vêt	ir	vêtir
v-prendre	.*pren	dre	prendre
v-mettre	.*mett	re	mettre
v-assaillir	(assaill|tressaill|défaill|saill|accueill|recueill|cueill)	ir	asaillir, cueillir
v-courir	.*cour	ir	courir
v-ouvrir	.*(couvr|ouvr|souffr|offr)	ir	couvrir, offrir
v-mourir	mour	ir	mourir
v-gésir		gésir	gésir
v-evoir	.*[cd]	evoir	recevoir, devoir
v-voir	(entre|re|)v	oir	voir
v-prévoir	(re|)prév	oir	prévoir
v-pourvoir	(re|)(dé|)pourv	oir	pourvoir
v-savoir	s	avoir	avoir
v-seoir	(ass|rass)oi	r	asseoir (je m'asseois)
v-surseoir	sursoi	r	surseoir
v-choir	(ch|rech|éch|déch)oi	r	échoir
v-inf	(capeyer|raire|ravoir|stupéfaire)		ravoir
v-avenir	aven	ir	avenir
v-parfaire	parfai	re	parfaire
v-être		être	être

TAB. 7 – Classes morphologiques verbales définies dans notre description morphologique du français.

du français (y compris ceux, comme dit plus haut, écrits à l'aide de deux caractères), nous avons introduit le phonème *ŋ* qui représente la terminaison des radicaux de verbes tels que *peindre*. Des règles de collision indiquent alors que *ŋ* devient *n* devant *r*, *nd* devant une autre consonne et *gn* devant une voyelle. Ceci étant fixé, le radical *peiŋ* correspond à un lemme qui suit régulièrement la table des verbes du troisième groupe en *-re* (table v-re3)[11].

7 Application au slovaque

Nous avons développé dans le formalisme METAMORPHO introduit précédemment une description morphologique relativement riche du slovaque, qui couvre toutes les catégories ouvertes (à l'exclusion des catégories fermées, dont pourtant certaines se déclinent, comme les divers types de numéraux ou de pronoms).

De même que pour le français, notre description utilise plus de « caractères » qu'il n'y a de lettres dans l'alphabet slovaque, afin de modéliser des différences de comportement. Ainsi, outre les « lettres » standard du slovaque (y compris ceux, une fois encore, écrits à l'aide de deux caractères), nous avons introduit le phonème *ŋ*. Dans le cas du slovaque, il représente un *nk* insécable[12].

Nous utilisons 22 classes de caractères différentes, dont certaines sont classiques et d'autres plus pragmatiques (voyelles, voyelles longues, voyelles brèves, leurs contreparties sans les liquides *l* et *r*, consonnes, consonnes et semi-consonne *j*, consonnes dures, consonnes mouillées, occlusives, non-occlusives, etc…).

Les règles de collisions sont relativement nombreuses (150 règles environ). Outre quelques phénomènes anecdotiques, ces règles modélisent principalement les phénomènes suivants :

- la *règle rythmique* selon laquelle, en slovaque, il est impossible que deux syllabes longues se suivent[13], sauf dans certains cas très précis,
- la disparition de la marque de mouillure de certaines consonnes devant certaines voyelles (ainsi, *ň*+*e* s'écrit *ne*),
- les modalités d'allongement de la dernière voyelle d'un radical au génitif pluriel de certains paradigmes nominaux féminins et neutres[14],

[11] Puisque pour cette table, `source` vaut *re*, le lexique comporte une entrée *peiŋre v-re3*.

[12] Un radical nominal féminin ou neutre en *-nk* dont le génitif pluriel allonge la voyelle précédente (*banka,bánk*) au lieu d'être en *-nok* ou *-niek* (selon la longueur de la voyelle précédente : *stránka,stránok*, *dovolenka,dovoleniek*) sera représenté par *ŋ*.

[13] Une syllabe longue comporte une voyelle longue ou une diphtongue.

[14] Par exemple, le génitif pluriel de *žena* (« femme ») est *žien* : on décide que la terminaison est dans ce cas `-`, et on ajoute une règle de collision appariant `e\c_-` et `ie\c_`, où `\c` désigne un caractère de la classe des consonnes.

– l'insertion d'une voyelle d'appui dans les radicaux se terminant par une suite de consonnes pour les formes à terminaison zéro[15].

Enfin, nous avons défini un nombre relativement raisonnable de classes flexionnelles permettant de représenter quasiment toute la morphologie des classes ouvertes du slovaque : les 63 classes se répartissent en 36 classes de conjugaison verbale, 21 classes de déclinaison nominale, 5 classes de déclinaison adjectivale (qui pourraient être ramenées à 3) et 1 classe d'adverbes (sans flexion, mais utilisée par la morphologie dérivationnelle et permettant de contraindre les radicaux possibles).

[15]On utilise pour cela le caractère spécial `$` représentant la fin du mot. Une règle de collision qui associe `bc_$` à `bec_$` permet alors de représenter l'alternance entre la forme nominative du singulier *vrabec* (« corbeau ») et les autres formes sans insertion du *-e-*, comme l'accusatif singulier *vrabca*.

Chapitre 5

Représentation intensionnelle du lexique par héritage de propriétés

Ce chapitre présente l'architecture que nous avons développée pour représenter les informations lexicales. Elle a pour objectifs de satisfaire quatre principes :

- la pertinence linguistique,
- l'utilisabilité dans des analyseurs syntaxiques ou syntaxico-sémantiques (représentation adéquate et indépendante du formalisme),
- la facilité d'entretien, de correction et de mise à jour (pour dénoter cette idée, nous emploierons désormais l'anglicisme « maintenabilité »),
- le regroupement modulaire d'informations lexicales diverses (morphosyntaxiques, syntaxiques, voire sémantiques).

Ces quatre critères conduisent inévitablement à un principe plus général, celui de la factorisation de l'information, ou, dit autrement, de la réduction des redondances autant que faire se peut.

L'ensemble de ces principes nous a conduit à introduire deux niveaux de lexiques : un lexique décrit de façon *intensionnelle*, et un lexique décrit de façon *extensionnelle*, obtenu à partir du précédent par *compilation* et utilisé par les analyseurs[1].

Dans ce chapitre, nous présentons ces deux types de lexique, ainsi que l'architecture par héritage de propriétés qui permet de passer de l'un à l'autre. Nous illustrerons notre propos à l'aide du Le*fff* (Lexique des Formes Fléchies du Français), lexique qui dans sa version 2 est un lexique morphologique et syntaxique du français à large couverture (Sagot *et al.*, 2005, 2006). Nous avons développé le Le*fff* à partir d'une version préliminaire, partielle et purement extensionnelle issue de travaux de Lionel Clément. Ce développement est toujours en cours, tant au niveau du formalisme lexical décrit dans ce chapitre qu'au niveau du contenu lui-même. Ce

[1] Ou en tout cas par ceux des analyseurs qui ne peuvent prendre directement en compte la forme intensionnelle.

lexique, dont la forme extensionnelle est disponible sur Internet, est utilisé dans les systèmes d'analyse développés au sein de l'équipe Atoll (dont SXLFG, voir chapitre 9), mais également exploité par d'autres équipes. De nombreuses techniques d'acquisition automatique ou semi-automatique ont été utilisées pour la constitution du Le*ffff* à côté de méthodes plus manuelles. Ces techniques font l'objet des deux chapitres suivants, ainsi que leur mise en œuvre au bénéfice du Le*ffff* (mais pas uniquement). En particulier, nous donnons quelques caractéristiques quantitatives du Le*ffff* 2 à la fin du chapitre 6.

1 Entrées lexicales

1.1 Qu'est-ce qu'une entrée lexicale ?

Le lexique intensionnel comme le lexique extensionnel sont constitués (entre autres) d'un ensemble d'entrées lexicales. Chaque entrée lexicale peut être considérée comme la représentation de certaines propriétés d'une entité linguistique. Au niveau intensionnel, ces entités sont des lemmes[2]. Au niveau extensionnel, il s'agit des formes fléchies[3]. Chaque entité linguistique élémentaire (forme ou lemme selon le cas considéré) peut être la vedette de plusieurs entrées : c'est le phénomène de l'*ambiguïté lexicale*.

Nous n'avons pas défini ce que nous entendons par *élémentaire*. En réalité, il s'agit d'une véritable question. En effet, s'il est classique de considérer le nom commun *loup* comme un lemme, la situation est moins claire, à des degrés variables, pour *loup-garou*, *gueule de loup*, *loup de mer*, *loup cervier*, *faim de loup*, *entre chien et loup*, *se jeter dans la gueule du loup*. La limite est floue entre lemme, « mot composé », expression figée, et véritable suite compositionnelle de lemmes ou de formes indépendants. Du reste, cette difficulté est double : lesquelles de ces entités, que nous appellerons *multi-mots*, doivent-elles former des entrées du lexique ? Mais également, comment représenter les entités complexes comme les co-occurrences ou les expressions figées, qui peuvent être modifiées (ainsi *se jeter à corps perdu dans la gueule du loup*) ?

[2]Nous verrons plus bas dans quelle mesure les phénomènes d'homonymie affectent la définition de ce qu'est un lemme.

[3]Dans certains lexiques, il s'agit également des lemmes. Nous pensons toutefois, tant pour des raisons théoriques que pour des raisons opérationnelles, que la forme fléchie est le niveau de granularité approprié. D'un point de vue opérationnel, il est toujours préférable de calculer une fois pour toutes tout ce qui peut l'être, et non pendant l'analyse. Si donc il est possible de se passer de l'intermédiaire qu'est le lemme pour obtenir les informations sur une forme fléchie dont on a besoin pour effectuer une analyse, alors il est préférable de s'en passer. Par ailleurs, les informations syntaxiques (sans parler des informations morphologiques) ne dépendent pas seulement du lemme : un participe passé passif ou un infinitif n'ont pas exactement les mêmes cadres de sous-catégorisation que les formes finies du même lemme.

```
boire v-ir3 @verbe_standard
```

TAB. 1 – L'entrée lexicale du lemme *boire* dans le lexique intensionnel. La classe morphologique `v-ir3` est celle des verbes standards du troisième groupe. La classe syntaxique `@verbe_standard` est celle des verbes transitifs directs, avec objet direct facultatif, et qui admettent une construction pronominale de type *Le vin se boit à 14 degrés*.

Par ailleurs, les liens de morphologie dérivationnelle entre lemmes peuvent justifier une hiérarchisation des lemmes eux-mêmes sous forme d'arbres (ou de graphes) de dérivation morphologique. C'est d'autant plus souhaitable que certains motifs de dérivation morphologique induisent une transformation bien précise sur les propriétés syntaxiques. Par exemple, un nom déverbal en *-eur* à partir d'un verbe du premier groupe est un déverbal agentif qui dénote l'agent de l'action exprimé par le verbe : il aura donc, en première approximation, le même cadre de sous-catégorisation que le verbe, à l'exception du sujet[4].

Nous ne prétendons pas apporter une réponse définitive à toutes ces questions, pour lesquelles la prise en compte simultanée des contraintes de pertinence linguistique et d'utilisabilité opérationnelle n'est pas aisée. En particulier, notre architecture, au stade actuel de son développement, ne permet pas de représenter des multi-mots structurés. Elle permet en revanche de représenter de façon prometteuse les relations de dérivation.

1.2 Forme intensionnelle et forme extensionnelle

1.2.1 Forme intensionnelle

Dans la forme intensionnelle du lexique, une entrée lexicale de base est la donnée d'un lemme et d'une classe syntaxique, c'est-à-dire la donnée d'un radical ou d'une forme fléchie exemplaire, d'une classe morphologique et d'une classe syntaxique (voir la table 1). Ces classes sont définies de façon indépendante en respectant les principes de pertinence linguistique, de maintenabilité, et de factorisation de l'information. Le chapitre précédent a traité de la représentation selon ces principes des classes morphologiques. Au cours de ce chapitre, nous verrons comment nous avons procédé pour décrire les classes syntaxiques à l'aide d'un graphe d'héritage de propriétés syntaxiques atomiques.

Nous avons mis en place également un moyen pour représenter les mécanismes de morphologie dérivationnelle, en tirant parti du couplage entre morphologie et syntaxe qui caractérise

[4]Naturellement, l'objet d'un nom déverbal n'a pas la même réalisation que l'objet d'un verbe. Mais la correspondance entre ces deux réalisations est systématique. Par exemple, à l'objet nominal d'un verbe correspond un « complément du nom » (syntagme prépositionnel) introduit par la préposition *de*. Ainsi, *Jean mange des pommes* est à mettre en parallèle avec *Jean est un mangeur de pommes*.

`blanc adj-blanc @adj_couleur`	*lemme de base*
`> adjectif_nominalisé`	*(un) blanc*
`> péjoratif-âtre`	*blanchâtre*
`> causatif_déadjectival-ir`	*blanchir*
`>> agentif-eur`	*(le/la) blanchisseur/euse*
`>> nom_d_action-age`	*(le) blanchissage*
`>> participe_présent_adjectivé`	*blanchissant(e)(s)*
`> nom_déadjectival-eur`	*(la) blancheur*

TAB. 2 – L'entrée lexicale du lemme *blanc* dans le lexique intensionnel, avec certains de ses dérivés.

ces mécanismes[5]. Ainsi, on peut indiquer en dessous d'un lemme de base un certain nombre de dérivations (lignes commençant par le symbole >). Chaque dérivation peut elle-même être suivie d'une dérivation secondaire (indiquée par >>), et ainsi de suite (voir table 2). Une dérivation n'est modélisée que par un seul identifiant, qui désigne à la fois :

– le mécanisme morphologique permettant de passer du lemme de base au lemme dérivé (c'est la valeur de l'attribut `name` de l'élément `derivation` correspondant dans notre formalisme de description morphologique décrit au chapitre précédent),
– un *foncteur syntaxique* permettant de passer de la structure syntaxique du lemme de base à celle du lemme dérivé.

Les foncteurs syntaxiques sont définis de façon parallèle aux classes syntaxiques, au sein d'un graphe d'héritage de foncteurs atomiques eux-même décrits selon la même syntaxe que les propriétés syntaxiques atomiques. La différence est que ces foncteurs agissent sur les informations associées au lemme de base pour en dériver les informations associées au lemme dérivé.

Notre architecture permet de gérer les phénomènes d'homonymie entre radicaux, ainsi que d'homonymie entre lemmes. En effet, il est relativement fréquent qu'un même radical puisse correspondre à différents lemmes. Ainsi, en français, la forme verbale infinitive *ressortir* a deux emplois différents selon qu'il s'agit de l'infinitif d'un verbe du deuxième groupe (*nous ressortissons [de tel pays]*) ou du troisième groupe (*nous ressortons [nous sortons à nouveau]*). Du reste, dans ce cas, les cadres de sous-catégorisation sont également différents. En pratique, on peut avoir plusieurs entrées dans le lexique intensionnel qui partagent le même radical, voire également la même classe flexionnelle[6]. On peut de plus identifier deux homonymes. Ainsi, notre lexique comporte deux entrées distinctes pour les lemmes *voler___fly* et *voler___steal*. Le

[5]Il y a naturellement des exceptions, soit au niveau morphologique soit au niveau syntaxique. Le but de notre mécanisme est de représenter les autre cas, qui sont majoritaires.

[6]On peut leur faire partager aussi la classe syntaxique, mais alors on a deux lignes identiques, ce qui n'est que pure redondance

```
bois    v  [pred='boire_____1<subj,(obj)>',cat=v,@P12s]
bu      v  [pred='boire_____1<subj,(obj)>',cat=v,@active,@Kms]
bu      v  [pred='boire_____1<(par-obj),subj>',cat=v,@passive,@être,@Kms]
...
```

TAB. 3 – Quelques entrées lexicales de formes fléchies du lemme *boire* dans le lexique extensionnel. On notera que l'on distingue le participe passé actif du participe passé passif, qui diffèrent entre autres par leur cadre de sous-catégorisation[8].

conjugueur sait gérer ces identifiants, qui correspondent à deux lemmes de classes syntaxiques différentes (pour faire simple, un verbe intransitif et un verbe transitif respectivement).

1.2.2 Forme extensionnelle

La forme extensionnelle du lexique est composée d'entrées lexicales qui sont des quadruplets : à une forme fléchie on associe une catégorie, un poids et une structure syntaxique (voir la table 3). La *catégorie* peut servir de base à un étiqueteur, peut correspondre à un terminal dans une grammaire non-contextuelle (comme le squelette d'une grammaire LFG), peut désigner une classe d'arbres dans le cas de grammaires TAG, ou servir à tout autre traitement nécessitant le groupement des formes en quelques demi-douzaines de classes. Le *poids* permet à toute application utilisant le lexique de favoriser certaines entrées par rapport à d'autres. Ainsi, notre processus de compilation du lexique associe aux multi-mots un poids supérieur à la somme des poids de ses composants pris individuellement. D'autres poids peuvent être renseignés manuellement. C'est le cas dans notre lexique du français, le L*efff*, pour certains mots pour lesquels on souhaite favoriser une catégorie par rapport à une autre (ainsi, des mots comme *sur*, *sous*, *par*, ou *pour* auront un poids inférieur en tant que noms communs par rapport à leur poids en tant que prépositions).

Enfin, la *structure syntaxique* regroupe les informations morphologiques et syntaxiques de façon appropriée aux traitements automatiques et inspirée par certains aspects du formalisme LFG[9]. Ces informations peuvent être de trois types :

– un prédicat syntaxico-sémantique sous-spécifié, le `pred`, qui attribue à toutes les formes sémantiquement pleines un identifiant du lemme et un cadre de sous-catégorisation à la LFG[10],

[9]Par rapport à des entrées lexicales du formalisme LFG, la principale différence est que nous n'associons jamais des *équations fonctionnelles* (ou un équivalent) aux entrées, mais des *structures fonctionnelles sous-spécifiées*. En réalité, on peut considérer ces structures sous-spécifiées comme le résultat de l'application d'équations fonctionnelles lexicales. Ainsi, ce que LFG noterait `(↑Subj) = (↑Vcomp Subj)` sera noté (directement ou via des macros, voir plus bas) : `[subj=[]1, vcomp=[subj=[]1]]`.

[10]La dernière version du L*efff* a un format plus évolué :

– des structures de traits sous-specifiées explicites,
– des macros lexicales et grammaticales, qui représentent des structures de traits sous-spécifiées (couples attributs-valeurs pour les macros lexicales, structures de dépendances sous-spécifiées pour les macros grammaticales),

Le prédicat syntaxico-sémantique sous-spécifié, le `pred`, attribue à toutes les formes sémantiquement pleines un identifiant du lemme et un cadre de sous-catégorisation à la LFG. Ce cadre de sous catégorisation liste par ordre d'obliquité[11] les arguments du prédicat sémantique. Chaque argument est la disjonction d'une ou plusieurs réalisations syntaxiques possibles, selon une terminologie inspirée de la grammaire LFG de Lionel Clément. Un sujet peut être réalisé sous la forme d'un pronom ou d'un groupe nominal (`subj`), d'une infinitive (`vsubj`) ou d'une complétive (`ssubj`). Un objet direct peut être réalisé sous la forme d'un pronom ou d'un groupe nominal (`obj`), d'une infinitive (`vcomp`), d'une complétive (`scomp`), ou d'une interrogative indirecte (`whcomp`). Les attributs sont des `acomp`, qu'ils soient nominaux ou adjectivaux[12]. Les compléments introduits par une préposition sont représentés comme en LFG par la concaténation de la préposition concernée et de la réalisation sans préposition correspondante. Ainsi, et à titre d'exemple, `de-scomp` représente une complétive en *de ce que*, `à-obj` représente un objet indirect introduit par *à*, et `pour-acomp` représente un attribut introduit par *pour* (*Jean prend Marie pour un génie*).

Les structures de traits que l'on veut associer à une entrée peuvent être représentées de deux façons : soit elles sont indiquées explicitement (il en est ainsi de ainsi de `cat=v` dans l'exemple de la table 3), soient elles sont indiquées sous la forme de *macros*, qui ne sont que des raccourcis, définis par ailleurs, pour des structures de traits (pouvant elles-même contenir des macros). L'intérêt des macros est que leur définition peut, le cas échéant, dépendre de l'utilisation que l'on fait du lexique. Dans la dernière entrée de la table 3, la macro `@Kms` représente par exemple la structure `[@K, @m, @s]`, dont l'explicitation complète dépend des définitions que l'on a pour ces trois macros. Les deux principaux systèmes d'analyse développés au sein de l'équipe ATOLL ont en effet des définitions différentes de certaines macros, dont la macro `@K`. Ainsi, le système FRMG explicitera par exemple `@Kms` en `[v-form = participle, gender = masc, number = sg]`, alors que SXLFG en fera `[v-form = past-participle, tense=none, gender = masc, number = sg]`. Ces différences montrent comment un même lexique syntaxique peut, grâce à des macros, être utilisé par différents systèmes reposant sur différentes grammaires voire différents formalismes.

[11] Ainsi, un participe passé passif a pour premier argument un `par-obj` facultatif et pour deuxième argument un sujet, avec les différentes réalisations possibles (selon les verbes, seulement un sujet nominal `subj` ou un sujet nominal ou phrastique `subj|vsubj|ssubj`).

[12] Alors qu'en LFG standard on distingue les *acomp* des *ncomp*.

Outre des informations simples comme une structure de traits morphologiques, les macros peuvent représenter des structures sous-spécifiées plus complexes. Par exemple, la macro `@CtrlSubj` désigne un phénomène de contrôle sujet. Elle est étendue, dans nos deux systèmes, en la structure sous-spécifiée suivante, où l'indice `1` sert à indiquer un partage de structures : `[subj=[]1, vcomp=[subj=[]1]]`. On peut la gloser de la façon suivante : le sujet de l'entrée lexicale courante est identique au sujet de son infinitive objet. L'explicitation des macros est prévue pour être effectuée une fois pour toutes au moment de la construction d'une structure de données opérationnelles à partir du lexique extensionnel. C'est ce qui se passe dans les systèmes FRMG et SXLFG.

2 Architecture par héritage de propriétés

Nous présentons dans la figure 1 l'architecture du lexique, c'est-à-dire la structure du processus permettant de compiler le lexique intensionnel en lexique extensionnel. Cette architecture repose sur un principe général d'héritage de propriétés atomiques. Ces propriétés atomiques sont regroupées en deux familles relativement étanches, les propriétés morphologiques et les propriétés syntaxiques. Le regroupement de propriétés morphologiques au sein de classes morphologiques, qui peuvent aussi se définir les unes par rapport aux autres par héritage non-monotone entre classes, a été vu au chapitre précédent. Les propriétés atomiques syntaxiques sont les points de départ d'une autre hiérarchie d'héritage, beaucoup plus profonde.

Ces deux ensembles de propriétés ne sont toutefois pas entièrement disjoints. D'une part les classes morphologiques peuvent attribuer à chaque forme une propriété syntaxique atomique, qui est donc spécifique à la forme et non à son lemme (contrairement aux autres propriétés syntaxiques). D'autre part les dérivations morphologiques induisent l'application de foncteurs syntaxiques sur les propriétés syntaxiques du lemme source pour produire les propriétés syntaxiques du lemme cible, avec autant de foncteurs qu'il y a de dérivations successives pour aller du lemme source au lemme cible.

Malgré ces écarts à la séparation entre propriétés morphologiques et propriétés syntaxiques, la distinction est suffisamment pertinente pour justifier un processus en deux phases pour la compilation du lexique intensionnel en lexique extensionnel. Le point de départ principal est un ensemble de fichiers de lemmes qui à chaque lemme associent une classe morphologique et une classe syntaxique, comme indiqué précédemment (voir table 1).

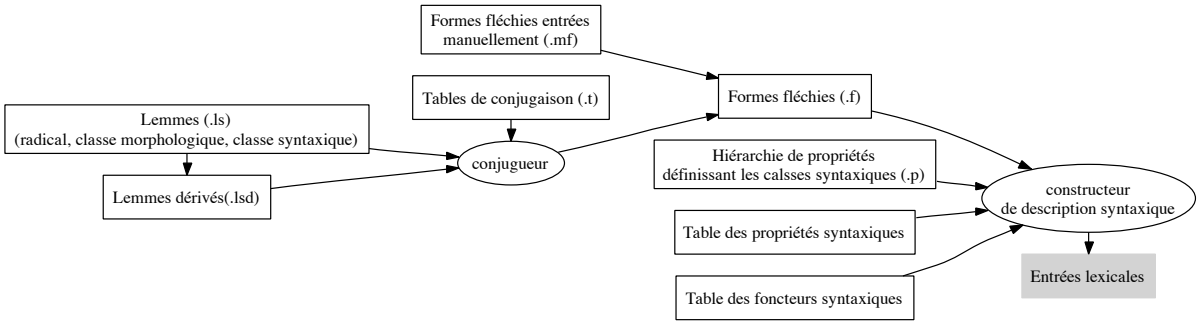


FIG. 1 – Architecture du lexique.

2.1 Phase morphologique

La première phase est donc *morphologique*. Le fichier de lemmes est donnée en entrée au conjugueur, qui a été construit comme décrit au chapitre précédent à partir de notre description morphologique du français. Le fichier de formes fléchies ainsi obtenu est complété par un fichier de formes fléchies entrées manuellement, pour gérer les variantes, abréviations courantes et autres phénomènes marginaux. Le fichier de formes fléchies associe à chaque entrée lexicale un lemme, une étiquette morphologique, la classe syntaxique de son lemme et une propriété syntaxique atomique issue de la morphologie, dite *propriété morphosyntaxique*. Cette dernière a pour rôle d'induire éventuellement des modifications dans les informations syntaxiques dont héritera la forme à partir de sa classe syntaxique. Ainsi, la propriété morphosyntaxique `Infinitive` attribuée à un infinitif verbal rend facultatif son sujet. Presque toutes les formes fléchies ont cependant un propriété morphosyntaxique qui est `Default`, laquelle n'induit aucun changement.

2.2 Phase syntaxique

La seconde phase est *syntaxique*. Comme indiqué précédemment, le fichier des lemmes associe à chaque lemme une classe syntaxique. Ces classes sont les nœuds d'un graphe (orienté et acyclique) d'héritage de propriétés syntaxiques atomiques, avec éventuellement des disjonctions. Par exemple, la classe verbale `@verbe_standard` est définie par une disjonction entre la classe `@verbe_transitif_direct` et la classe `@verbe_pronominal`. Ces deux classes n'étant pas définies elles-même via des disjonctions, chaque forme fléchie d'un lemme verbal comme *boire*, auquel est associée la classe `@verbe_standard`, correspondra à deux entrées du lexique : une entrée associée à une construction transitive directe (*Jean boit un verre*, et une entrée associée à une construction pronominale *Le vin se boit avec modération*.

Les propriétés atomiques sont définies de façon indépendante du graphe d'héritage. Elles construisent progressivement l'entrée lexicale.

Dans un troisième fichier, les propriétés syntaxiques sont définies par une ou plusieurs opérations parmi les suivantes :

- apport à la structure syntaxique d'une *macro* syntaxique définie dans un fichier séparé par une représentation en structures de traits avec partage de valeur,
- modification du `pred`, soit en indiquant qu'il est vide (pour les mots vides), soit en en changeant le lemme par rapport au lemme morphologique (pronoms personnels...), soit en appliquant sur le cadre de sous-catégorisation une expression régulière quelconque,
- définition de la catégorie de l'entrée lexicale,
- attribution d'un poids autre que le poids par défaut, pour (dé)favoriser l'emploi d'une entrée lexicale pendant la désambiguïsation de la sortie d'un l'analyseur syntaxique,
- dédoublement d'un patron en deux variantes différant par la propriété morphosyntaxique (on construit ainsi une structure syntaxique pour les infinitifs à partir de la valeur courante de la structure syntaxique standard ; les deux structures évoluent alors indépendamment l'une de l'autre).

Une telle structure hiérarchique permet de modifier aisément les informations associées à tous les verbes partageant une classe ou une propriété. Ceci facilite la correction d'erreurs et, plus généralement, améliore la maintenabilité du lexique. Du reste, c'est pour une part à partir de considérations relativement voisines qu'ont été introduites des concepts tels que la *hiérarchie de types HPSG* ou les *templates LFG*.

Mais l'intérêt principal de cette structure est plus profond. En effet, il s'avère que les classes syntaxiques que l'on construit ainsi regroupent des lemmes qui partagent également des propriétés communes au niveau de l'interface syntaxe-sémantique. On retrouve là, quoique par une démarche différente, des classes qui, pour les verbes, ont des points communs avec les classes de verbes de Levin (Levin, 1993).

3 Informations de sémantique lexicale

La forme intensionnelle du lexique associe aux lemmes une ou plusieurs classes syntaxiques, c'est-à-dire des classes qui associent au lemme des cadres de sous-catégorisation. Ces cadres regroupent trois types d'informations :

- une structure prédicative (au niveau des dépendances sémantiques) qui attribue au lemme des arguments sémantiques[13] (obligatoires ou facultatifs),

[13] Dont les types sont généralement appelés *rôles thématiques*.

– une structure syntaxique (au niveau des dépendances syntaxiques) qui lie un certain nombre d'arguments syntaxiques (obligatoires ou facultatifs) à un (ou zéro) argument sémantique[14],
– des informations sur les réalisations possibles de chaque argument *syntaxique*.

Il est naturel, à ce stade, de chercher à représenter également les *restrictions de sélection*, c'est-à-dire des informations sur les réalisations possibles de chaque argument *sémantique*.

En effet, les contraintes qui pèsent sur la réalisation d'un syntagme sont de deux ordres : elles peuvent être syntaxiques (on ne peut dire **Je retarde partir*) ou sémantique (on ne peut dire **Je mange mon départ*). Cependant, les contraintes sémantiques sont moins rigides que les contraintes syntaxiques[15], comme le montrent toute une gamme d'usages, qui vont des « sens figurés » aux métaphores et aux licences poétiques.

En première approximation, on peut considérer que ces contraintes sémantiques peuvent se représenter comme des contraintes reliant le lemme et la ou les têtes de chacun de ses arguments[16] si l'argument est non-prépositionnel, ou comme des contraintes reliant le lemme, la (ou les) préposition(s)[17] et les têtes des syntagmes gouvernés par cette (ces) préposition(s).

Pour toutes ces raisons, ces contraintes de restriction de sélection peuvent être représentées de deux façons différentes :

Sous forme symbolique : les propriétés syntaxiques atomiques associées par le graphe d'héritage aux classes syntaxiques peuvent être complétées par des propriétés sémantiques atomiques, représentées sous forme de clauses logiques. C'est ce que nous avons fait dans notre formalisme Méta-RCG décrit au chapitre 10. Un exemple de telle clause, où l'on représente par une forme du lemme courant, est PATIENT(@, *Head*) → COMESTIBLE(*Head*). Il est raisonnable, à titre d'exemple, de faire hériter le lemme d'infinitif *manger* d'une telle clause. Naturellement, des contraintes sémantiques représentées de façon symbolique et donc binaire peuvent n'être utilisées à grande échelle que si l'on dispose de mécanismes de tolérance permettant de ne pas toujours générer d'échec en cas de non-satisfaction.

Sous forme probabiliste : on peut construire directement un modèle probabiliste représentant la probabilité d'occurrence de toute dépendance sémantique possible (*lemme - rôle thématique - dépendant direct* ou *lemme - rôle thématique - préposition - dépendant indirect*). Ceci nécessite naturellement un corpus d'apprentissage, qui peut ne pas avoir été annoté manuellement, comme nous le verrons au chapitre suivant.

[14] C'est ici qu'interviennent des phénomènes comme la diathèse ou le sujet impersonnel.

[15] Les contraintes syntaxiques elles-mêmes ne sont pas complètement rigides. Mais c'est un autre problème, celui de la différence entre langue normée et langue performée, que nous aborderons de façon plus globale au chapitre 8.

[16] Il peut y avoir plusieurs têtes en cas de coordination.

[17] Ici encore il peut y en avoir plus d'une : *Ce livre a été écrit pour et avec Mademoiselle X.*

Dans les deux cas, et dans une perspective d'analyse automatique, on peut utiliser ces contraintes soit pendant l'analyse (pour filtrer plus vite les analyses inappropriées d'un point de vue sémantique) soit après l'analyse, pour désambiguïser le résultat.

La prise en compte pendant l'analyse de contraintes sémantiques symboliques est une des motivations principales du développement du formalisme Méta-RCG, qui fait l'objet du chapitre 10. En effet, une telle approche nécessite l'utilisation de formalismes aux propriétés adaptées (clôture par intersection, dite *non-linéarité*).

La prise en compte après l'analyse, au moment de la désambiguïsation, de contraintes exprimées sous forme probabiliste a fait l'objet de travaux préliminaires de notre part. Nous avons en particulier développé un module de désambiguïsation des forêts de dépendances produites par le système FRMG sur cette base. Ces travaux, qui sont complémentaires à ceux sur l'utilisation d'heuristiques de désambiguïsation (voir chapitre 9), ne sont pas détaillés dans ce document.

Chapitre 6

Acquisition automatique de lexiques morphologiques

Les travaux sur l'acquisition d'informations lexicales sont principalement orientés vers la terminologie (Daille, 2000), l'acquisition de collocations (Dunning, 1993) ou des propriétés de sous-catégorisation (Briscoe et Carroll, 1997), et présupposent toutes la disponibilité d'informations élémentaires telles que la catégorie (ou *partie du discours*) et la classe flexionnelle, habituellement appelées *lexique morphologique*. Cependant, l'acquisition de ces informations élémentaires n'est pas étudiée très souvent dans la littérature, en partie parce que ce problème est tout à la fois simple et largement résolu pour l'anglais, et bien qu'il soit un prérequis à l'acquisition d'informations plus avancées. Pour des langues telles que le français, les ressources lexicales à large couverture ne sont pas nombreuses, et ne sont pas toujours librement disponibles.

Notre approche permet d'extraire d'un corpus important une liste de lemmes avec un certain nombre d'informations qui leur sont associées (partie du discours, classe morphologique, ainsi que le préfixe lorsque c'est pertinent). Le lexique obtenu peut alors être comparé à d'autres lexiques disponibles, souvent commerciaux ou munis de licences restrictives, tels que la liste d'entrées lexicales extractible des bases de données de l'ATILF (Dendien et Pierrel, 2003), MulText (Ide et Véronis, 1994), ou ABU (ABU, 1999). Ces lexiques ont été principalement construits par des lexicographes humains. Pour cette raison, ils ont été coûteux à développer et sont sujets à des erreurs et à des incomplétudes. Les résultats de nos travaux sont complémentaires à ces lexiques, puisque de tels problèmes peuvent être partiellement évités par la mécanisation du processus, tout en ayant d'autres inconvénients, comme la sur-génération. Notre méthode reposant sur des statistiques de corpus, elle est à mettre en rapport avec les techniques d'extraction terminologique. Cependant, notre objectif n'est pas d'acquérir des termes (qui peuvent couvrir plusieurs mots) mais des entrées lexicales.

Notre méthode repose sur la disponibilité d'un corpus brut de taille raisonnable[1] de la langue étudiée, ainsi que d'une description morphologique qui permette à la fois la flexion de lemmes et la lemmatisation ambiguë de formes. Elle sait tirer parti de la présence d'informations de morphologie dérivationnelle. Nous l'avons mise en œuvre à l'aide des descriptions morphologiques du français et du slovaque décrites au chapitre 4 dans le formalisme que nous y avons présenté. L'intérêt de l'application au français est directement lié à l'utilisation du lexique obtenu pour l'apprentissage d'informations syntaxiques et sémantiques mais surtout pour l'analyse automatique dans nos systèmes décrits dans la dernière partie de ce document. L'intérêt de l'application au slovaque, pour le moment, est double : elle démontre l'efficacité de notre méthode pour une langue à la morphologie assez différente de celle du français, tirant parti de la richesse de la morphologie de cette langue pour apprendre un lexique non restreint aux seuls verbes, mais elle montre aussi son utilité pour le développement rapide de lexiques *ex nihilo* (à l'exception du corpus brut) pour des langues peu dotées.

C'est une première version de notre méthode, développée en collaboration avec Lionel Clément et Bernard Lang, qui a donné lieu à l'acquisition du lexique Le*fff* 1 des verbes français (Clément *et al.*, 2004; Clément et Sagot, 2004). Une nouvelle version, qui repose sur un modèle probabiliste plus convaincant et plus sophistiqué et une implémentation entièrement réécrite, a permis l'acquisition d'un lexique du slovaque, toutes catégories ouvertes confondues (Sagot, 2005a). C'est cette dernière version qui est décrite dans ce chapitre. En parallèle à nos premières expériences, et indépendamment de nous, Oliver et ses collaborateurs ont mené des travaux comparables Oliver *et al.* (2003); Oliver et Tadić (2004) pour acquérir ou enrichir des lexiques de verbes, noms et adjectifs. Par rapport aux expériences d'Oliver et à nos premiers travaux sur ce sujet, la méthode présentée ici est une amélioration sur au moins trois points : la prise en compte de la morphologie dérivationnelle, le formalisme utilisé pour les descriptions morphologiques, et l'utilisation d'un modèle probabiliste approprié.

1 Modèle probabiliste et méthodologie d'acquisition

Le schéma général de notre méthodologie d'acquisition de lexiques morphologiques peut se résumer à l'itération, un nombre quelconque de fois, de la séquence suivante :

1. Lemmatisation ambiguë du corpus On génère tous les lemmes possibles susceptibles d'être à l'origine des formes présentes dans le corpus.

2. Classement des lemmes hypothétiques On ordonne ces lemmes selon leur vraisemblance à l'aide d'un modèle probabiliste qui met en œuvre le principe suivant : un lemme est

[1] En réalité, les lemmes que l'on peut apprendre aisément sont ceux attestés plus de 3 à 6 fois dans corpus. Par conséquent, plus le corpus est volumineux, plus il sera possible d'apprendre un lexique important.

d'autant plus vraisemblable que de nombreuses formes différentes sont attestées dans le corpus, en proportions respectives cohérentes avec ce qui est observé pour les autres lemmes ayant la même catégorie[2]. De plus, les liens de morphologie dérivationnelle qui peuvent exister entre deux lemmes tend à renforcer l'un si l'autre est très vraisemblable.

3. Validation manuelle des lemmes les plus vraisemblables On produit automatiquement à partir des lemmes les plus vraisemblables une interface de validation manuelle permettant de remplir progressivement un lexique validé. Ce lexique validé est utilisé par les itérations suivantes de la séquence complète pour ajuster les paramètres du modèle probabiliste.

Ces trois phases sont décrites dans les paragraphes suivants.

Il est bien sûr envisageable de n'exécuter qu'une seule fois les deux premières phases de cette séquence, de se passer de validation manuelle, et de ne garder que les lemmes dont la vraisemblance (et la fréquence) dépasse un certain seuil. Le lexique obtenu est alors incomplet ou bruité (suivant la valeur du seuil), mais il peut servir de base à l'acquisition complètement automatique de ressources supplémentaires, comme un étiqueteur morphosyntaxique (voir plus bas). Une telle ressource peut en effet guider une nouvelle itération de la séquence d'apprentissage morphologique, en fournissant pour chaque forme des hypothèses pondérées sur sa catégorie. Cette nouvelle itération verra ainsi ses résultats améliorés par rapport à l'itération précédente, et ce sans aucune intervention manuelle. Nous pensons toutefois qu'il est indispensable de passer par une validation humaine, quitte à acquérir un étiqueteur après quelques itérations de la séquence. Le guidage de nouvelles itérations de la séquence par un tel étiqueteur permet en effet de combiner les informations apportées par la validation manuelle à celles apportées par l'étiqueteur.

Comme vu plus haut, nous appelons *forme décorée* une forme munie de son lemme et de ses traits morphologiques tels que la catégorie, le genre, le nombre, le temps, la personne, etc. Une classe flexionnelle est un opérateur qui calcule un ensemble de formes fléchies à partir d'une entrée unique dite *forme canonique*[3]. Nous appelons *lemme* un radical muni d'une classe flexionnelle. Ainsi, la forme verbale *manger* a pour lemme dans notre lexique le couple *(mang, v-er)*, où *v-er* est la classe flexionnelle (unique) des verbes dits du premier groupe (cf. chapitre 4). Notre objectif est donc d'acquérir un lexique des formes fléchies qui les relie à leur(s) lemme(s) en n'utilisant comme données d'entrée qu'un corpus brut et une description morphologique de la langue.

[2] À ce stade, on peut avoir l'impression que le serpent se mort la queue. En réalité, on peut utiliser cette idée telle quelle à condition de l'implémenter à l'aide d'un algorithme de point fixe. C'est ce que nous faisons effectivement.

[3] Cette forme canonique peut en toute rigueur ne pas être une des formes produites par la classe flexionnelle. En pratique, c'est rarement le cas, même si nous en aurons un exemple plus loin.

1.1 Génération et flexion des lemmes hypothétiques

Après l'avoir segmenté, on élimine du corpus les formes présentes dans une liste constituée manuellement de mots appartenant à des classes fermées (pronoms, certains adverbes, prépositions, etc.)[4].

Une fois extraites les formes pertinentes du corpus, ainsi que leur nombre d'occurrences, il faut construire l'ensemble de lemmes hypothétiques : il s'agit de l'ensemble des lemmes que l'on peut construire en accord avec la description morphologique de la langue et qui ont parmi leurs formes au moins une qui est attestée dans le corpus (parmi les formes pertinentes)[5]. Ceci est réalisé par le lemmatiseur ambigu construit à partir de la description morphologique (cf. Chapitre 4). On élimine de ces lemmes ceux qui ont été manuellement indiqués comme faux par les éventuelles itérations précédentes de la séquence d'apprentissage. Les lemmes hypothétiques ainsi construits qui ne sont pas des hapax[6] sont alors fléchis. Le résultat est donc un ensemble volumineux de formes fléchies, c'est-à-dire de quadruplets *(radical, classe morphologique, forme non-décorée, étiquette morphosyntaxique)*. A titre d'illustration, à partir de seulement 19 942 formes pertinentes différentes extraites de notre corpus du slovaque de 160 000 mots, la lemmatisation ambiguë génère 844 133 de ces quadruplets issus de 68 870 lemmes non-hapax (parmi 86 763 en tout) ainsi que 2 985 liens de dérivation entre ces lemmes[7].

1.2 Classement des lemmes hypothétiques

À ce point du processus, le but est de classer les lemmes hypothétiques générés à l'étape précédente de telle façon que les lemmes les mieux classés soient (dans l'idéal) tous corrects, et que les moins biens classés soient tous faux. Pour cela, il faut disposer d'un moyen de modéliser

[4]La constitution de cette liste est partiellement automatisée par la phase de validation manuelle décrite ci-dessous. En effet, on peut signaler que certains lemmes hypothétiques ont été générés à partir de formes appartenant à des classes fermées. Ces formes seront filtrées lors des itérations suivantes de la séquence d'apprentissage.

[5]Il y en a toujours au moins une, puisque l'opération de lemmatisation ambiguë est théoriquement exactement l'opération (ensembliste) inverse de l'opération de flexion.

[6]C'est-à-dire les lemmes attestés par deux occurrences de la même forme ou deux formes différentes, ou bien dérivés d'un lemme qui satisfait la condition précédente.

[7]Ces chiffres sont différents de ceux indiqués dans Sagot (2005a) pour deux raisons : le nombre de lemmes générés est inférieur, malgré une augmentation à la marge de la couverture de notre description morphologique du slovaque, en raison d'une plus grande factorisation des classes morphologiques et de contraintes plus précises sur les radicaux. Le nombre de formes fléchies générées est très inférieur en raison d'un traitement plus performant de la dérivation morphologique : dans la version utilisée par Sagot (2005a), les formes fléchies d'un lemme dérivé étaient considérées comme des formes fléchies du lemme d'origine, avec une étiquette morphologique complexe permettant de reconstituer le lemme dérivé. Cela induisait une duplication des informations : tout lemme dérivé étant aussi un lemme hypothétique à part entière, ses formes étaient générées par ce lemme mais aussi par tous les lemmes dont il était un dérivé potentiel. La version actuelle ne génère les formes qu'une seule fois, en tant que formes du lemme dérivé, et les relations de dérivation sont représentées à part par des triplets *(lemme d'origine, lemme dérivé, type de dérivation)*.

la plausibilité de chaque lemme (c'est-à-dire la probabilité qu'il soit correct). Nous avons choisi de calculer la vraisemblance de chaque lemme conditionnellement au corpus. Puisque nous n'avons pas les moyens de la calculer directement, nous utilisons un algorithme itératif de point-fixe qui itère un certain nombre de fois une série de calculs selon le modèle suivant.

1.2.1 Idée générale

L'idée générale est la suivante. Nous allons tenter d'attribuer à chaque lemme une plausibilité, c'est-à-dire la probabilité qu'il soit correct. Les lemmes dont la plausibilité sera nulle seront donc rejetés. Pour cela, nous allons partir d'une première estimation, calculée au plus juste mais de façon arbitraire, de ces plausibilités. Cette première estimation nous permet de calculer, pour chaque forme, la probabilité qu'elle a d'être issue d'un lemme donné. On peut alors en déduire, pour chaque forme (attestée ou non) de chaque lemme hypothétique, si elle a tendance à confirmer ou à infirmer la plausibilité de chaque lemme dont elle peut être issue. Ces tendances permettent alors de calculer une estimation nouvelle (et, espérons-le, meilleure) de la plausibilité de chaque lemme. Et on peut recommencer autant de fois que nécessaire jusqu'à ce que les plausibilités se soient stabilisées d'une fois sur l'autre.

Prenons un exemple. Considérons le corpus est composé des trois mots *manger*, *mange* et *mangerons*. Supposons que les seuls lemmes hypothétiques sont les deux lemmes verbaux du premier groupe dont les infinitifs sont *manger* et *mangerer*. Supposons enfin que l'estimation initiale rend ces deux lemmes équiprobables (notre modèle conduirait en fait à un autre résultat, du reste meilleur, mais peu importe ici). Il est certain que les formes *manger* et *mange* sont des formes fléchies du premier lemme, alors que pour *mangerons* les deux lemmes sont possibles, de façon équiprobable. Les « tendances » évoquées plus haut sont alors les suivantes[8] : les formes *manger* et *mange* rendent le lemme d'infinitif *manger* certain, car il est le seul à même d'avoir ces deux formes pour formes fléchies. La forme *mangerons* n'induit à ce stade aucune tendance, puisqu'on a pour le moment aucun élément permettant de l'attribuer à l'un ou l'autre lemme. Après cette première itération du calcul, on a donc un lemme certain, d'infinitif *manger*, et d'un lemme à la plausibilité intermédiaire, d'infinitif *mangerer*. Ce résultat induit, lors de la seconde étape, à rendre bien plus grande la probabilité que la forme *mangerons* vienne du lemme d'infinitif *manger*, déjà certain, par rapport à celle qu'il vienne du lemme d'infinitif *mangerer*, mal confirmé. Ceci rend la plausibilité de *mangerer* encore plus faible. Après quelques itérations du processus, le lemme d'infinitif *mangerer* est complètement éliminé, et le lemme correct est le seul à être conservé.

[8] Pour simplifier, on laisse de côté, dans cet exemple, l'influence des formes non attestées.

1.2.2 Modèle et algorithme

Appelons *forme* une forme fléchie pertinente f et $occ(f)$ le nombre d'occurrences de f. On appelle *token* une occurrence particulière d'une forme. Une forme attestée $occ(f)$ fois est donc la forme de $occ(f)$ tokens distincts.

Considérons l'expérience suivante : on choisit au hasard un token. La probabilité que la forme de ce token soit la forme f est $P(f) = occ(f)/n_{tot}$, où n_{tot} le nombre total de tokens (le nombre de mots une fois retirés les mots erronés ou appartenant à des classe fermée). Les valeurs de $occ(f)$ et de n_{tot} sont bien sûr toutes deux connues. Soit $\mathcal{L}_f$ l'ensemble des lemmes hypothétiques qui ont f parmi leurs formes fléchies. On appelle $P(l)$ la probabilité que le token choisi soit une forme fléchie du lemme l, et $P(f|l)$ la probabilité conditionnelle d'avoir choisi une occurrence de f sachant que l'on a choisi une forme fléchie de l.

Supposons qu'on commence la $i+1$-ième étape de l'algorithme de point fixe. L'étape précédente a permis de calculer pour chaque lemme des estimations $P_i(f|l)$ et $P_i(l)$ des probabilités $P(f|l)$ et $P(l)$. Le but est donc de mettre à jour notre estimation de ces probabilités, en calculant des estimations $P_{i+1}(f|l)$ et $P_{i+1}(l)$. L'estimation des $P_0(f|l)$ et des $P_0(l)$ est détaillée plus bas.

Pour calculer ces nouvelles estimations, nous allons utiliser un certain nombre de formules qui sont vraies sur les probabilités réelles (lorsque i tend vers l'infini, en cas de convergence), en remplaçant dans ces formules les probabilités par nos estimations de rang i ou $i + 1$.

On écrit tout d'abord :

$$P_{i+1}(f|l) = \frac{P(f) - \sum_{l' \in \mathcal{L}_f, l' \neq l} P_i(l') P_i(f|l')}{P_i(l)}.$$

Il nous reste donc à calculer $P_{i+1}(l)$. Pour cela, nous allons estimer la probabilité $\Pi(l)$ (tout à fait distincte de $P(l)$) que l soit un lemme correct[9]. L'idée est la suivante. Nous allons examiner la contribution à cette probabilité de chacune des formes, attestées ou non. Les formes de l attestées dans le corpus vont donc augmenter $\Pi(l)$, alors que ses formes non attestées vont diminuer $\Pi(l)$. Pour faciliter cette approche, on introduit l'*odds* (ou la *cote*[10]) d'un lemme l défini par

$$O_l = \frac{\Pi(l)}{1 - \Pi(l)}.$$

[9] Bien sûr, si certains lemmes ont été déjà validés, soit que l'on parte d'un lexique non-vide, soit que l'on ait déjà effectué une ou plusieurs étapes de validation manuelle, alors $\Pi(l) = 1$ pour tous les lemmes l déjà connus.

[10] C'est effectivement la traduction de l'anglais *odds*. Mais le terme anglais est utilisé très majoritairement dans la littérature.

En effet, la formule de Bayes peut être exprimée comme une formule sur les *odds* de la façon suivante [11] : la prise en compte d'une nouvelle information I (ici, le fait que f est ou n'est pas attesté dans le corpus) multiplie l'*odds* de l'hypothèse « le lemme l est correct » par l'*odds ratio* (ou *rapport de cotes*) $OR_l(f)$ suivant :

$$OR_{i+1}(l|f) = \frac{P(I\text{si } l \text{ est correct})}{P(I \text{ si } l \text{ est faux})}.$$

L'information qu'une forme est attestée étant aussi importante que celle qu'une forme ne l'est pas, on doit faire ce calcul pour toutes les formes de l, qu'elles soient attestées ou non. Si f est attestée, la formule précédente s'écrit :

$$OR_{i+1}(l|f) = \frac{\sum_{l\in\mathcal{L}_f} P_i(f,l)}{\sum_{l'\in\mathcal{L}_f, l'\neq l} P_i(f,l)}.$$

Si f n'est pas attestée, il faut donc évaluer la probabilité de ne pas trouver la forme f dans le corpus sachant ce même corpus, d'une part si l est correct et d'autre part si l est faux, puisque l'*odds ratio* est le rapport entre ces deux probabilités. On peut montrer que ce rapport est égal à la probabilité de n'avoir pas choisi f sachant que l'on a choisi une forme issue de l. Pour estimer cette probabilité à l'étape $i+1$, on utilise la probabilité que la forme choisie ait reçu un suffixe flexionnel donné sachant la classe morphologique de l (ceci est possible grâce à la connaissance des probabilités de la forme $P_i(f'|l)$ et des informations morphologiques associées aux formes f' issues de l).

[11] En voici la démonstration. Soit A et B deux événements. La formule de Bayes, sous sa forme classique, s'énonce :

$$P(A|B) = \frac{P(A,B)}{P(B)}.$$

Soit maintenant un ensemble de données E et une hypothèse H concernant ces données. L'odds de l'hypothèse H est définie par $O(H) = P(H)/P(\neg H)$. L'odds de l'hypothèse H conditionnée à E est donc exprimée par $O(H|E) = P(H|E)/P(\neg H|E)$. Le rapport entre ces deux odds (odds ratio), qui est le résultat $OR(H,E)$ de la prise en compte de E pour évaluer la probabilité de H, est donc :

$$OR(H,E) = \frac{O(H|E)}{O(H)} = \frac{P(H|E)/P(\neg H|E)}{P(H)/P(\neg H)} = \frac{P(H|E)}{P(H)} \cdot \frac{P(\neg H)}{P(\neg H|E)}$$

Or, puisque $P(E) = 1$, et par application de la formule de Bayes classique, $P(H|E) = \frac{P(H,E)}{P(E)} = P(H,E)$ (et de même pour $\neg H$. Et donc :

$$OR(H,E) = \frac{P(H,E)}{P(H)} \cdot \frac{P(\neg H)}{P(\neg H,E)}$$

D'où par application de la formule de Bayes classique :

$$OR(H,E) = \frac{P(E|H)}{P(E|\neg H)}.$$

C'est la formule utilisée dans cette partie.

Une fois tous les *odds ratio* de l calculés, on les multiplie à un *odd* initial (c'est-à-dire avant toute prise en compte des informations contenues dans le corpus) donné par $O^0(l) = 1$ (cela correspond à une probabilité initiale pour l d'être correct égale à $\Pi^0(l) = 1/2$). La seule exception est évidemment constituée par les lemmes déjà validés ou déjà connus, qui ont un *odd* infini ($\Pi^0(l) = 1$). Si l'on appelle $\mathcal{F}_l$ l'ensemble de toutes les formes fléchies de l, l'*odd* de l sachant le corpus est donc obtenu par : $O_{i+1}(l) = O^0(l) * \prod_{f \in \$mathcalF_l} O_{i+1}(l|f)$. En réalité, cette valeur est légèrement modifiée pour prendre en compte les relations de morphologie dérivationnelle (liens entre lemmes, préfixes,...).

On peut maintenant calculer la probabilité $\Pi_{i+1}(l) = O_{i+1}(l)/(1 + O_{i+1}(l))$ que l soit correct. Si l'on définit le nombre d'occurrences de l par la formule $occ_i(l) = \sum_{f \in \mathcal{F}_l} occ(f).P_i(l|f)$, on peut alors calculer $P_{i+1}(l)$, en écrivant $P_{i+1}(l) = occ(l).\Pi_{i+1}(l)/n_{tot}$. En effet, cette formule doit être vraie après convergence, puisqu'alors $\Pi_{i=\infty}(l)$ ne devrait valoir que 0 ou 1. Son utilisation à chaque itération permet au modèle de converger, bien que nous ne disposions d'aucune preuve formelle de la convergence du processus, et encore moins du fait que le processus ne donne que des $\Pi(l)$ valant 0 ou 1. Cependant, les résultats empiriques valident cette méthode.

Une fois $P_{i+1}(l)$ calculé, on peut itérer cette série de calculs jusqu'à convergence empirique (en pratique, il faut i_{max}=10 à 15 itérations). Après la dernière itération, les lemmes sont ordonnés selon leur probabilité $\Pi_{i_{max}}(l)$ d'être corrects. Les lemmes qui ont une probabilité égale à 1 sont ordonnés selon $occ_{i_{max}}(l)$. Lorsque c'est pertinent, les lemmes dont on a acquis le fait qu'ils sont dérivés d'un autre lemme sont associés à leur lemme d'origine.

Pour mettre en œuvre cet algorithme de point fixe, il nous faut donc encore disposer d'estimations aussi pertinentes que possible des probabilités $P_0(l)$ et $P_0(f|l)$. L'idée est de calculer une estimation initiale $P_0(l|f)$ de la probabilité $P(l|f)$ en disant successivement :

1. les différentes étiquettes morphosyntaxiques e possibles pour une forme f sont équiprobables[12],
2. les lemmes possibles pour un couple (f, e) sont équiprobables.

Donc si l'on note $\mathcal{E}_f$ l'ensemble des étiquettes morphosyntaxiques possibles pour la forme f, et $\mathcal{L}_{f,e}$ l'ensemble des lemmes ayant f parmi leurs formes d'étiquette e, on écrit donc :

$$P_0(l|f) = \sum_{e \in \mathcal{E}_f} P(l|e, f) \cdot P(e|f) = \sum_{e \in \mathcal{E}_f} \frac{1}{card(\mathcal{L}_{f,e})} \cdot \frac{1}{card(\mathcal{E}_f)}.$$

[12]Sauf si on utilise les résultats d'un tagger non-déterministe, typiquement le tagger endogène appris automatiquement à partir d'une session précédente d'apprentissage du lexique (cf plus bas), ou un tagger exogène (c'est-à-dire qui repose sur une autre ressource, qu'il s'agisse d'un corpus annoté manuellement ou de règles écrites à la main). En effet, dans ce cas, on dispose d'une estimation plus fine de la probabilité des étiquettes sachant la forme.

Ceci permet d'obtenir $P_0(l) = \sum_{f \in \mathcal{F}_l} P_0(l|f) \cdot P(f)$ puis, par la formule de Bayes, $P_0(f|l) = \frac{P_0(l|f) \cdot P(f)}{P_0(l)}$.

1.3 Validation manuelle

La première exécution de l'algorithme présenté ci-dessus conduit à un résultat qui est bon mais qui comporte néanmoins du bruit, principalement à cause de mots mal orthographiés[13] ou identifiés à tort comme pertinents (voire mal taggés en cas d'utilisation d'un taggeur). Nous les appelons les *formes incorrectes*. Ceci peut avoir deux types de conséquences :

- une ou plusieurs formes incorrectes peuvent rendre probable un lemme faux qui n'a que des formes incorrectes ou non-attestées (on parle de *lemmes issus de formes incorrectes*), comme par exemple **démesurer*, qui repose sur le nom *démesure* et des formes de l'adjectif *démesuré* tout en étant renforcé par le verbe *mesurer* dont il ne diffère que par un préfixe correct,
- une ou plusieurs formes incorrectes peuvent s'associer à des formes correctes pour donner à un lemme correct une classe morphologique incorrecte et/ou un radical incorrect (on parle de *lemmes mal devinés*), comme par exemple *reconstruir*, qui est généré par de nombreuses formes du lemme verbal correct *reconstruire* ainsi que par une occurrence de la forme incorrecte *reconstruir*.

Naturellement, les erreurs peuvent aussi être la conséquence de mauvaises décisions prises par l'algorithme d'apprentissage lui-même, introduisant ainsi des lemmes incorrects avec une probabilité $\Pi(l) = 1$ bien qu'ils soient issus de formes toutes correctes. C'est en particulier le cas quand un lemme est peu attesté, le faible nombre de formes qui en sont issues ne permettant pas toujours de faire les bons choix en connaissance de cause. Il en est ainsi du lemme verbal incorrect *rendormer* appris lors de la première exécution du processus en lieu et place du lemme correct *rendormir*. Cependant, une telle situation arrive très rarement pour peu que le lemme soit attesté plus de quelques fois (voir 6.2).

Le processus de validation manuelle est effectué sur la liste ordonnée des lemmes générée à l'étape précédente. Le but de cette étape est de répartir les lemmes les mieux classés entre les catégories suivantes :

- les lemmes corrects, ajoutés au lexique,
- les lemmes mal devinés,

[13] Les mots mal orthographiés posent un problème délicat dans la mesure où ils sont inévitables dans un corpus. De plus, certaines fautes d'orthographe sont systématiques, faisant ainsi apparaître des ensembles cohérents de formes mal orthographiées, qui sont interprétées avec un bon niveau de confiance comme dérivant d'un lemme commun mal orthographié. Il en est ainsi par exemple du lemme faux *appeller*, bien attesté par toutes les formes à deux *l* du lemme correct *appeler* ainsi que par toute une série de formes mal orthographiées comme *appellons*, *appeller*, *appellait*, etc…

byť	v-byť
európsky, a, e	adj-l
ten článok , bez článku , o článku (lok) , tie články , s/so článkami	nc-mid:Gu:0:Leu:ami
ten štát , bez štátu , o štáte (lok) , tie štáty, s/so štátmi	nc-mid:Gu:0:Leu:mi
ta únia, tie únie , sto únií, o úniách (lok)	nc-fiv
členský, á, é	adj-b
mať, mám, majú	v-ať
ta komisia, tie komisie , sto komisií, o komisiách (lok)	nc-fiv
to právo	nc-no
môcť, môžem, môžu, mohol	v-môcť
ta rada, tie rady	nc-fdv
ta oblasť , tie oblasti	nc-fm2
to opatrenie	nc-nie
ta návrha, tie návrhy	nc-fdv
ta krajina, tie krajiny	nc-fdv
spoločný, á, é	adj-b
to rámce	nc-ne
ta politika, tie politiky	nc-fdv
iný, á, é	adj-b
ten cieľ , tí cieli , s/so cieľmi	nc-mac:i:0:mi
ten rok , bez roka , o roku (lok) , tie roky, s/so rokmi	nc-mid:Ga:0:Leu:mi
ta voda, tie vody	nc-fdv

FIG. 1 – Première page web de validation du lexique du slovaque après la première itération de l'algorithme, et avant la première session de validation manuelle. Les lemmes sont ordonnés en fonction de leur plausibilité et de leur fréquence. Le lemme le mieux attesté est donc *byť* (être).

- les lemmes issus de formes correctes et incorrectes, dont on retire du corpus la forme les ayant rendus certains,
- les lemmes issus de formes incorrectes, dont on retire du corpus toutes les formes.

Cette étape de validation peut être effectuée très rapidement et sans connaissances linguistiques spécifiques, à l'aide d'une interface de validation présentant sous forme de page web les lemmes hypothétiques ainsi que certaines formes significatives données dans un contexte raisonnable, ainsi que spécifié par l'élément `show` de la description morphologique (cf. chapitre 4). Les quatre classes précédentes correspondent à quatre « boutons radio » associés à chaque lemme et parmi lesquels il faut choisir, qui sont respectivement de couleur verte, rouge, grise et noire. Un extrait d'une telle page web est donné dans la figure 1.

2 Application au français et au slovaque

2.1 Verbes français

Comme indiqué précédemment, la première expérience d'acquisition de lexique morphologique que nous avons menée a été effectuée *ex nihilo* à l'aide d'une version préliminaire de la méthode décrite dans ce chapitre (Clément *et al.*, 2004), sur un corpus journalistique (issu du *Monde diplomatique*) de 25 millions de mots. Quasiment aucune information de morphologie dérivationnelle n'était exploitée par cette version préliminaire. Pour cette raison, et parce que seuls les lemmes verbaux ont une morphologie riche en français, nous avons restreint l'apprentissage aux seuls lemmes verbaux. Pour cela, nous avons identifié les formes verbales à l'aide d'un étiqueteur morphosyntaxique (nous n'avons utilisé aucune autre information produite par cet étiqueteur). Pour ne pas acquérir les informations lexicales éventuellement présentes (de façon implicite) dans de nombreux étiqueteurs, nous avons entraîné l'étiqueteur (exogène) `TreeTagger` sur un corpus étiqueté de 700 000 mots (le corpus de Paris 7) avec un lexique qui ne comportait que des lemmes appartenant à des classes fermées (articles, pronoms, prépositions, conjonctions, etc.) mais aucun appartenant à une classe ouverte (noms, verbes, adjectifs, adverbes)[14]. Tout autre étiqueteur ne reposant pas sur un lexique des classes ouvertes aurait pu être utilisé.

Il est bien clair que cette phase n'est toutefois pas entièrement satisfaisante, puisqu'elle repose indirectement sur un corpus annoté. Dans l'état de notre système au moment de cette expérience, se passer de l'étiqueteur ne permettait que l'apprentissage des lemmes verbaux les mieux attestés (c'est-à-dire les plus fréquents). Mais l'ajout récent de nombreuses informations de morphologie dérivationnelle nous fait penser que c'est possible, surtout si on utilise la technique, elle aussi plus récente que cette expérience, d'apprentissage automatique d'un étiqueteur endogène à partir des résultats de notre méthode d'apprentissage de lexique morphologique (voir plus bas). En effet, la morphologie verbale du français est suffisamment riche pour pouvoir apprendre sans étiqueteur les lemmes verbaux les plus fréquents. La morphologie dérivationnelle permet alors à tous les noms, adjectifs et adverbes directement ou indirectement dérivés des ces lemmes verbaux d'être appris également. La couverture du lexique ainsi obtenu devrait suffire à l'apprentissage d'un étiqueteur endogène de qualité acceptable, rendant ainsi entièrement endogène le processus d'apprentissage. Nos premières expériences confirment cette intuition, mais nous n'avons pas encore pu la mettre en œuvre pour un apprentissage *ex nihilo* et à grande échelle d'un lexique du français.

[14] On a donc appris des probabilités de transitions pour toutes les étiquettes, mais des probabilités d'émission pertinentes seulement pour les classes fermées (les classes ouvertes étant toutes regroupées sous un même « mot inconnu », elles reçoivent des probabilités d'émission uniformes).

Le résultat de l'expérience décrite ici montre que la validation manuelle est grandement facilitée par la qualité des résultats produits par l'algorithme de point fixe. Après la première exécution de cet algorithme, et donc avant toute validation manuelle, le 100ème lemme incorrect le mieux classé dans la liste des lemmes possibles est classé 3 337ème (soit un taux de précision de 97,0% sur les 3 336 lemmes les mieux classés)[15]. 7 500 verbes ont une probabilité d'être corrects égale à 1, parmi lesquels 5 250 ont un nombre d'occurrences strictement supérieur à 1 (en effet, cette version du système d'apprentissage ne filtrait pas les hapax). Il est intéressant de noter que presque tous les lemmes incorrects bien classés proviennent de fautes d'orthographes dans le corpus ou d'erreurs de l'étiqueteur (respectivement 19% et 76% des 100 lemmes incorrects les mieux classés), mais presque pas de mauvais choix de l'algorithme d'apprentissage (5%). La table 1 donne le nombre de lemmes corrects et incorrects de probabilité $\Pi(l) > 0.97$ en fonction de leur nombre d'occurrences[16] après la première exécution de l'algorithme (et donc, une fois encore, avant toute validation manuelle)[17]. Ces résultats ne sont pas faits pour être parfaits, puisque seuls les lemmes les mieux classés sont proposés à la validation manuelle après cette première itération, et ne sont donnés ici qu'à titre d'illustration. En effet, de meilleurs résultats dès la première itération auraient pu être obtenus facilement en augmentant les contraintes sur les formes pertinentes (par exemple, on aurait pu rejeter toute forme n'apparaissant qu'une seule fois, ou toute forme taggée comme forme verbale avec une fréquence inférieure à un certain seuil). Mais cela aurait été au détriment de la couverture de la méthode, et de nombreux lemmes n'auraient pu être appris. Ici, la minimisation du silence est prioritaire par rapport à celle du bruit, puisque la validation manuelle fait diminuer le bruit au fil des itérations de la séquence complète.

Après seulement quelques heures de validation manuelle et d'itérations de la séquence complète d'apprentissage, nous avons obtenu un lexique de presque 5 000 lemmes générant approximativement 200 000 formes. Parmi ces lemmes, un certain nombre sont absents de la référence standard qu'est Bescherelle (1990), bien qu'ils soient attestés dans un corpus ne relevant pas d'une langue de spécialité[18]. À l'inverse, environ 1 500 verbes de Bescherelle (1990) ne sont pas acquis, principalement parce qu'ils ne sont pas attestés dans le corpus. On peut se convaincre

[15] On notera que ces chiffres mesurent certes la pertinence de l'approche, mais aussi la taille et la qualité du corpus : un corpus plus petit ou de moins bonne qualité (fautes d'orthographe plus fréquentes,...) aurait inévitablement conduit à une précision inférieure.

[16] Pour les lemmes à préfixes, les nombres d'occurrences ont été augmentés artificiellement de 10 unités. Voir la note précédente.

[17] Les lemmes considérés comme corrects sont ceux qui sont présents dans le lexique après plusieurs itération du cycle complet d'apprentissage et de validation manuelle.

[18] Quelques exemples parmi ceux commençant par la lettre *a* : *américaniser*, *arcbouter* (alors que seul *arc-bouter* est dans Bescherelle (1990), *artificialiser*, *aryaniser*, de nombreux dérivés en *auto-* comme *(s') autoaccuser*, *autocélébrer*, *(s') autodétruire*,..., *aéroporter*, ... Quelques autres exemples : *cloner*, *cliquer*, *télécharger*, *paramétrer*, *encrypter*, *zapper*, *zoomer*,...

Après une itération, lemmes l tels que $\Pi(l) > 0.97$		
$occ(l)$	Lemmes corrects	Lemmes incorrects
≥ 20	2 831	66
≥ 10	3 403	164
≥ 5	3 771	277
≥ 2	4 293	674
> 1	4 390	807

TAB. 1 – Premiers résultats intermédiaires de l'apprentissage automatique des lemmes verbaux du français présents dans un corpus journalistique de 25 millions de mots. Ces résultats sont obtenus avant toute phase de validation manuelle permettant de rejeter les lemmes issus de formes mal orthographiées ou mal étiquetées.

de ce dernier résultat à l'aide du test suivant. Pour évaluer nos résultats, nous avons étiqueté notre corpus en utilisant une version de `TreeTagger` que nous avons entraînée avec un autre lexique morphologique librement utilisable, celui de ABU (1999). Seuls 1,64% des mots étiquetés comme formes verbales par cet étiqueteur ne sont pas couverts par notre lexique, ce qui est à comparer au taux moyen d'erreurs de ce même étiqueteur, qui est de 3,25%. Une vérification manuelle rapide montre qu'en effet ces 1,64% sont presque exclusivement dûs à des erreurs de l'étiqueteur[19].

Le lexique ainsi obtenu a constitué le premier noyau de notre Lexique des Formes Fléchies du Français, le Le*fff*, dont il est la première version (Le*fff* 1). Cette version a été mise en ligne sous licence libre au printemps 2004 sur le site web du Le*fff* (`http://www.lefff.net/`).

2.2 Slovaque

Pour des généralités sur la langue slovaque, on se reportera à la figure 1.

Nous avons utilisé pour nos expériences sur le slovaque un corpus relativement petit de 150 000 mots représentant 20 000 tokens différents. Ce corpus contient des textes produits par l'Union Européenne (dont le Projet de Traité Constitutionnel) et des articles libres d'utilisation trouvés sur l'Internet (tant scientifiques que journalistiques).

En utilisant la méthode décrite dans ce chapitre, et après quelques itérations de la séquence d'apprentissage (y compris 3 à 4 heures seulement de temps cumulé de validation manuelle), nous avons acquis un lexique du slovaque contenant environ 4 000 entrées générant plus de

[19] A cette même époque, et donc avec la même version de notre système d'apprentissage, nous avons également mené des expériences très préliminaires d'apprentissage de lemmes adjectivaux sur un corpus de français de spécialité, en l'espèce un corpus de botanique de 3 millions de mots. Après la première passe, parmi les 1 000 adjectifs les mieux classés, 350 sont inconnus du lexique de ABU (1999), et presque tous étaient de fait des lemmes spécialisés corrects.

66 500 formes fléchies soit 40 000 tokens différents[20]. Ces formes couvrent 88% des formes attestées dans le corpus qui n'ont pas été manuellement éliminées (comme les prépositions, les adverbes, les particules, les pronoms, etc...). Par construction, la précision du lexique est de 100% puisque notre lexique est validé manuellement.

Quoique préliminaires[21], ces résultats sont très prometteurs, en particulier au vu du temps de validation relativement court. Cela montre d'une part la faisabilité d'un processus automatique d'acquisition lexicale *ex nihilo*, même sur un corpus relativement restreint. Ceci est dû en partie au fait que le slovaque ait une morphologie riche, et pourrait donc être reproduit facilement sur n'importe quelle langue pour laquelle on dispose d'un module morphologique permettant la flexion et la lemmatisation ambiguë. D'autre part, l'application de cette méthode a conduit à un lexique du slovaque qui sera très bientôt disponible librement. Bien qu'il ne soit pas encore à large couverture, ce lexique est intéressant à la fois en raison de la présence d'informations de morphologie dérivationnelle et en raison du fait qu'il est le lexique d'un corpus réel et contient par conséquent de nombreux mots nouveaux ou spécialisés qui sont moins souvent présents dans les lexiques ou les dictionnaires[22], comme par exemple *korpusový*, adjectivisation du mot *korpus* (« corpus »).

3 Morphologie dérivationnelle

Initialement introduite en particulier pour permettre un apprentissage plus efficace de lemmes peu attestés mais dérivés de lemmes connus ou bien attestés, la morphologie dérivationnelle est apparue également comme un complément intéressant au lexique morphologique proprement dit. En effet, les liens de dérivation qui existent entre lemmes appris peuvent être conservés sans difficulté, puis utilisés pour tout un ensemble de tâches, comme par exemple l'apprentissage de cadres de sous-catégorisation pour les noms et adjectifs déverbaux.

L'apprentissage des liens de morphologie dérivationnelle est donc un sous-produit du mécanisme d'apprentissage morphologique décrit ci-dessus. Cependant, et à moins de disposer d'une description exhaustive, exceptions et cas particulier y compris, les relations de dérivation entre lemmes ne sont apprises qu'à la condition de procéder de motifs de dérivation fréquents et productifs. Il est toutefois intéressant d'essayer d'évaluer dans quelle mesure l'apport d'informations de morphologie dérivationnelle dans la description morphologique d'une langue

[20] En effet, un même token peut être une forme fléchie de plusieurs lemme, et pour un même lemme correspondre à différentes étiquettes morphosyntaxiques.

[21] En particulier, le corpus utilisé devrait être plus grand. Nous prévoyons de faire des expériences en ce sens dans un avenir proche.

[22] Comme par exemple le *SLEX99 elektronický lexikón slovenského jazyka* (Lexique électronique de la langue slovaque SLEX99), consultable en ligne à l'adresse `http ://www.forma.sk/onlines/slex/`.

améliore la précision et la couverture du mécanisme d'apprentissage automatique du lexiques morphologiques.

Nous avons donc, *a posteriori*, relancé l'apprentissage de lexique *ex nihilo* pour le slovaque, comme si aucune validation n'avait encore jamais eu lieu, et ce de deux façons :

– d'une part en prenant en compte les informations de morphologie dérivationnelles,

– d'autre part sans les prendre en compte.

Lorsqu'on regarde les 1000 lemmes les mieux classés dans chacune des deux expériences, on constate que le nombre de lemmes corrects (au regard de la validation manuelle effectuée précédemment) passe de 767 à 900. Par ailleurs, parmi ces 900 lemmes corrects dans le cas où la morphologie dérivationnelle est prise en compte, 126 ne sont attestées que par 3 occurrences ou moins, dont 60 par une seule occurrence : pour ces lemmes, un apprentissage sans l'aide de la morphologie dérivationnelle aurait été bien plus difficile à effectuer. La prise en compte de la morphologie dérivationnelle n'a donc pas seulement pour effet l'apprentissage des relations de dérivation, elle permet également d'améliorer significativement les performances du mécanisme d'apprentissage lui-même.

4 Acquisition automatique d'un étiqueteur morphosyntaxique

4.1 Principe général

Le mécanisme d'acquisition automatique de lexique décrit précédemment génère comme sous-produit, pour chaque forme, la probabilité qu'elle corresponde à une étiquette morphosyntaxique donnée. Il se trouve que c'est typiquement l'information dont on peut avoir besoin pour construire un étiqueteur morphosyntaxique. Toutefois, les probabilités issues de l'apprentissage de lexique sont des probabilités lexicales (probabilités d'émission). Nous avons donc développé un algorithme de point fixe, dont le principe sous-jacent est le même que celui sur lequel repose notre algorithme d'apprentissage de lexique : à partir d'une estimation initiale des probabilités recherchées (ici, les probabilités d'émission de chaque étiquette pour chaque forme), on construit un nouveau modèle probabiliste (n-gramme) qui sert à calculer une nouvelle estimation des probabilités recherchées. Ce processus peut être répété jusqu'à convergence (empirique), produisant à chaque étape i un nouveau corpus étiqueté $\mathcal{C}_i$ et un nouvel étiqueteur $\mathcal{T}_i$. Si le processus est arrêté après $i = N$ étapes, on obtient donc finalement un étiqueteur morphosyntaxique n-gramme acquis automatiquement, l'étiqueteur $\mathcal{T}_N$. On notera que l'apprentissage du modèle n-gramme va donc se faire à partir d'un corpus annoté (automatiquement) de façon *ambiguë*, ce qui n'est traditionnellement pas le cas lorsque l'on entraîne un étiqueteur sur un corpus annoté manuellement.

Il faut toutefois faire attention à trois problèmes :

– l'apprentissage automatique de lexique se fait sur les classes ouvertes, et il faut donc détecter et/ou traiter spécifiquement les mots appartenant aux classes fermées,
– si l'on se place après qu'une étape de validation manuelle au moins a eu lieu, on est en droit d'accorder une plus grande confiance aux probabilités fournies par le modèle d'apprentissage de lexique lorsqu'elles concernent des formes fléchies de lemmes validés (par opposition aux formes fléchies de lemmes encore ni validés ni rejetés),
– il faut prendre en compte au mieux les mots inconnus, c'est-à-dire les mots qui ne sont pas des formes fléchies de lemmes vraisemblables tout en n'ayant pas été identifiés non plus (manuellement ou automatiquement) comme appartenant à une classe fermée.

En revanche, par rapport aux méthodes habituelles d'entraînement d'étiqueteurs morpho-syntaxiques, nous disposons d'un avantage certain : au moment de l'étiquetage par $\mathcal{T}_i$, c'est-à-dire lors de la production du corpus annoté $\mathcal{C}_{i+1}$, on dispose du corpus annoté précédent $\mathcal{C}_i$. Ceci signifie que les probabilités de transition que l'on va utiliser pour la construction du corpus $\mathcal{C}_{i+1}$ peuvent prendre en compte non seulement du contexte gauche du mot (son passé) dans $\mathcal{C}_{i+1}$ mais aussi son contexte droit dans $\mathcal{C}_i$. De plus, les probabilités présentes dans le corpus $\mathcal{C}_i$ peuvent servir de probabilité d'émission pour la construction du corpus annoté $\mathcal{C}_{i+1}$. L'intérêt est que l'on peut alors faire en sorte que les probabilités associées à un mot dépendent du mot précédent mais aussi du mot suivant, sans toutefois recourir à des mécanismes de calcul des k meilleurs chemins : nos étiqueteurs s'exécutent en temps linéaire en la longueur de la phrase, alors que les étiqueteurs cherchant un ou plusieurs meilleurs chemins dans un automate pondéré s'exécutent en temps cubique[23].

4.2 Mise en œuvre

Nous avons donc à définir trois mécanismes :

1. la construction du corpus étiqueté $\mathcal{C}_0$ à partir des probabilités d'émission données par le modèle d'apprentissage de lexique,
2. la construction de l'étiqueteur $\mathcal{T}_i$ à partir du corpus annoté $\mathcal{C}_i$,
3. la construction du corpus annoté $\mathcal{C}_{i+1}$ en utilisant l'étiqueteur $\mathcal{T}_i$ et le corpus annoté $\mathcal{C}_i$.

Contrairement à ce que nous avons fait pour l'apprentissage de lexiques morphologiques, nous n'allons pas décrire en détail ces étapes, mais en donner seulement les principes généraux.

La construction du corpus $\mathcal{C}_0$ se fait à partir des idées suivantes :

[23] De ce fait, ces étiqueteurs nécessitent impérativement une segmentation en phrases, ce qui n'est pas le cas des nôtres, puisqu'il s'exécutent en temps linéaire.

- si pour un mot donné on dispose d'étiquettes probabilisées, on les utilise pour annoter de façon ambiguë,
- sinon, et si le mot considéré a une fréquence supérieure à un certain seuil (nous utilisons 1/2000), alors on lui attribue une étiquette égale au mot lui-même et identifiée comme telle,
- sinon, et si le mot considéré est connu comme appartenant à une ou plusieurs classes fermées, et que ces classes ont été renseignées manuellement, on lui attribue comme étiquettes les identifiants de ces classes fermées de façon équiprobable,
- sinon, on attribue au mot considéré une étiquette spéciale ? indiquant qu'on n'a pour le moment aucune idée de son étiquette.

La construction de l'étiqueteur $\mathcal{T}_i$ à partir du corpus $\mathcal{C}_i$ consiste en la construction :

- de *bigrammes d'étiquettes*,
- de *bigrammes de traits* (bigrammes de genres, de nombres, de cas, etc...).

De plus, ces bigrammes sont de deux types : les bigrammes avant et les bigrammes arrière. Un bigramme avant modélise la probabilité d'un événement sur un mot sachant un événement sur celui qui suit, alors qu'un bigramme arrière modélise la probabilité d'un événement sur un mot sachant un événement sur celui qui précède.

Enfin, la construction du corpus annoté $\mathcal{C}_{i+1}$ se fait selon les principes suivants :

- on appelle *probabilité bigramme arrière* (respectivement *probabilité bigramme avant*) la moyenne géométrique entre la probabilité du bigramme arrière (resp. avant) d'étiquettes et la moyenne des bigrammes arrière (resp. avant) de traits correspondant aux étiquettes en jeu,
- on appelle *probabilité arrière* la somme des probabilités bigrammes arrière pondérées par les probabilités concernée pour le mot précédent dans $\mathcal{C}_{i+1}$,
- on appelle *probabilité avant* la somme des probabilités bigrammes avant pondérées par la probabilité de l'étiquette concernée pour le mot suivant dans $\mathcal{C}_i$,
- la probabilité contextuelle d'une étiquette pour un mot donné est la moyenne harmonique de la probabilité arrière et de la probabilité avant,
- la probabilité d'une étiquette pour un mot donné dans $\mathcal{C}_{i+1}$ est, à une normalisation près, la moyenne géométrique entre la probabilité contextuelle de cette étiquette et la probabilité de cette étiquette dans $\mathcal{C}_i$.

De plus, pour tout mot ayant l'étiquette ? avec une probabilité non nulle dans $\mathcal{C}_i$, on fait comme si cette probabilité était deux fois moindre et on répartit entre tous les étiquettes possibles l'autre moitié de cette probabilité. Ceci sert à remplacer l'étiquette ?, qui note une absence d'information venant du modèle d'apprentissage de lexique, par des étiquettes pertinentes. Enfin, et seulement pour la construction de $\mathcal{C}_1$, pour tout mot qui n'est la forme fléchie d'un lemme

validé manuellement, on fait comme si toutes les probabilités de ses étiquettes étaient deux fois moindre et on répartit entre tous les tags possibles la probabilité 1/2 ainsi « récupérée ». Ceci sert à nuancer les informations venant du modèle d'apprentissage de lexiques.

Chapitre 7

Développement d'un lexique syntaxique et sémantique

Une fois le formalisme de représentation des informations lexicales défini, le développement d'un lexique syntaxique peut se répartir en deux types de tâches :

1. la détection d'erreurs et de manques dans le lexique,
2. l'acquisition automatique d'informations syntaxiques et sémantiques, permettant de corriger les erreurs et les manques détectés précédemment.

Pour passer de la version 1 du Le*fff*, lexique morphologique des verbes français, à sa version 2, lexique syntaxique couvrant toutes les catégories, nous avons développé des techniques qui permettent de ne pas réaliser les deux tâches ci-dessus que de façon manuelle. Cependant, nous n'avons pas développé de système d'acquisition automatique de lexiques syntaxiques, à l'instar de ce que nous avons fait pour les lexiques morphologiques. De fait, les travaux en ce sens (voir chapitre 2) semblent assez limités en termes de pertinence linguistique : ils nécessitent souvent un nombre important de pré-requis (classes syntaxiques pré-établies,...), nécessitent une validation manuelle bien plus complexe et coûteuse que pour l'acquisition de lexiques morphologiques, et ne permettent pas toujours l'extraction de toutes les informations nécessaires (phénomènes de contrôle, de montée, attributifs,...). Il nous a semblé toutefois possible de mettre en œuvre deux grandes familles de techniques, une pour chacune des tâches identifiées ci-dessus :

1. l'identification automatique de formes fléchies qui ne sont pas ou pas correctement renseignées dans le lexique, par une technique de fouille d'erreurs sur des sorties d'analyseurs syntaxiques utilisant ce lexique,
2. l'acquisition automatique (avec validation manuelle) d'informations lexicales ciblées par reconnaissance et comptabilisation de motifs morphosyntaxiques.

Nous allons étudier plus en détails des techniques que nous avons mises en œuvre et qui relèvent de ces deux familles[1]. Dans ce type de travaux, notre approche est la suivante : nous utilisons des techniques automatiques pour présenter des informations ou des hypothèses d'une façon qui permette au mieux de préparer le travail manuel du linguiste. Autrement dit, nous ne prétendons pas nous substituer au linguiste, mais plutôt lui faciliter le travail. Cette approche nous semble réaliser le meilleur équilibre entre efficacité du développement du lexique et qualité de son contenu.

En conclusion de ce chapitre, nous reviendrons rapidement sur le Le*fff* 2, l'ensemble des techniques utilisées pour son développement, ainsi que ses caractéristiques quantitatives.

1 Fouille d'erreurs sur des sorties d'analyseurs syntaxiques

1.1 Principes

1.1.1 Idée générale

Nous proposons un modèle probabiliste permettant de repérer les formes fléchies potentiellement sources d'erreurs à partir d'un corpus de phrases soumis à analyse. Pour exploiter au mieux les formes détectées par le modèle et en particulier pour pouvoir identifier la source de l'erreur, un environnement de visualisation a été mis en place. L'ensemble a été testé sur les résultats d'analyse produits sur plusieurs corpus de plusieurs centaines de milliers de phrases et deux systèmes d'analyse syntaxique distincts, à savoir FRMG et SXLFG$_L$. Ces travaux ont été effectués en collaboration avec Éric de La Clergerie (Sagot et Éric de La Clergerie, 2006).

L'idée que nous avons mise en œuvre, qui généralise et raffine une idée proposée par van Noord (2004), est la suivante. Pour identifier les incomplétudes et les incorrections d'un système d'analyse syntaxique, on peut analyser un corpus de taille conséquente et étudier à l'aide d'outils statistiques ce qui différencie les phrases pour lesquelles l'analyse a réussi de celles pour lesquelles elle a échoué.

L'application la plus simple de cette idée consiste à chercher les formes qui se retrouvent fréquemment dans des phrases qui n'ont pas pu être analysées. Nous allons donc chercher,

[1] Nous avons également utilisé d'autres méthodes, plus simples, moins originales, ou de façon trop préliminaire pour être présentées en détail. Par exemple, nous avons utilisé notre correcteur orthographique SXSpell (voir chapitre 11) pour détecter dans un gros corpus les formes inconnues du lexique et leur chercher des corrections orthographiques. En fonction de l'existence ou non, pour chaque forme inconnue, d'une correction à faible coût, et en fonction du nombre d'occurrences de cette forme, on peut constituer automatiquement une liste de formes fléchies inconnues du lexique et devant probablement y figurer. Cette technique est bien plus manuelle que notre méthode globale d'acquisition de lexique morphologique (chapitre 6) ou que notre technique de fouille d'erreurs dans les sorties d'analyseurs syntaxiques (voir plus bas), mais elle permet d'augmenter rapidement la couverture du lexique pour des catégories à la flexion moins riche que les verbes, et ne nécessite pas la mise en œuvre d'analyseurs.

pour chaque phrase dont l'analyse a échoué, quelle est la forme qui a le plus de chances d'être la cause de cet échec : c'est une forme *suspecte*. Naturellement, le lexique n'est pas la seule source d'erreur possible. La chaîne de traitement pré-syntaxique, qui se place en amont de l'analyseur, peut induire toutes sortes d'erreurs. Et les analyseurs eux-mêmes reposent sur des grammaires imparfaites et incomplètes. Ce deuxième point peut être limité par la comparaison des résultats obtenus avec deux analyseurs différents, reposant sur deux grammaires différentes. C'est ce que nous avons fait. On pourrait faire de même à l'aide de deux chaînes de traitement pré-syntaxiques différentes, mais nous ne disposons que de celle que nous avons développé, SXPipe (voir Chapitre 11). Cela fait donc partie de la phase de validation et d'exploitation manuelle des résultats que d'identifier si c'est bien le lexique (Le*fff*, en l'occurrence) qui est en cause ou non pour chaque forme fléchie identifiée comme suspecte.

Ce principe de base sur les formes s'étend naturellement aux lemmes et aux séquences de formes (ou de lemmes).

Une version plus simple de cette idée est utilisée par van Noord (2004), sans toutefois chercher à identifier une forme suspecte pour chaque phrase, et donc sans prendre en compte le fait qu'il y a, dans toute phrase non analysable, une cause d'erreur[2].

1.1.2 Modèle probabiliste

On suppose donc que le corpus utilisé est découpé en phrases, elles-mêmes découpées en formes. On note p_i la i-ième phrase. On note $o_{i,j},\ (1 \leq j \leq |p_i|)$ les occurrences des formes constituant p_i, et on désigne par $F(o_{i,j})$ les formes correspondantes. Enfin, on note erreur la fonction qui à une phrase p_i associe 1 si l'analyse de p_i a échoué, et 0 sinon.

Soit $\mathcal{O}_f$ l'ensemble des occurrences d'une forme f dans le corpus : $\mathcal{O}_f = \{o_{i,j} | F(o_{i,j}) = f\}$. Le nombre d'occurrences de f dans le corpus est donc $|\mathcal{O}_f|$.

Définissons tout d'abord le *taux de suspicion moyen global* $\overline{S}$, qui est la probabilité moyenne qu'une occurrence donnée d'une forme soit la cause d'un échec d'analyse. Nous faisons l'hypothèse que l'échec de l'analyse d'une phrase est dû à une cause unique, et donc ici à une forme unique. Cette hypothèse, qui n'est pas nécessairement vérifiée, simplifie le modèle et donne des résultats pertinents. En notant $\mathrm{occ}_{\mathrm{total}}$ le nombre total de formes dans le corpus, on a donc :

$$\overline{S} = \frac{\Sigma_i \mathrm{erreur}(p_i)}{\mathrm{occ}_{\mathrm{total}}}$$

[2]La formule suivante est utilisée par van Noord (2004), qui ne descend pas au niveau des occurrences et des phrases individuelles : $\mathrm{err}(f) = \frac{\#\text{phrases ratées avec } f}{\#\text{phrases avec } f}$.

Soit une forme f qui est la j-ième forme de la phrase p_i, c'est-à-dire que $F(o_{i,j}) = f$. Supposons que l'analyse de p_i a échoué : $\mathrm{erreur}(p_i) = 1$. On appelle *taux de suspicion* de la j-ième forme $o_{i,j}$ de la phrase p_i la probabilité, notée $S_{i,j}$, que l'occurrence $o_{i,j}$ de la forme f soit responsable de l'échec de l'analyse de p_i. Si au contraire l'analyse de la phrase p_i a réussi, ses occurrences ont un taux de suspicion nul.

On définit alors le *taux de suspicion moyen* S_f de la forme f comme étant la moyenne des taux de suspicion de ses occurrences (toutes phrases confondues) :

$$S_f = \frac{1}{|\mathcal{O}_f|} \cdot \sum_{o_{i,j} \in \mathcal{O}_f} S_{i,j}$$

Nous utilisons un algorithme de recherche de point fixe, en itérant un certain nombre de fois les calculs suivants. Supposons que l'on vient de terminer la n-ième itération : nous connaissons pour chaque phrase p_i et pour chaque occurrence $o_{i,j}$ de cette phrase l'estimation de son *taux de suspicion* $S_{i,j}$ par la n-ième itération, estimation notée $S_{i,j}^{(n)}$. On peut en déduire l'estimation de rang $n+1$ pour le taux de suspicion moyen de chaque forme f, noté $S_f^{(n+1)}$:

$$S_f^{(n+1)} = \frac{1}{|\mathcal{O}_f|} \cdot \sum_{o_{i,j} \in \mathcal{O}_f} S_{i,j}^{(n)}$$

Ce taux[3] nous permet de calculer une nouvelle estimation des taux de suspicion des occurrences, en attribuant à chaque occurrence d'une phrase p_i un taux de suspicion $S_{i,j}^{(n+1)}$ égal à cette estimation $S_f^{(n+1)}$ du taux de suspicion moyen S_f de la forme correspondante, puis normaliser à l'échelle de la phrase. Ainsi :

$$S_{i,j}^{(n+1)} = \mathrm{erreur}(p_i) \cdot \frac{S_{F(o_{i,j})}^{(n+1)}}{\sum_{1 \le j \le |p_i|} S_{F(o_{i,j})}^{(n+1)}}$$

[3]Nous avons également fait des expériences où S_f est estimé à l'aide d'un autre estimateur, le *taux de suspicion moyen lissé*, noté $\tilde{S}_f^{(n)}$, qui prend en compte le nombre d'occurrences de f. En effet, la confiance que l'on peut accorder à l'estimation $S_f^{(n)}$ est d'autant plus faible que le nombre d'occurrences occ_f est faible. D'où l'idée de lisser $S_f^{(n)}$ en le remplaçant par un barycentre $\tilde{S}_f^{(n)}$ de $S_f^{(n)}$ et de $\overline{S}$ dont le coefficient de pondération λ dépend de $|\mathcal{O}_f|$: si $|\mathcal{O}_f|$ est grand, $\tilde{S}_f^{(n)}$ sera proche de $S_f^{(n)}$; s'il est petit, il sera plus proche de $\overline{S}$:

$$\tilde{S}_f^{(n)} = \lambda(|\mathcal{O}_f|) \cdot S_f^{(n)} + (1 - \lambda(|\mathcal{O}_f|)) \cdot \overline{S}.$$

Dans ces expériences, nous avons utilisé la fonction de lissage $\lambda(|\mathcal{O}_f|) = 1 - e^{-\beta|\mathcal{O}_f|}$ avec $\beta = 0.1$. Mais ce modèle couplé au classement selon $M_f = S_f \cdot \ln|\mathcal{O}_f|$ (voir plus bas) donne des résultats similaires au modèle sans lissage. Nous présentons donc le modèle non lissé, qui a l'avantage de ne pas faire intervenir de fonction de lissage choisie empiriquement.

À ce stade, la $(n+1)$-ième itération est terminée, et on peut recommencer à nouveau ces calculs, jusqu'à convergence sur un point fixe. L'amorçage de cet algorithme se fait en posant, pour une occurrence $o_{i,j}$ dans la phrase p_i, $S_{i,j}^{(0)} = \text{erreur}(p_i)/|p_i|$. Autrement dit, si l'analyse de p_i a échoué, on part d'une estimation où toutes ses occurrences ont une probabilité égale d'être la cause de l'échec.

Après quelques dizaines d'itérations de ce processus, on obtient donc des estimations du taux de suspicion moyen de chaque forme, ce qui permet

- de repérer les formes les plus vraisemblablement responsables d'erreurs,
- pour chaque forme f, d'identifier les phrases non analysables s_i où une de ses occurrences $o_{i,j} \in \mathcal{O}_f$ est un des principaux suspects et où $o_{i,j}$ a un taux de suspicion parmi les plus élevés parmi les occurrences de f.

L'algorithme a été implémenté en *perl*, en optimisant les structures de données pour réduire les coûts en mémoire et en temps. En particulier, les phrases stockent une liste d'occurrences qui pointent directement sur la structure associée à chaque forme.

1.1.3 Extensions du modèle

Ce modèle donne déjà de très bons résultats, comme nous le verrons à la section 7.1.3. Cependant, on peut l'étendre de différentes façons, dont nous avons déjà implémenté certaines.

Tout d'abord, il est possible ne pas se restreindre aux formes. De fait, nous ne travaillons pas sur les formes seules, mais sur des couples formés d'une forme (une entrée du lexique) et du ou des tokens qui leur correspondent dans le texte brut (un token est une portion de texte délimitée par des espaces ou des tokens de ponctuation).

Par ailleurs, on peut vouloir rechercher la cause de l'échec de l'analyse d'une phrase pas seulement dans la présence d'une forme dans cette phrase, mais aussi dans la présence d'un bigramme voire d'un trigramme de formes. Il suffit pour cela de généraliser la notion d'occurrence : on peut dire que les occurrences associées à une phrase sont un ensemble de faits simultanés caractéristiques (mais non spécifiques) de cette phrase. Par exemple, en définissant les occurrences associées à une phrase comme l'ensemble des formes et des bigrammes de formes de cette phrase, on obtient des résultats très intéressants.

L'autre généralisation possible consisterait à savoir prendre en compte des faits qui ne sont pas simultanés, mais qui sont des hypothèses concurrentes, qu'il faut par conséquent probabiliser aussi. Nous n'avons pas encore mis en œuvre un tel mécanisme. Il serait pourtant très intéressant, car il permettrait de dépasser le stade de la forme ou des n-grammes de formes et d'obtenir des généralisations, par exemple au niveau des lemmes.

1.2 Mise en œuvre

Nous avons appliqué ces principes pour rechercher les causes d'erreurs dans les sorties de deux systèmes d'analyse syntaxique profonde pour le français, FRMG et SXLFG$_L$. Pour cela, nous avons analysé avec ces deux systèmes d'une part les 40 000 phrases du corpus EASy et d'autre part 300 000 à 600 000 phrases du *Monde diplomatique*. Enfin, nous avons développé un environnement complet de visualisation des résultats de cette fouille d'erreurs. Cette section détaille ces trois points successivement.

1.2.1 Analyseurs

Les deux systèmes d'analyse que nous avons utilisés reposent sur des analyseurs syntaxiques profonds non-probabilistes. Ils ont en commun :

- le lexique syntaxique Le*fff* 2 (Sagot *et al.*, 2005), qui comporte 600 000 entrées (couvrant 400 000 formes distinctes) ; une entrée regroupe informations morphologiques, cadres de sous-catégorisation syntaxique (lorsque pertinent) et informations syntaxiques complémentaires, en particulier pour les formes verbales (contrôles, attributifs, impersonnels,...) ;
- la chaîne de traitement pré-syntaxique SXPipe (Sagot et Boullier, 2005), qui convertit un texte brut en suite de treillis de mots présents dans le Le*fff* ; SXPipe comprend entre autres des modules de segmentation en phrases, de tokenization et correction orthographique, de détection d'entités nommées, et d'identification non-déterministe des mots composés.

En revanche, FRMG et SXLFG$_L$ utilisent des analyseurs complètement différents, ne reposant ni sur le même formalisme, ni sur la même grammaire, ni sur le même générateur d'analyseurs. La comparaison des résultats de fouille d'erreurs sur les sorties de ces systèmes nous permet donc de distinguer les erreurs dues à Le*fff* ou à SXPipe d'une part, et celles dues à l'une ou l'autre des grammaires d'autre part. Voyons plus en détails les caractéristiques de ces deux analyseurs.

L'analyseur FRMG (Thomasset et Éric Villemonte de la Clergerie, 2005) s'appuie sur une grammaire compacte TAG du français générée à partir d'une méta-grammaire. La compilation et l'exécution de l'analyseur se déroule dans le cadre du système DYALOG (Villemonte de la Clergerie, 2005).

L'analyseur SXLFG$_L$ repose sur le générateur d'analyseurs LFG efficaces et robustes SXLFG, présenté au chapitre 9. Présentons toutefois dès maintenant les grandes lignes de son fonctionnement. L'analyse est effectuée en deux phases. Tout d'abord, un analyseur à la Earley construit une forêt partagée représentant l'ensemble des analyses en constituants qui satisfont le squelette non-contextuel de la grammaire. Puis les structures fonctionnelles sont construites de bas

en haut, éventuellement en plusieurs passes. L'efficacité de l'analyseur repose sur diverses techniques de représentation compacte de l'information, sur l'emploi systématique de méthodes de partage de calcul et de structures, sur des techniques d'évaluation paresseuse, et sur l'élagage heuristique et quasiment non-destructif en cours d'analyse. Enfin, la grammaire utilisée pour la construction de SXLFG$_L$ par SXLFG est un descendant de la grammaire LFG utilisée par Lionel Clément pour son système XLFG.

Nos deux analyseurs implémentent aussi également des techniques évoluées de rattrapage et de tolérance d'erreurs, mais elles sont inutiles dans le cadre des travaux décrit ici, puisque nous ne cherchons qu'à distinguer les phrases qui ont une analyse complète (sans utilisation de techniques de rattrapage) de celles qui n'en ont pas.

1.2.2 Corpus

Nous avons analysé, à l'aide de ces deux systèmes, les corpus suivants :

Corpus MD : Il s'agit de plusieurs centaines de milliers de phrases d'un corpus d'articles du *Monde diplomatique* dont le style est naturellement exclusivement journalistique[4].

Corpus EASy : Il s'agit du corpus d'environ 40 000 phrases constitué pour la campagne d'évaluation des analyseurs syntaxiques EASy. Nous n'avons utilisé que les corpus bruts, sans prendre en compte le fait que pour environ 10% des phrases une annotation manuelle est disponible. Le corpus EASy regroupe des sous-corpus de styles variés : journalistique, littéraire, législatif, médical, transcription d'oral, courrier électronique, questions, etc.

Les deux corpus sont relativement bruts dans le sens où aucun nettoyage n'a été effectué pour éliminer certaines séquences de caractères ne pouvant pas être considérées comme des phrases.

La table 1 donne quelques informations générales sur les corpus utilisés et sur les résultats obtenus par les deux analyseurs. Il est à noter que les deux analyseurs n'ont pas exploité exactement le même jeu ni le même nombre de phrases pour MD, et qu'ils n'utilisent pas exactement la même notion de phrase.

1.2.3 Environnement de visualisation des résultats

Nous avons développé un outil de visualisation des résultats de la fouille d'erreurs, qui permet de les parcourir et de les annoter. Il s'agit d'une page WEB qui fait usage de techniques de génération dynamique, en particulier via *javascript*. Un exemple est montré figure 1.

[4] L'expérience présentée en 12.1 consiste en l'analyse par SXLFG$_L$ du même corpus. Cependant, elle est plus récente que celle décrite ici. Elle a donc produit de meilleurs résultats, mais n'a pas été effectuée sur autant de phrases.

corpus	#phrases	#succès (%)	#formes	#occ	$\overline{S}$ (%)	Date
MD/FRMG	330 938	136 885 (41.30%)	255 616	10 422 926	1.86%	Juil. 05
MD/SxLFG$_L$	630 000	337 437 (53.56%)	373 986	15 840 364	1.85%	Déc. 05
EASy/FRMG	39 872	16 477 (41.32%)	61 135	878 156	2.66%	Déc. 05
EASy/SxLFG$_L$	39 872	21 067 (52.84%)	61 135	878 156	2.15%	Déc. 05

TAB. 1 – Informations générales sur les corpus et les résultats d'analyse.

Pour cela, les suspects sont triés selon une mesure M_f modélisant pour une forme f l'intérêt qu'a l'utilisateur à tenter de corriger dans ses ressources l'erreur correspondante. Un utilisateur souhaitant privilégier les erreurs presque certaines sur les erreurs fréquentes peut visualiser les suspects en fonction de $M_f = S_f$. Un utilisateur souhaitant privilégier la fréquence d'une erreur potentielle au détriment de sa crédibilité peut les visualiser en fonction de[5] $M_f = S_f|\mathcal{O}_f|$. Une position intermédiaire, adoptée ici, consiste à les trier en fonction de $M_f = S_f \cdot \ln|\mathcal{O}_f|$.

L'environnement de visualisation permet de naviguer entre les suspects (triés) dans un menu à gauche de la page (**A**). Lorsque le token suspect est égal à la forme, seul le token est indiqué. Sinon, le token est séparé de la forme par « / ». Si l'on sélectionne un suspect, la partie droite de la page indique toutes sortes d'informations à son sujet. Après avoir indiqué son rang dans le classement selon la mesure M_f choisie (**B**), un champ est disponible pour ajouter ou éditer un commentaire associé au suspect (**D**). Ces commentaires, destinés au dépouillement des résultats par des linguistes ou par les développeurs des analyseurs et des grammaires, sont stockés dans une base de données (SQLITE). Des informations statistiques sont également fournies pour f (**E**), dont son nombre d'occurrences occ_f, le nombre d'occurrences de f appartenant à des phrases non analysables, l'estimation finale de son taux de suspicion moyen S_f, et le pourcentage $\mathrm{err}(f)$ de phrases non analysables parmi celles où f apparaît. Ces indications sont complétées par un bref historique (**F**) montrant la convergence du S_f. La partie inférieure de la page donne le moyen d'identifier les causes d'erreurs provenant de f en montrant d'une part (**G**) les entrées de f dans le lexique Lefff, et d'autre part (**H**) les phrases non analysables où f est le suspect principal et où une de ses occurrences a un taux de suspicion spécialement élevé.

La page avec commentaires peut être envoyée par mail, par exemple au gestionnaire de Lefff ou au développeur de tel ou tel analyseur (**C**).

1.3 Résultats

Pour MD/FRMG, l'algorithme a effectué 200 itérations en environ 2700 s sur un PC à 3.2 Ghz avec 1 Go de mémoire vive. Il a proposé 18 492 formes dites *pertinentes* (sur les

[5] Soit une forme f. Le taux de suspicion S_f peut être vu comme la probabilité qu'une occurrence particulière de f induise une erreur. Par conséquent, $S_f|\mathcal{O}_f|$ modélise le nombre d'occurrences de f en tant que source d'erreur.

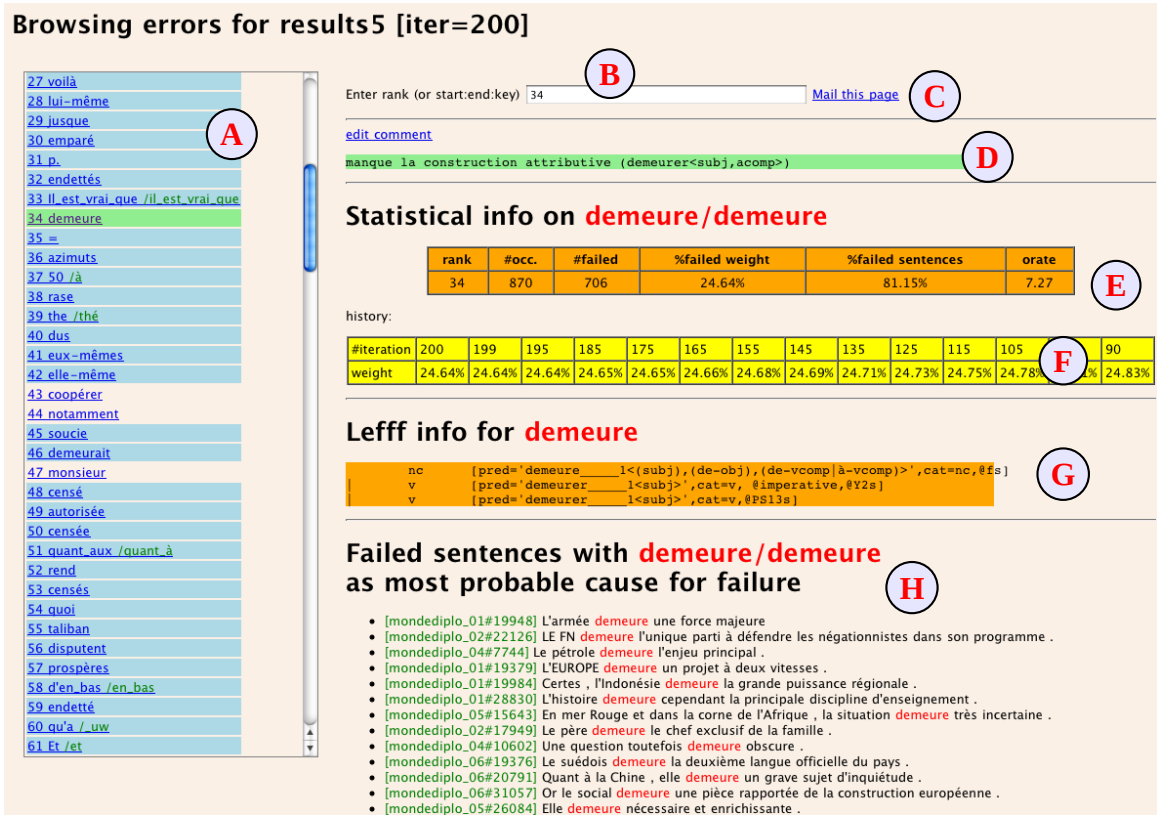


FIG. 1 – Visualisation du résultat de la fouille d'erreurs (illustré pour MD/FRMG).

#occ	$> 100\,000$	$> 10\,000$	> 1000	> 100	> 10
# formes	13	84	947	8345	40 393
# formes suspectes (%)	1 (7.6%)	13 (15.5%)	177 (18.7%)	1919 (23%)	12 022 (29.8%)

TAB. 2 – Répartition des suspects pour MD/FRMG.

255 616 possibles), une forme pertinente étant une forme f qui vérifie les contraintes (arbitraires) de seuil suivantes[6] : $S_f^{(200)} > 1,5 \cdot \overline{S}$ et $|\mathcal{O}_f| > 5$.

Nous ne pouvons pas encore assurer la convergence théorique de l'algorithme[7]. En pratique, sur les 1000 premières formes retournées pour MD/FRMG, la dernière itération fait varier le taux de suspicion de moins de 0,01% en moyenne (le maximum est de 0,06%, et seules 10 formes varient de plus de 0,01%). En pratique, il n'est pas nécessaire de réaliser 200 itérations et on obtient déjà de bons résultats avec une cinquantaine d'itérations seulement.

Pour un « petit » corpus comme EASy/FRMG, les 200 itérations prennent 260s, retournent 2678 formes (sur les 61 125 possibles). Les indications de convergence sur les 1000 premières formes sont équivalentes à celles obtenues pour MD/FRMG.

La table 2 donne une idée de la répartition des suspects par rapport à leur fréquence, montrant que les formes rares ont proportionnellement plus de chances d'être suspectes. La forme suspecte la plus fréquente est « " » avec $S_f = 9\%$, résultant en partie de problèmes de segmentation.

1.3.1 Causes d'erreurs identifiées

Les tables 3 et 4 montrent des extraits des résultats obtenus en appliquant notre méthode sur les résultats d'analyse du corpus MD par FRMG. Elles donnent respectivement les 20 couples token(s)/forme les mieux classés, et ceux classés 100 à 109, pour montrer que les résultats restent pertinents bien au-delà des premiers rangs. Outre le taux de suspicion et le nombre d'occurrences, nous donnons le taux $\mathrm{err}(f)$ (utilisé par van Noord (2004)) qui est le pourcentage d'échecs d'analyse parmi les phrases comportant une occurrence de la forme f, ainsi qu'une analyse manuelle de la cause de l'erreur identifiée automatiquement.

On constate que les résultats sont très bons : les deux tableaux identifient correctement des erreurs, réparties en quatre sources : les erreurs dans le lexique Le*fff*, les erreurs dans la chaîne de traitement pré-syntaxique SXPipe, les imperfections de la grammaire, mais aussi les problèmes issus du corpus en raison du fait qu'il s'agit d'un corpus brut.

[6]Ces seuils éliminent des formes à la sortie des résultats mais il est à noter que toutes les formes et leurs occurrences sont utilisées lors des itérations.

[7]Mais l'algorithme présente des ressemblances troublantes avec des algorithmes itératifs et convergents proposés pour la recherche de distributions de probabilités à entropie maximale sous un ensemble de contraintes (Berger *et al.*, 1996).

Rang	Token(s)/forme	$S_f^{(200)}$	$\lvert\mathcal{O}_f\rvert$	$\mathrm{err}(f)$	M_f	Origine de l'erreur
1	;	65%	13379	100%	6.16	FRMG : « ; » ne peut terminer les phrases dans FRMG
2	*	83%	199	100%	4.40	corpus : phrases comportant le seul mot « * »
3	(...)	58%	1835	100%	4.37	SXPipe : on devrait en faire un mot ignorable
4	coll/coll.	74%	257	93%	4.09	FRMG : ne sait pas analyser « L'Harmattan , coll. »
5	jour	44%	3005	100%	3.49	Le*fff* (ancienne version) : manquait la forme « jour »
6	[...]	61%	173	100%	3.16	SXPipe : on devrait en faire un mot ignorable
7	feu	47%	643	100%	3.01	Le*fff* : « faire long feu » ; FRMG : adj. pré-déterminant
8	date	45%	749	100%	2.97	FRMG (ancienne version) : pb sur les verbes support
9	confiance	44%	666	100%	2.89	FRMG(ancienne version) : pb sur les verbes support
10	80/à	57%	138	100%	2.83	SXPipe : bug (idem aux rangs 12 : 70/à, 13 : 60/à et 14 : 30/à)
11	etc/etc.	44%	493	100%	2.74	Le*fff* : ce n'est pas une ponctuation...
15	voici	43%	269	93%	2.43	FRMG : grammaire incomplète
16	qu'elle/quelle	30%	3052	96%	2.37	SXPipe (ancienne version) : bug (id. 17 :qu'elles/quelles)
17	contrairement	39%	313	100%	2.23	Le*fff* : Manque la notion de sous-cat adverbiale (ici *à-obj*)
18	fiche	63%	34	94%	2.23	Le*fff* : manque comme nom commun (!)
19	emparé	66%	29	100%	2.22	Le*fff* : Manquent les constr. pronominales (*s'emparer (de)*)
20	demeurent	37%	398	87%	2.21	Le*fff* : Manque la construction attributive

TAB. 3 – Analyse des 20 premières formes (classées selon $M_f = S_f \cdot \ln |\mathcal{O}_f|$).

Rang	Token(s)/forme	$S_f^{(200)}$	$\lvert\mathcal{O}_f\rvert$	$\mathrm{err}(f)$	M_f	Origine de l'erreur
100	investir	30%	136	86%	1.48	Le*fff* : l'objet direct n'est pas obligatoire
101	clôt	47%	23	96%	1.48	Le*fff* : Manquent les constr. pronominales (*se clore*)
102	demain	25%	378	81%	1.48	Le*fff* et FRMG : Peut former un GN saturé (*pour demain*)
103	Seuls/seuls	29%	169	95%	1.48	FRMG : adj. pré-déterminant (id. au rang 104)
105	autoproclamé	50%	19	100%	1.47	Le*fff* : Manque la construction attributive
106	renchérit	45%	25	92%	1.46	Le*fff* et FRMG : Traiter les constructions narratives
107	emparée	50%	18	100%	1.46	Le*fff* : Manquent les constr. pronom. (id. rang 109, cf. rang 19)
108	Mille/_NUMBER	56%	13	100%	1.45	SXPipe : *Mille et une* pas reconnu comme un nombre

TAB. 4 – Analyse des formes de rang 100 à 109 (classées selon $M_f = S_f \cdot \ln |\mathcal{O}_f|$).

Rang	Token(s)/forme		Origine de l'erreur
1	(...)	3,55	SXPipe : en faire un mot que l'on peut ignorer (_EPSILON)
2	[...]	2,85	SXPipe : comme pour *[...]*
3	demeurent	2,23	Le*fff* : Manque la construction attributive
4	Premières/Premiere	2,09	SXPipe : bug
5	emparé	2,05	Le*fff* : Manquent les constr. pronominales (*s'emparer*)
6	endettés	2,04	Le*fff* : Manque en tant qu'adjectif
7	lui-même	1,99	Le*fff* : À mettre comme un adjectif spécial
8	endetté	1,87	Le*fff* : comme pour *endettés*
9	elle-même	1,81	Le*fff* : À mettre comme un adjectif spécial
10	d'en_bas/en_bas	1,79	Le*fff* et grammaires : Constr. 'prep+adv' pour certains adv
11	larvée	1,78	Le*fff* : Manque comme adjectif
12	–/-	1,75	grammaires : gestion du tiret parenthétique

TAB. 5 – Couples token(s)/forme maximisant la moyenne harmonique $\bar{M}_f$ des mesures M_f^{FRMG} et $M_f^{\text{SXLFG}_L}$ obtenues par FRMG et SXLFG$_L$.

Pour le corpus EASy, les résultats sont également pertinents, mais parfois plus délicats à interpréter, suite à la taille limitée du corpus, à sa grande hétérogénéité et à la présence de sous-corpus de courriers électroniques et de transcription de corpus oraux qui introduisent beaucoup de bruit. L'importance des erreurs de segmentation (dues à la fois à SXPipe et au corpus lui-même, déjà segmenté) s'en trouve renforcée.

1.3.2 Comparaison entre les deux analyseurs

Nous avons complété l'étude des résultats séparés de nos deux systèmes d'analyse par un couplage des deux résultats. Nous avons calculé pour cela pour chaque mot la moyenne harmonique des mesures $M_f = S_f \cdot \ln |\mathcal{O}_f|$ obtenus pour chaque système d'analyse. Les résultats sont très intéressants, car ils identifient des erreurs provenant en majorité des ressources partagées entre les deux systèmes (le lexique Le*fff* et la chaîne de traitement pré-syntaxique SXPipe). Bien que certaines erreurs sont issues d'incomplétudes communes aux deux grammaires, c'est néanmoins un moyen efficace d'obtenir une première répartition entre sources d'erreurs différentes.

À titre d'illustration, la table 5 montre quels sont les 10 couples token(s)/forme dont la moyenne harmonique des taux de suspicion obtenus par FRMG et SXLFG$_L$ est la plus grande (sur les résultats d'analyse du corpus MD).

1.4 Conclusions et perspectives

L'analyse de corpus importants permet donc de mettre en place des techniques de fouille d'erreurs, pour identifier les incomplétudes et les inexactitudes dans les différentes ressources utilisées par un système d'analyse syntaxique complet, et en particulier le lexique syntaxique. La technique décrite ici et mise en œuvre sur les formes (ou les couples token(s)/forme) nous a déjà permis de détecter nombre d'erreurs et de manques dans notre lexique Le*fff*, de révéler des comportements inappropriés dans notre chaîne de traitement pré-syntaxique SXPipe, et de mettre en avant le manque de couverture de nos grammaires pour certains phénomènes.

Nous souhaitons mettre en place différentes améliorations et extensions de ce travail. Tout d'abord, l'interface est améliorable, de même que l'implémentation de l'algorithme itératif.

Ensuite, nous pensons intégrer au modèle la possibilité que les faits pris en compte (ici les occurrences) ne soient pas nécessairement tous certains, mais puissent être parfois en concurrence les uns avec les autres. Par exemple, pour une forme donnée, plusieurs lemmes sont souvent possibles. Les probabiliser permettrait donc de rechercher les *lemmes* les plus suspects.

Nous comptons également développer un module permettant non seulement de détecter des erreurs, par exemple dans le lexique, mais également de proposer une correction[8]. Pour cela, nous envisageons, pour chaque phrase dont l'analyse a échoué, de relancer l'analyse en remplaçant les informations lexicales associées au suspect principal de la phrase par des informations sous-spécifiées. Ces informations pourraient être obtenues par exemple par généralisation, selon certaines règles, des informations associées au suspect dans le lexique. On pourrait aussi traiter le suspect principal comme s'il était un mot inconnu, à qui l'on associerait des informations très peu contraignantes. L'étude statistique des analyses ainsi obtenues permettrait alors de proposer des corrections pour le mot en question. Une telle méthodologie permettrait de faire le lien entre la technique de détection d'erreurs présentée jusqu'ici et les techniques d'acquisition d'informations syntaxiques atomiques présentées ci-dessous.

2 Reconnaissance et comptabilisation de motifs morphosyntaxiques

2.1 Principes

Comme dit précédemment, nous pensons qu'il est aujourd'hui difficile de développer des techniques d'acquisition automatique de lexiques syntaxiques comparables à ce que nous avons

[8]Une fois encore, nous n'avons pas pour objectif de remplacer le linguiste, mais de lui faciliter la tâche en lui proposant une correction. Si la correction proposée par la machine est la bonne, le linguiste n'a qu'à la valider, ce qui est bien plus rapide que de l'effectuer manuellent.

mis en place pour l'acquisition de lexiques morphologiques. Ceci tient à plusieurs causes, parmi lesquelles :

- la complexité et la diversité des structures syntaxiques possibles,
- le fait qu'une forme donnée puisse correspondre à plusieurs structures syntaxiques, dont certaines peuvent être rares,
- la difficulté qu'il y a à apprendre des informations structurelles sur des corpus bruts,
- le coût de développement et le volume très insuffisant des corpus annotés manuellement par rapport à ce qui serait nécessaire pour cette tâche.

Il nous semble cependant possible d'acquérir des informations syntaxiques atomiques ciblées, évitant ainsi ces problèmes :

- on ne recherche plus des structures mais des valeurs de vérités (cette forme a-t-elle, parfois ou toujours, une propriété syntaxique atomique donnée ?),
- on laisse au linguiste le travail de répartition des informations acquises en structures syntaxiques différentes,
- on peut se passer de corpus annotés manuellement, bien qu'il soit préférable de disposer de corpus étiquetés (si l'on dispose d'un étiqueteur, on peut construire des corpus étiquetés très volumineux, ce qui compense en partie les erreurs d'étiquetage).

L'idée est de définir des motifs morphosyntaxiques qui correspondent à une propriété syntaxique atomique puis de les rechercher dans un corpus étiqueté très volumineux. Il n'est pas indispensable de chercher à éviter le bruit ou le silence. Par exemple, on peut chercher les constructions figées de type V N (telles que *avoir peur*) en cherchant dans un très gros corpus les suites de deux formes dont la première est étiquetée comme verbe et la seconde comme nom commun (à condition qu'il ne s'agisse pas d'un nom de jour ou de certains autres noms). On peut même chercher les cas où ce motif est suivi d'une préposition, pour rechercher les compléments obliques régis par ces constructions (*avoir besoin de*, *faire face à*,...).

Nous avons donc défini un petit langage de description de motifs, et développé un compilateur qui construit à partir d'un fichier de motifs un analyseur spécifique qui reconnaît ces motifs dans un corpus donné en entrée. Nous avons appliqué cette méthode sur un gros corpus journalistique étiqueté. Pour diminuer au maximum le silence, nous avons recherché certains motifs dans le corpus original, d'autres dans une version épurée des formes étiquetées comme adverbes, et d'autres dans une version épurée des formes étiquetées comme adverbes ou adjectifs.

Cette méthode nous a permis d'une part d'enrichir le Le*fff* de diverses façons (par exemple, toutes les constructions figées de type *V N* et leurs cadres de sous-catégorisation ont été obtenus par cette méthode, ainsi qu'un ensemble d'objets indirects en *de-obj* et en *à-obj* qui manquaient dans le Le*fff*), mais aussi d'apprendre des données statistiques sur les dépendances syntaxiques.

C'est de cette manière que nous avons acquis les données nécessaires aux expériences préliminaires de désambiguïsation sémantique de forêts d'analyse syntaxique que nous avons menées sur le résultat de l'analyse de corpus journalistique par le système FRMG (voir 5).

2.2 Description et identification des motifs morphosyntaxiques

2.2.1 Langage de description de motifs

Nous avons défini un langage très simple de description de motifs. On peut le considérer comme un langage régulier qui prend en entrée un corpus étiqueté. Une règle se présente sous la forme suivante :

$$\textit{motif\_complet} > \textit{relation}.$$

Le *motif_complet* est une suite de motifs élémentaires, qui peuvent reconnaître un mot unique, séparés par un espace. Un motif élémentaire est de la forme :

$$p_+ : r_+ \setminus p_- : r_-,$$

où les p sont des préfixes d'étiquettes et les r des expressions régulières sur les formes fléchies. Un tel motif élémentaire reconnaît une forme étiquetée pour peu que son étiquette commence par p_+, que la forme fléchie soit reconnue par r_+, que son étiquette ne commence pas par p_-, et que la forme fléchie ne soit pas reconnue par r_-. Les r sont facultatives (et leur absence supprime le « : » correspondant). La partie $p_- : r_-$ est facultative (et son absence supprime le « \ » correspondant).

La *relation* est de la forme *prédicat*$(n_1, \ldots, n_k)$, et a la signification suivante : une relation de type *prédicat* est construite entre le n_1-ième,..., et le n_k-ième lemme associé par l'étiqueteur aux formes fléchies correspondantes dans le motif. On peut remplacer un n_i par une chaîne de caractère entre guillemets, et alors cette chaîne sera utilisée à la place du lemme.

Prenons quelques exemples. Considérons la règle suivante :

```
D- A- N-C- > qual(3,2)
```

Elle reconnaît les motifs *déterminant - adjectif - nom commun*, et construit pour ces motifs une relation *qual* entre le nom commun et l'adjectif. Un autre exemple :

```
\P- D- N-C- V-\V--K > subj(4,3)
```

Cette règle reconnaît les séquences *déterminant - nom commun - verbe* qui ne sont pas précédées d'une préposition et dont le verbe n'est pas un participe passé (étiquette V--K). Elle construit, en cas de reconnaissance, une dépendance subj entre le nom commun et le verbe. Soit enfin la règle suivante :

```
V- N-C-\:(ensemble|point|monsieur|lundi|grâce) P-:à > v-n-prep(1,2,3)
```

Elle reconnaît les séquences *verbe - nom commun - préposition « à »*, pour peu que le nom commun ne soit pas l'un de ceux indiqués[9], et construit alors une relation de v-n-prep entre le verbe, le nom commun et la préposition *à* qui introduit peut-être un complément oblique sous-catégorisé par la construction. Naturellement, rien ne garantit qu'il s'agisse bien d'une construction figée de type *V N*, et encore moins que la préposition *à* introduise effectivement un complément sous-catégorisé. Ce n'est que l'analyse statistique des motifs reconnus dans un corpus volumineux (typiquement par un simple comptage) qui fera apparaître des tendances pertinentes.

2.2.2 Compilation des motifs

Une telle technique, puisqu'elle se place au niveau des formes fléchies et des lemmes, nécessite le traitement de corpus importants (au moins quelques dizaine de millions de mots). Ce qui signifie que la reconnaissance des motifs doit être efficace. Nous avons donc développé un compilateur permettant de construire automatiquement, à partir d'un fichier de motifs (au format indiqué ci-dessus), un analyseur en *perl*.

2.3 Mise en œuvre et résultats

Comme indiqué précédemment, nous avons défini un certain nombre de motifs, qui s'appliquent à trois variantes de corpus étiqueté : le corpus complet, le corpus privé des formes étiquetées comme adverbes, et le corpus privé des formes étiquetées comme adverbes ou adjectifs. Nous avons donc compilé ces motifs en trois analyseurs disctincts, chacun étant destiné à une de ces trois variantes.

Les motifs ne sont pas directement reconnus sur le corpus étiqueté, mais sur un sous-produit de ces corpus qui contient l'ensemble des n-grammes de couples forme/étiquette assortis de leur nombre d'occurrences, où n est la largeur maximale de la fenêtre requise par les motifs. Ces

[9]Pour simplifier la règle, nous ne les avons pas tous indiqués. Il s'agit principalement de forclusifs (comme *point*), des adverbes (comme *ensemble*) et des têtes de prépositions composées (comme *grâce* pour *grâce à*) susceptibles d'être mal étiquetés comme noms communs, ainsi que des jours de la semaine. Pour la règle équivalente avec la préposition « de », nous avons également exclu les titres (comme *monsieur*).

n-grammes sont relativement longs à construire, mais permettent ensuite une recherche rapide des motifs.

Nous avons utilisé environ 180 millions de mots du *Monde diplomatique*, étiqueté par une version de `TreeTagger` entraîné sur le corpus Paris 7. Seules les versions sans adverbes et sans adverbes ni adjectifs ont été utilisées. La version sans adverbe comporte 168 millions de mots qui constituent plus de 66 millions de 4-grammes différents.

Pour l'instant, nous n'avons exploité que trois catégories de motifs, que nous allons étudier successivement : des motifs de détection de constructions figées à objet nominal sans déterminant (type *avoir peur (de)*, des motifs ayant trait aux compléments (directs ou indirects) sous-catégorisés par des verbes, et des motifs d'identification des relations entre noms communs et adjectifs épithètes.

2.3.1 Figements *V N* / *V N prep*

Comme indiqué plus haut, un certain nombre de motifs recherchant des suites *verbe - nom commun*, éventuellement suivis d'une préposition, permettent de débroussailler certaines constructions verbales figées, celles du moins où l'objet direct est figé, et réalisé par un nom commun sans déterminant[10]. Le L*efff* ne comportant initialement aucune des informations nécessaires à la modélisation de ces constructions (à deux ou trois prototypes près), nous avons utilisé de tels motifs, puis traité les résultats en sorte de construire automatiquement une liste de constructions *V N* les plus probables (car les plus fréquemment soupçonnées), avec des cadres de sous-catégorisation envisagés[11].

Appliqués à la version du corpus sans adverbes ni adjectifs[12], nous obtenons des résultats dont un extrait est donné dans le tableau 6 (on notera que nous avons laissé les constructions avec le verbe *être*).

2.3.2 Sous-catégorisation

Les motifs d'extraction d'informations de sous-catégorisation ont un double rôle :

[10]Certaines de ces constructions sont des constructions à verbe support, d'autres non. Nous ne faisons pas de différence ici, nous recherchons simplement à répertorier les constructions à objet direct sans déterminant.

[11]Ce traitement procède comme suit. Si une séquence *verbe - nom commun* est suivie dans moins de 10% des cas d'une préposition, on ne propose aucun complément sous-catégorisé. Dans le cas contraire, on propose un complément introduit par chaque préposition représentant au moins 30% des occurrences du patron *verbe - nom commun - préposition*. On considère qu'il n'y a pas facultativité du ou des compléments que si le patron *verbe - nom commun - préposition* correspond à plus de 95% des occurrences du patron *verbe - nom commun*.

[12]Ce qui permet, à côté de *j'ai peur*, de prendre en compte des séquences telles que *je n'ai pas peur* ou *j'ai souvent peur*, puisque les adverbes ont été éliminés (y compris le forclusif *pas*, traité comme un « adverbe » négatif).

Rang	Occurrences	Verbe	Nom	Cadre de sous-catégorisation
1	33002	avoir	besoin	(de-obj)
2	19842	tenir	compte	(de-obj)
3	17803	avoir	lieu	(de-obj)
4	11224	rendre	compte	(de-obj)
5	11013	être	question	(de-obj)
6	9352	avoir	raison	(de-obj)
7	8694	avoir	droit	(à-obj)
8	8253	faire	face	(à-obj)
9	8107	faire	partie	(de-obj)
10	6944	être	temps	(de-obj)
24	2929	avoir	tendance	à-obj
29	2442	avoir	confiance	(dans-obj,en-obj)
46	1270	prendre	soin	(de-obj)
47	1236	faire	peur	(à-obj)
48	1230	faire	mention	(de-obj)
49	1214	faire	foi	
50	1183	avoir	hâte	(de-obj)
100	540	interjeter	appel	
200	240	être	source	de-obj
300	158	faire	pitié	(à-obj)
400	112	faire	naufrage	
500	83	faire	silence	

TAB. 6 – Constructions de type V N (avec objet nominal sans déterminant). Ces résultats, extraits automatiquement, sont des suggestions à destination d'un linguiste chargé de développer un lexique syntaxique. On indique les constructions les plus fréquentes, quelques constructions intéressantes, ainsi qu'une séquence de 5 constructions et quelques constructions moins bien classées pour montrer la pertinence des résultats même au-delà constructions les mieux classées.

– vérifier que les cadres de sous-catégorisation présents dans le L*efff* sont le plus exhaustifs et le moins sur-générateurs possibles,
– acquérir des informations quantitatives, en particulier sur les têtes possibles (et leur fréquence) des sujets et compléments sous-catégorisés, en vue d'une exploitation sous forme de contraintes probabilistes de restriction de sélection (voir la fin du chapitre 5)[13].

Nous considérons à la fois les sujets et les compléments directs et indirects, en recherchant les syntagmes nominaux ou prépositionnels qui suivent immédiatement les formes verbales (ou qui les précèdent immédiatement, pour les sujets)[14], mais aussi les pronoms clitiques. Ces derniers sont d'autant plus pertinents qu'ils produisent beaucoup moins de bruit que les syntagmes nominaux ou prépositionnels. Ils ne permettent toutefois que l'extraction d'informations moins riches : tous les compléments ne sont pas cliticisables, et un même clitique non nominatif peut être ambigu quant à son cas et quant à la préposition introductrice du complément qu'il « remplace ».

Nous avons mis en place une heuristique permettant d'extraire des occurrences trouvées de ces motifs une ébauche de cadre de sous-catégorisation, en utilisant en particulier un mécanisme permettant de regrouper certaines prépositions en familles (en particulier pour extraire les locatifs, notés *loc-prep*). Les résultats, dont un aperçu est donné table 7, ne sont pas véritablement intéressants en soi : ils ne prennent leur intérêt que par comparaison avec un lexique existant ou en guise d'aide au linguiste qui développe un lexique. En particulier, ils regroupent tous les compléments trouvés, sans pouvoir distinguer plusieurs cadres possibles un même lemme.

On peut étendre cette technique au-delà des verbes, par exemples aux noms ou aux adjectifs, pour identifier ceux d'entre eux qui sous-catégorisent des compléments prépositionnels. Nous comptons faire ces expériences dans un avenir proche.

2.3.3 Adjectifs qualificatifs

Les motifs d'extraction de relation entre noms communs et adjectifs qualiticatifs épithètes ont également un double rôle :

– comme précédemment, acquérir des informations quantitatives sur les fréquences des dépendances entre nom commun et adjectif épithète,
– acquérir des informations quantitatives sur l'antéposition (certains adjectifs peuvent être antéposés au nom commun qu'ils qualifient, parfois de façon quasi-systématique).

Pour illustrer ceci, la table 8 donne les adjectifs détectés comme étant les plus souvent antéposés.

[13]Cette idée a été utilisée sous une forme ou une autre depuis longtemps (voir par exemple l'introduction de Basili *et al.* (1994)), en particulier pour tenter de résoudre les problèmes de rattachement de groupes prépositionnels.
[14]Sauf celles précédées d'un auxiliaire être, pour éviter les passifs

Rang	Occurrences	Verbe	Ébauche de cadre de sous-catégorisation
1	81391	être	(obj|vcomp)
2	52782	avoir	(obj),(à-obj|à-vcomp)
3	25717	faire	(obj|vcomp)
4	23494	pouvoir	obj|vcomp
5	11520	devoir	obj|vcomp
6	8624	permettre	(scomp|obj),(de-obj|de-vcomp/à-obj)
7	6743	faillir|falloir	(scomp|obj|vcomp),(à-obj)
8	6655	prendre	(obj)
9	6535	voir	(obj|vcomp),(loc-obj/à-obj)
10	6414	comprendre	(obj),(à-obj)
11	6375	donner	(scomp|obj),(à-obj|à-vcomp)
12	5708	aller	(obj|vcomp),(loc-obj)
13	5165	savoir	(obj|vcomp)
14	5145	dire	(obj|vcomp),(à-obj)
15	5025	vouloir	scomp|obj|vcomp
16	4548	venir	(scomp|obj|vcomp),(de-obj|de-vcomp)
17	4329	passer	(obj),(de-obj|de-vcomp)
18	4304	trouver	(scomp|obj),(loc-obj)
19	4018	devenir	(obj)
20	4002	rester	(obj),(loc-obj|loc-vcomp/à-obj|à-vcomp)
96	1511	entraîner	(obj),(loc-obj)
97	1470	ajouter	(obj),(à-obj)
98	1468	paraître	(scomp|obj|vcomp),(loc-obj/à-obj|à-vcomp)
99	1450	marquer	(obj),(par-obj)
100	1443	soutenir	(scomp|obj),(par-obj)
996	80	détailler	(obj),(de-obj)
997	80	démissionner	(scomp|obj),(de-obj)
998	80	découdre	(obj),(de-obj)
999	80	applaudir	(obj),(à-obj)
1000	80	amputer	(obj),(de-obj)

TAB. 7 – Informations de sous-catégorisation extraites pour les verbes, à l'exclusion des constructions avec pronom réfléchi. On notera que *obj* peut désigner en fait un attribut nominal (cf. *être*), et que l'étiqueteur utilisé ne sait pas distinguer *falloir* de *faillir*. Enfin, un « / » indique une alternative entre compléments obliques, sans que l'on sache s'il sépare deux compléments qui peuvent être réalisés en même temps ou deux compléments mutuellement exclusifs.

Rang	Occurrences	Adjectif	Fréq. d'antéposition
1	192373	autre	99%
2	2383	quelques	98%
3	718	piètre	97%
4	36247	honorable	97%
5	63328	petit	97%
6	126729	grand	96%
7	49842	bon	95%
8	17474	vieux	95%
9	26315	jeune	94%
10	12242	mauvais	94%
11	44373	certain	94%
12	2625	prétendu	94%
13	3201	joli	94%
14	21689	beau	93%
15	22558	meilleur	93%
16	10732	gros	93%
17	1376	brave	92%
18	10792	moindre	92%
19	35498	ancien	92%
20	49156	seul	91%
21	9935	excellent	91%
22	3488	fameux	90%
23	5450	double	89%
24	106806	nouveau	86%
24	16245	haut	85%

TAB. 8 – Adjectifs épithètes les plus fréquemment détectés comme antéposés, à l'exclusion des adjectifs numéraux et des adjectifs dont le nombre d'occurrences est inférieur à 300.

3 Conclusion : le Le*fff* 2

Au niveau morphologique, la méthode décrite au chapitre 6 a permis l'acquisition automatique de nombreux lemmes verbaux. Pour les noms, les adjectifs et les adverbes, nous sommes partis du lexique Multext (Ide et Véronis, 1994). Pour les catégories fermées, nous sommes partis d'un travail purement manuel. Par la suite, au fil de son utilisation dans nos outils (analyseurs, correcteur orthographique,...), du développement de notre formalisme morphologique MetaMorpho et du développement des techniques présentées dans ce chapitre, le Le*fff* a beaucoup évolué, y compris au niveau morphologique (listes de lemmes, informations morphologiques).

Mais le Le*fff* 2 est également un lexique syntaxique, comme indiqué au chapitre 5. Les techniques présentées dans ce chapitre nous ont aidé à développer, corriger et compléter ses informations syntaxiques. Naturellement, le travail purement manuel (développement des classes syntaxiques,...) a joué un rôle très important, tout comme les informations (extensionnelles) issues du travail de Lionel Clément qui nous a servi de point de départ. Malheureusement, les informations syntaxiques du Le*fff* 2, dans sa version actuelle, n'ont un degré de précision et de couverture satisfaisant que pour les verbes et les catégories fermées. Il manque encore pour les noms et les adjectifs (voire les adverbes) des informations de sous-catégorisation précises (pour le moment, la majorité des noms et des adjectifs ont tous un cadre de sous-catégorisation par défaut, qui est permissif).

Aujourd'hui[15], le Le*fff* 2 regroupe 404 483 formes fléchies dans 625 720 entrées (dont certaines sont factorisées). Ceci recouvre, entre autres, 6 798 lemmes verbaux, 37 673 lemmes nominaux (sans compter les noms propres), et 10 053 lemmes adjectivaux. À titre d'exemple, au niveau intensionnel, nous avons défini 228 classes syntaxiques pour les verbes.

Le Le*fff* 2, comme indiqué précédemment, est utilisé dans tous nos outils, mais aussi par d'autres équipes. En effet, il est accessible librement sur Internet, sur le site du Le*fff*, sous une licence libre (`www.lefff.net`).

Actuellement, le développement du Le*fff* se poursuit activement. Le format de représentation des cadres de sous-catégorisation dans le lexique extensionnel est en train d'évoluer, dans le cadre de discussions autour du projet *LexSynt* qui regroupe les principaux acteurs de la recherche sur les lexiques syntaxiques pour le français. Par ailleurs, nous travaillons à l'intégration dans le Le*fff* de mécanismes permettant son couplage avec des données statistiques acquises sur corpus (à partir de corpus, traités ou non par des étiqueteurs ou des analyseurs). Nous envisageons également de travailler sur les problématiques de représentation et d'acquisition d'informations plus sémantiques. Enfin, nous commençons un travail important, en collaboration avec d'autres équipes, sur la modélisation et l'intégration au Le*fff* de lexèmes complexes, et en par-

[15] Avril 2006, version 2.1.

ticulier d'expressions figées. Il s'agit là d'une problématique incontournable, intéressante mais difficile, à l'interface entre lexiques et grammaires.

Troisième partie

Utilisation des informations lexicales pour l’analyse automatique

Chapitre 8

Méthodologies d'analyse automatique

Avant de présenter les formalismes linguistiques, les analyseurs et les systèmes d'analyse sur lesquels nous avons travaillé, il nous semble nécessaire de présenter notre vision de la formalisation de la langue et de l'analyse automatique. Nous présenterons donc d'abord les différents faits linguistiques qu'il nous semble nécessaire de formaliser pour permettre l'analyse automatique d'énoncés[1]. Puis nous discuterons de l'apparente contradiction qui existe entre l'utilisation de formalismes génératifs pour effectuer cette formalisation de la langue et le continuum d'acceptabilité et de compréhensibilité que forment les énoncés que l'on est amené à analyser, dès lors que l'on cherche à traiter de corpus réels. Enfin, nous présenterons, en la justifiant autant que possible, une architecture possible pour l'analyse automatique.

Les chapitres suivants reposent pour tout ou partie sur les fondements présentés dans ce chapitre. Cependant, ces fondements ne nous sont apparus que progressivement au cours de nos travaux, et nous n'avons pas encore implémenté complètement l'architecture que nous proposons à la fin de ce chapitre. Toutefois, comme nous le verrons plus bas, les deux chapitres suivants présentent successivement deux analyseurs, reposant sur deux formalismes différents, qui sont deux approximations différentes quoique complémentaires de cette architecture. Ils ont chacun leurs avantages et leurs inconvénients, et sont des réalisations indispensables à l'implémentation effective de l'architecture complète que nous proposons. Cette implémentation, actuellement en développement, a déjà donné lieu à la réalisation d'un premier composant, un « chunker », écrit sous la forme d'une grammaire LFG et dont une évaluation est proposée au chapitre 12.

[1] Par *analyse* nous dénotons la tâche consistant à associer à un énoncé une structure syntaxique voire syntaxico-sémantique. Toutes les applications du traitement automatique des langues ne nécessitent pas une analyse : pour certaines, un étiquetage (déterministe) est suffisant. Mais ce n'est pas l'objet de ce chapitre ni des suivants.

1 Principes de formalisation du langage

Nous n'avons pas pour objectif le développement d'une théorie linguistique. Mais la construction d'analyseurs automatiques ne peut se faire sans une définition motivée des structures que ces analyseurs ont pour tâche de construire.

1.1 Niveaux de formalisation de la langue

Toute modélisation du langage est la modélisation d'une interface entre sens et énoncés linguistiques. Par conséquent, toute modélisation du langage doit nécessairement étudier la nature de cette interface, c'est-à-dire définir la nature de la relation entre éléments sémantiques de représentation du monde et éléments linguistiques (p.ex. phonèmes). Cette relation prend la forme d'une double articulation, ainsi nommée par Martinet mais introduite par Saussure. La première articulation est celle selon laquelle tout énoncé s'articule en une suite de *signes* regroupant à la fois un sens, le *signifié*, et une forme vocale (ou, par abus de langage, textuelle), le *signifiant*[2]. La deuxième articulation, quant à elle, est celle selon laquelle le signifiant d'un signe de première articulation s'articule en une succession d'unités distinctives (les phonèmes).

La formalisation de la langue telle que nous l'étudions ici est donc située au niveau de la première articulation. Toutefois, cette articulation peut être elle même analysée comme faisant intervenir des unités linguistiques de différentes natures, ou *niveaux*, dont Benveniste montre qu'elles comportent toutes de façon *nécessairement et simultanément inséparable* une *forme* et un *sens* (Benveniste, 1966, ch. X), que l'on peut identifier à un signifiant et un signifié[3]. En généralisant ces constats, on peut en venir avec Mel'čuk généraliser les notions de signifiant et de signifié à ces unités linguistiques appartenant à un niveau ou à un autre, et à affirmer que *de façon générale, est signifié linguistique tout élément linguistique qui correspond à un autre élément linguistique plus près*[4] *de la surface phonétique. Symétriquement, est signifiant linguistique tout élément linguistique qui correspond à un autre élément linguistique plus près du sens.*

Il est généralement conclu à partir de telles considérations que tout modèle du langage doit introduire un certain nombre de niveaux *disjoints* et souvent *ordonnés*, tout élément appartenant à un certain niveau étant signifiant pour un élément du niveau immédiatement précédent et signifié pour un élément du niveau immédiatement suivant. On identifie ces niveaux, ou le

[2] On peut considérer que cette idée est déjà présente dans la Grammaire dite de Port-Royal Lancelot et Arnauld (1660) : Ainſi, l'on peut conſiderer deux choſes dans ces ſignes : La premiere ; ce qu'ils ſont par leur nature, c'eſt à dire, en tant que ſons & caracteres. La ſeconde ; leur ſignification ; c'eſt à dire, la maniere dont les hommes s'en ſeruent pour ſignifier leurs penſées.

[3] Benveniste limite cette analyse aux unités des niveaux inférieurs à celui de la phrase.

[4] Bien que la relation d'ordre ainsi présupposée ne soit pas évidente.

regroupement de plusieurs de ces niveaux, aux différentes disciplines linguistiques : phonétique, morphologie, syntaxe, sémantique, analyse du discours, voire pragmatique. La plupart des formalismes actuels se limitent cependant aux niveaux morphologique, syntaxique, et parfois sémantique. De plus, on trouve parfois des raffinements à l'intérieur de ces niveaux, ou d'autres niveaux en amont ou en aval. Ces niveaux sont par exemple structurels en HPSG (avec les différents types de traits), explicites en LFG (structures fonctionnelles, structures de constituants) et en GUST (sémantique, syntaxe, morphologie).

Cette stratification des niveaux n'a pourtant rien de nécessaire. Nous verrons justement qu'elle peut être remise en cause pour différentes raisons. Nous conserverons le terme *niveau*, mais sans l'affubler de la notion de successivité qu'il a donc souvent. Nous qualifierons ainsi de *niveau* de la description linguistique une classe cohérente de faits linguistiques de même nature.

1.2 Morphèmes, chunks, syntagmes et dépendances

Au niveau de la première articulation, les faits linguistiques mis en jeu dans les travaux en linguistique formelle et en linguistique computationnelle semblent pouvoir se grouper en quatre niveaux (tels que définis ci-dessus) :

- les *morphèmes*, dont la morphologie décrit la combinaison en unités syntaxiquement atomiques[5] appelées *formes*,
- les *chunks*[6], suites continues et non-enchâssées de formes (Abney, 1991) dont la réalité est visible (entre autres) au niveau prosodique, et qui disposent d'une *tête*, unité syntaxiquement et sémantiquement principale,
- les *syntagmes*, qui sont des combinaisons structurées, enchâssées et éventuellement discontinues de chunks et de formes n'appartenant à aucun chunk, disposant d'au moins une tête (mais éventuellement de plusieurs en cas de coordination),
- les *dépendances syntaxiques*, qui sont des liens asymétriques entre deux formes (ou éventuellement chunks ou syntagmes), indiquant une hiérarchie syntaxique,
- les *dépendances sémantiques*, qui sont des liens asymétriques entre deux formes, indiquant une hiérarchie sémantique ; on notera qu'à une dépendance syntaxique peut correspondre une dépendance sémantique de même sens, mais que cela n'a rien de nécessaire : pour une dépendance sémantique (resp. syntaxique) donnée, la dépendance syntaxique (resp. sémantique) correspondante peut être en sens inverse, ou ne pas exister.

[5]Par *syntaxiquement atomiques* nous voulons dire que ces combinaisons sont insécables, sauf à donner naissance à des combinaisons différentes.

[6]Nous emprunterons à regret ce terme à l'anglais, faute de bon équivalent français. Le terme de *syntagme non-récursif*, parfois employé, introduit une inutile confusion avec la notion de *syntagme*.

1.3 Lexique et sémantique

Comme nous l'avons vu au chapitre 1, nombre de formalismes grammaticaux développés actuellement sont dits *lexicalisés*, c'est-à-dire que l'on ne peut décrire de structure qui ne soit associée à un élément du lexique (c'est le cas par exemple des LTAG, pour TAG lexicalisés, cf. Group (1995)) : il n'y a pas à proprement parler de grammaire, en dehors des règles générales de combinaison de ces structures. Ces formalismes, ainsi que d'autres formalismes fortement (quoique non complètement) lexicalisés ont inévitablement tendance à se présenter sous la forme de lexiques très volumineux et très redondants Clément et Kinyon (2003), puisque des structures identiques ou similaires sont souvent associées à tous les mots ayant un comportement identique ou similaire (morphologique, syntaxique, sémantique,...). Or il se trouve que ces redondances sont essentiellement des invariants de sémantique : si deux éléments du lexique ont une structure en commun, ou si leurs structures ont des parties ou des éléments communs, c'est parce que les deux éléments du lexique transcrivent des sens qui ont des points communs eux-mêmes. De ce fait, la linguistique informatique porte depuis quelques années un intérêt de plus en plus important à la sémantique, faisant ainsi progresser la notion de modèle linguistique depuis celle de « simple » système formel vers celle, justifiée précédemment, de modèle d'interface entre le langage et une représentation du monde. Chacun des modèles existant met en jeu une quantité plus ou moins grande d'informations sémantiques : faible en TAG, elle est déjà plus importante dans des formalismes comme HPSG ou GUST.

Cette prise de conscience a mis en valeur le lien (en partie au moins) causal entre la sémantique et les structures associées aux éléments du lexique[7]. Ce lien est décomposé *de facto* en deux volets : un lien entre une représentation structurée des sens (ontologies, comme WordNet ou ses dérivés, cf. Miller *et al.* (1990), ou Vossen *et al.* (1997)) et un certain nombre de comportements (caractéristiques grammaticales), puis un lien entre ces comportements et les structures des formalismes linguistiques. Le premier lien est étudié par des travaux comme ceux de Pustejovsky (cf. typiquement Pustejovsky (1995)), Levin (cf. p.ex. Levin (1993)), Dorr (cf. Dorr et Jones (1995)) ou Saint-Dizier (cf. p.ex. Saint-Dizier (1998)), et prend généralement le nom de *sémantique lexicale*. Le second lien est principalement représenté par les travaux sur les *métagrammaires* (Candito, 1996, 1999). Toutefois, ces travaux ne sont pas utilisés directement aujourd'hui pour l'analyse ou la génération de texte. Les travaux de sémantique lexicale restent principalement descriptifs, et les travaux sur les métagrammaires ne servent qu'à générer, stocker et garder à jour des grammaires classiques, obtenues par compilation (initialement des

[7] Ces structures peuvent elles-même contenir des informations sémantiques. L'idée dont il est question ici est que ces informations sémantiques pourraient ne pas être complémentaires des autres informations, mais être au moins partiellement leur cause.

grammaires TAG, mais des essais ont été tentés pour obtenir des LFG, cf. Clément et Kinyon (2003)).

1.4 Formaliser l'interaction entre tous les faits linguistiques

Des deux paragraphes précédents, on déduit qu'une formalisation satisfaisante de la langue devrait respecter les deux principes suivants :

- la factorisation de l'information, problématique que nous avons abordée dans les chapitres concernant la représentation d'informations lexicales,
- la prise en compte de tous les faits linguistiques, et en particulier (outre la morphologie) les chunks, les syntagmes, les dépendances syntaxiques et les dépendances sémantiques, avec toutes les contraintes qui en régissent le fonctionnement (topologie, cadres de sous-catégorisation, sémantique lexicale, mais également phénomènes de figement,...).

Pourtant, comme nous l'avons vu au chapitre 1, la plupart des formalismes linguistiques sont des formalismes à décorations, dont le squelette est un formalisme de réécriture *linéaire* (grammaires non-contextuelles, TAG) : dans une phrase, chaque forme peut être vue comme une ressource que l'on ne peut consommer qu'une seule fois (chaque forme ne peut être utilisé que dans une seule relation structurelle). Les structures construites sont alors des arbres, qui ne représentent qu'un seul type de faits linguistiques. Parmi tous ceux évoqués précédemment, un choix doit donc être fait, ainsi qu'une approximation supplémentaire afin de se satisfaire de structures arborescentes. La nature de ce choix permet de répartir les formalismes usuels en deux familles principales :

- soit seuls les chunks et les syntagmes sont pris en compte par le squelette, en faisant l'approximation que les syntagmes sont tous continus et ne comportent qu'une seule tête (d'où des traitements tout à fait bancals de la coordination) : ce sont les *grammaires de constituants*, ou *grammaires syntagmatiques*,
- soit seules les dépendances syntaxiques sont prises en compte par le squelette, en faisant le plus souvent l'approximation qu'elles induisent une structure d'arbre : ce sont les *grammaires de dépendances*.

Naturellement, une telle partialité du point de vue n'est pas du tout satisfaisante, et d'autres faits linguistiques, de nature variées, doivent être pris en compte. C'est le rôle des décorations, qui servent donc en un sens de béquille « fourre-tout » aux formalismes squelettes, béquille rendue nécessaire par leur puissance d'expression limitée, et singulièrement par leur linéarité. Nous verrons au début du chapitre 10 que cette situation est effectivement bien peu satisfaisante, et ce à la fois pour des raisons computationnelles et pour des raisons linguistiques, que l'on peut résumer dès maintenant en quelques points :

- la différence entre faits linguistiques traités par le squelette et faits linguistiques traités par les décorations est arbitraire,
- les faits linguistiques choisis pour être traités par le squelette ne forment pas nécessairement une structure d'arbres (constituants discontinus, graphes de dépendance,...),
- la stratification ainsi induite peut empêcher la modélisation d'interactions entre certains types de contraintes,
- la complexité algorithmique des décorations est le plus souvent exponentielle (elles reposent presque toujours sur l'unification de structures de traits),
- deux mécanismes différents coexistent ou se succèdent dans l'analyseur (perte d'efficacité, difficulté de faire du rattrapage d'erreurs, perte d'efficacité ou de précision des techniques de désambiguïsation en cours d'analyse,...)

C'est pourtant une solution qui comporte un certain nombre d'avantages :

- bien que de façon peu satisfaisante, elle permet malgré tout la prise en compte de nombreuses classes de faits linguistiques, et donc une modélisation raisonnablement pertinente d'un sous-ensemble important des phénomènes ; ainsi en LFG, la morphologie est principalement traitée par le lexique, les chunks et les syntagmes (considérés comme continus) sont traités par le squelette (c-structures), et les dépendances sont traitées dans les f-structures, quoique d'une façon qui a tendance à entretenir la confusion entre dépendances syntaxiques et dépendances sémantiques[8].
- les technologies à mettre en place pour construire des analyseurs sont mieux connues, plus répandues, et pour certaines très efficaces (cf. chapitre 9),
- il existe des formalismes largement étudiés et utilisés (y compris par des linguistes de terrain), comme LFG, fb-(L)TAG, HPSG, etc.

2 Désambiguïsation

L'analyse d'une phrase conduit souvent, en cas de succès, à plus d'une analyse, c'est-à-dire à plus d'une structure regroupant un ensemble cohérent de faits linguistiques démontrant la correction (parfaite ou imparfaite) de la phrase analysée. Dans la pratique, si l'on dispose d'une grammaire raisonnable, une phrase comportant un nombre moyen de mots conduit souvent à un très grand nombre d'analyses : c'est le problème de l'*ambiguïté*.

Le moyen le plus direct de réduire l'ambiguïté est naturellement de faire en sorte que le modèle associe un nombre minimal d'analyses à une phrase donnée. Ceci ne peut être atteint

[8] D'où, entre autres, des difficultés de traitement de l'auxiliation.

que de deux façons : soit l'on réduit la richesse des structures produites[9], soit l'on augmente le nombre de contraintes associées à chaque analyse, c'est-à-dire que l'on augmente *de façon contrôlée* la richesse des structures produites. Il faut en effet éviter que les contraintes ajoutées soient elles-mêmes si ambiguës que les ambiguïtés qu'elles permettent de lever soient en plus petit nombre que celles qu'elles introduisent.

Mais face aux ambiguïtés effectivement introduites par le modèle, on peut se poser plusieurs questions lors de l'analyse :

- Comment choisir une ou plusieurs « meilleures » analyses parmi l'ensemble des analyses complètes (de toute la phrase) : c'est ce que nous appellerons la *désambiguïsation globale*.
- Comment choisir, en cours d'analyse, une ou plusieurs « meilleures » sous-analyses et ne continuer qu'avec elles, en espérant avoir gardé le plus possible de sous-analyses conduisant à des analyses complètes : c'est ce que nous appellerons la *désambiguïsation interne*.
- Avant l'analyse proprement dite, comment sélectionner pour chaque forme un sous-ensemble des décorations morphologiques qu'elle est susceptible de porter, afin de réduire l'ambiguïté que devra gérer l'analyseur : c'est ce que l'on appelle généralement l'*étiquetage morphosyntaxique*.
- Lorsque l'analyseur tombe sur une alternative, comment faire en sorte, si l'on veut obtenir aussi vite que possible au moins une analyse, qu'il sache quel choix essayer en premier : c'est ce que nous appellerons l'*ordonnancement*.

Bien que nous pensions qu'il s'agit d'un problème fondamental, tant d'un point de vue théorique (informatique et linguistique, voire psycholinguistique) que pratique, nous n'avons pas encore travaillé sur l'ordonnancement. Nous garderons donc en tête le fait qu'une architecture de modélisation et d'analyse de la langue doit pouvoir intégrer un ordonnanceur, mais nous ne décrirons pas les différentes techniques qui permettent de l'implémenter.

Les trois premiers problèmes, quant à eux, sont des variantes d'un même problème plus général : étant donné un état de l'analyseur représentant un certain degré d'ambiguïté, comment réduire cette ambiguïté — partiellement ou totalement, c'est-à-dire en ne conservant qu'une seule (sous-)analyse — en minimisant le risque d'empêcher ainsi la construction de certaines analyses complètes[10]. Nous dénoterons ce problème par le terme générique de *désambiguïsation*.

[9] Un « analyseur » construisant simplement le DAG des formes constituant une phrase produit des structures simples, mais en nombre relativement limité.

[10] On notera donc qu'il y a une frontière très nette entre grammaire et techniques de désambiguïsation : la grammaire sert à construire des analyses, les techniques de désambiguïsation servent à choisir parmi ces analyses. D'un point de vue génerativiste, on peut reformuler cela comme suit : la grammaire sépare les énoncés agrammaticaux des énoncés grammaticaux, les techniques de désambiguïsation choisissent, pour les énoncés grammaticaux, parmi leurs analyses.

On peut regrouper les techniques de désambiguïsation (qui peuvent s'appliquer à chacun des trois problèmes regroupés sous ce terme) en deux familles, que nous allons évoquer successivement :

- les techniques par règles, ou techniques *heuristiques*,
- les techniques qui reposent sur un modèle *probabiliste*.

Dans les deux cas, le paradigme est le même : on associe à chaque (sous-)analyse un poids, on classe les (sous-)analyses selon ces poids, et on ne conserve que la ou les (sous-)analyses les mieux classées. Ce qui différencie ces deux familles est la nature et le mode de calcul de ce poids.

2.1 Désambiguïsation heuristique

Un mécanisme de désambiguïsation heuristique repose sur une *fonction de classement* qui permet d'associer à chaque (sous-)analyse un poids, uniquement en fonction des données utilisées par l'analyseur (analyses existantes, DAG d'entrée,...). La fonction de classement[11] peut calculer par exemple :

- des paramètres structurels de la (sous-)analyse, tels que sa profondeur ou sa « platitude »,
- des paramètres linguistiques de la (sous-)analyse, afin par exemple de ne retenir que les analyses maximisant le nombre de dépendances argumentales ou minimisant le nombre de syntagmes nominaux non-saturés (tels que les noms communs sans déterminants),
- des paramètres inter-analyses, afin par exemple d'éliminer une sous-analyse ayant certaines propriétés dès lors qu'une autre sous-analyse existe ayant certaines autres propriétés.

L'un des avantages de ces fonctions de classement est qu'elles ne dépendent pas de données externes qui résultent d'un apprentissage sur corpus. C'est ce qui différencie les techniques de désambiguïsation heuristiques des techniques de désambiguïsation probabilistes. Mais c'est aussi la cause de leur inconvénient principal : sauf à être motivées linguistiquement, elles ont une certaine part d'arbitraire.

L'autre avantage des techniques de désambiguïsation heuristiques est que l'on peut attribuer à chaque règle un certain degré de confiance. Quand nous introduirons la notion de backtrack, nous verrons que l'on peut avoir besoin de remettre en cause des choix faits par les techniques de désambiguïsation. Savoir quelles règles sont plus dignes de confiance et quelles règles sont plus risquées peut être à cet égard très intéressant.

[11] Une fonction de classement construit un ordre partiel entre les (sous-)analyses. Le cas échéant, cet ordre partiel peut correspondre à une partition des (sous-)analyses en deux classes, une classe des (sous-)analyses rejetées et une classe des (sous-)analyses conservées.

2.2 Désambiguïsation probabiliste

Les techniques de désambiguïsation probabilistes ont pour principe le calcul de la probabilité de chaque (sous-)analyse compte tenu d'un corpus d'apprentissage sur lequel a été appris un modèle probabiliste. La fonction de classement du paragraphe précédent est donc remplacée par un modèle probabiliste. Le plus souvent, ce modèle probabiliste procède par combinaison de n-grammes calculés le long des structures en jeu (arcs de dépendance ou de constituance, DAG d'entrée, chaîne de dérivation).

Dans le cas particulier de l'étiquetage morphosyntaxique, on parle d'*étiquetage probabiliste*. Puisque l'on se place en amont de l'analyse proprement dite, la seule structure disponible est le DAG d'entrée : le modèle est généralement un modèle n-gramme calculé sur le DAG.

L'avantage des techniques probabilistes de désambiguïsation est leur finesse et leur fondement empirique. Mais elles reposent sur la disponibilité de corpus d'apprentissage, ce qui induit des inconvénients majeurs. En particulier, de tels corpus ne sont pas toujours facilement disponibles ni adaptés[12], et ils attestent souvent une certaine forme de la langue (par exemple la langue journalistique) qui ne permet pas la construction de modèles vraiment génériques (par exemple adaptés au traitement de corpus de courrier électronique ou de langues de spécialité).

Par ailleurs, comme évoqué au chapitre 1, il est parfois fait usage de mécanismes de désambiguïsation probabiliste d'une façon excessive, en utilisant des modèles à la fois simples et très ambigus de la langue, et en se reposant sur la phase de désambiguïsation pour choisir la bonne analyse. Cette remarque ne remet pas en cause la pertinence des techniques probabilistes de désambiguïsation, mais seulement certaines utilisations qui en sont faites.

2.3 Méthodologies complexes de désambiguïsation

Naturellement, il est rare que l'on se contente d'une seule règle de désambiguïsation heuristique. En revanche, peu de systèmes utilisent à la fois des méthodes heuristiques et un modèle probabiliste pour désambiguïser (si l'on exclut l'étiquetage morphosyntaxique, très largement utilisé). L'utilisation de deux modèles probabilistes différents, l'un pour la désambiguïsation interne et l'autre pour la désambiguïsation globale[13], est en revanche plus répandue. C'est un exemple d'une technique que l'on peut utiliser avec tous les types de techniques de désambiguïsation, appelée reclassement (*reranking*). L'idée est que l'analyse est faite en s'aidant d'un

[12] On peut cependant imaginer la construction de corpus analysés automatiquement à l'aide d'analyseurs n'utilisant pas de techniques de désambiguïsation probabiliste, puis l'utilisation de tels corpus pour l'apprentissage de modèles probabilistes. Il pourrait s'agir là d'un moyen approprié pour utiliser des méthodes probabilistes tout en se passant de la phase coûteuse d'annotation manuelle de gros corpus.

[13] En toute rigueur, le modèle utilisé pour la désambiguïsation interne est souvent également utilisé à la racine, avant que l'on ne mette en œuvre l'autre modèle sur celles des analyses complètes qui ont été conservées.

modèle de désambiguïsation interne qui est simple, pour garantir une bonne efficacité de l'analyseur. Une fois à la racine, il ne reste qu'un nombre raisonnable d'analyses (on peut choisir par exemple les 10 meilleures si l'on est dans un cadre probabiliste). On fait le pari que la « bonne » analyse est parmi celles-ci, et on fait alors usage d'un modèle plus complet, plus complexe et plus coûteux en temps pour départager ces quelques analyses. Sur les meilleurs[14] analyseurs probabilistes pour l'anglais, le reclassement permet de gagner une dizaine de pourcents (en relatif) sur la précision des analyses (la mesure utilisée étant en général la proportion de mots pour lesquels le gouverneur a été correctement trouvé).

3 Robustesse

Comme nous l'avons vu au chapitre 1, il existe un grand nombre de formalismes pour la modélisation des langues. Ces formalismes sont des langages formels dans lesquels sont écrits des descriptions des langues appelées *grammaires*. Ces grammaires décrivent aussi bien que possible les contraintes qui caractérisent les phrases « correctes » de la langue. La vérification de ces contraintes construit les relations structurelles qui relient les mots des phrases correctes.

Comme nous l'avons vu, les formalismes peuvent se classer en deux catégories, selon que les contraintes ainsi énoncées sont strictes ou violables. Si elles sont strictes, comme dans le cas le plus fréquent, celui des formalismes à décorations, le formalisme est dit *génératif*. Si elles peuvent être violées, le formalisme est dit *non-génératif*. Les formalismes génératifs, qui sont de loin les plus répandus, ont l'avantage d'avoir des propriétés théoriques et algorithmiques claires, sur lesquelles on peut s'appuyer pour développer des analyseurs (ou des générateurs) aussi efficaces que possible. À l'inverse, le point de vue non-génératif a l'immense intérêt de chercher à extraire de l'information de tout énoncé, y compris les énoncés manifestement incorrects[15] et les énoncés corrects mais non décrits par la grammaire (souvent incomplète).

3.1 Formalismes génératifs et réalité de la langue

Au premier abord, les formalismes génératifs sont pourtant en contradiction avec la réalité de la langue. En effet, une grammaire d'un formalisme génératif partitionne les énoncés en deux

[14] Selon un classement pour le moins artificiel : les analyseurs pour l'anglais sont pour la plupart développés en utilisant une partie du Penn Treebank comme corpus d'apprentissage, et une autre partie de ce même corpus journalistique comme corpus d'évaluation. Mais les classements ainsi obtenus ne sont pas nécessairement pertinents pour indiquer les performances de ces analyseurs sur d'autres types de corpus — performances qui ne sont pas excellentes.

[15] Par *énoncés manifestement incorrects*, nous voulons parler des énoncés ressentis comme incorrects par les locuteurs natifs de la langue. C'est une définition floue, mais il est délicat d'être plus précis, puisque nous ne voulons pas nécessairement parler de tous les énoncés qui sont incorrects aux termes des grammaires classiques normatives (grammaires scolaires, etc…).

classes disjointes : les énoncés corrects, qui peuvent être analysés, et les énoncés incorrects, sur lesquels rien ne peut être dit. Au contraire, on peut caractériser la réalité de la langue par un double continuum : un *continuum d'acceptabilité* (production d'énoncés à grammaticalité variable) et un *continuum de compréhensibilité* qui lui est orthogonal (compréhensibilité variable des énoncés, y compris ceux qui sont pleinement grammaticaux et ceux qui sont pleinement agrammaticaux).

Mais cette contradiction est le résultat de la confusion de deux notions :

– la *langue normée*, qui est l'ensemble des énoncés perçus comme étant canoniquement corrects,
– la *langue performée*, ensemble des énoncés performés (ou susceptibles de l'être) dont il est possible d'extraire du sens et de la structure.

Cette différence est d'autant plus sensible que l'on considère des sources s'éloignant de la langue écrite générale (on peut par exemple définir un axe allant des corpus journalistiques aux corpus de transcription de l'oral, en passant par les corpus de spécialité et les corpus de courrier électronique).

Une fois cette distinction faite, il est clair que choisir une définition générative de la langue normée n'empêche pas d'affirmer l'existence des continuums d'acceptabilité et de compréhensibilité. En effet, il est raisonnable qu'une définition formelle (générative) de la langue normée exclue les énoncés qui ne sont pas parfaitement grammaticaux. Mais ceci n'empêche pas, au moins en théorie, que l'on utilise également une telle grammaire générative pour extraire un maximum d'informations des phrases non reconnues, voire de savoir mesurer leur taux d'acceptabilité ou de compréhensibilité .

3.2 Analyse non-générativiste à l'aide d'un formalisme génératif : la robustesse

Un tel point de vue permet l'utilisation de formalismes génératifs avec un point de vue non-générativiste : face à un énoncé qui n'est pas reconnu par une grammaire générative (soit parce qu'elles ne rentrent pas dans la langue normée, soit parce que la grammaire est incomplète), on peut malgré tout chercher à extraire le maximum d'informations sous une forme ou une autre, telle que : suivantes :

– analyses légèrement incorrectes,
– analyses correctes d'un énoncé légèrement différent de celui que l'on avait en réalité en entrée,
– analyses partielles (plusieurs sous-analyses de sous-parties de l'énoncé, ou analyses issues de la vérification de certaines contraintes mais pas toutes),

C'est ce que nous appellerons *l'analyse non-générativiste à l'aide d'un formalisme génératif*, et que nous résumerons par le terme de *robustesse*. Les mécanismes mis en œuvre sont des mécanismes de rattrapage et de tolérance d'erreurs.

3.3 Robustesse et ambiguïté

En un sens, les techniques de robustesse sont des techniques de désambiguïsation. En effet, face à un énoncé qui ne respecte pas toutes les contraintes énoncées par la grammaire, un nombre considérable de corrections sont envisageables, ou, de façon duale, d'analyses partiellement incorrectes. Une part importante du travail effectué par les mécanismes de robustesse est précisément de sélectionner un nombre raisonnable de corrections (ou d'analyses partiellement incorrectes), si possible de façon à « minimiser » la distance entre l'énoncé corrigé et l'énoncé original (ou le « degré d'incorrection » des analyses produites).

Mais à l'inverse, les techniques de désambiguïsation (désambiguïsation interne et étiquetage morphosyntaxique) peuvent effectuer de mauvais choix qui empêchent la construction d'analyses complètes de l'énoncé considéré. Les techniques de robustesse classique peuvent alors intervenir, mais il est bien plus satisfaisant de se rendre compte, dans pareille situation, que la cause du problème est dans un élagage excessif dû aux désambiguïsations. En effet, on peut alors remettre en cause les résultats de désambiguïsations de plus en plus anciennes, jusqu'à se retrouver dans une situation où l'on peut effectivement construire une analyse complète. C'est ce que l'on appelle le *backtrack*[16]. Pour le moment, nos principaux analyseurs ne savent pas encore mettre en œuvre cette technique. Pourtant, l'analyseur que nous sommes en train de construire le peut : la forêt d'analyse construite par le chunker (déjà opérationnel) n'est pas réellement élaguée par les techniques de désambiguïsation. Les parties rejetées de la forêt sont simplement marquées comme telles, avec différents niveaux de rejet, de sorte que l'on peut revenir en arrière progressivement, jusqu'à pouvoir poursuivre les traitements — traitements qui n'existent pas encore.

Une autre idée, complémentaire mais pas du tout incompatible avec la notion de backtrack, est dénotée en général par le terme d'*analyse itérative*. Le principe en est le suivant : on peut considérer qu'une proportion importante des énoncés que l'on peut être amené à analyser ne comportent pas de surprises ni de phénomènes très particuliers. En conséquence, une grammaire relativement simple, et des techniques de désambiguïsation très filtrantes ont une chance significative de mener à au moins une bonne analyse, et auront des performances inégalables en termes d'efficacité. Il est donc raisonnable de commencer toute analyse par une tentative avec

[16] Cette technique, intellectuellement très satisfaisante, nécessite pourtant que les techniques de désambiguïsation effectuent une bonne hiérarchisation des énoncés rejetés, au risque de voir l'efficacité de l'ensemble diminuer de façon significative.

une grammaire simple et des méthodes très filtrantes de désambiguïsation. En cas d'échec, on peut alors tenter l'analyse avec une grammaire plus couvrante et des méthodes moins aventureuses de désambiguïsation, et ainsi de suite. On peut considérer le paradigme de l'analyse itérative comme une technique de robustesse car les dernières grammaires utilisées, en cas d'échec de toutes les précédentes, peuvent être des grammaires sur-génératrices, c'est-à-dire qu'elles reconnaissent des énoncés linguistiquement incorrects, soit en étant très permissives, soit en ne vérifiant tout simplement pas certains types de contraintes (comme les contraintes morphologiques, ou à l'autre extrémité les contraintes sémantiques). C'est une technique qui a l'avantage de préserver l'efficacité en moyenne de l'analyseur dans son ensemble, mais il n'est pas toujours facile, à partir d'une grammaire jugée « normale », d'en extraire des grammaire plus simples ou à l'inverse de les dégrader ou les généraliser en grammaires sur-génératrices.

3.4 Robustesse et multiplicité des contraintes

Nous avons vu, à propos des problèmes posés par l'ambiguïté des grammaires, qu'une des façons de limiter cette ambiguïté est d'augmenter de façon contrôlée le nombre de contraintes prises en compte par la grammaire. Ceci n'est pas sans conséquence sur la problématique de la robustesse. En effet, on constate facilement que les contraintes les plus souvent prises en compte sont celles qui sont également le moins souvent violées : les contraintes morphologiques sont bon an mal an presque toujours respectées, et les contraintes syntaxiques (ordre des mots, respect des cadres de sous-catégorisation, structuration de la phrase) sont relativement bien satisfaites (et bien qu'existent des phénomènes tels que les zeugmas, ou les inévitables incorrections présentes entre autres dans les corpus de transcription de l'oral).

Mais la prise en compte de contraintes sémantiques, pourtant importantes, nécessite la disponibilité de techniques convenables de robustesse. En effet, le non-respect de contraintes telles que les restrictions de sélection n'est pas toujours le résultat d'incorrections de langage, loin s'en faut : des phénomènes tels que les métaphores, les synecdoques, les métonymies voire les oxymores sont des exemples à la fois fréquents et parfaitement corrects de « violation » des contraintes sémantiques que l'on est en droit de vouloir prendre en compte dans la grammaire.

4 De la théorie à la pratique

À ce stade de notre présentation des différentes techniques que l'on peut mettre en œuvre au sein des analyseurs, il est clair qu'un analyseur complet devrait pouvoir utiliser des techniques de désambiguïsation à la fois heuristiques et probabilistes, disposer de techniques permettant le backtrack ainsi que l'analyse itérative, mais aussi le reclassement (reranking). De plus, un tel

analyseur devrait idéalement reposer sur un formalisme et une grammaire (ou plusieurs grammaires si l'on utilise le paradigme de l'analyse itérative) qui permettent la prise en compte d'un maximum de familles de faits linguistiques, des contraintes qui les régissent, et des interactions entre ces familles.

Restent trois questions :

- l'étiquetage morphosyntaxique est-il nécessaire voire souhaitable ?
- selon quelles modalités est-il raisonnable de faire usage de techniques de désambiguïsation interne, technique qui introduit par nature un certain risque d'échec supplémentaire ?
- quand et comment prendre en compte les dépendances sémantiques et les contraintes qui les régissent (restrictions de sélection) ?

Avant de proposer les grandes lignes d'une architecture qui nous semble raisonnable pour la modélisation et l'analyse des langues (et singulièrement du français), nous discuterons successivement de ces trois questions.

4.1 Faut-il utiliser un étiqueteur morphosyntaxique ?

L'utilisation d'un étiqueteur morphosyntaxique en amont des analyseurs est une pratique habituelle. Elle a l'avantage de réduire considérablement l'ambiguïté, en particulier sous sa forme probabiliste[17], à l'aide de modèles n-grammes (typiquement $n = 3$, avec les techniques de repli appropriées). L'intérêt de ces modèles est qu'ils peuvent être très efficacement implémentées avec des automates finis. Toutefois, les performances des étiqueteurs morphologiques plafonnent autour de 90% à 98% (suivant la complexité des traits associés aux formes), ce qui a pour effet que la plupart des phrases de longueur significative ont au moins une étiquette erronée. De nombreux travaux tentent de surmonter ces problèmes (étiquetage ambigu, étiquettes sous-spécifiées,...), mais les résultats sont parfois peu convaincants. Or des techniques efficaces telles que celles que nous utilisons dans nos analyseurs permettent de se passer de l'étiquetage morphologique. C'est donc un choix tout à fait raisonnable.

4.2 Quand désambiguïser ?

Une autre question importante est de savoir si l'on veut privilégier l'efficacité apportée par la désambiguïsation interne, ou si l'on refuse les risques qui lui sont associés. En effet, la désambiguïsation interne peut éventuellement éliminer des sous-analyses qui auraient pu conduire à des analyses globales, ce qui peut même conduire à un échec de l'analyse de la phrase bien que celle-ci soit grammaticale.

[17] Voir cependant Boullier (2003c) pour un exemple d'étiquetage ambigu non-probabiliste

Naturellement, une partie de la réponse à cette question constitue un choix délibéré : la désambiguïsation interne permet une augmentation importante de l'efficacité (voire de la complexité), mais elle induit un certain risque.

Pourtant, il est possible de faire usage de la désambiguïsation interne sans augmenter de façon significative ce risque. On peut en effet considérer que certaines sous-analyses possèdent un certain type de saturation linguistique à laquelle correspond certaines propriétés. Ainsi, le verbe principal d'une relative trouve tous ses compléments au sein de cette relative : une règle éliminant toutes les analyses d'une relative dans lesquelles les contraintes de sous-catégorisation ne sont pas respectées est une règle peu risquée, car une violation de ces contraintes au niveau de la relative conduira presque sûrement à une violation de ces mêmes contraintes une fois l'analyse complètement achevée.

Il y a donc moyen de contrôler, au moins partiellement, le risque que l'on prend à faire usage de la désambiguïsation interne. Mais cela reste globalement un choix de la part du développeur de grammaires et d'analyseurs.

4.3 Quand et comment utiliser les informations sémantiques ?

La dernière des trois questions que nous nous posions précédemment concerne les informations de sémantique lexicale. En effet, peu de grammaires à large couverture intègrent la prise en compte d'informations sémantiques, au point qu'on peut considérer qu'il y a un fossé important entre les modélisations de la syntaxe, qui sont à l'œuvre dans les analyseurs les plus courants, et les modélisations de la sémantique, qui restent souvent cantonnées au niveau théorique.

Les contraintes de sémantique lexicale sont pourtant cruciales. Sans donner plus de poids à cet argument informel, un étranger produit souvent des énoncés grammaticalement incorrects mais tout à fait compréhensible, parce que les contraintes dites de restriction de sélection suffisent souvent à reconstruire la structure syntaxique de l'énoncé, même en l'absence des marqueurs syntaxiques correspondants.

Il reste à savoir si ces contraintes doivent être prises en compte après l'analyse syntaxique proprement dite, sous la forme d'une sorte de post-traitement, ou si au contraire leur pertinence désambiguïsatrice ne fait pas pencher pour une utilisation *en parallèle* aux autres familles de faits linguistiques. Nous reviendrons sur cette question dans le chapitre sur le formalisme Méta-RCG, mais nous pouvons dès maintenant indiquer que la prise en compte de différents types de faits en parallèle nécessite l'utilisation de formalismes non-linéaires. Si l'on veut rester dans le cadre plus standard des formalismes à décorations, ces contraintes sémantiques sont rejetées dans les décorations, et tous les problèmes liés à cette architecture se posent alors (voir chapitre 10).

Cependant, les contraintes sémantiques sont moins bien respectées que les autres, ainsi que nous l'avons évoqué précédemment. La prise en compte d'informations sémantiques ne peut donc se faire de façon satisfaisante et à grande échelle qu'à la condition d'être couplées à des mécanismes de robustesse adéquats.

4.4 Architecture possible pour l'analyse automatique

Après toutes ces remarques, il nous semble que l'analyse automatique peut être réalisée dans de bonnes conditions selon l'architecture suivante (et en considérant comme un pré-requis évident la disponibilité d'un lexique regroupant toutes les informations lexicales nécessaires) :

1. Chaîne de traitement pré-syntaxique pour prendre en entrée des corpus bruts (sans étiquetage),
2. Analyseur proprement dit, avec à tous les niveaux des techniques de robustesse et de désambiguïsation variées :
   - analyseur non-contextuel reposant sur une grammaire de chunks, avec désambiguïsation globale multi-niveaux (pour permettre différents niveaux de backtrack) ;
   - analyseur reposant sur un formalisme non-linéaire pour construire les syntagmes, les dépendances syntaxiques et les dépendances sémantiques ; cet analyseur prend en entrée la forêt issue du chunker et la complète ;
   - équations fonctionnelles à la LFG sur la forêt ainsi obtenue, pour calculer les traits peu discriminants tels que les traits morphologiques.

Trois systèmes d'analyses seront présentés, qui sont des approximations différentes de cette architecture, avec certains avantages et certains inconvénients :

- deux systèmes d'analyse reposant sur des grammaires LFG et le constructeur d'analyseurs SXLFG : SXLFG_L, qui repose sur une grammaire à large couverture du français, et SXLFG_B, chunker reposant sur une grammaire à la LFG et qui est un premier pas vers l'implémentation de l'architecture ci-dessus ;
- un système d'analyse reposant sur un nouveau formalisme, Méta-RCG, qui est un formalisme non-linéaire : c'est un « autre premier pas » vers l'architecture ci-dessus.

Le système SXLFG_L se distingue de l'architecture ci-dessus sur deux points : tout d'abord, c'est un formalisme à décorations (et à squelette linéaire), avec tous les problèmes associés à ces formalismes ; ensuite, nous n'avons pas encore développé de façon directement utilisable des mécanismes de désambiguïsation probabiliste. Mais c'est un système d'analyse efficace, robuste, et à large couverture. Le système Méta-RCG, quant à lui, n'est pas un système à large couverture. Il ne dispose pour le moment d'aucun mécanisme de robustesse. Mais il repose sur

un formalisme non-linéaire, et montre tout l'intérêt de ces formalismes pour la prise en compte en parallèle de contraintes de natures diverses.

Chapitre 9

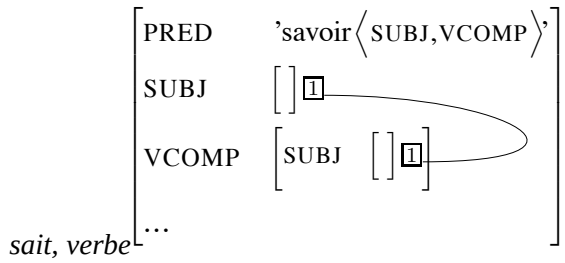
Analyse syntaxique LFG

Le développement d'un nouvel analyseur syntaxique pour le formalisme LFG (*Lexical-Functional Grammars*, cf. p. ex. Kaplan (1989)) n'est pas en soi très original. Il en existe déjà un certain nombre, comme ceux de Andrews (1990) ou Briffault *et al.* (1997). Mais le plus connu des systèmes d'analyse LFG est sans conteste le projet XLE (Xerox Linguistic Environment), qui est le successeur du Grammars Writer's Workbench Kaplan et Maxwell (1994); Riezler *et al.* (2002); Kaplan *et al.* (2004). XLE est un projet d'importance, qui concentre un grand nombre de compétences linguistiques et techniques, repose sur un point de vue similaire au nôtre quant à l'équilibre entre analyse de surface et analyse profonde, et a été utilisé avec succès pour analyser des corpus bruts de taille importante.

Toutefois, ces analyseurs n'utilisent pas toujours de la manière la plus complète possible les différentes techniques algorithmiques de partage de calcul et de représentation compacte de l'information qui permettent d'écrire un analyseur efficace malgré le fait que le formalisme LFG, comme de nombreux formalismes qui reposent sur l'unification, est NP-complet. Naturellement, notre but n'est pas d'écrire un nouvel XLE mais d'étudier comment robustesse et efficacité peuvent être atteints dans l'analyse syntaxique LFG de corpus bruts.

La construction des structures de constituance (ou c-structures) ne pose théoriquement[1] pas de problème particulier, car elles sont décrites par des grammaires non-contextuelles (CFG) sous-jacentes aux LFG et appelées ici grammaires *squelettes*. En effet, les algorithmes généraux d'analyse pour les CFG sont bien connus (Earley, GLR, CKY,...). En revanche, la construction efficace des structures fonctionnelles (ou f-structures) est beaucoup plus problématique. Nous avons développé un module de calcul de ces structures qui partage les sous-structures communes à plusieurs analyses. De plus, des mécanismes de rattrapage d'erreur à tous les niveaux permettent d'obtenir un analyseur robuste.

[1] Même si, en pratique, la disponibilité d'un *bon* analyseur est déjà plus délicate.




```
sait   verbe   [pred="savoir<subj,vcomp>", ... , @CtrlSubj];
@CtrlSubj = [subj=[]1, vcomp=[subj=[]1]];
```

Verbe → sait
(↑ pred) = 'savoir <subj,vcomp>'
(↑ vcomp subj) = (↑ subj)

FIG. 1 – Entrée lexicale (partielle) pour une forme verbale à contrôle sujet, codage de cette entrée via une macro et règle correspondante en LFG standard.

En parallèle, nous avons adapté une grammaire LFG du français pour obtenir un analyseur du français à la fois efficace et raisonnablement couvrant. Nous y reviendrons au chapitre 12.

1 Formalisme employé

Le formalisme employé est une variante de LFG, qui se distingue du formalisme standard sur deux points principaux. Tout d'abord, les entrées lexicales ne contiennent ni équations fonctionnelles ni règles lexicales. Il ne s'agit là que d'optimisations : d'une part les équations fonctionnelles sont remplacées dans le lexique par la structure fonctionnelle correspondante, qui est le résultat de l'application de ces équations ; d'autre part on remplace les règles lexicales par le résultat de leur application (on aura donc des entrées lexicales correspondant par exemple à la forme impersonnelle, au lieu d'indiquer que la règle lexicale « impersonnel » est applicable sur l'entrée adéquate). Ainsi, les entrées lexicales sont des triplets contenant une forme fléchie, une catégorie (c'est-à-dire un terminal de la grammaire squelette) et une structure fonctionnelle sous-spécifiée avec partage possible de structures. Un exemple d'entrée lexicale, avec la façon dont elle est encodée et la règle LFG standard correspondante, est indiquée à la figure 1 (elle définit une forme fléchie *sait* qui est de la catégorie *verbe* et dont la structure fonctionnelle associée contient elle-même une structure fonctionnelle vide et partagée, repérée par 1).

La seconde différence importante est que les équations fonctionnelles sont interprétées en sorte qu'elles ne puissent pas *modifier* les structures fonctionnelles associées aux non-terminaux de partie droite d'une règle. D'une manière générale, le calcul de l'information est strictement

CLITICS → (cld) clg (cll)	`CLITICS -> (cld) clg (cll)`
↓1 : (↑ à-obj = ↓1)	`$1 : ($$ à-obj = $1)`
(↑ obj de-obj) = ↓2	`$$ obj de-obj = $2)`
↓3 ∈ (↑ adjunct)	`$3 < ($$ adjunct)`

FIG. 2 – Les clitiques dans le cas d'une extraction du génitif de l'objet (règle simplifiée et codage). Les terminaux `cld`, `clg` et `cll` désignent respectivement les pronoms clitiques datif, génitif (*en*) et « locatif » (*y*).

synthétisé. Le contenu de la ou des structures fonctionnelles attachées au non-terminal de la partie gauche d'une règle de la grammaire squelette ne peut qu'être une constante ou se calculer (par unification) à partir des structures fonctionnelles attachées aux symboles de la partie droite de cette règle (structures directement données dans le lexique pour les symboles terminaux, comme vu précédemment). Toutefois, le calcul des structures fonctionnelles associées au symbole de partie gauche peut utiliser des *copies locales* des structures fonctionnelles associées aux symboles de partie droite. Ces copies locales peuvent être modifiées, ce qui a pour effet que la puissance d'expression n'est pas diminuée par rapport au modèle LFG standard. La seule différence est que l'obtention des structures fonctionnelles associées aux nœuds internes de la forêt nécessiterait un nouveau parcours de la forêt. Nous n'avons pas implémenté ce parcours, considérant que ce que nous construisons est suffisant : la forêt représentant les structures en constituants, les structures fonctionnelles associées à la racine de la forêt, dites structures fonctionnelles *principales*, et des repères permettant d'associer des sous-structures des structures fonctionnelles principales à des non-terminaux de la forêt (voir plus bas). Cette vision purement synthétisée, qui ne restreint pas la puissance d'expression de LFG, est la clef de notre stratégie d'évaluation, car elle permet d'introduire du partage de calcul pendant la construction des structures fonctionnelles et de réaliser cette construction sur la forêt d'analyse et non sur les analyses individuelles prises successivement (voir plus bas).

À titre d'exemple, considérons la règle permettant de décrire la disposition des clitiques d'un verbe fini en cas d'extraction clitique du génitif modifieur de l'objet (exemple : *Jean en mange une partie*). La règle complète et sa représentation en SXLFG sont données dans la figure 9.1. Comme Clément et Kinyon (2001), nous écrivons toutes les équations fonctionnelles au même niveau, en remplaçant l'opérateur ↓ associé au n-ième symbole de partie droite par $\downarrow n$.

Les opérateurs fonctionnels utilisés sont les suivants (un identifiant est un nom d'attribut, une expression régulière d'attributs pouvant utiliser les opérateurs de disjonction et d'itération de Kleene, ou un identifiant sous-spécifié de type `($`i `pcas)-obj`) :

unification (opérateur « = ») : l'équation est vérifiée si et seulement si l'unification entre les deux structures opérandes est possible et si elle réussit ; un échec de l'unification fait

échouer le calcul de la structure `$$` (c'est-à-dire ↑) associée au non-terminal de partie gauche,

parcours constructif de structure (opérateur « ») : avec un chemin vers une structure S en partie gauche et un identifiant i en partie droite (identifiant atomique ou expression régulière), rend la sous-structure de S de nom i en la créant si elle n'existe pas,

parcours non-constructif de structure (opérateur « `.` ») : comme le précédent, mais en cas de non-existence cela échoue ; cet opérateur est utilisé lorsque la partie droite est une expression régulière, pour parcourir toutes les structures existantes qui sont reconnues par l'expression régulière, sans créer de structure supplémentaire (s'il n'y en a pas, échec),

vérification de la présence (resp. absence) d'un attribut (opérateur « `+` » resp. « `-` ») : l'équation est vérifiée si la structure opérande existe (resp. n'existe pas),

unification « contrainte » (opérateur « `=c` ») : comme l'unification, mais la structure finale d'une analyse ne doit pas comporter d'attribut ayant subi une unification contrainte sans avoir subi également une unification,

ajout dans la liste d'adjoints (opérateur « `<` ») : ajoute le premier opérande au second, ce dernier devant être une liste d'adjoints, c'est-à-dire un chemin se terminant par un identifiant déclaré de type *liste d'adjoints* (classiquement `adjunct`),

conditionnement (opérateur « `:` ») : permet de contraindre la vérification d'une équation au fait qu'un symbole optionnel de partie droite soit effectivement réalisé ; par exemple, une équation de la forme `$1 :(`*éq*`)` est vérifiée soit parce que le premier symbole de la partie droite (`cld` dans l'exemple de la figure 9.1) dérive dans la chaîne vide, soit parce que l'équation *éq* est vérifiée.

Enfin, un mécanisme permet de représenter pour une structure fonctionnelle donnée les structures en constituants qui lui correspondent (voir plus bas).

2 L'analyseur SXLFG : analyse standard

2.1 Architecture globale

Le cœur d'un analyseur[2] produit par SXLFG est un analyseur CFG général qui traite le squelette CFG de la grammaire LFG. C'est un analyseur à la Earley (voir chapitre 3) qui repose sur un automate coin-gauche sous-jacent et qui est une évolution de Boullier (2003b).

[2]Nous appelons *analyseur* un programme construit à partir d'une grammaire et fournissant une ou plusieurs analyses selon cette grammaire pour une entrée donnée. Par conséquent, un constructeur d'analyseurs produit deux analyseurs différents à partir de deux grammaires différentes.

L'ensemble des analyses produites par cet analyseur est représenté par une forêt partagée. En réalité, cette forêt partagée peut être vue elle-même comme une grammaire non-contextuelle dont les productions sont des productions instanciées du squelette[3]. L'évaluation des équations fonctionnelles est réalisée au cours d'un parcours de bas en haut dans cette forêt. Un module de désambiguïsation, qui élimine les f-structures non sélectionnées, peut être alors invoqué à n'importe quel nœud de la forêt, y compris bien sûr à sa racine.

L'entrée de l'analyseur est un DAG de mots tel qu'en produit la chaîne de traitement pré-syntaxique SXPipe, décrite au chapitre 11 (tous les mots étant alors connus du lexique, y compris certains mots spéciaux représentant les tokens inconnus du corpus initial). Ce DAG est converti par le *lexeur* en un DAG de lexèmes (un lexème étant ici un symbole terminal du squelette CFG associé avec une structure fonctionnelle sous-spécifiée).

2.2 L'analyseur non-contextuel

Les évolutions de l'analyseur Earley par rapport à celui décrit dans Boullier (2003b) sont de deux types : il accepte les DAG en entrée et il dispose de mécanismes de rattrapage d'erreurs. Ce second point sera traité dans la section 9.3.1. Savoir prendre des DAG en entrée ne nécessite pas de changements considérables dans l'algorithme d'Earley, au moins du point de vue théorique[4]. Puisque l'analyseur Earley est guidé par un automate fini coin-gauche qui définit un sur-langage régulier du langage défini par le squelette CFG, cet automate accepte également les DAG en entrée (ce qui correspond à l'intersection de deux automates finis).

2.3 Calcul des f-structures

2.3.1 Remarques générales

Comme noté dans Kaplan et Bresnan (1982), si le nombre d'analyses CFG (c-structures) croît exponentiellement par rapport à la longueur de l'entrée, la construction et la vérification des f-structures associées prend un temps exponentiel. Nos expériences montrent que le squelette des grammaires LFG à large couverture peut être hautement ambigu (cf. chapitre 12). Ceci veut dire que l'analyse (complète) de longues phrases serait irréalisable. Bien qu'il soit pos-

[3]Si A est un symbole non-terminal de la grammaire squelette G, $A_{i..j}$ est un symbole non-terminal instancié si et seulement si $A \overset{+}{\underset{G}{\Longrightarrow}} a_i \dots a_{j-1}$, où $w = a_1 \dots a_n$ est la chaîne d'entrée et $\overset{+}{\underset{G}{\Longrightarrow}}$ la fermeture transitive de la relation *dérive*.

[4]Si i est un nœud du DAG et si l'on a une transition sur le terminal t vers le nœud j (sans perte de généralité, on peut supposer $j > i$) et si l'item Earley $[A \rightarrow \alpha \bullet t\beta, k]$ est un élément de la table $T[i]$, alors on peut ajouter à la table $T[j]$ l'item $[A \rightarrow \alpha t \bullet \beta, k]$ s'il n'y est pas déjà. Il faut faire attention à ne commencer une phase PREDICTION dans une table $T[j]$ que si toutes les phases Earley (PREDICTOR, COMPLÉTION et LECTURE) sont déjà achevées dans toutes les tables $T[i]$, $i < j$.

sible, en analyse non-contextuelle, de calculer et de stocker en un temps polynomial un nombre exponentiel (ou même non borné) d'arbres d'analyse dans une forêt partagée, ce résultat ne peut être transposé dans le cas des f-structures pour plusieurs raisons[5]. Cependant, ce comportement rédhibitoire (et bien d'autres) pourraient bien ne pas avoir lieu dans les applications pratiques de TAL, sans compter qu'un certain nombre de techniques, dont certaines sont décrites à la section 9.2.4, peuvent être appliquées pour restreindre cette explosion combinatoire.

Le calcul efficace de structures se combinant par unification est encore un champ de recherche actif. Cependant, ce problème est simplifié si le calcul des structures considérées peut se faire de façon cumulative, comme c'est le cas en LFG. Dans ce cas, le squelette (c'est-à-dire la forêt partagée) n'a pas besoin d'être modifié pendant la résolution des équations fonctionnelles. Si l'on adopte une stratégie de parcours de bas en haut dans la forêt partagée, les informations stockées dans les f-structures sont calculées de façon *synthétisée*. Ceci signifie que l'évaluation des f-structures sur une sous-forêt[6] n'est effectuée qu'une seule fois, même si cette sous-forêt est partagée par de nombreux nœuds pères. En réalité, l'effet d'une évaluation complète des équations fonctionnelles est d'associer à chaque nœud de la forêt partagée un ensemble de structures fonctionnelles partielles qui ne dépendent que des descendants dudit nœud (mais pas de ses nœuds pères ou frères).

Dans le cas où l'analyse est un succès, le résultat de l'analyseur LFG est l'ensemble des *structures fonctionnelles principales* (complètes et cohérentes si possible), c'est-à-dire l'ensemble des f-structures associées à la racine de la forêt partagée. De tels ensembles de f-structures (qu'il s'agisse ou non de f-structures principales) pourraient être factorisés en une seule f-structure contenant des valeurs disjonctives, comme dans XLE. Nous avons décidé de ne pas utiliser de telles valeurs disjonctives complexes, sauf pour les valeurs atomiques, mais plutôt d'associer à toute f-structure (principale ou non) un identifiant unique : deux f-structures identiques auront toujours le même identifiant tout au long du processus, et ne seront donc pas dupliquées. L'expérience montre que cette stratégie est payante et que le nombre total de f-structures distinctes construites au long d'une analyse complète reste tout à fait raisonnable, sauf peut-être dans certains cas pathologiques.

[5]Par exemple, il est possible, en LFG, de définir des f-structures qui encodent les analyses CFG individuelles. Si une forêt partagée de taille polynomiale en la taille de l'entrée représente un nombre exponentiel d'analyses, le nombre de f-structures différentes associées à la racine de la forêt partagée serait ce même nombre exponentiel d'analyses. En d'autres termes, il y a des cas où aucun partage de calcul des f-structures n'est possible.

[6]Si le squelette non-contextuel G est cyclique (c'est-à-dire $\exists A$ t.q. $A \xRightarrow[G]{+} A$), la forêt peut être un graphe général et non seulement un DAG. Bien que le générateur d'analyseurs non-contextuels utilisé sache traiter ce cas, nous l'excluons explicitement dans SXLFG. Naturellement, cette (petite) restriction ne veut pas dire que les structures fonctionnelles cycliques soient interdites. SXLFG sait gérer les f-structures cycliques, qui peuvent être un moyen élégant de représenter certaines relations linguistiques.

2.3.2 Mise en œuvre

Le calcul des f-structures se fait au cours d'un ou de plusieurs parcours de bas en haut de la forêt produite par le squelette non-contextuel (stratégie ascendante). Chacun de ces parcours est appelé une *passe*. À chaque passe, on fait toujours en sorte que le calcul des f-structures associées au symbole instancié[7] de partie gauche d'une production instanciée ne commence que lorsque l'on a terminé le calcul de toutes les f-structures associées à tous les symboles de partie droite de cette production instanciée. Dans le cas où un symbole de partie droite est un symbole terminal, les f-structures qui lui sont associées proviennent directement du lexique. Une fois le calcul fini, toutes les f-structures associées au symbole instancié de partie gauche sont regroupées dans un ensemble, auquel viennent s'ajouter toutes les f-structures calculées sur d'autres productions instanciées ayant le même symbole instancié en partie gauche.

L'évaluation des équations fonctionnelles associées à une production instanciée de la forêt partagée se fait comme suit. Une évaluation particulière des équations se fait en sélectionnant, pour chaque symbole instancié de la partie droite, une seule f-structure parmi l'ensemble des f-structures qui lui sont associées. Sur une production instanciée donnée, on effectue cette évaluation pour toutes les combinaisons de f-structures de partie droite possibles[8]. Chaque évaluation particulière peut ne produire aucun résultat (en cas d'échec d'unification), un résultat unique, ou plusieurs résultats, ce dernier cas n'ayant lieu qu'en présence d'équations comportant des expressions régulières sur les chemins, puisque plusieurs chemins peuvent conduire à un succès. Autrement dit, si l'on sélectionne un seul chemin pour chaque expression régulière qui intervient dans les équations, on obtient soit aucune f-structure résultat, soit une seule.

L'ordre dans lequel les équations fonctionnelles sont évaluées n'a aucune importance, à l'exception des équations comportant l'opérateur non-standard de *parcours non-constructif de structures* (opérateur « . »). Ces équations sont évaluées après toutes les autres, ce qui permet aux attributs référencés par cet opérateur d'avoir été construits par d'autres équations de la même règle[9].

L'évaluation se fait dans des structures de travail. Si la f-structure associée à un symbole instancié de partie droite est utilisée, elle est potentiellement copiée dans une structure de travail. En réalité, un mécanisme d'unification paresseuse a été mis en place, à l'image de ce qui

[7] Rappelons qu'un symbole instancié est de la forme $A_{i..j}$, où i et j sont des positions si l'entrée de l'analyseur est une chaîne, ou des numéros d'états si c'est un DAG.

[8] Naturellement, tout échec en cours d'évaluation disqualifie toutes les combinaisons comprenant les f-structures en jeu dans cet échec.

[9] On peut construire des jeux d'équations où l'ordre dans lequel sont évaluées les équations comportant l'opérateur « . » change les résultats. Mais d'une part le développeur de grammaire peut contrôler l'ordre d'évaluation des équations, et d'autre part une telle situation semble ne pas arriver dans la pratique lorsque l'on développe des grammaires réelles.

se passe dans XLE[10]. Ce mécanisme permet de ne copier une f-structure dans la structure de travail que si elle est susceptible d'être modifiée. À la fin d'une évaluation réussie, la structure de travail associée au symbole instancié de partie gauche est sauvegardée dans une structure globale et reçoit alors son identifiant unique (voir ci-dessous). Cet identifiant est ajouté, s'il ne s'y trouve pas déjà, à l'ensemble des identifiants déjà calculés précédemment pour ce symbole instancié (sur la même production instanciée ou sur d'autres). Les structures de travail associées aux symboles instanciés de partie droite ne sont pas sauvegardées, et ce même si elles ont été modifiées, pour éviter qu'une autre production instanciée ayant en partie droite ce même symbole instancié ne récupère ces modifications qui ne le concernent pas du tout. C'est ce qui fait le caractère *synthétisé* de l'évaluation des f-structures. C'est aussi ce qui fait que SXLFG n'associe pas directement les mêmes f-structures aux nœuds internes de la forêt que le modèle LFG standard[11] (au niveau de la racine, cette différence n'existe pas). Nous considérons en effet qu'il est plus pertinent d'associer à ceux des nœuds internes de la forêt qui le méritent non pas une f-structure particulière mais plutôt des références à des sous-structures des f-structures associées à la racine. Pour cela, nous permettons aux (sous-)f-structures de référencer des nœuds de la forêt (c'est-à-dire de la c-structure). Ceci est fait grâce à un attribut spécial, nommé A_{ij}, qui accumule dans une liste des identifiants de nœuds de la forêt (plus spécifiquement, des identifiants de productions instanciées), à l'aide d'une équation spéciale qui demande l'ajout de l'identifiant de la production instanciée courante à l'attribut A_{ij} de la f-structure associée à son symbole instancié de partie gauche[12].

En général, l'association d'un identifiant unique à une f-structure quelconque ne pose pas de problème : c'est son contenu qui permet de déterminer un identifiant. Cependant, c'est plus problématique dans le cas de f-structures cycliques. Il faut évidemment assurer une identification cohérente (une f-structure cyclique d'identifiant f qui fait référence à elle-même, directement ou à travers des sous-structures, se fera référence par l'identifiant f). Nous avons opté pour un compromis qui garantit, grâce à l'évaluation paresseuse, que deux ensembles identiques de

[10]Lorsque deux f-structures sont unifiées, nous ne copions que leurs parties *communes* qui sont nécessaire à la vérification de l'unifiabilité des deux f-structures. Ceci restreint la quantité de copies entre deux nœuds frères aux seules parties où ils interagissent. Naturellement, les nœuds-frères originaux sont laissés inchangés (et peuvent ainsi être utilisés dans un autre contexte).

[11]Bien que nous ne l'ayons pas réalisé, et comme suggéré précédemment, il serait facile de dupliquer les f-structures des nœuds internes et de compléter ces copies par les informations propres à chaque contexte. On obtiendrait alors une implémentation complète du modèle LFG standard. Le mécanisme des A_{ij}, évoqué ci-dessous, joue un rôle équivalent, et de façon mieux contrôlable.

[12]Si l'évaluation de ces équations permettant le calcul des valeurs des A_{ij} était effectuée conjointement au calcul des autres équations, on pourrait se retrouver avec un nombre exponentiel de f-structures qui ne différeraient que par la valeur de leur champ A_{ij}. SXLFG permettant de spécifier pour chaque attribut un numéro de passe, on ne fait le calcul des A_{ij} que dans une passe qui suit tous les autres calculs. La ou les autres passes ayant alors éliminé un très grand nombre d'arbres invalides, le calcul des A_{ij} se fait sur un nombre restreint d'arbres, assurant par conséquent un gain potentiellement exponentiel en temps d'analyse.

f-structures cycliques portent les mêmes identifiants dans de nombreux cas. Cependant, si un ensemble de f-structures cycliques est modifié, chaque f-structure de cet ensemble recevra un nouvel identifiant, même si ce nouvel ensemble pré-existait dans la structure globale.

2.4 Désambiguïsation interne et globale

Les applications des systèmes d'analyse ont souvent besoin d'un résultat désambiguïsé, nécessitant ainsi l'application de techniques de désambiguïsation sur les sorties ambiguës des analyseurs tels que ceux générés par SXLFG. Dans notre cas, ceci implique le développement de procédures de désambiguïsation permettant de choisir l'analyse la plus probable (ou les analyses les plus probables) parmi les f-structures principales. Ensuite, la forêt partagée est élaguée, en sorte de ne garder que les c-structures qui sont compatibles avec la ou les f-structures principales choisies (si l'on ne choisit qu'une seule f-structure, on n'obtient en général qu'une seule c-structure, mais ce n'est pas nécessairement le cas). C'est ce que l'on appelle la *désambiguïsation globale*.

D'un autre côté, sur tout nœud interne de la forêt, un nombre potentiellement très grand de f-structures peut être obtenu. Si rien n'est fait, ces nombreuses structures peuvent conduire à une explosion combinatoire qui empêche l'analyse de se terminer en un temps raisonnable. C'est la raison pour laquelle il semble approprié de permettre au développeur de la grammaire de répertorier un ensemble de symboles non-terminaux qui disposent d'une certaine forme de saturation linguistique qui rend possible l'application sur ces non-terminaux de techniques de désambiguïsation[13]. Ainsi, certains non-terminaux du squelette CFG qui correspondent à des portions de phrases linguistiquement saturées (par exemple des propositions) peuvent se voir attribuer une liste (ordonnée) de méthodes de désambiguïsation, chaque non-terminal pouvant recevoir sa propre liste. Ceci permet un filtrage immédiat en cours d'analyse des f-structures associées aux nœuds internes concernés qui auraient certainement (ou très probablement) conduit à des f-structures principales incomplètes, incohérentes, ou non sélectionnées par les mécanismes de désambiguïsation globale. Par ailleurs, cela conduit inévitablement à une réduction parfois très importante des temps d'analyse. Ce point de vue est en réalité une généralisation de la désambiguïsation classique (la désambiguïsation globale), en ce qu'elle en permet l'application sur d'autres nœuds que le nœud racine. Nous l'appelons *désambiguïsation interne*. On notera que nous employons le terme *désambiguïsation* dans un sens large, puisqu'une désambi-

[13] Une telle approche est plus satisfaisante qu'un *skimming* à l'aveugle qui stoppe l'analyse normale de la phrase lorsque le temps ou l'espace utilisé dépasse une certaine limite fixée par avance, et qui la remplace par une analyse dégradée qui s'impose de faire une quantité de travail bornée sur chaque non-terminal restant Riezler *et al.* (2002); Kaplan *et al.* (2004))

guïsation, globale ou interne, peut conduire à plus d'une analyse. En ce sens, *désambiguïsation* est à prendre comme un raccourci pour *désambiguïsation totale ou partielle*.

Les techniques de désambiguïsation sont en général partagées entre techniques probabilistes et techniques à règles. L'architecture de SXLFG permet l'implémentation des deux types de techniques, pourvu que les calculs puissent être réalisés sur les structures fonctionnelles. Elle permet d'associer un poids à chaque f-structure associée au non-terminal instancié considéré[14]. Appliquer une règle de désambiguïsation revient à éliminer toutes les f-structures qui ne sont pas optimales au sens de cette règle. Les règles associées au non-terminal courant sont appliquées en cascade (comme indiqué ci-dessus, la liste des règles et par conséquent leur nombre et leur ordre d'application peut varier d'un non-terminal à un autre).

Après l'application de ce mécanisme de désambiguïsation sur les structures fonctionnelles, la forêt partagée (qui représente les structures de constituants) est filtrée afin de correspondre exactement à la f-structure ou aux f-structures conservée(s). En particulier, si la désambiguïsation est complète (seule une f-structure a été conservée), ce filtrage conduit en général à une c-structure unique (un arbre).

3 Techniques d'analyse robuste

3.1 Rattrapage d'erreurs dans l'analyseur non-contextuel

La détection d'une erreur dans l'analyseur Earley[15] peut être due à deux phénomènes différents qui peuvent se cumuler : le squelette CFG n'a pas une couverture suffisante, ou l'entrée n'est pas une phrase correcte de la langue. Naturellement, bien que l'analyseur ne puisse pas faire la différence entre ces deux causes, les développeurs de analyseurs et de grammaires doivent les gérer différemment. Dans les deux cas, l'analyseur doit être en mesure de faire du *rattrapage* afin de continuer l'analyse et d'analyser correctement, autant que faire se peut, les portions correctes et incorrectes en préservant des relations raisonnables entre elles. La gestion des erreurs dans les analyseurs est un domaine de recherche qui a été principalement étudié dans le cas déterministe et rarement dans le cas général d'analyseurs non-déterministes, y compris pour les analyseurs non-contextuels.

L'idée que nous avons mise en œuvre est la suivante. Lorsque l'on arrive à un point j où l'analyse est bloquée, on va rechercher un intervalle $p..q$ incluant j ($p \leq j \leq q$), qui permettrait de poursuivre l'analyse s'il recouvrait des terminaux différents de ceux qui sont effectivement

[14] De nombreux travaux tels que ceux de Hindle et Rooth (1991) montrent la pertinence d'une désambiguïsation au niveau des relations de dépendance.

[15] Rappelons ici que l'algorithme d'Earley, comme l'algorithme GLR, a la propriété du préfixe valide. Ceci est encore vrai lorsque l'entrée est un DAG.

présents. Autrement dit, on cherche, avec différentes stratégies de recherche possibles, un intervalle de la chaîne ou du DAG d'entrée pour lequel il existe (au moins) une correction permettant la poursuite de l'analyse. Les entiers p et q sont alors appelés *bornes* du rattrapage. On voit que la philosophie sous-tendant cette méthodologie est que l'on construit dans tous les cas des analyses complètes, qui respectent la grammaire, quitte à « corriger » la chaîne ou le DAG d'entrée (c'est-à-dire en réalité quitte à faire comme s'il était différent de ce qu'il est)[16].

Soit une chaîne ou un DAG source, d'état initial 1 et d'état final n. Supposons que l'analyse est bloquée dans l'analyseur Earley à la position j, c'est-à-dire dans la table $T[j]$[17]. On appelle *signature* d'une tentative de rattrapage tout couple d'entiers (p, q) qui entoure ce point j ($1 \leq p \leq j \leq q \leq n$), p et q étant appelées *bornes* de cette tentative, comme indiqué précédemment. On appelle *stratégie de rattrapage* toute suite de signatures : l'application d'une stratégie consiste à essayer de se rattraper en essayant successivement les signatures de la stratégie, jusqu'à trouver un rattrapage ayant la bonne signature, ce qui met un terme à l'application de la stratégie. Signalons que si la signature $(1, n)$ est dans la stratégie, le succès est certain (voir ci-dessous).

L'écrivain de la grammaire peut choisir une stratégie parmi celles proposées par défaut dans SXLFG, mais peut également écrire sa propre stratégie. Comme nous allons le comprendre, une signature (p, q) telle que $q - j > j - p$ a tendance à favoriser la préservation de la partie du texte déjà analysée au détriment de la correction de la portion du texte non encore examinée. On a l'inverse si $q - j < j - p$. Si deux signatures (p, q) et (p', q') sont telles que $q < q'$, la deuxième tentative va corriger davantage de symboles que la première. Si $p' < p$ elle réutilisera une plus faible quantité des analyses (partielles) déjà réalisées. Notons qu'une signature (p, q) peut être essayée même s'il n'existe pas de chemin dans le DAG entre les noeuds p et q.

Les quatre stratégies proposées par SXLFG sont qualifiées de *avant*, *arrière*, *mixte-avant* et *mixte-arrière* (cf. tableau 1). La stratégie avant privilégie les choix déjà réalisés par l'analyseur (et a donc tendance à sauter du texte source sans l'analyser). La stratégie arrière remet en cause les choix déjà réalisés par l'analyseur. Les stratégies mixtes visent à minimiser les portions du

[16]Il aurait été possible également de construire des analyses qui respectent la chaîne ou le DAG d'entrée, quitte à produire des forêts ne respectant pas la grammaire ou le lexique. Dans le cas de LFG, il aurait alors fallu définir le comportement des équations fonctionnelles sur les nœuds agrammaticaux de la forêt obtenue, ce qui n'est pas nécessairement satisfaisant. Un dernier choix consiste à toucher non pas à l'entrée (chaîne ou DAG) ou à la grammaire, mais à la sortie de l'analyseur non-contextuel, c'est-à-dire à produire une structure qui n'est pas une forêt, mais un ensemble de forêts constituant des analyses partielles. On peut considérer cette dernière possibilité comme dégradant l'analyseur non-contextuel en un analyseur de surface (*shallow parser*) en cas d'échec de l'analyse globale. Mais notre objectif étant de construire des analyses profondes, nous avons jugé préférable de construire dans tous les cas une analyse globale.

[17]Nous laissons de côté dans cette description le cas où l'entrée est un DAG et où l'on est bloqué simultanément à plusieurs endroits du DAG, qui sont non-comparables par la relation de précédence linéaire. Ce cas est cependant géré par SXLFG.

avant	(j,j), $(j,j+1)$, …, (j,n), $(j-1,j+1)$, …, $(j-1,n)$, …, $(1,n)$
arrière	(j,j), $(j-1,j)$, …, $(1,j)$, $(j-1,j+1)$, …, $(1,j+1)$, …, $(1,n)$
mixte-avant	(j,j), $(j,j+1)$, $(j-1,j)$, $(j,j+2)$, $(j-1,j+1)$, $(j-2,j)$, …, $(1,n)$
mixte-arrière	(j,j), $(j-1,j)$, $(j,j+1)$, $(j-2,j)$, $(j-1,j+1)$, $(j,j+2)$, …, $(1,n)$

TAB. 1 – Description des stratégies de rattrapage dans l'analyseur non-contextuel. Les positions (ou états) initiale et finale sont notées 1 et n, et la position où l'analyse est bloquée est notée j.

source qui ne seront finalement pas analysées, avec une priorité aux choix déjà effectués par l'analyseur dans la stratégie mixte-avant.

Pour une signature (p,q), le rattrapage procède comme suit. On examine tous les items $[A \rightarrow \alpha\bullet\beta, i]$ de la table $T[p]$. Pour chacun de ces items on va examiner le suffixe $\beta = X_1 \ldots X_k$ de gauche à droite, jusqu'à trouver un X_h dont une chaîne dérivée commence par un symbole terminal r tel qu'il existe une transition sortante sur r depuis le noeud q du DAG. Si c'est le cas, l'item $[A \rightarrow \alpha X_1 \ldots \bullet\, X_h \ldots X_k, i]$ est ajouté à la table $T[q]$. Ceci garantit au moins une transition valide de $T[q]$ sur r. Sinon, on a atteint la position réduction de la production courante et on examine alors un autre item de la table $T[p]$[18].

Lorsque tous les items de $T[p]$ ont été examinés, si la table $T[q]$ est non vide, la signature (p,q) est déclarée *productive*, et le rattrapage est un succès. L'analyse normale reprend donc sur la table $T[q]$. On est sûr qu'aucune nouvelle erreur ne sera détectée dans $T[q]$. En revanche, une autre erreur peut très bien être détectée en $T[q+1]$ (ou dans les tables suivantes) et donc déclencher un nouveau rattrapage. Ce nouveau rattrapage peut bien sûr avoir une signature productive (p',q') qui englobe la borne précédente (p,q) ($p' \leq p$ et $q' > q$) et qui, par conséquent,

[18]Notons que nous n'effectuons pas la réduction précédente car cette opération nous ferait considérer une signature de la forme (i,q), avec $i \leq p$. Si $i < p$, on serait amené à tenter un rattrapage qui n'a pas la signature actuellement considérée pour tenter un rattrapage. De plus, si la stratégie de rattrapage est raisonnable, il y a de bonnes chances pour que l'on tente plus tard cette signature (i,q). Si en revanche $i = p$, la signature est la bonne, mais l'item $[B \rightarrow \gamma \bullet A\delta, m]$ est nécessairement un item de $T[p]$, et à ce titre a déjà été ou sera étudié par ailleurs.

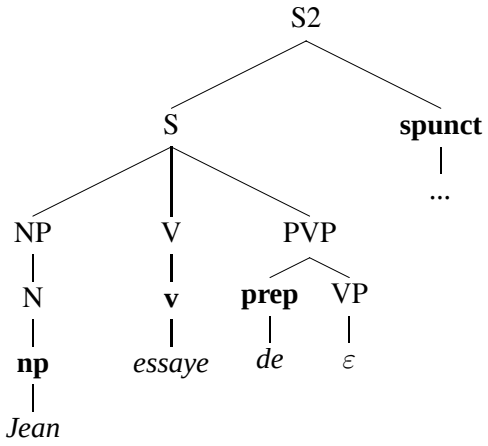


FIG. 3 – Structure en constituants simplifiée pour la phrase incomplète « Jean essaye de... ». La dérivation de VP dans la chaîne vide est le résultat d'un rattrapage avant, et conduira à une structure fonctionnelle incomplète (pas de « pred » dans la sous-structure correspondant au nœud VP).

invalide et remplace le rattrapage précédent. Si $p' > q$, les deux rattrapages coexistent. Le rattrapage avec la borne $q' > q$ n'est pas essayé si $p < p' \leq q$ (et produit donc un échec pour cette signature). La raison est double. Le cas $p < p' < q$ nous ferait essayer un rattrapage dans une table $T[q']$ inaccessible (car sautée par le rattrapage précédent). Le cas $p' = q$ correspond en fait à une *composition* de deux signatures (p, q) et (p', q') en une nouvelle signature (p, q') plus large, laquelle sera, si la stratégie le permet, essayée plus tard.

Nous avons indiqué que si la signature $(1, n)$ est dans la stratégie, le succès de cette stratégie est certain. Cela provient du fait que par construction de l'analyseur Earley, la table $T[1]$ contient l'item $[S' \rightarrow \bullet S\,\$, 0]$, où $\$$ désigne le marqueur de fin de chaîne, S est l'axiome de la grammaire, et S' le super-axiome (si w est une phrase, la production $S' \rightarrow S\,\$$ permet donc de reconnaître la chaîne $w\$$). Le mécanisme décrit précédemment permet de mettre l'item $[S' \rightarrow S \bullet \$, 0]$ dans la table $T[n]$ car $\$$ est par définition une transition sortante valide pour le noeud n. Cette situation est appelée *rattrapage trivial*.

Un exemple d'analyse produite à l'aide d'un rattrapage est montrée dans la figure 3 : dans ce cas, aucune transition sortante sur **spunct** n'est disponible après avoir reconnu **prep**. Un rattrapage avant est donc tenté, et réussit moyennant l'insertion d'un VP « vide » après le symbole **prep**, permettant ainsi la construction d'une analyse complète.

3.2 Structures fonctionnelles incohérentes, incomplètes ou partielles

Le calcul des structures fonctionnelles échoue si et seulement si aucune structure fonctionnelle cohérente et complète n'est trouvée. Ceci peut arriver parce que les contraintes d'unifi-

$$
\left[\begin{array}{l}
\text{pred = \textbf{'essayer <subj, de-vcomp>'}, v[2..3]} \\
\\
\text{subj} = \left[\begin{array}{l}
\text{pred = \textbf{'Jean <(subj)>'}, np[1..2]} \\
\text{det} = + \\
\text{hum} = + \\
A_{ij} = \{R_9^{182}, R_{26}^{177}, R_{28}^{170}\}
\end{array}\right]_{F68} \\
\\
\text{de-vcomp} = \left[\begin{array}{l}
\text{pred = \textbf{'de <obj|...>'}, prep[3..4]} \\
\text{vcomp} = \left[\begin{array}{l}
\text{subj} = []_{F68} \\
A_{ij} = \{R_{84}^{162}\}_2
\end{array}\right]_{F69} \\
\text{pcase = de} \\
A_{ij} = \{\}_2
\end{array}\right]_{F70} \\
\\
\text{number = sg} \\
\text{person = 3} \\
\text{mode = indicative} \\
\text{tense = present} \\
A_{ij} = \{R_{33}^{130}, R_{48}^{119}, R_{49}^{134}\}
\end{array}\right]
$$

FIG. 4 – Structure fonctionnelle incomplète simplifiée pour la phrase incomplète « Jean essaye de... ». Les identifiants de sous-structures sont indiqués en indice (comme $F70$). Comme indiqué précédemment, une règle peut indiquer au parseur de stocker un identifiant de la production instanciée courante dans le champ spécial A_{ij} de la structure associée au non-terminal de sa partie gauche. Les valeurs atomiques de la forme R_p^q représentent donc des productions instanciées, permettant ainsi de faire le lien entre sous-structures fonctionnelles et non-terminaux de la c-structure.

cation spécifiées par les équations fonctionnelles n'ont pas pu être vérifiées, ou parce que les structures fonctionnelles résultantes sont incohérentes ou incomplètes. Sans entrer dans les détails, l'incohérence est principalement due à ce que des contraintes de sous-catégorisation ont échoué.

La première tentative de calcul des f-structures se fait en respectant toutes les contraintes de cohérence et de complétude, sauf lors du calcul des structures principales (c'est-à-dire associées à l'axiome). Ainsi, le résultat peut être soit un ensemble non vide de structures cohérentes et complètes, soit un ensemble non vide de structures incohérentes ou incomplètes dont toutes les sous-structures sont cohérentes et complètes, soit aucune structure. structure.

Un échec de cette première tentative conduit à une seconde évaluation des f-structures sur la forêt partagée pendant laquelle les vérifications de cohérence et de complétude sont relâchées (un exemple en est donné dans la figure 4). En cas de succès, on obtient des structures principales incohérentes ou incomplètes. Bien sûr, cette seconde tentative peut elle-même échouer. On cherche alors dans la forêt partagée un ensemble de *nœuds maximaux* qui ont des f-structures

(éventuellement incomplètes ou incohérentes) et dont les nœuds pères n'ont pas de f-structure. Ils correspondent à des analyses partielles disjointes. Le processus de désambiguïsation présenté à la section 9.2.4 s'applique à tous les nœuds maximaux.

3.3 Sur-segmentation des phrases inanalysables

En dépit de toutes ces techniques de rattrapage, l'analyse échoue parfois, et aucune analyse n'est produite. Ceci peut arriver parce qu'un délai maximal donné en paramètre a expiré avant la fin du processus, ou parce que l'analyseur Earley a réalisé un rattrapage trivial (en raison d'une couverture insuffisante de la grammaire, ou parce que la phrase d'entrée est tout simplement trop loin d'être correcte : fautes de grammaire, phrases incomplètes, phrases trop bruitées, ...).

Pour cette raison, nous avons développé une sur-couche de SXLFG qui réalise une *sur-segmentation* des phrases inanalysables. L'idée est qu'il arrive fréquemment que des portions de la phrase d'entrée soient analysables comme phrases, bien que la phrase dans son ensemble ne le soit pas. Pour cette raison, nous redécoupons en *segments* les phrases inanalysables (segmentation de niveau 1) ; puis, si nécessaire, nous redécoupons les segments de niveau 1[19] (segmentation de niveau 2), et ainsi de suite avec 5 niveaux de segmentation[20]. Une telle technique suppose naturellement que la grammaire reconnaît aussi les syntagmes voire les chunks (ce qui est linguistiquement justifié, pour reconnaître par exemple les phrases nominales) mais aussi les terminaux isolés (ce qui est moins pertinent linguistiquement). En un sens, c'est une généralisation de l'utilisation d'une grammaire de FRAGMENTS telle que décrite dans Riezler *et al.* (2002); Kaplan *et al.* (2004).

Nos 5 niveaux de segmentation sont les suivants :

niveau 1 : segmentation sur les (quasi-)frontières de phrases : ceci inclut les frontières de phrases détectées par un segmenteur s'il n'a pas été utilisé précédemment (par exemple parce que le texte était déjà segmenté à l'aide d'un segmenteur différent), ou les quasi-frontières de phrases (« ; », identifiants d'éléments dans une liste, ...),

niveau 2 : segmentation sur les ponctuations fortes (tirets isolés, points d'exclamation ou d'interrogation, guillemets, etc.),

niveau 3 : segmentation sur les ponctuations faibles (typiquement « , »),

niveau 4 : segmentation sur les coordinations (telles que « *et* » ou « *ou* »),

niveau 5 : segmentation sur les frontières de mots.

[19] Une phrase peut être segmentée en deux segments de niveau 1, le premier étant analysable. Dans ce cas, seul le second sera sur-segmenté en segments de niveau 2. Et seuls les segments de niveau 2 qui ne sont pas analysables seront sur-segmentés, et ainsi de suite.

[20] Le dernier niveau de segmentation segmente la phrase d'entrée en terminaux isolés, afin de garantir que toute entrée est analysée, et en particulier pour ne pas abandonner l'analyse de phrases pour lesquelles des segments de niveau 1 ou 2 sont analysables, mais qui ont des portions qui ne sont « analysables » qu'au niveau 5.

4 Conclusion

Nous avons présenté SXLFG, générateur d'analyseurs LFG efficaces et robustes. Son utilisation avec différentes grammaires et tous les outils nécessaires autour de l'analyseur pour analyser des corpus réels est présentée aux chapitres 11 et 12. Les systèmes d'analyse ainsi obtenus sont évalués au chapitre 12 à la fois en termes d'efficacité, de robustesse et parfois de précision, montrant les excellentes performances de SXLFG.

Chapitre 10

Analyse non-linéaire : les Méta-RCG

Il existe de nombreux formalismes pour le traitement automatique des langues. Mais nombre d'entre eux reposent sur les grammaires non-contextuelles (comme en LFG, voir le chapitre précédent), ou sur une extension de ces grammaires qui préservent leur propriété de clôture par substitution (voir chapitre 1). Ceci rend nécessaire l'ajout, par dessus ce *squelette syntaxique*, des *décorations* (cf chapitre 1), c'est-à-dire d'un mécanisme supplémentaire, habituellement fondé sur des structures combinées par unification le long de ce squelette syntaxique (il en est ainsi des structures fonctionnelles de LFG). Cependant, il s'avère qu'une telle architecture peut être évitée si l'on utilise un formalisme qui a la propriété de clôture non par substitution mais par intersection[1]. Nous appelons *formalismes non-linéaires* de tels formalismes[2]. De tels formalismes ont un double avantage :

1. ils permettent de représenter de façon purement syntaxique des phénomènes effectivement *syntaxiques*, qui, quoique parfois simples, ne peuvent pas être représentés correctement (quand ils peuvent l'être) avec des formalismes dont les structures syntaxiques sont des arbres,
2. ils permettent de faire interagir des contraintes de différentes natures sur un pied d'égalité, en s'affranchissant d'une architecture à deux niveaux et de l'unification de structures dont le nombre de traits serait déraisonnable ; c'est le cas par exemple pour les contraintes de sémantique lexicale (restriction de sélection), dont la prise en compte pendant l'analyse a motivé les travaux évoqués dans ce chapitre.

[1]Rappelons le résultat de Ginsburg (1975) qui montre qu'un formalisme qui étend les grammaires non-contextuelles tout en ayant une expressivité strictement inférieure à celle des langages de type 0 (ce qui est évidemment souhaitable) ne peut pas être clos à la fois par substitution et par intersection.

[2]Nous avons conscience de la confusion que cette terminologie peut induire au regard de l'existence de formalismes qui permettent d'effectuer l'analyse en temps *linéaire* en la longueur de la phrase d'entrée. Toutefois, de tels formalismes sont si simples que nous pensons qu'il ne saurait y avoir de réelle ambiguïté dans ce chapitre. En tout état de cause, *non-linéaire* désigne dans tous ce chapitre un formalisme clos par intersection, indépendamment de la complexité en temps de ses algorithmes d'analyse.

L'objectif de ce chapitre est de présenter un tel formalisme, que nous avons défini et mis en œuvre à travers :

- une grammaire du français (et un environnement de développement de grammaires),
- un analyseur,
- des outils de visualisation des analyses sous formes de *vues partielles* qui reprennent les points de vue traditionnels tels que les constituants, les dépendances, la sémantique prédicative, ou même les boîtes topologiques.

Ce chapitre est construit comme suit. Tout d'abord, nous montrerons quels sont les inconvénients des formalismes à décorations, motivant ainsi notre propos (Sagot et Boullier, 2004). Puis nous montrerons au contraire combien l'introduction explicite de la notion de non-linéarité dans les modèles formels des langues peut permettre une modélisation satisfaisante de la syntaxe et de son interaction avec la sémantique lexicale. Nous présenterons alors le formalisme non-linéaire que nous avons développé, appelé Méta-RCG (Sagot, 2005c,b). Puis nous illustrerons les intérêts de ce formalisme grâce à la mise en œuvre que nous en avons faite à travers une grammaire Méta-RCG du français et un environnement d'analyse.

1 Points faibles des formalismes à décorations

Nous avons présenté au chapitre précédent un analyseur LFG, c'est-à-dire un analyseur qui repose sur un formalisme à décorations, et dont nous confirmerons au chapitre 12 les très bonnes performances. Si nous allons présenter ce que nous considérons comme les inconvénients de tels formalismes, ce n'est donc pas pour en conclure qu'il ne faut pas les utiliser, mais que cela vaut le coup d'étudier l'autre possibilité, celle des formalismes de réécriture clos par intersection. Et ce d'autant plus qu'ils n'ont pas été souvent mis en œuvre à grande échelle pour la formalisation et l'analyse de langues naturelles[3].

Comme dit précédemment, la plupart des formalismes linguistiques déployés à grande échelle font usage de formalismes à décorations qui reposent sur des formalismes squelette, appelés *squelettes syntaxiques*, qui sont *clos par substitution*. Ce formalisme squelette constitue le premier niveau du formalisme complet. Il s'agit le plus souvent de grammaires non-contextuelles ou de TAG. L'expressivité limitée de ces squelettes rend inévitable le second niveau, formé de décorations calculées sur le squelette par des opérations d'unification. Ces architectures à deux

[3]Cet état de fait est peut être en partie dû à l'importance historique de la notion de Famille Abstraite de Langages (Abstract Family of Languages), au faible nombre de sous-familles strictes des langages contextuels qui soient clos par intersection tout en étendant les langages non-contextuels, et à un certain nombre de difficultés qui surviennent lorsque l'on veut analyser non pas des chaînes mais des treillis de mots. Nous proposons plus bas une réponse partielle à ce dernier point.

niveaux induisent donc une séparation formelle stricte entre *structures syntaxiques*, produites par le squelette, et *attributs syntaxiques*, représentés dans les décorations.

1.1 Arguments computationnels

D'un point de vue computationnel, cette séparation n'est pas entièrement satisfaisante. De fait, la plupart des formalismes à décorations ont un squelette syntaxique de complexité assez faible, mais ne spécifient pas très clairement l'expressivité de leurs décorations. Par exemple, LFG et HPSG sont des formalismes à décorations dont le formalisme squelette est celui des grammaires non-contextuelles[4]. Mais leurs décorations leur donnent un pouvoir d'expression très important : le formalisme LFG est NP-complet, et le formalisme HPSG n'est même pas décidable (dans le cas général).

On peut imaginer deux grandes familles d'algorithmes pour les analyseurs de tels formalismes :

- on peut générer d'abord toutes les analyses squelette, et ensuite calculer les décorations sur ces analyses, ce qui permet de faire chuter l'ambiguïté de l'analyse grâce à l'analyse par le squelette (bien plus efficace que l'analyse complète), qui peut donc être vu comme un guide (c'est ce qui est fait dans SXLFG),
- on peut calculer en une seule passe à la fois les structures squelette et les décorations, ce qui permet d'élaguer les mauvaises sous-analyses à la volée.

Les deux options sont envisageables, et il n'y a pas réellement d'argument décisif en faveur de l'une ou l'autre option. Comme nous l'avons vu au chapitre précédent, SXLFG fait le choix de calculer d'abord la forêt partagée (le squelette), puis de calculer les décorations dans une seconde phase. Dans le cas d'un squelette non-contextuel, comme dans SXLFG, on peut effectivement tirer parti de la complexité relativement faible de l'analyse par le squelette. Mais le nombre d'analyses induites par le squelette peut être colossal (nous verrons au chapitre 12 qu'il nous est arrivé, pour une phrase de notre corpus de test, de construire au cours de la première phase d'une analyse par SXLFG une forêt représentant plus de 3.10^{73} arbres). À l'inverse, les algorithmes en une passe permette de tirer parti de l'élimination dynamique de sous-structures squelettes qui sont rejetées par les décorations. Cependant, on perd un temps important à calculer des décorations sur des sous-structures qui sont plus tard rejetées par le squelette lui-même. Ceci peut être très coûteux étant donnée la complexité souvent plus élevée des calculs de décorations par rapport à l'analyse par le squelette. Ces difficultés de gestion de l'ambiguïté sont particulièrement flagrants dès lors qu'une phrase comporte un certain nombre de mots inconnus. Tous ces problèmes, ajoutés aux problèmes algorithmiques et pratiques qui apparaissent

[4]Même si le cas de HPSG est moins explicite que celui de LFG.

dès que l'on met en jeu l'unification, compliquent le développement d'analyseurs efficaces, et rendent presque nécessaire l'introduction de techniques heuristiques ou probabilistes de filtrage en cours d'analyse (voir le chapitre précédent), ce qui n'est pas toujours très satisfaisant ni d'un point de vue pratique ni d'un point de vue théorique.

Ces problèmes d'ambiguïtés croisées ont une traduction directe sur les problèmes de rattrapage d'erreurs : il est très délicat de développer pour des formalismes à décorations (formalismes à deux niveaux) des analyseurs munis de techniques satisfaisantes de rattrapage d'erreurs[5], pour la raison suivante. Si une phrase d'entrée n'est pas reconnue par la grammaire, le rôle d'un mécanisme de rattrapage d'erreur tels que ceux présentés pour SXLFG est de permettre la construction d'une analyse de la phrase qui viole le plus petit nombre possible de contraintes grammaticales. Si certaines de ces contraintes sont représentées dans le squelette, alors que d'autres le sont dans les décorations, il est très difficile de faire interagir ces deux niveaux et de leur faire calculer de concert l'analyse « la moins fausse ». De fait, SXLFG n'implémente pas une telle technique : les mécanismes de rattrapage d'erreurs de l'analyseur non-contextuel sous-jacent sont totalement décorrélés des mécanismes de tolérance et de rattrapage d'erreurs qui sont au niveau du calcul des décorations (c'est-à-dire ici des équations fonctionnelles). On pourrait cependant vouloir, en cas d'échec du calcul des décorations, que le squelette non-contextuel propose des analyses « légèrement fausses » pour essayer de voir si, sur ces analyses, le calcul des décorations ne fonctionne pas mieux. Dans un formalisme sans décorations, des algorithmes à la fois plus simples, plus efficaces et plus intuitifs peuvent être implémentés.

C'est une des motivations de l'utilisation des TAG que de surmonter ces difficultés, en prenant en compte plus d'informations (et donc plus de complexité) au niveau du squelette que ne le font les squelettes non-contextuels, ce qui permet de ne manipuler que des décorations relativement simples (au moins dans les TAG à décorations, ou *Feature-based TAGs*). Mais les TAG n'ont pas non plus l'expressivité suffisante pour modéliser correctement la syntaxe des langues. L'introduction des TAG multi-composants (*Multi-Component TAGs*, ou *MC-TAG*), et d'un certain nombre de variantes, n'ont pas permis de conserver la simplicité qui fait l'élégance des TAG, n'ont toujours pas la complexité requise pour modéliser tous les phénomènes syntaxiques, et n'ont pas conduit à des systèmes d'analyses efficaces à grande échelle (ce qui suppose entre autres des analyseurs efficaces et des grammaires à grande couverture).

Les formalismes pour la linguistique sont donc ou trop peu expressifs pour pouvoir se passer de décorations reposant sur l'unification, ou trop expressifs et donc trop complexes algorithmiquement, comme Prolog ou les Grammaires de Clauses Définies (*Definite Clause Grammars*, *DCG*).

[5] C'est le cas aussi des techniques d'apprentissage automatique de grammaires ou de paramètres de grammaires, mais ce domaine dépasse le cadre de nos travaux.

1.2 Arguments linguistiques

De façon peut-être plus importante, la séparation entre structures syntaxiques et attributs syntaxiques n'est pas réellement justifiée d'un point de vue linguistique.

Tout d'abord, la position exacte de cette séparation varie d'un formalisme à l'autre et d'une langue à l'autre. Par exemple, une adjonction englobante en TAG sera vue en LFG comme deux non-terminaux indépendants de la grammaire non-contextuelle sous-jacente, reliés l'un à l'autre au niveau fonctionnel, c'est-à-dire au niveau des attributs. Et un même phénomène linguistique peut avoir des réalisations suffisamment différentes dans deux langues données pour qu'il soit modélisé dans une grammaire de l'une au niveau structurel (squelette) et dans une grammaire de l'autre au niveau des attributs (décorations) : par exemple, les groupes génitifs sont des syntagmes introduits par la préposition *de* en français, alors que ce sont des syntagmes nominaux dont le génitif est marqué par le cas dans la plupart des langues slaves.

De plus, il existe un certain nombre de phénomènes qui sont à l'évidence purement syntaxiques mais dont la modélisation nécessite une expressivité importante. C'est ce qui a conduit à la définition de la notion de formalisme *faiblement sensible au contexte* (*Mildly Context-Sensitive*, ou *MCS*) par Joshi (1985), et dont les TAG sont un exemple, afin d'essayer de trouver un niveau de complexité intermédiaire entre les grammaires non-contextuelles et les grammaires sensibles au contexte. Rappelons leur définition (déjà donnée au chapitre 1). Un formalisme est dit faiblement sensible au contexte s'il vérifie les 4 conditions suivantes :

MCS1. Il étend strictement les grammaires non-contextuelles,

MCS2. L'analyse est de complexité polynomiale,

MCS3. Son expressivité lui permet de décrire des langages modélisant des versions faibles de dépendance à longue distance et d'accord multiple (typiquement les langages $\mathcal{M}_4 = \{a^n b^n c^n\}$, $\mathcal{S}_2 = \{a^n b^m c^n d^m\}$ et *2-copie*, c'est-à-dire $\mathcal{C}_2 = \{ww \mid w \in \{a, b\}^*\}$.)

MCS4. Il satisfait la propriété de Croissance Constante (*Constant Growth Property*, *CGP*)[6].

MCS1 semble naturel, MCS2 est nécessaire pour garantir une certaine efficacité computationnelle, et MCS3 est justifié par un certain nombre de phénomènes linguistiques bien attestés. Cependant, le statut de MCS4 est moins clair, et aucune preuve convaincante de cette hypothèse n'est disponible dans la littérature, comme remarqué par exemple par Groenink (1996), Sagot et Boullier (2004) ou Stabler (2003). Par ailleurs, un certain nombre de phénomènes attestés semblent bien être au-delà de l'expressivité des formalismes MCS, comme nous allons le voir. Puisque tous les squelettes syntaxiques utilisés actuellement sont MCS, il semble que

[6]De façon informelle, un langage vérifie la propriété de Croissance Constante si ses éléments (ses phrases) respectent la condition suivante. Si on classe ses phrases par longueur croissante, la différence entre les longueurs de deux phrases consécutives est bornée par une constante. Un formalisme vérifie cette propriété si les langages qu'il permet de définir la vérifient.

la seule solution soit ou bien d'analyser ces phénomènes purement syntaxiques en utilisant de façon inélégante des attributs *ad hoc*, ou bien d'utiliser des squelettes plus complexes. Cette dernière solution semble raisonnable, mais elle n'est utilisée aujourd'hui que par des formalismes reposant sur Prolog (comme les DCG) ou les Grammaires de Propriétés, dont la complexité est exponentielle[7].

1.2.1 Nombres chinois et génitifs en géorgien ancien

Il existe dans certaines langues des constructions sporadiques qui ne sont pas analysables par des grammaires MCS car elles ne respectent pas la CGP (condition MCS4). Classiquement, il en est ainsi des nombres chinois Radzinski (1991) et du génitif en géorgien ancien[8] Michaelis J. (1996). En revanche, les Grammaires Non-Contextuelles Parallèles Multiples (PMCFG) introduites par Kaji *et al.* (1992) couvrent ces deux cas tout en étant strictement moins puissantes que les formalismes couvrant tout *PTIME* Groenink (1996).

Naturellement, on pourrait considérer ces cas comme des cas marginaux dont il n'est pas nécessairement souhaitable de pouvoir rendre compte. En particulier, dans le cas des nombres chinois, il peut être objecté (Radzinski, 1991) que les contraintes qui font que la CGP n'est pas respectée pourraient ne pas être de nature linguistique, mais être issues des propriétés mathématiques des objets dénotés. Par ailleurs, il serait satisfaisant de disposer d'une grammaire pour le géorgien ancien permettant un nombre arbitraire de génitifs empilés, mais il n'est pas toujours considéré que cela soit indispensable (Kallmeyer, 1997). Cependant, au moins deux autres phénomènes, bien moins sporadiques, mettent en cause l'idée selon laquelle les langages MCS suffisent au traitement des langues[9]. Il s'agit des exemples bien connus que sont le « scrambling » et les constructions coordonnées à verbes multiples.

---

[7] Ce qui ne veut pas dire, loin de là, que nous ne sommes pas séduits par ces approches, et en particulier par les Grammaires de Propriétés, qui combinent pertinence linguistique, robustesse et une assez bonne efficacité. Mais nous faisons dans ce chapitre l'hypothèse que l'on peut modéliser la langue par un formalisme de complexité au plus polynomiale. Notre propos vise donc à explorer les pistes susceptibles de valider ou d'invalider cette hypothèse.

[8] Les auteurs cités montrent que le génitif en géorgien ancien n'est pas semi-linéaire. En réalité le génitif en géorgien ancien, pouvant être réduit à $\{a_1 b\, a_2 g b\ \ldots a_k g^{k-1} b\}$, ne respecte même pas la CGP, ce qui est plus fort.

[9] Nous laissons de côté, par manque d'arguments satisfaisants, une certaine classe de questions en chinois mandarin, dénotées en anglais par le terme *A-not-A questions*. Ces constructions semblent utiliser un mécanisme de copie d'un syntagme verbal complet. Le terme de *A-not-A questions* vient de ce qu'un élément négatif sépare les deux copies de *A*, lesquelles doivent absolument être identiques. Stabler (2003) montre que ce phénomène, comme de nombreux autres dans de nombreuses langues qui procèdent du même schéma, a la double propriété de mettre en œuvre un mécanisme complexe (la copie) dans un but simple, puisque la contribution sémantique de la portion *not-A* est atomique : elle transforme l'affirmation *A* en question. La question de savoir si ce mécanisme est MCS est ouverte. Stabler (2003) conclut de la sorte : *although the literature seems to lack any MCS proposals for A-not-A […], the prospects for such proposals look reasonable*.

1.2.2 Scrambling en allemand et coordonnées à verbes multiples en néerlandais

Le scrambling en allemand[10] peut être décrit de la façon suivante : dans une subordonnée comprenant des complétives, les compléments des différents verbes sont placés entre le sujet du verbe principal et les verbes. Les verbes sont alors généralement ordonnés du plus enchâssé au verbe principal, mais les compléments sont dans un ordre quelconque (on trouvera des exemples dans Becker *et al.* (1991)). Il est montré par Boullier (1999) que le langage utilisé par Becker *et al.* (1992) pour modéliser le scrambling est insuffisant, et utilise un langage plus vaste qu'il appelle SCR[11] Ce langage est au-delà de la puissance des TAG Becker *et al.* (1991) et même de celle des LCFRS Becker *et al.* (1992). Il semble pourtant vérifier la CGP. Cependant, comme indiqué plus haut, aucun formalisme grammatical n'a été proposé jusqu'à présent qui soit simultanément MCS tout en ayant une puissance strictement plus grande que les LCFRS. Il semble donc raisonnable de chercher un formalisme grammatical qui ne soit pas MCS pour décrire le scrambling en allemand[12].

Les constructions coordonnées à verbes multiples en néerlandais sont analysées par Groenink (1996) : il exhibe une classe d'exemples C_1 non exprimables dans le cadre des LCFRS[13], qui regroupe des exemples ayant un nombre de verbes arbitraire, ce qui nécessite une argumentation sur la pertinence de cet argument (cf. ci-dessous). Il a été prouvé par la suite qu'elle n'était pas même semi-linéaire Michaelis J. (1996). Cependant, bien qu'une PMCFG puisse analyser cette classe d'exemples, Groenink (1996) montre qu'on peut l'étendre à une classe plus large C_2, encore moins acceptable linguistiquement que C_1, que même les PMCFG, qui subsument strictement les LCFRS et ne respectent pas la CGP, ne peuvent décrire[14].

1.2.3 La puissance nécessaire pour décrire le langage naturel est *PTIME*

Même en rejetant la classe C_2 d'exemples du néerlandais, les exemples ci-dessus semblent indiquer que les formalismes MCS ne suffisent pas à décrire le langage naturel[15]. Toutefois,

[10]Ce phénomène existe dans d'autres langues, comme le japonais.

[11]Il s'agit approximativement du langage suivant : soit un alphabet de terminaux partitionné en $\{n_1, \ldots, n_l\}$ et $\{v_1, \ldots, v_m\}$ et muni d'une correspondance h indiquant, si $v = h(n)$, que n est un argument de v. Une suite de terminaux est reconnue par SCR si et seulement si elle est de la forme $\pi(n_1, \ldots, n_p)v_1 \ldots v_q$ où π est une permutation, les n_i et les v_j sont deux à deux distincts, et pour tout n_i il existe un et un seul v_j tel que $h(n_i) = v_j$.

[12]On peut noter dès ici que le fait qu'un tel formalisme ne serait peut-être pas de la puissance minimale pour exprimer SCR n'est pas nécessairement un problème. En effet, si l'on peut exhiber des grammaires analysables sans véritable surcoût par rapport aux grammaires MCS mais permettant d'exprimer simplement et élégamment le scrambling, rien ne justifie dans la pratique la quête de la plus petite grammaire possible incluant SCR.

[13]Il s'agit de son fragment (1.14) répété en (3.39).

[14]Il s'agit du cas où l'on ne se restreint pas à un seul verbe à montée à l'infinitif parmi les verbes multiples coordonnés (cf. paragraphe 3.3 de Groenink (1996)).

[15]Notons à cet égard que si l'on accepte la classe d'exemples C_2, l'insuffisance des PMCFG à les traiter ne prouve rien de plus : on pourrait imaginer trouver un formalisme grammatical agréable subsumant les grammaires MCS mais non comparable avec les PMCFG.

les trois premières conditions MCS semblent devoir être vérifiées par les langues naturelles : MCS1 a été justifiée plus haut, MCS3 a été explicitement introduite pour que les grammaires MCS puissent traiter divers phénomènes linguistiques courants, et MCS2 est justifiée au moins par les besoins pratiques de complexité. Seule MCS4 peut donc être violée : la modélisation du langage naturel passe par des grammaires ne vérifiant pas la CGP.

Toutefois, la problématique de la distinction compétence/performance ne rend pas la chose si simple, par exemple pour la classe d'exemples C_1 du néerlandais. De fait, Manaster-Ramer (1987) indique que le résultat dépend du point de vue que l'on adopte sur ce que l'on inclut dans la langue et ce que l'on en exclut. Toutefois, il nous semble artificiel de vouloir chercher le formalisme minimal permettant de décrire une langue donnée, en allant jusqu'à s'autoriser à borner *ex abrupto* certains paramètres sans justification linguistique claire, surtout s'il existe des formalismes permettant de s'affranchir de telles limitations sans pour autant entraîner un surcoût significatif.

La problématique compétence/performance est encore plus forte concernant l'insuffisance des PMCFG pour décrire la classe C_2. En réalité, on peut penser qu'une telle classe d'exemples, peu convaincante en termes de performance mais acceptable (quoiqu'à la marge) en termes de compétence, est plus ou moins du niveau maximal de complexité de la langue concernée, lequel est disponible mais presque jamais utilisé dans une grammaire de la langue. Ainsi, un formalisme grammatical permettant de traiter le néerlandais devrait avoir la puissance nécessaire au traitement de C_2. Mais il devrait pour ce faire exprimer sa pleine puissance, utilisée très rarement dans une grammaire complète du néerlandais. En poursuivant ce raisonnement, et puisque les PMCFG ne suffisent pas, on ne peut que conclure que le niveau de complexité requis est le seul autre niveau immédiatement supérieur à celui des grammaires MCS pour lequel il existe des formalismes grammaticaux aux bonnes propriétés, à savoir tout *PTIME* (cf. partie 2).

Par ailleurs, nous n'avons parlé jusqu'à présent que de puissance générative faible. Or si l'équivalence faible entre le néerlandais et un langage analysable avec une PMCFG n'est pas certaine, l'« équivalence forte » entre structures linguistiques et analyse par une PMCFG, que l'on souhaite évidemment, est encore plus douteuse. Cela renforce l'intuition selon laquelle les grammaires adéquates à l'analyse des langues ne respectent pas la CGP, ne sont pas même exprimables par une PMCFG, et nécessitent un formalisme grammatical couvrant tout *PTIME*.

2 La non-linéarité

Les arguments que nous venons de développer conduisent donc à ceci : les formalismes à décorations ont un squelette syntaxique de complexité insuffisante, ce qui induit la nécessité d'utiliser des décorations. Et la complexité nécessaire est celle que l'on obtient en étendant les

grammaires non-contextuelles en considérant leur clôture par intersection. C'est bien grâce à l'abandon de la clôture par substitution au profit de la clôture par intersection que l'on est en mesure de modéliser convenablement la syntaxe. C'est ce que nous appelons la *non-linéarité* de la syntaxe.

Mais ceci n'est qu'un cas particulier d'un phénomène plus large : très souvent, plusieurs contraintes linguistiques concernant une même portion[16] de phrase doivent être exprimées. Dans les formalismes clos par substitution, ceci ne peut être réalisé, sauf à sélectionner une seule famille de contraintes qui sont implémentées dans le squelette, les autres étant traitées dans des décorations. La famille de contraintes concernée peut être par exemple les contraintes de constituances (par exemple en LFG ou en TAG) ou les contraintes de dépendances (comme dans les Grammaires de Dépendances), créant au passage un clivage entre points de vue concurrents sur la modélisation de la syntaxe, concurrence par conséquent induite tout à fait artificiellement.

À l'inverse, la propriété de clôture par intersection permet la formalisation directe de multiples contraintes concernant la même portion de la chaîne d'entrée, ou même de portions qui se superposent seulement partiellement. C'est la raison pour laquelle nous disons que de tels formalismes, ainsi que leurs grammaires, sont *non-linéaires*. Naturellement, un formalisme de réécriture (sans décorations) ne peut qu'être non-linéaire, puisque par définition il ne peut pas déléguer à des décorations le traitement de certaines contraintes. Nous allons montrer dans la section suivante qu'il est effectivement possible de définir un formalisme non-linéaire sans décorations pour modéliser de façon satisfaisante les langues et permettre leur analyse efficace (en temps polynomial en la longueur de la phrase d'entrée). Le reste de cette section montrera quels sont les bénéfices computationnels et linguistiques de la non-linéarité, et quels sont les propriétés que devrait avoir un formalisme non-linéaire pour la linguistique.

2.1 Expressivité et complexité des formalismes non-linéaires

Comme suggéré précédemment, nous faisons l'hypothèse, tout au long de ce chapitre, qu'il est possible d'analyser les langues avec une complexité polynomiale. Les deux formalismes les plus étudiés qui étendent les grammaires non-contextuelles[17] sont les *simple Literal Movement Grammars* (*sLMG*, Groenink (1995, 1996)) et les Grammaires à Concaténation d'Intervalles (RCG, Boullier (2003a, 2004)). Ces deux formalismes couvrent tout *PTIME*, l'ensemble des langages reconnaissables en temps polynomial[18]. Nous avons présenté les RCG, de façon in-

[16] Nous utilisons à dessin le mot *portion*, qui est imprécis, car nous voulons garder une terminologie agnostique au regard de la distinction entre formalismes traitant de *sous-chaînes* de la chaîne d'entrée (comme les LMG, voir plus bas) et ceux traitant d'*intervalles* (comme les RCG).

[17] Nous excluons donc les langages réguliers, bien qu'ils soient clos par intersection.

[18] De façon plus précise, *PTIME* est l'ensemble des langages reconnaissables en un temps polynomial en la longueur de la chaîne d'entrée par une machine de Turing finie et déterministe.

formelle puis formelle, au chapitre 1. Les grammaires sLMG sont également des ensembles de clauses à la Horn concernant des prédicats sur des portions de la chaîne d'entrée. Mais contrairement à ce qui se passe en RCG, où les portions en question sont les intervalles, les portions considérées par les sLMG sont les sous-chaînes de la chaîne d'entrée. Par ailleurs, il est possible de convertir une sLMG en une RCG, et c'est une opération simple, ou une RCG en une sLMG, bien que cette opération soit plus délicate, en particulier si la RCG est fortement non-combinatoire. Ces conversions conduisent à des graphes d'analyse comparables dans le formalisme source et le formalisme cible.

Naturellement, c'est la non-linéarité qui apporte une expressivité supérieure. Grâce à cette non-linéarité, des grammaires très simples peuvent définir des langages qui ne sont même pas MCS. Par exemple, et comme nous l'avons vu au chapitre 1, le langage $\mathcal{L}_{exp} = \{a^{2^p}\}$ est reconnu par la grammaire suivante (qui est interprétable à la fois comme une RCG et comme une sLMG[19]) :

$$\begin{array}{lcl} \text{S}(X\ Y) & \rightarrow & \text{EQLEN}(X,\ Y)\ \text{S}(X) \\ \text{S}(a) & \rightarrow & \varepsilon \\ \text{EQLEN}(a\ X,\ a\ Y) & \rightarrow & \text{EQLEN}(X,\ Y) \\ \text{EQLEN}(\varepsilon,\ \varepsilon) & \rightarrow & \varepsilon \end{array}$$

Répétons ici ce que nous avons déjà dit au chapitre 1 sur l'analyseur induit par cette grammaire. On peut remplacer les deux clauses définissant le prédicat EQLEN par un mécanisme en temps constant. Si l'on a préalablement vérifié, comme c'est supposé par Boullier (2003a), que la chaîne d'entrée n'est constituée que de symboles a (c'est le travail du *lexeur*), alors l'analyse se fait en temps logarithmique (donc sub-linéaire) en la longueur de la phrase. Pourtant, ce langage ne respecte même pas la CGP (Propriété de Croissance Constante).

La puissance d'expression des RCG et des sLMG permet d'exprimer tous les phénomènes linguistiques dont nous avons dit précédemment qu'ils sont au-delà des langages MCS, comme cela a été montré par exemple par Groenink (1996) ou Boullier (2003a).

Il est important de remarquer que cette expressivité plus grande n'augmente pas la complexité d'analyse de langages qui appartiennent à des classes de complexité inférieures. Par exemple, les grammaires non-contextuelles peuvent être trivialement converties en sLMG ou en RCG fortement équivalentes, et analysées dans ces formalismes en $O(n^3)$. Il a également été montré par Boullier (2000) qu'une TAG peut toujours être convertie en une RCG s'analysant avec la même complexité (qui est $O(n^6)$ ou moins, selon la grammaire[20]). De plus, Boullier

[19] Dans Groenink (1995), le symbole →est utilisé comme dans nos exemples, et comme en RCG. Mais le symbole `:-` peut également être trouvé pour les (s)LMG, comme par exemple dans Groenink (1996)

[20] Si par exemple la grammaire ne comporte pas (de schémas) d'arbres englobants, la grammaire est une *TIG* (*Tree Insertion Grammar*), comme indiqué au chapitre 1, analysable en $O(n^3)$.

(2003a) montre que l'intersection de deux grammaires non-contextuelles peut être trivialement représentée par une RCG non-linéaire, et analysée en $O(n^3)$, c'est-à-dire sans augmentation de la complexité par rapport aux grammaires non-contextuelles. Enfin, l'exemple ci-dessus montre même qu'il est possible d'analyser en temps sub-linéaire des langages qui ne respectent même pas la Propriété de Croissance Constante.

2.2 Modélisation des langues et non-linéarité

2.2.1 La syntaxe des langues est non-linéaire

Comme dit précédemment, les formalismes linéaires pour la modélisation des langues font nécessairement usage de décorations, car de nombreuses contraintes doivent être exprimées qui concernent les mêmes portions de la chaîne d'entrée. C'est donc que les langues sont par essence non-linéaires. Un exemple très simple en est par exemple le phénomène des verbes à contrôle du sujet, comme dans la phrase suivante :

(RCG_1) Marie veut manger

L'existence d'une relation syntaxique entre *veut* et *Marie* est claire, tout comme entre *veut* et *manger*. L'existence d'une relation syntaxique entre *Marie* et *manger* n'est pas moins évidente, et s'exprime en disant que *veut* est un verbe à contrôle sujet. On peut s'en convaincre par exemple (et entre autres nombreux indices) par le fait que le sujet de la phrase *Marie* est assujetti à des contraintes de restrictions de sélection provenant de *manger* (c'est ainsi que **Marie veut pleuvoir* est agrammatical)[21]. Ce phénomène très simple et très courant est un exemple typique de non-linéarité élémentaire, qui nécessite un traitement relativement complexe dans les formalismes reposant sur un squelette linéaire. À l'inverse, la grammaire-jouet suivante construit une structure syntaxique raisonnable – quoiqu'excessivement simplifiée – pour *Marie veut manger* (comme la grammaire précédente, celle-ci peut s'interpréter à la fois comme une sLMG ou comme une RCG) :

PHRASE(*S V Vi*)	$\rightarrow$	VERBE(*V*) SUJ(*S*, *V*) OBJ(*Vi*, *V*, *S*)
OBJ(*Vi*, *V*, *S*)	$\rightarrow$	V_CTRL_SUJ(*V*) INF(*Vi*) SUJ(*S*, *Vi*)
SUJ(*Marie*, *veut*)	$\rightarrow$	ε
SUJ(*Marie*, *manger*)	$\rightarrow$	ε
VERBE(*veut*)	$\rightarrow$	ε
V_CTRL_SUJ(*veut*)	$\rightarrow$	ε
INF(*manger*)	$\rightarrow$	ε

[21] C'est encore plus clair dans des langues qui mettent en œuvre des procédés de montée des clitiques, comme en allemand.

On peut paraphraser cette grammaire comme suit. Une phrase, c'est-à-dire une chaîne qui satisfait l'axiome PHRASE, est la concaténation d'un sujet S, d'un verbe fini V et d'un infinitif Vi, qui satisfait les faits linguistiques suivants (clause 1) : V est un verbe fini (prédicat VERBE), S est le sujet de V (prédicat SUJ) et Vi est l'objet de V (prédicat OBJ, auquel on passe également le sujet S). Les deux premiers de ces faits sont garantis respectivement par les clauses 5 et 3, qui disent que *veut* est un verbe fini et que *Marie* peut être le sujet de *veut*. Le dernier des trois faits vient de la seconde clause, qui demande à V d'être un verbe à contrôle sujet (prédicat V_CTRL_SUJ, qui demande à Vi d'être un verbe à l'infinitif (prédicat INF), et qui demande à S, c'est-à-dire au sujet de V, d'être *également* le sujet de Vi. Ces trois faits sont eux-mêmes directement exprimés par les clauses 6, 7 et 4 respectivement.

Un exemple comme celui-ci donne un aperçu du fait que l'utilisation de la non-linéarité est naturelle lorsqu'on décrit les langues, même à un niveau purement syntaxique. Un autre exemple très intéressant, que nous ne développerons pas ici, est la coordination[22]. Des articles tels que ceux de Groenink (1996) ou de Boullier (2003a) donnent des exemples d'utilisation de la non-linéarité pour décrire de façon linguistiquement adéquate des phénomènes syntaxiques complexes. Toutes ces considérations montrent que les formalismes non-linéaires polynomiaux (comme les RCG et les sLMG) permettent la construction de structures syntaxiques qui sont des *graphes* et pas seulement des arbres, ce qui semble tout à fait naturel, et ce sans décorations reposant sur l'unification.

2.2.2 Utiliser la non-linéarité pour faire interagir différents types de contraintes linguistiques

Mais il serait dommage d'en rester là. La non-linéarité permettant d'exprimer différents prédicats concernant les mêmes portions de la phrase, il est naturel d'essayer de voir comment la mettre en œuvre pour ajouter des contraintes linguistiques supplémentaires par rapport à ce qui est fait en général dans les grammaires. Dans l'exemple ci-dessus, le prédicat SUJ(*Marie*, *manger*) était directement asserté dans la grammaire. Mais dans une grammaire plus réaliste, on procéderait d'une autre façon. Si *Marie* est un sujet raisonnable pour *manger*, c'est parce que *Marie* est un syntagme nominal (SN) correct (peu importe ici pourquoi exactement), parce que *manger* est un verbe correctement accordé à ce syntagme nominal, et parce que *Marie* a des propriétés sémantiques qui en font un agent raisonnable du verbe à l'actif *manger*. Cette relation sémantique, qui peut être modélisée par un prédicat à deux arguments, est ce que l'on appelle une restriction de sélection (si on la voit comme une contrainte) ou un élément de

[22] Nous avons pourtant travaillé sur ce sujet, mais dans un cadre différent, celui de l'extension du formalisme des TAG à décorations pour gérer les coordinations elliptiques. On se reportera par exemple à Seddah et Sagot (2006).

sémantique prédicative (si on la voit comme un produit du processus d'analyse ou comme une représentation partielle simple du sens de la phrase).

La non-linéarité permet de modéliser une telle caractérisation d'un sujet valide de manière très simple, par exemple comme suit[23] :

$$\text{SUJ}(S, V) \quad \rightarrow \quad \text{ACCORD}(S, V)\ \text{SN}(S)\ \text{AGENT}(S, V)$$

2.2.3 Concilier non-linéarité et ambiguïtés locales

Telle quelle, cette caractérisation est cependant trop simpliste, pour la raison suivante. Le prédicat *SN(S)* pourrait être vrai, mais de façon ambiguë, c'est-à-dire en étant la racine de plusieurs sous-analyses. Dans ce cas, différentes analyse de *S* par *SN* pourrait correspondre à différents syntagmes nominaux ayant différentes propriétés, dont par exemple celle d'être ou de ne pas être un agent valide de *V*. C'est le problème de l'interaction entre non-linéarité et ambiguïtés locale, conséquence directe de la non-clôture par substitution. Pour cette raison, on pourrait vouloir passer en premier argument au prédicat *AGENT* non pas *S* mais l'*analyse* de *S* par *SN*. Cependant, ceci n'est pas possible car cela irait au-delà de l'expressivité des formalismes polynomiaux, puisqu'il peut y avoir un nombre exponentiel de façons différentes pour *SN(S)* d'être vrai.

Cependant, il est possible de passer en premier argument de *AGENT* quelque chose qui est strictement entre *S*, trop peu informatif, et l'analyse de *S* par *NP*, trop informative. La quantité d'informations que l'on a le droit de passer doit être polynomiale en la longueur de la phrase avec un exposant borné (par une constante)[24]. Une possibilité est de définir des *signatures* des sous-analyses de *NP(S)* linguistiquement justifiées, comme par exemple la ou les têtes[25] (au sens linguistique) et les traits morphosyntaxiques associés. En effet, c'est une hypothèse raisonnable que de considérer que les propriétés sémantiques de *S* peuvent être vérifiées sur ses têtes.

Cette discussion est en réalité très générale. C'est une conséquence directe de la non-linéarité (c'est-à-dire de la clôture par intersection), et peut être énoncé comme suit : dans un formalisme non-linéaire polynomial, il n'y a aucun moyen d'accéder à une sous-analyse complète particulière calculée ailleurs, parce que cela dépasserait *PTIME*. Ceci a d'ailleurs une autre conséquence : si l'on veut remplacer la chaîne d'entrée (c'est-à-dire une liste de symboles) par un DAG de symboles, il n'y aura, dans le cas général, aucun moyen de garantir qu'un

[23] Cet exemple n'est pas censé montrer comment on procède dans une vraie grammaire du français. Le but est ici d'illustrer le fait que la non-linéarité permet de faire interagir ensemble des contraintes de natures différentes. En particulier, nous avons caché dans un prédicat ACCORD toutes les contraintes d'accord, et nous ne vérifions pas que le verbe est à l'actif, qu'il n'est pas impersonnel, etc. . .

[24] Pour être plus précis, cette information doit pouvoir être représentée par un nombre connu à l'avance (et donc borné) d'intervalles de la chaîne d'entrée, passés sous forme d'arguments supplémentaire au prédicat qui en a besoin.

[25] Il y en a plusieurs en cas de coordination.

unique chemin dans ce DAG sera utilisé par une analyse entière. Cependant, comme expliqué plus bas, des restrictions raisonnables à la structure du DAG d'entrée permettent de contourner ce problème.

3 Un formalisme linguistique non-linéaire : les Méta-RCG

Dans les sections précédentes, nous avons vu que les formalismes non-linéaires polynomiaux semblent des alternatives intéressantes aux formalismes usuels qui reposent sur un squelette syntaxique linéaire assorti de décorations reposant sur l'unification. Dans cette section, nous allons présenter de façon succincte un formalisme non-linéaire polynomial que nous avons développé dans le but de l'utiliser à des fins de modélisation linguistique, le formalisme des Méta-RCG, mais également une grammaire du français dans ce formalisme, l'analyseur associé, et les analyses qu'il produit.

Comme dit plus haut, les deux principaux formalismes non-linéaires qui étendent les grammaires non-contextuelles sont les RCG et les sLMG. Nous avons donc commencé par choisir celui de ces deux formalismes qui serait la base de notre formalisme linguistique (c'est-à-dire son formalisme d'implémentation). Rappelons la différence principale entre RCG et sLMG : les RCG traitent de prédicats sur des intervalles de la chaîne d'entrée, les sLMG traitent de prédicats sur des sous-chaînes de la chaîne d'entrée. On est donc ramenés aux deux questions suivantes :

1. Les faits linguistiques sont-ils des faits sur les sous-chaînes de la phrase ou sur ses intervalles ?
2. Est-il plus facile d'écrire une RCG ou une sLMG ?

Nous avons vu précédemment que la réponse à la deuxième question est en faveur des RCG. La première question est cependant plus pertinente. Elle est également en faveur des RCG pour la raison suivante (Sagot et Boullier, 2004) : d'un point de vue linguistique, deux occurrences de la même sous-chaîne dans une phrase donnée sont deux objets linguistiques différents (par exemple, il peut s'agir de deux homonymes, ils peuvent être impliqués dans des relations différentes, etc...). Ainsi, il semble plus naturel de traiter avec des intervalles, qui peuvent être reliés avec les concepts de *position*, de *place* ou de *trace* (intervalle de longueur nulle) que l'on trouve dans certaines théories linguistiques. Pour cette raison, mais aussi à cause de l'existence d'un analyseur RCG efficace (Boullier (2004), voire chapitre 3), notre formalisme linguistique polynomial non-linéaire repose sur les RCG. Il en étend la syntaxe mais pas l'expressivité, d'où son nom, *Méta-RCG*.

3.1 Présentation des Méta-RCG

Soit $G_{RCG} = (N, T, V, P, S)$ une RCG, comme définie au chapitre 1. Nous allons définir une *Méta-RCG* G_{MRCG} qui étend G_{RCG} (l'extension concerne les grammaires, pas nécessairement les langages). Pour cela, nous allons tout d'abord passer en revue un certain nombre de remarques et de définitions.

3.1.1 Têtes

Quoique controversée pour bien des raisons, la notion de *tête d'un syntagme* est largement répandue dans la littérature linguistique, et a été utilisée à grande échelle par de nombreux formalismes linguistiques, comme les HPSG (Pollard et Sag, 1994), mais aussi, entre autres, en LFG ou en Grammaires de Dépendances. Nous définissons les têtes en étendant la notion d'argument d'un prédicat RCG de la façon suivante. Nous appelons *argument Méta-RCG*, ou simplement *argument* l'un des objets suivants :

- un argument RCG, c'est-à-dire comme vu précédemment la concaténation d'éléments de $V \cup T$,
- un *argument à têtes et coordonnants*, ou *argument TC*, c'est-à-dire un élément de V suivi de l'opérateur $^\wedge$ ou de l'opérateur !,
- un *argument à tête unique*, ou *argument TU*, c'est-à-dire un élément de V suivi de l'opérateur $^+$,
- un *argument d'ajout de tête*, ou *argument AT*, c'est-à-dire un argument de la forme $V_1{}^+ V_2{}^+ V_3^\wedge$.

Un argument qui n'est pas un argument RCG standard est appelé *argument syntagmatique*.

La signification de ces différents types d'arguments peut se définir comme suit. Si $Syntagme$ est un intervalle dénotant un syntagme, $Syntagme^\wedge$ dénote un couple de listes, la première étant la liste des intervalles couvrant ses têtes et la seconde la liste des intervalles couvrant les coordonnants qui les séparent. Ainsi, si la sous-chaîne correspondant à $Syntagme$ est *une pomme ou une poire*, une grammaire raisonnable produira une analyse telle que $Syntagme^\wedge$ comprend une liste têtes formée des deux intervalles couvrant (exactement) *pomme* et *poire*, et une liste de coordonnants formée de l'unique intervalle couvrant *ou*.

L'argument $Syntagme!$ dénote la même chose que $Syntagme^\wedge$, mais le nombre de têtes (respectivement de coordonnants) doit être exactement égal à 1 (respectivement 0).

Un argument à tête unique V^+ crée un argument syntagmatique constitué d'une liste de têtes contenant un seul intervalle, à savoir V, et une liste vide de coordonnants.

Enfin, un argument d'ajout de tête $V_1{}^+V_2{}^+V_3^\wedge$ est un argument syntagmatique dont les têtes sont la concaténation de V_1 et de la liste de têtes contenue dans $V_3^\wedge$, et dont les coordonnants sont la concaténation de V_2 et de la liste de coordonnants contenue dans $V_3^\wedge$.

Par exemple, et simplement pour illustrer notre propos, une clause analysant (récursivement) un groupe nominal coordonné pourrait avoir la forme suivante :

NP(*Det Head Coord Np2*, *Head*$^+$*Coord*$^+$*Np2*$^\wedge$) $\rightarrow$ NOUN(*Head*) DET(*Det*, *Head*)
COORD(*Coord*) NP(*Np2*, *Np2*$^\wedge$)

3.1.2 Attributs et numéros d'homonymes

Un *attribut* peut être défini comme un vecteur (fini) $\vec{F} = (f_1, \ldots, f_n)$. Une *valeur constante* de l'attribut $\vec{F}$ est un élément de $\vec{F}$. Une *valeur variable* de l'attribut $\vec{F}$ est la concaténation de l'opérateur \$ et d'un élément d'un ensemble V_f de symboles de variables d'attributs (par exemple, si $g \in V_f$, $\$g$ est une valeur variable valide pour $\vec{F}$). On appelle *liste d'attributs* une liste de noms d'attributs. Un *couple attribut-valeur*, ou *couple a-v*, est de la forme *F=v*, où *F* est le nom d'un attribut et v une valeur constante ou variable de $\vec{F}$. Enfin, une *liste de couples attribut-valeur* est une séquence de couples a-v dans laquelle un attribut ne peut apparaître qu'une seule fois.

Nous appelons *numéro d'homonyme* un vecteur similaire à un attribut, défini par $\vec{HN} = (0, \ldots, h_{max})$. Tout d'abord, nous définissons un *argument terminal* comme un argument composé d'au plus une variable, laquelle variable dénote un intervalle de longueur au plus 1. Le rôle d'un numéro d'homonyme est alors d'associer à certains arguments terminaux de certains prédicats un numéro spécial qui permet de distinguer plusieurs terminaux (c'est-à-dire plusieurs mots) homonymes. Pour chaque prédicat, une fonction spéciale *arguments-à-HN* donne la liste des positions de ses arguments qui ont de tels numéros d'homonymes. Mais ces numéros d'homonymes ne sont pas visibles dans les clauses Méta-RCG elles-mêmes.

3.1.3 Contextes

Nous définissons un *élément de contexte* $Ctxt$ comme étant un élément de l'ensemble V des symboles de variables, éventuellement suivi de l'opérateur / ou de l'opérateur $^+$, et éventuellement suivi de l'opérateur : et d'une liste d'attributs. Le rôle des éléments de contexte est de modéliser les dépendances à longue distance. Pour cette raison, et bien qu'ils soient définis de façon déclarative comme tout notre formalisme, ils sont plus facile à décrire d'un point de vue opérationnel.

Les dépendances à longue distance seront traitées en deux étapes qui ont des points communs avec le trait SLASH en HPSG, et que l'on peut décrire informellement comme suit. À un

certain point de l'analyse, un élément de contexte $Ctxt$ est construit à partir du syntagme $Ctxt$ concerné puis « empilé » dans le *contexte* du prédicat courant A, dont on a spécifié qu'il a dans son contexte un élément $Ctxt$. Tous les prédicats qui ont dans leur contexte un élément $Ctxt$, qui sont reliés à A par de tels prédicats, et pour lesquels aucune nouvelle valeur n'est explicitement donnée, héritent de cet élément de contexte. Cet élément est ainsi transporté vers d'autres régions de l'analyse. À un endroit (éventuellement tout à fait éloigné) de l'analyse qui dispose de cet élément de contexte $Ctxt$, celui-ci, ses têtes ou ses traits morphologiques peuvent être rapatriés (ou *dépilés*) dans des arguments de prédicats, et utilisés.

Si l'on associe à un élément de contexte une liste d'attributs (derrière l'opérateur :), cette liste doit être empilée en plus de l'intervalle lui-même (si nom fait partie de cette liste d'attributs, la valeur empilée est celle de l'attribut du prédicat de partie gauche de nom $Ctxt.nom$). L'opérateur / signifie que la liste d'attributs associée à l'intervalle $Ctxt$, qui est alors obligatoire, est empilé dans le contexte, mais pas l'intervalle proprement dit. L'opérateur $^+$, qui suppose que l'intervalle concerné est un syntagme, indique que les têtes (et les coordonnants) de l'intervalle doivent être empilés en plus de l'intervalle.

La percolation d'éléments de contexte d'un point à l'autre de l'analyse se fait au moyen d'une fonction spéciale que nous devons définir, appelée $contexte$. Cette fonction est définie sur l'ensemble N des noms de prédicats, et, pour un prédicat donné, renvoie une liste d'éléments contextuels. La signification de cette fonction est la suivante : étant donné un nom de prédicat A, les éléments de contexte présents dans $contexte(A)$ sont associés à toutes les occurrences de prédicats A. Supposons que, dans une clause donnée, le même élément de contexte $Ctxt$ (avec ses têtes et/ou ses attributs) est associé à plus d'un prédicat. Ceux de ces prédicats qui sont en partie droite et qui redéfinissent explicitement la valeur de $Ctxt$, via l'opérateur $=$, récupèrent cette valeur. Tous les autres partageront une même valeur, qui est celle du prédicat de partie gauche. En cas de besoin, on peut accéder à cette valeur via l'opérateur $=$ qui est alors utilisé en partie gauche. Cet opérateur $=$, qui est un opérateur d'égalité entre intervalles pour l'intervalle et ses têtes, et un opérateur d'égalité de valeurs pour les attributs, représente par conséquent à la fois l'empilage (en partie droite) et le dépilage (en partie gauche) d'éléments contextuels. Et pour cause : d'un point de vue déclaratif, ces opérations sont identiques.

Par exemple, si un élément de contexte $Ctxt$ est associé aux noms de prédicats A et B, alors la clause A(X) $\rightarrow$B(X), où rien ne redéfinit de valeur pour $Ctxt$, va garantir le fait que $Ctxt$ est passé de A à B exactement de la même façon que l'est X.

On définit un *couple élément contextuel - valeur*, ou *couple c-v*, ou *équation contextuelle*, comme étant une expression de la forme $Ctxt = Intervalle$ ou $Ctxt.attribut = valeur$. Nous en verrons des exemples après avoir défini la syntaxe complète des Méta-RCG.

3.1.4 Méta-RCG

Les clauses Méta-RCG étendent les clauses RCG, puisque les prédicats sont remplacés par des prédicats Méta-RCG de la forme

$$\mathrm{A}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_p)[\phi_1 \ldots \phi_j \ldots \phi_q]\{\kappa_1 \ldots \kappa_k \ldots \kappa_r\},$$

où les α_i sont des arguments Méta-RCG, les ϕ_j sont des couples a-v (équations sur les attributs), et les κ_k sont des couples c-v (équations contextuelles). La partie concernant les attributs et celle concernant les éléments contextuels sont facultatives. Par exemple, et en utilisant *DepLD* comme abréviation de *syntagme à dépendance longue-distance*, voici une clause Méta-RCG valide (on suppose que $contexte(\mathrm{NP}) = \{\{\mathrm{DepLD}^+ : \mathrm{nombre,cas}\}\}$) :

$$\begin{aligned} \mathrm{SV}(V, V^+)\{\mathrm{DepLD.cas=accusatif}\} \quad \rightarrow \quad & \mathrm{VERBE}(V) \\ & \mathrm{OBJ}(\mathit{DepLD}, \mathit{DepLD}^+, V)[\mathrm{nbr=DepLD.nbr}]\{\mathrm{DepLD}=0\}, \end{aligned}$$

où *DepLD=0* est un raccourci pour *DepLD est un intervalle de longueur vide*. On peut gloser cette clause de la façon suivante : on peut faire de l'intervalle *V* un syntagme verbal (prédicat *SV*) dont *V* est lui-même la tête, dans un contexte où l'on dispose d'une dépendance à longue distance *DepLD* de cas accusatif, si *V* est un verbe (prédicat VERBE) et si l'on peut faire de *DepLD* l'objet direct de ce verbe (cet objet direct ayant le même nombre que *DepLD* et n'ayant plus lui même de dépendance à longue distance à consommer).

3.2 Conversion d'une Méta-RCG en RCG

Comme indiqué précédemment, toute grammaire Méta-RCG peut être convertie en une RCG fortement équivalente. Ceci permet d'utiliser l'analyseur RCG de Pierre Boullier pour obtenir un analyseur Méta-RCG, pour peu que l'on développe un compilateur de Méta-RCG en RCG. Mais cela permet surtout d'avoir un analyseur purement syntaxique, c'est-à-dire sans décorations, malgré la syntaxe Méta-RCG qui « simule » certains types de décorations. Les détails de ce processus de compilation de Méta-RCG en RCG ne sont pas très intéressants, et ne seront pas développés ici[26]. Nous nous contenterons d'indiquer que l'ensemble des informations supplémentaires (équations sur les attributs et équations contextuelles) sont passées sous la forme d'arguments supplémentaires aux prédicats RCG[27].

[26] C'est pourtant un processus qui est très loin d'être trivial, à la fois d'un point de vue théorique et d'un point de vue d'implémentation.

[27] Les attributs voient par exemple leurs valeurs représentées par des intervalles, en sorte que l'intervalle $0..i$ représente la i-ième valeur possible de l'attribut associé à un argument donné. Si le nombre maximum de valeurs distinctes possibles pour un trait (maximum connu statiquement) est supérieur à la longueur de la phrase, on complète la phrase par un nombre approprié de terminaux spéciaux.

Nous avons développé un tel compilateur, appelé `mrcg2rcg`. À partir d'une Méta-RCG, ce compilateur produit donc une RCG fortement équivalente, mais aussi un fichier regroupant toutes les informations nécessaires à la conversion des analyses RCG produites par l'analyseur de Pierre Boullier en analyses Méta-RCG.

3.3 Projection partielle d'une analyse globale en analyses classiques

Comme indiqué précédemment, la non-linéarité des Méta-RCG permet de traiter au même niveau les phénomènes de dépendance, de constituance, de sémantique lexicale, et d'autres. De plus, le fondement même des RCG, à savoir la concaténation d'intervalles, rend la vision topologique de la langue inhérente à toute grammaire Méta-RCG. Par conséquent, l'analyse d'une phrase obtenue à partir d'une grammaire Méta-RCG regroupe des faits linguistiques participant de ces diverses classes de phénomènes.

Il est ainsi aisé d'extraire d'une analyse Méta-RCG des *vues partielles* qui constituent des analyses classiques (en constituants, en dépendances, en boîtes topologiques, en relations prédicats-arguments sémantiques, etc.). Cette extraction se fait par *projection partielle*, c'est-à-dire qu'aucune information n'est calculée à partir de l'analyse Méta-RCG, elles en sont extraites par projection. Nous avons donc défini un langage de spécification de projections des analyses Méta-RCG, des spécifications dans ce langage de divers types de projections, et un module d'application de ces spécifications pour obtenir ces vues partielles. Les projections partielles déjà disponibles construisent des vues en constituants, en dépendances, en boîtes topologiques, et en sémantique prédicative. Enfin, nous avons défini un outil de conversion de ces vues partielles pour générer automatiquement des vues graphiques.

3.4 L'interface entre lexique et grammaire

Une partie importante du développement d'un formalisme linguistique réside dans la définition d'une interface entre la grammaire et le lexique. Bien que cela soit possible en théorie, il n'y a aucune raison de contraindre une Méta-RCG à être lexicalisée, c'est-à-dire à inclure dans toute clause au moins un symbole terminal dans au moins un argument du prédicat de partie gauche. Cependant, un grand nombre d'informations est fourni par les terminaux, qui sont ici les mots[28]. Cette information doit être à la fois représentée et utilisée d'une façon qui, comme

[28]Puisque l'on souhaite inclure les propriétés de sémantique lexicale dans la grammaire, l'approche traditionnelle de nombre de formalismes consistant à associer à chaque mot une *catégorie*, dont la valeur est utilisée comme symbole terminal de la grammaire, n'est pas satisfaisante. Cela induirait une perte trop importante d'informations ou nécessiterait un tel nombre de catégories qu'il n'y aurait aucun avantage par rapport à prendre, comme nous le faisons, les mots eux-même comme terminaux.

l'ensemble du formalisme, doit être à la fois utilisable efficacement et acceptable linguistiquement.

De nombreuses options peuvent être envisagées, qui tirent parti des propriétés de clôture des RCG :

1. Compiler une énorme grammaire avec autant de terminaux que de formes fléchies dans le lexique.
2. Compiler d'un côté la partie non lexicalisée de la grammaire, et construire d'autre part un ou plusieurs modules de clauses lexicalisées, appelés *pseudo-grammaires*. De tels modules peuvent être obtenus par compilation de clauses Méta-RCG en clauses RCG puis par utilisation du générateur d'analyseurs RCG. Mais on peut aussi construire un module spécifique unique de façon indépendante du formalisme à proprement parler. Il pourrait profiter de la structure hiérarchique des informations lexicales, utiliser des algorithmes spécifiques pour permettre le passage à l'échelle, et être capable de compléter le graphe d'analyse RCG produit par la partie non lexicalisée à l'aide de sous-graphes cohérents.
3. Compiler la partie non lexicalisée de la grammaire, et, pour chaque phrase, générer et compiler dynamiquement l'ensemble des clauses lexicalisées mettant en jeu les mots présents dans cette phrase (qui forme une *pseudo-grammaire*).

Indépendamment de ce choix, les clauses lexicalisées peuvent être représentées soit telles quelles, ou peuvent être obtenues par un processus d'héritage au sein d'une ontologie lexicale.

4 Analyser le français avec une Méta-RCG

Le développement du formalisme Méta-RCG s'est fait en parallèle avec celui d'une grammaire et d'un analyseur du Français. Par manque de place, nous ne sommes pas en mesure de donner ici un fragment de notre grammaire. En revanche, nous allons donner un aperçu de sa couverture par divers exemples de phénomènes linguistiques et de phrases correctement analysées. Puis nous illustrerons certains aspects de la structure de notre grammaire par deux exemples.

4.1 Une Méta-RCG du français

Dans son état actuel, notre Méta-RCG du français comporte 370 clauses grammaticales (c'est-à-dire non lexicales) se compilant en une RCG à 625 clauses, d'arité 73 et de degré maximal 40.

4.1.1 Prise en compte de la sémantique lexicale

C'était une des motivations du développement du formalisme Méta-RCG, d'une grammaire du français dans ce formalisme et du système d'analyse associé que de pouvoir modéliser la langue d'une façon qui intègre les contraintes simples de sémantique lexicale (restrictions de sélection). Nous avons vu comment la prise en compte de telles contraintes dans la grammaire permet de les utiliser au cours de l'analyse, et non après. L'effet induit est naturellement que l'analyse obtenue comporte un certain nombre de prédicats de sémantique lexicale qui décrivent la structure prédicative de la phrase analysée.

Notre Méta-RCG du français met en œuvre ceci de façon systématique. Tout mot plein (nom, verbe, adjectif, adverbe) est vu comme un prédicat pouvant avoir un ou plusieurs arguments sémantiques. En généralisant de manière peut-être abusive l'emploi de certains termes, nous disons par exemple que le nom que modifie un adjectif est son *patient*, qu'un modifieur prépositionnel d'un nom introduit par *de* est l'*agent*, le *patient* ou le *possesseur* du nom qu'il modifie (cf. les exemples suivants :*le départ de Paul*, *le licenciement de Paul*, *le chien de Paul*)[29], etc. Nous avons défini pour cela un nombre très restreint de types d'arguments sémantiques correspondant à un certain nombre de prédicats. Ces prédicats sont appelés par des prédicats représentant les fonctions syntaxiques (c'est ainsi que le *sujet* d'un verbe, selon la diathèse de ce dernier, sera aussi son *agent* ou son *patient*), et appellent eux-mêmes des prédicats modélisant les restrictions de sélection (c'est ainsi que l'*agent* d'une forme du verbe *manger* devra valider un prédicat vérifiant qu'il est *animé*).

On constate alors un des intérêts de l'approche non-linéaire : si l'on commence une analyse faisant d'un inanimé le sujet d'une forme active de *manger*, l'analyse échoue immédiatement, alors que dans un système tel que SXLFG elle continue jusqu'à échouer bien plus tard, voire jusqu'à réussir selon des critères purement syntaxiques[30].

4.1.2 Des constituants et des dépendances aux relations de discours

Notre grammaire modélise donc les dépendances syntaxiques et les dépendances sémantiques (sémantique prédicative simple). Naturellement, elle modélise également les propriétés morphologiques, les rapports de constituance, ainsi que les rapports purement topologiques,

[29]Il en est de même d'un déterminant possessif : outre sa fonction de détermination, il joue le rôle d'*agent*, de *patient* ou de *possesseur* du nom qu'il détermine dans des exemples tels que respectivement *mon départ*, *mon licenciement* et *mon chien*.

[30]Naturellement, ceci a une contrepartie au niveau des problématiques de robustesse. La prise en compte de contraintes supplémentaires telles que les contraintes de sémantique lexicale, qui sont plus souvent violées (emplois figurés, métaphores,...), nécessite des mécanismes de rattrapage en cas d'échec d'autant plus perfectionnés. Nous avons un certain nombre d'idées précises sur ce qu'il conviendrait de faire pour mettre en œuvre de tels mécanismes dans un analyseur Méta-RCG (plus exactement dans l'analyseur RCG sous-jacent), mais nous ne les avons pas encore implémentés.

capturés très naturellement par les notions d'intervalle et de concaténation qui sont au cœur du formalisme.

Nous avons cherché à voir dans quelle mesure nous pouvions tirer parti de la non-linéarité pour aller au-delà de l'échelle de la phrase, et instaurer des relations entre phrases distinctes. Pour cela, nous avons utilisé le mécanisme des contextes pour analyser plusieurs phrases successivement en passant d'une phrase à la suivante un certain nombre d'informations, et en particulier le prédicat principal (verbal) de la phrase précédente ainsi que ses propriétés aspectuelles (représentées pour le moment par son temps). Un certain nombre de clauses, donc certaines sont ancrées sur des lexèmes appelés *connecteurs*, et d'autres sont non-lexicalisées[31], construisent des relations de discours entre les phrases. Il s'agit pour le moment d'une partie très prototypale de notre grammaire, mais elle montre la faisabilité d'un traitement grammatical des relations de discours dans le formalisme Méta-RCG. Une méthode similaire pourrait être très facilement mise en œuvre pour traiter, également via les contextes, des problèmes tels que l'anaphore. Mais il s'agit là de travail encore à venir.

4.1.3 Interface lexique-grammaire

Dans l'état actuel de notre système d'analyse, nous utilisons un lexique construit comme suit. Nous disposons d'un lexique syntaxico-sémantique[32] qui est un graphe d'héritage dont la forme est proche de la forme intensionnelle du L*efff* (voir chapitre 5), à ceci près que les classes ne sont pas purement syntaxiques, les propriétés atomiques dont héritent les nœuds étant des clauses Méta-RCG dont certaines concernent la sémantique lexicale. Les nœuds terminaux de la hiérarchie sont associés à des formes, à des lemmes, ou à des classes de lemmes. Une forme qui correspond à un nœud terminal dans cette hiérarchie se voit attribuer les clauses dont ce nœud hérite. Dans le cas contraire, la partie morphologique du L*efff* est utilisée pour obtenir les lemmes possibles pour cette forme. Si un lemme correspond à un nœud terminal dans la hiérarchie, il se voit attribuer les clauses dont ce nœud hérite. À défaut, on recherche s'il existe un nœud associé à une classe de lemmes qui lui correspond (par exemple le nœud **tion/nom commun* pour le lemme *construction/nom commun*). Dans le pire des cas, des nœuds par défauts existent pour chaque catégorie.

Enfin, et ce à titre provisoire, la prise en compte des clauses lexicalisées, et donc construites par héritage dans le lexique, se fait dynamiquement : pour chaque phrase, une nouvelle pseudo-

[31] Il en est ainsi, à titre d'exemple, d'une clause qui relie deux phrases par une relation de discours de type *arrière-plan* pour peu qu'aucun connecteur ne les relie, que la première phrase soit à l'imparfait et la seconde au passé composé ou au passé simple.

[32] Sa couverture correspond *grosso modo* à ce qui est nécessaire pour analyser le corpus EUROTRA évoqué plus haut.

grammaire lexicalisée est construite, compilée, et liée au module non-lexicalisé pré-existant à la volée. C'est la propriété de modularité des RCG qui permet une telle architecture.

4.1.4 Phénomènes couverts

Notre grammaire Méta-RCG du français couvre un certain nombre de phénomènes, comme illustré dans le tableau 1 (la plupart des phrases sont issues du corpus EUROTRA, cf. Danlos et Laurens (1991)).

4.2 Deux exemples d'analyses

Pour valider le formalisme des Méta-RCG, et en parallèle à son élaboration, nous avons développé (et développons encore) une grammaire du français dans ce formalisme. Comme indiqué précédemment, un des intérêts de la non-linéarité des Méta-RCG – et c'est une des motivations pour ce formalisme – réside dans le fait qu'on puisse faire interagir des contraintes de différentes natures, et en particulier des contraintes *morphologiques*, *syntaxiques* et *sémantiques* (au sens de la sémantique lexicale, restreinte aux restrictions de sélection et à la sémantique prédicative).

Pour illustrer ceci, nous allons montrer sur deux exemples la façon dont l'analyse se passe. La première phrase *Un avocat mange un avocat*, met en œuvre les mécanismes de numéros d'homonymes et les contraintes de sémantique lexicale pour désambiguïser complètement cette phrase, malgré l'ambiguïté lexicale du nom *avocat*. La seconde phrase, *Pierre veut une bière et dormir*, est un exemple de coordination hétérogène (entre un groupe nominal et une infinitive) mais aussi un exemple de verbe à contrôle sujet (*veut*). Naturellement, pour ne pas rentrer dans des détails complexes qui prendraient trop de place, nous ne présenterons que des parties, qui plus est simplifiées, du processus d'analyse, en mettant l'accent sur le traitement des mécanismes évoqués.

Par la suite, les clauses Méta-RCG instanciées respecteront les conventions suivantes. Un intervalle $\langle i..j \rangle_w$ couvrant la sous-chaîne s sera représentée par $s_{i..j}$. Si cet intervalle a un numéro d'homonyme h, il sera représenté par $s^h_{i..j}$. S'il a des têtes, les terminaux (les mots) correspondant seront soulignés, et leurs numéros d'homonymes indiqués en exposant (toute tête a un numéro d'homonyme).

4.2.1 Ambiguïté et sémantique lexicales : *Un avocat mange un avocat*

Considérons la phrase suivante :

(RCG_2) Un avocat mange un avocat

Phénomène	Exemple	#a.
Désambiguïsation lexicale	La décision appartiendra à la Commission Le pilote conduit l'avion / La route conduit à Rome	1 1/1
Mot inconnu	Jean mange un bortsch	1
Négation et adverbe dans le noyau verbal	Les mesures que l'industrie a prises n'ont malheureusement pas renversé cette tendance	1
Verbe support	Le développement des entreprises présente un intérêt économique	1
Sujet phrastique	Que Luc ait renversé cette tendance gêne Marie	1
Complétives introduites par une préposition	Luc a pris conscience de ce que Marie renverse cette tendance	1
Complétive adverbiale, négation, passif	Bien que le problème ait été analysé, la Commission n'a pas pris les décisions nécessaires	1
Relative à extraction longue distance	Les mesures que la Commission a proposé que le Conseil prenne sont acceptées	1
Préposition composée, infinitive	Jean a réussi à force d'avoir lu le livre	1
Pronom relatif composé et ambiguïté lexicale	Le Japon par qui l'Europe sait que les mesures sont acceptées prospère	2
Extraction d'un complément sous-catégorisé par un nom	Pierre raconte l'agression qui a été commise par Jean contre Marie	1
Participiale modifieur du sujet	En étant abandonnée par la Commission, la proposition a été refusée	1
Participiale "active" apposée	Luc parti, l'Europe refusera le projet	1
Participiale "passive" apposée	La proposition abandonnée, le projet a été refusé par le Conseil	1
Sous-catégorisation nominale	La collaboration du Japon avec l'Europe pour conserver les marchés industriels engage la recherche	1
Coordination hétérogène	Pierre veut une bière et dormir	1
Difficultés multiples	Les entreprises dans lesquelles le Japon veut que la Commission accepte que l'Europe investisse fabriquent des ordinateurs	1
Coordination, verbes à contrôle et extraction génitive à longue distance	Paul aime la Normandie dont je sais que Pierre regarde Marie et Paul manger une pomme et une poire verte	1

TAB. 1 – Quelques phénomènes reconnus par notre grammaire Méta-RCG ("#a." veut dire "nombre d'analyses").

Cette phrase, syntaxiquement, est extrêmement simple. Nous ne l'utilisons comme exemple que pour donner un aperçu des clauses de bases de notre grammaire et pour montrer le rôle des numéros d'homonymes. De plus, les clauses seront simplifiées, par souci de lisibilité. En particulier, les prédicats et les clauses traitant de positions topologiques vides (modifieurs de phrases, adverbes, clitiques, dépendants à longue distance, etc…) seront ignorés. Enfin, pour simplifier encore, les attributs ne seront pas indiqués, sauf lorsque c'est absolument nécessaire à la présentation. Certaines parties de l'analyse, qui ne sont pas fondamentales pour la compréhension générale, sont remplacées par des descriptions textuelles en italiques. Ceci étant dit, voici comment notre grammaire analyse cette phrase (*NV* est l'abréviation de *Noyau Verbal* et *CV* celle de *Complexe Verbal*) :

PHRASE(*un avocat mange un avocat*$_{0..5}$)	$\rightarrow$	VSEM_IND(*mange*$^{0}_{2..3}$)
		PHRASE2(*un avocat* $\underline{\textit{mange}}^{0}$ *un avocat*$_{0..5}$)
VSEM_IND(*mange*$^{0}_{2..3}$)	$\rightarrow$	VERBE(*mange*$^{0}_{2..3}$)
		LEX(*mange*$^{0}_{2..3}$)[mode=Ind]
VERBE(*mange*$^{0}_{2..3}$)	$\rightarrow$	ε
LEX(*mange*$^{0}_{2..3}$)[mode=Ind]	$\rightarrow$	ε
PHRASE2(*un avocat* $\underline{\textit{mange}}^{0}$ *un avocat*$_{0..5}$)	$\rightarrow$	SUBJECT(*un* $\underline{\textit{avocat}}^{0}{}_{0..2}$, *mange*$^{0}_{2..3}$)
		VP($\underline{\textit{mange}}^{0}$ *un avocat*$_{2..5}$){Subj=*un* $\underline{\textit{avocat}}^{0}{}_{0..2}$}
VP($\underline{\textit{mange}}^{0}$ *un avocat*$_{2..5}$)	$\rightarrow$	NV($\underline{\textit{mange}}^{0}_{2..3}$, *un avocat*$_{3..5}$)
NV($\underline{\textit{mange}}^{0}_{2..3}$, *un avocat*$_{3..5}$)	$\rightarrow$	CV($\underline{\textit{mange}}^{0}_{2..3}$, $_{3..3}$, $_{3..3}$)
		ROLES($_{2..2}$, $_{2..2}$, $_{2..2}$, *un avocat*$_{3..5}$, *mange*$^{0}_{2..3}$)
ROLES($_{2..2}$, $_{2..2}$, $_{2..2}$, *un avocat*$_{3..5}$, *mange*$^{0}_{2..3}$)	$\rightarrow$	OBJECT(*un* $\underline{\textit{avocat}}_{3..5}{}^{1}$, *mange*$^{0}_{2..3}$)
		ROLES(*un* $\underline{\textit{avocat}}_{3..5}{}^{1}$, $_{2..2}$, $_{2..2}$, $_{5..5}$, *mange*$^{0}_{2..3}$)

Avant de poursuivre l'analyse, il est nécessaire de clarifier la signification des arguments du prédicat ROLES. En réalité, les trois premiers arguments (il y en a cinq dans la grammaire réelle) dénotent les rôles syntaxico-sémantiques associés au verbe. Le premier rôle est le rôle accusatif, le second est le rôle datif, le troisième est le rôle génitif. Le prédicat récursif ROLES « consomme » à chaque appel un complément dans la liste des compléments pas encore analysés (son quatrième argument), l'analyse, et, s'il remplit un rôle encore vide, remplit l'argument correspondant de l'appel de partie droite de ROLES avec ledit complément. Quand l'intervalle des compléments non encore analysés est vide, le remplissage des rôles est terminé, et des vérifications peuvent être faites sur les rôles impossibles ou obligatoires. Ceci étant dit, voici comment l'analyse se poursuit.

ROLES(*un* $\underline{avocat}^{1}_{3..5}$, $_{2..2}$, $_{2..2}$, $_{5..5}$, $mange^{0}_{2..3}$)	→	Vrai car $mange^{0}$ est un verbe transitif et car le premier argument de ROLES, qui est le rôle accusatif, est rempli par *un* $\underline{avocat}^{1}_{3..5}$
SUBJECT(*un* $\underline{avocat}^{0}_{0..2}$, $mange^{0}_{2..3}$)	→	SUBJ_SEM($avocat^{0}_{1..2}$, $mange^{0}_{2..3}$) NP(*un* $\underline{avocat}^{0}_{0..2}$)
SUBJ_SEM($avocat^{0}_{1..2}$, $mange^{0}_{2..3}$)[dth=act]	→	AGENT($avocat^{0}_{1..2}$, $mange^{0}_{2..3}$)
AGENT($avocat^{0}_{1..2}$, $mange^{0}_{2..3}$)	→	ANIME($avocat^{0}_{1..2}$)
OBJECT(*un* $\underline{avocat}^{1}_{3..5}$, $mange^{0}_{2..3}$)	→	OBJ_SEM($avocat^{1}_{4..5}$, $mange^{0}_{2..3}$)[dth=pass] NP(*un* $\underline{avocat}^{1}_{3..5}$)
OBJ_SEM($avocat^{1}_{4..5}$, $mange^{0}_{2..3}$)[dth=pas]	→	PATIENT($avocat^{1}_{4..5}$, $mange^{0}_{2..3}$)
PATIENT($avocat^{1}_{1..2}$, $mange^{0}_{2..3}$)	→	COMESTIBLE($avocat^{1}_{4..5}$)
NP(*un* $\underline{avocat}^{0}_{0..2}$)	→	Vrai car *un* $\underline{avocat}^{0}_{1..2}$ est un groupe nominal valide
NP(*un* $\underline{avocat}^{1}_{3..5}$)	→	idem
ANIME($avocat^{0}_{1..2}$)	→	ε
COMESTIBLE($avocat^{1}_{4..5}$)	→	ε

On constate donc que le processus global peut se résumer ainsi :

1. Identifier la partie sémantique du verbe principal (le participe passé s'il y a un ou plusieurs auxiliaires, la forme finie sinon), et en faire la tête de la phrase,
2. Identifier tout le noyau verbal (composants verbaux, clitiques et adverbes infixes) ;
3. Analyser le sujet, qui est d'une part un syntagme nominal (prédicat NP ; les syntagmes nominaux peuvent être des groupes nominaux, des infinitives ou des complétives) et d'autre part un argument sémantique du verbe (prédicat SUBJ_SEM, qui est souvent vrai si AGENT est vrai sur les mêmes arguments, mais ce n'est pas le cas au passif ou pour les impersonnels) ; le sujet est empilé dans le contexte (non utilisé pour la phrase considérée pour le moment) ;
4. Identifier l'un après l'autre tous les compléments post-verbaux (ici seulement un), et les analyser à la fois comme syntagmes nominaux et comme arguments sémantiques du verbe (prédicats $*\_SEM$).

Naturellement, cette présentation simplifiée n'explique pas comment sont traités les attributs du sujet ou d'un complément, les clitiques, les dépendances à longue distance, les modifieurs, les relatives, etc. . .C'est seulement le cœur de la grammaire.

Pour illustrer, nous donnons figure 1 l'arbre de constituants extrait automatiquement de l'analyse complète de la phrase.

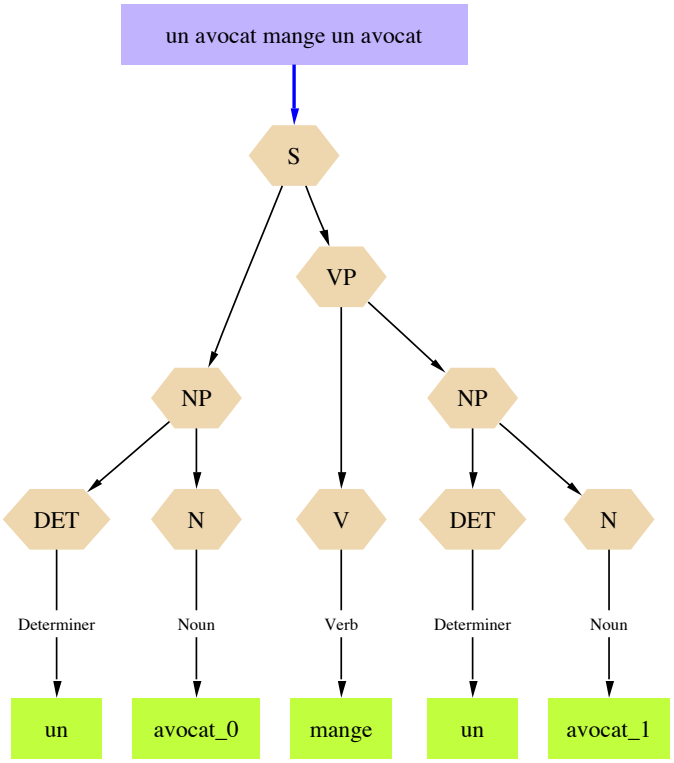


FIG. 1 – Vue en constituants de l'analyse de la phrase (RCG_2).

4.2.2 Coordination hétérogène et verbe à contrôle : *Pierre veut une bière et dormir*

Considérons la phrase suivante :

(RCG_3) Pierre veut une bière et dormir

Comme indiqué précédemment, cette phrase met en œuvre deux phénomènes : une coordination hétérogène entre un groupe nominal (*une bière*) et une infinitive (*dormir*), et un verbe à contrôle sujet (*veut*) qui fait de son sujet (*Pierre*) le sujet d'un des termes coordonnés formant son infinitif objet (*dormir*). Nous ne donnerons pas l'analyse complète de la phrase, mais seulement les parties concernées par ces deux phénomènes.

Tout d'abord, nous allons voir comment est traitée la coordination hétérogène. Nous faisons la distinction entre un groupe nominal et un syntagme nominal. Nous appelons *syntagme nominal* un syntagme (par ailleurs non nécessairement continu) qui joue dans la phrase un rôle canoniquement dévolu à un groupe nominal (entité à une seule tête, laquelle est un nom ou un pronom plein, et qui elle non plus n'est pas nécessairement continue). Ainsi, un groupe nominal, une infinitive ou une complétive sont pour nous des syntagmes nominaux. Cette distinction permet de modéliser simplement un grand nombre de phénomènes, y compris le fait qu'une infinitive et un groupe nominal peuvent effectivement être coordonnés, comme dans la phrase que nous étudions ici. Cela permet également de modéliser le fait, par exemple, que de nombreux verbes peuvent avoir pour objet direct une infinitive ou une complétive[33]. Des prédicats tels que OBJ_SEM peuvent toutefois contraindre l'objet direct d'un verbe à être ou à ne pas être une infinitive, par exemple. Mais dans notre grammaire, nous considérons ces contraintes comme des contraintes sur l'objet, et non comme des différences fondamentales entre types de syntagmes, au moins en ce qui concerne la réalisation d'un objet. D'où l'analyse suivante pour l'objet de *veut*, dans laquelle les contextes n'ont pas été indiqués, puisqu'ils seront utilisés et donc montrés plus bas (NP signifie *groupe nominal*).

[33] Cette possibilité n'est pas toujours contrôlée par des propriétés syntaxiques du lemme verbal, mais relève bien souvent de contraintes de sémantique lexicale, c'est-à-dire de restriction de sélection. Ainsi, le fait qu'un verbe puisse avoir ou non une infinitive comme objet direct est très souvent corrélé au fait qu'il puisse avoir ou non le nom déverbal correspondant comme objet direct, ce qui montre que la sémantique de l'objet est plus pertinente que sa réalisation.

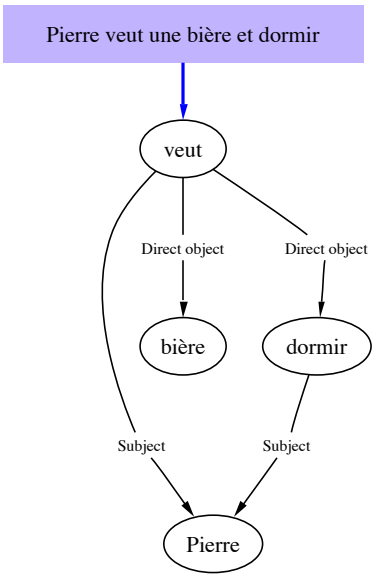


FIG. 2 – Vue en dépendances de l'analyse de la phrase (RCG_3).

OBJECT(*une bière et dormir*$_{2..6}$, *veut*$^0_{1..2}$)	$\rightarrow$	OBJ_SEM(*bière*$^0_{3..4}$, *veut*$^0_{1..2}$)
		OBJ_SEM(*dormir*$^0_{5..6}$, *veut*$^0_{1..2}$)
		NP(*une bière*^{0} *et dormir*$^0_{2..6}$) .
OBJ_SEM(*bière*$^0_{3..4}$, *veut*$^0_{1..2}$)	$\rightarrow$	NOUN(*bière*$^0_{3..4}$) .
OBJ_SEM(*dormir*$^0_{5..6}$, *veut*$^0_{1..2}$)	$\rightarrow$	*[Voir plus bas]*
NP(*une bière*^{0} *et dormir*$^0_{2..6}$)	$\rightarrow$	NP(*une bière*$^0_{2..4}$)
		NP(*dormir*$^0_{5..6}$)
		COORD(*et*$^0_{4..5}$)
		[prédicats de traitement
		des attributs] .
NP(*une bière*$^0_{2..4}$)	$\rightarrow$	NG(*une bière*$^0_{2..4}$) .
NG(*une bière*$^0_{2..4}$)	$\rightarrow$	*[Analyse standard d'un groupe nominal]*
NP(*dormir*$^0_{5..6}$)	$\rightarrow$	INFINITIVE(*dormir*$^0_{5..6}$) .
INFINITIVE(*dormir*$^0_{5..6}$)	$\rightarrow$	*[Voir plus bas]*

Dans notre grammaire, tout syntagme nominal peut potentiellement être une infinitive. Le fait que *veut* est un verbe à contrôle sujet n'a en réalité qu'un seul impact : il permet au prédicat OBJ_SEM(*dormir*$^0_{5..6}$, *veut*$^0_{1..2}$) d'être vrai. Ainsi, le syntagme nominal qui est l'objet de *veut*^{0} peut effectivement être une infinitive. L'analyse de cette infinitive, qui est ici *dormir*, utilise l'élément contextuel Subj de la façon suivante. De même que pour l'analyse de la phrase étudiée précédemment, l'analyse du sujet, ici *Pierre*$^0_{0..1}$, empile ce syntagme dans l'élément

contextuel Subj (voir la PHRASE2-clause dans l'analyse de la phrase précédente). Le prédicat INFINITIVE, qui hérite de cet élément contextuel via successivement VP, VK, ROLES, OBJECT et NP, l'utilise pour former la dépendance syntaxico-sémantique grâce à la clause suivante (où VK_INF signifie *noyau verbal d'une infinitive*, et où le contexte de INFINITIVE n'est pas montré) :

INFINITIVE($\underline{dormir}^{0}_{5..6}$){Subj=$\underline{Pierre}^{0}_{0..1}$ Subj.gender=masc Subj.number=sing}
$\rightarrow$ VK_INF($\underline{dormir}^{0}_{5..6}$, $Pierre^{0}_{0..1}$)[gender=masc number=sing]

Pour illustrer, nous donnons figure 2 le graphe de dépendances extrait automatiquement de l'analyse complète de la phrase.

Chapitre 11

Mise en œuvre de systèmes d'analyse complets

Un système d'analyse est une architecture logicielle permettant d'effectuer l'analyse automatique de textes bruts. Naturellement, une telle architecture est centrée autour d'un analyseur, produit à l'aide d'un générateur d'analyseur appliqué à une grammaire, et secondé d'un lexique. Cependant, la réalité des corpus que l'on peut être amené à analyser est parfois loin de la langue représentée par la grammaire et le lexique que l'on utilise. Malgré les techniques de robustesse incluses dans les analyseurs, il est par conséquent indispensable de savoir mettre en œuvre deux composants supplémentaires pour construire un véritable système d'analyse :

- une chaîne de traitement pré-syntaxique dont le rôle est (entre autres) de découper le texte brut en phrases (segmentation) et les phrases en mots (tokenisation), d'effectuer les corrections orthographiques nécessaires et de reconnaître les entités nommées, c'est-à-dire (pour nous) toutes les suites de symboles qui se construisent de façon productive, ne peuvent être analysées de façon compositionnelle par la grammaire générale, et peuvent être considérés comme un seul « mot » par l'analyseur (ainsi les adresses, les nombres, les dates, etc.) ; toutes ces opérations peuvent (et parfois doivent) être effectuées de façon non-déterministe ;
- un gestionnaire global dont le rôle est à la fois d'articuler les éléments entre eux mais aussi de mettre en œuvre des méthodes de contrôle et de rattrapage globales : délais maximaux, gestion de grappes de machines, génération d'un journal des événements, analyse à l'aide d'un second analyseur en cas d'échec du premier, analyse à l'aide de versions dégradés de l'analyseur standard, etc.

Nous avons développé de tels outils. La suite de ce chapitre présente ainsi notre chaîne de traitement pré-syntaxique, nommée SXPipe, et notre environnement d'analyse.

1 Traitements pré-syntaxiques

Le pré-traitement de texte brut avant l'analyse syntaxique est souvent considéré comme une tâche aisée et peu intéressante. Cependant, l'expérience montre que cette étape est cruciale lorsque l'on traite des corpus réels et que les outils disponibles ne sont pas toujours satisfaisants, par exemple parce qu'ils ne disposent pas d'un module de correction orthographique ou parce qu'ils ne gèrent pas (ou mal) les ambiguïtés.

Lorsque l'on est amené à analyser des corpus de qualité variées, comme cela a été le cas pendant la campagne d'évaluation EASy (voir chapitre 12), on est face à des textes bruts dont la qualité varie de correcte à très faible. Ainsi, il est indispensable de développer un système de pré-traitement très robuste pour segmenter en phrases et en mots ces textes parfois extrêmement bruités avec une perte minimale d'information et sans perdre le lien entre les formes de sortie[1] et les tokens originels du corpus[2].

Nous présentons tout d'abord une vue d'ensemble de l'architecture de notre chaîne SXPipe, décrite dans Sagot et Boullier (2005). Puis nous décrivons les différents composants, à savoir plusieurs étapes de reconnaissance d'entités nommées, deux étapes de segmentation (en phrases et en mots) et de correction orthographique, et une étape non-déterministe d'identification de multi-mots et de correction d'erreurs d'accentuation et de majuscule, qui produit le résultat final, à savoir un DAG de formes. Nous terminons par une rapide évaluation de notre système.

1.1 Architecture globale

L'architecture globale de notre système de traitement pré-syntaxique SXPipe est illustrée par la figure 1. Pendant tout le processus, les tokens d'entrée sont conservés dans des *commentaires*

[1] Rappelons ici les définitions données au chapitre 2. Nous employons *forme* comme une traduction de l'anglais *word-form* au sens de Clément et de La Clergerie (2004), et le mot *token* pour désigner, comme dans Clément et de La Clergerie (2004), les séquences de caractères présents dans le corpus et séparés par des espaces ou des signes de ponctuation clairement identifiés. Enfin, nous appelons *mot* une forme simple ou un composant de mot composé. Ainsi, *a priori* comporte deux mots mais constitue une seule forme, *aux* est un seul token et un seul mot mais deux formes, et *donne-moi* est un seul token, mais deux mots et deux formes. Une forme composée de plusieurs mots est appelée *multi-mot*.

[2] Ceci est indispensable, par exemple dans le contexte d'une campagne d'évaluation, mais aussi si l'on veut comparer le résultat de plusieurs analyseurs, pour pouvoir relier la sortie des analyseurs aux tokens du corpus. En effet, une forme peut couvrir plusieurs tokens, et un token plusieurs formes.

(entre accolades et complétés par leur position dans la chaîne d'entrée) qui sont immédiatement suivis du mot associé[3]. Par exemple[4],

contactez-moi␣au␣1␣av.␣Foch,␣75016␣Paris,␣ou␣par␣e-mail␣à␣my.name@my-email.com.

deviendra, si l'on laisse de côté les ambiguïtés[5]

{contactez$_{0..1}$} contactez {-moi$_{1..2}$} moi {au$_{2..3}$} à {au$_{2..3}$} le {1 av. Foch, 75016 Paris$_{3..9}$} _ADDRESS {,$_{9..10}$} , {ou$_{10..11}$} ou {par$_{11..12}$} par {e-mail$_{12..13}$} e-mail {à$_{13..14}$} à {my.name@my-email.com$_{14..15}$} _EMAIL {.$_{15..16}$} . {.$_{15..16}$} _SENT_BOUND.

1.2 Détection des frontières de phrases et reconnaissance d'entités nommées.

Les corpus réels ne sont pas comme les phrases de linguistes. Ils comportent des séquences de tokens qui ne sont pas analysables, ni morphologiquement ni syntaxiquement, mais procèdent de patrons productifs. Ce qui veut dire qu'elles doivent être reconnues avant la phase de correction orthographique. La plupart d'entre elles sont regroupées sous le terme d'*entités nommées* Maynard *et al.* (2001). Cependant, nous utiliserons ce terme dans un sens légèrement plus large, en y incluant toutes les séquences de tokens de ce type, y compris celles qui ne sont généralement pas considérées comme des entités nommées (par exemple les nombres). Nous appelons *grammaire locale* une grammaire qui reconnaît une certaine famille d'entités nommées.

Nous avons développé un ensemble de grammaires locales robustes[6], implémentées sous la forme de programmes *perl* mettant en jeu un grand nombre d'expressions régulières.

Certaines entités nommées contiennent des caractères qui sont habituellement des marques de ponctuation, en particulier le point (par exemple dans les URL), mais aussi la virgule (dans les adresses) ou toute sorte d'autres caractères (par exemple dans les smileys). C'est pourquoi certaines grammaires locales doivent être appliquées avant la segmentation. Dans notre système, ceci concerne les classes suivantes d'entités nommées :

adresses e-mail avec détection des espaces erronés,

URL avec détection de nombreux cas d'erreurs et de nombreux formats,

[3]Nous utilisons les conventions suivantes : un mot artificiel (par exemple un identifiant d'entité nommée) commence par un « _ » ; dans le corpus, les caractères « _ », « *{* » et « *}* » sont remplacés par les mots artificiels *_UNDERSCORE*, *_O_BRACE* et *_C_BRACE*, qui sont donc des entrées du lexique. Ainsi, ces trois caractères sont disponibles comme méta-caractères.

[4]Dans cet article, le symbole « ␣ » représente de manière plus visible une espace, et donc une frontière de tokens ou de mots.

[5]On notera que le même token peut être utilisé plusieurs fois de suite, pour gérer les agglutinées (ainsi *au* $_{2..3}$). Par ailleurs, le token spécial *_SENT_BOUND* sera expliqué plus tard.

[6]Par robuste, nous voulons dire que les entités nommées comportant des erreurs sont également reconnues, comme par exemple *ttp␣ :/url.avec.erreurs.com␣/index.html*.

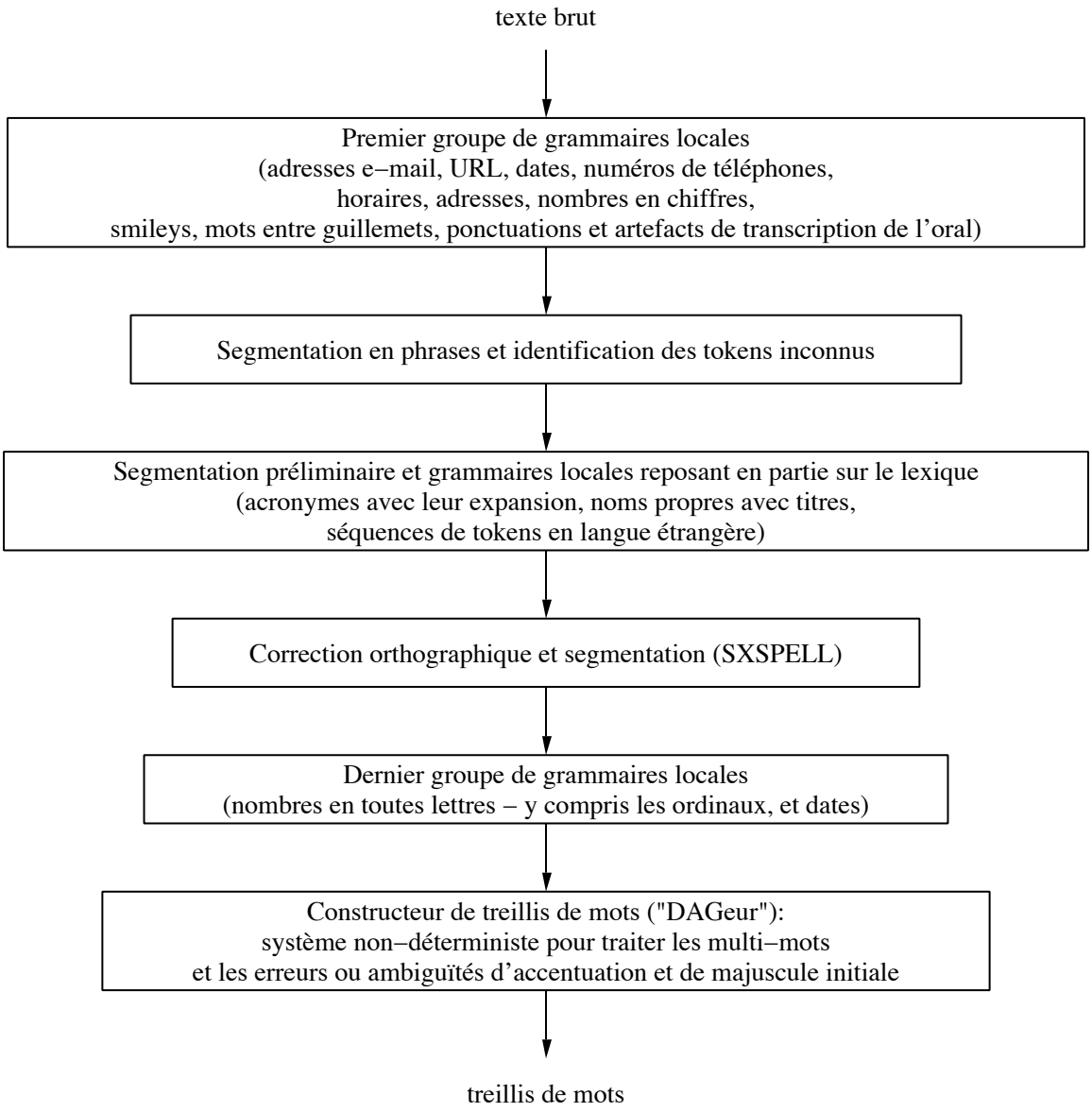


FIG. 1 – Architecture globale du système.

dates sous différents formats ainsi que les intervalles de dates (par exemple, *du 29 au 31 janvier* devient *du _DATE au _DATE*, alors que *29*, lorsqu'il est isolé, n'est reconnu que comme un nombre),

numéros de téléphone dans divers formats,

horaires dans de nombreux formats ainsi que les intervalles horaires (par exemple, *2-3 heures*, *3 ou 4 minutes*, etc.),

adresses postales dans de nombreux formats différents,

nombres en chiffres sous différents formats, ainsi que les ordinaux comportant des chiffres (par exemple *2ème*),

smileys tels que *:-)* ou *:D*,

mots entre guillemets : *un «test»* devient *un {«test»} test*,

artefacts de formatage pour traiter divers phénomènes spéciaux de ponctuation (comme remplacer *(␣...␣)* par un mot unique *(...)*) et les artefacts provenant de la transcription de l'oral (les répétitions d'un mot plus de deux fois, ou plus d'une fois s'il appartient à une liste pré-définie, ou encore les marqueurs d'hésitation comme *euh*, etc.).

Après avoir appliqué ces grammaires locales, nous segmentons le texte en phrases. Cette tâche est réalisée par un très gros ensemble d'expressions régulières *perl* qui étendent les idées simples proposées par exemple par Grefenstette et Tapanainen (1994), et qui prennent en compte une liste de mots connus comportant un point (souvent des abréviations). Nous sommes capables de traiter correctement toutes sortes de faux positifs et de faux négatifs qui apparaissent inévitablement dans un corpus réel. À l'issue de cette étape, le mot artificiel *_SENT_BOUND* représente les frontières de phrase.

Nous appliquons alors le tokeniseur et correcteur orthographique décrit dans la partie suivante dans un mode dégradé, en ce sens qu'aucune correction orthographique n'est faite, mais que le texte est tokenisé (découpé en mots) de la même façon qu'il le serait si les corrections étaient faites. Le but est d'identifier les tokens de la chaîne d'entrée qui ne peuvent pas être analysés comme des mots connus (c'est-à-dire présents dans le lexique) ou des combinaisons de mots connus (comme *anticommuniste-né*, *donne-m'en* ou *scénario-catastrophe*).

Une fois les mots inconnus identifiés (au sens du paragraphe précédent), interviennent alors des grammaires locales spéciales qui prennent en compte cette information supplémentaire. Elles reconnaissent les classes suivantes d'entités nommées :

acronymes suivis ou précédés de leur expansion, avec diverses variantes typographiques possibles,

noms propres précédés par un titre (comme *Dr.* ou *Mr*),

séquences en langue étrangère autre que le français.

Les deux dernières grammaires locales méritent plus d'explication. Elles reposent sur la technique suivante. Soit $w_1 \dots w_n$ une phrase dont les mots sont les w_i. Nous définissons une fonction d'étiquetage t qui associe (grâce à des expressions régulières) une étiquette $t_i = t(w_i)$ à chaque mot w_i, où les t_i sont pris dans un petit ensemble fini d'étiquettes possibles (respectivement 9 et 12 pour les deux grammaires locales concernées). Ainsi, une séquence d'étiquettes $t_1 \dots t_n$ est associée à $w_1 \dots w_n$. Ensuite, un (gros) ensemble de transducteurs finis transforme $t_1 \dots t_n$ en une nouvelle séquence d'étiquettes $t'_1 \dots t'_n$. Si dans cette dernière la sous-séquence $t'_i \dots t'_j$ correspond à un certain patron, la séquence de mots correspondante $w_i \dots w_j$ est considérée comme reconnue par la grammaire locale.

Considérons par exemple l'énoncé

Peu␣après␣,␣le␣Center␣for␣irish␣Studies␣publiait . . .,

où *Center*, *irish* et *Studies* ont été identifiés comme mots inconnus. On associe à cet énoncé les étiquettes suivantes : `cnpNEEucn`...(`c` correspond à *initiale en majuscule*, `n` à *probablement français* (cas par défaut), `p` à *ponctuation*, `N` à *connu comme français*, `E` à *connu comme étranger* et `u` à *inconnu*). Des expressions régulières sur ces étiquettes mènent à `cnpNeeeen`..., où *e* correspond à *étranger*, ce qui signifie que *Center␣for␣irish␣Studies* est reconnu comme une séquence en langue étrangère[7]. La phrase devient alors (*_ETR* correspond à *séquence en langue étrangère*) :

Peu␣après␣,␣le␣{Center for irish Studies} _ETR␣publiait . . .

1.3 Découpage en mots et correction orthographique

1.3.1 Un correcteur de mots isolés : SXSpell

La prochaine étape dans notre chaîne de traitement est le correcteur orthographique. Les corpus réels ont des taux variables de fautes d'orthographe, qui vont de quasiment zéro (p. ex. dans les corpus littéraires) à des taux très élevés (p. ex. dans les corpus de courrier électronique). De plus, s'ils restent non corrigés, les mots mal orthographiés deviennent des mots inconnus pour l'analyseur. Ceci doit être évité au maximum, car les mots inconnus se voient attribuer des informations syntaxiques par défaut qui conduisent à la fois à une faible précision et à une très forte ambiguïté au niveau lexical qui se répercute au niveau syntaxique. C'est pourquoi nous avons développé un correcteur orthographique, nommé SXSpell, en collaboration avec Pierre Boullier.

[7] En fait, nous avons aussi développé un outil prototype d'identification de la langue pour de telles séquences. Dans cet exemple, la réponse correcte — anglais — est trouvée.

De nombreux travaux ont déjà été réalisés dans le domaine de la correction orthographique (cf. par exemple la synthèse de Kukich (1992)). Les techniques de correction des mots pris isolément se regroupent principalement en deux catégories : les techniques avec entraînement et les techniques sans entraînement. Les principales techniques avec entraînement sont les techniques stochastiques (souvent fondées sur les n-grams) et les réseaux de neurones. Les techniques sans entraînement reposent sur la *distance minimum de correction* (pour *minimum edit distance*, et fondées sur des opérations telles que l'insertion, la suppression, le remplacement et l'interversion) ou sur des *règles* (fondées sur des règles de réécriture qui peuvent être dépendantes du contexte et dont l'origine vient de la phonologie à états finis). Ces dernières sont clairement plus puissantes et mieux adaptées à la tâche[8], mais les opérations citées peuvent également être utiles en tant que telles. Ainsi, notre correcteur est un correcteur à règles, mais ces opérations sont également disponibles pour écrire des règles sous-spécifiées.

L'application d'une règle est appelée *correction élémentaire*. Nous associons à chaque règle un *poids local* et un *poids de composition*. Le coût total d'une correction est la somme des coûts locaux de toutes les corrections élémentaires, à laquelle on ajoute la somme des coûts de composition si plus d'une correction élémentaire a été appliquée. Ceci permet d'avoir un coût total plus élevé que la somme des coûts locaux. La meilleure correction est alors celle de coût minimum.

Notre objectif était de disposer d'une implémentation efficace de ces techniques simples même en utilisant des règles réalistes et donc nombreuses et un lexique orthographique construit à partir du Le*fff*[9] couvrant (notre lexique orthographique pour le français a plus de 400 000 mots). Pour atteindre cet objectif, nous considérons le lexique orthographique comme un automate à états finis déterministe $\mathcal{F}$, le mot d'entrée comme un transducteur fini $\mathcal{T}_w^0$, et les règles de réécriture comme des transducteurs fini $\mathcal{T}^i (i > 0)$. Tout d'abord, nous calculons le transducteur fini $\mathcal{T}_w^{all}$ de toutes les séquences possibles de caractères qui peuvent être obtenues à partir de w par l'application des règles, ainsi que leur coût[10]. Puis nous extrayons de $\mathcal{T}_w^{all}$ tous les mots qui existent effectivement dans le lexique, en calculant l'intersection de $\mathcal{F}$ avec $\mathcal{T}_w^{all}$.

[8] Un exemple simple en est le fait suivant. Le son [o] pouvant s'écrire (entre autres) *o* ou *eau*, il est raisonnable de savoir corriger l'un en l'autre. Il est alors plus naturel et plus approprié par rapport aux calculs de coût de correction de voir cette opération comme un remplacement de *eau* par *o* que de la considérer comme la succession de deux suppressions et d'un remplacement.

[9] Nous appelons *lexique orthographique* un lexique contenant tous les mots connus, c'est-à-dire tous les mots simples (qui sont donc des formes) et tous les composants de multi-mots. Il s'oppose au *lexique morphologique* qui associe à chaque forme (mot simple ou multi-mot) des informations morphologiques. Ainsi, *instar* est une entrée du lexique orthographique, mais pas du lexique syntaxique. Ce dernier comprend en revanche une entrée pour *à l'instar de*.

[10] Bien sûr, un seuil peut être donné en paramètre pour empêcher le calcul de corrections déraisonnablement coûteuses.

La difficulté de cette approche n'est pas la théorie sous-jacente, qui est bien connue, mais vient de la taille des automates à manipuler. En effet, avec un nombre de règles de l'ordre de quelques centaines, l'automate $\mathcal{T}_w^{all}$ a facilement des milliards et des milliards de chemins. Et il doit être intersecté avec $\mathcal{F}$ et ses 400 000 chemins. Pour cette raison, nous avons utilisé de manière intensive des techniques de tabulation et de représentation compacte. Il faut admettre que la faisabilité d'une telle approche n'était pas claire *a priori*, mais nous obtenons de très bons résultats, à la fois en terme de qualité (avec des règles adéquates) et de temps de réponse (avec un coût-seuil adéquat).

1.3.2 Correction orthographique en contexte

La correction orthographique ne peut être effectuée exclusivement au niveau du mot. En effet, au moins quatre phénomènes impliquent l'environnement d'un mot pendant son identification comme élément du lexique ou pendant sa correction :

- les mots commençant par une majuscule,
- les mots en position initiale dans une phrase (ce qui interagit fortement avec le point précédent),
- les multi-mots qui sont la conséquence de morphologie dérivationnelle productive (par exemple *anti-Bush*) ou de l'agglutination syntaxique (par exemple *préchoisis-t'en* qui doit être segmenté en *pré-_ choisis t' en*, où « -_ » est par convention la marque des préfixes),
- les fautes d'orthographe (ou typographiques) concernant plusieurs tokens (ainsi *corre ction* au lieu de *correction*) ou plus d'un mot (par exemple *unproblème* au lieu de *un problème*).

Ainsi, nous avons développé, en collaboration avec Pierre Boullier, un correcteur orthographique en contexte qui sait prendre en compte ces phénomènes et envoyer des requêtes au correcteur de mots isolés de SXSpell, afin de segmenter en mot et de corriger le texte simultanément (nous ne corrigeons pas les mots à initiale majuscule, mais d'autres mots inconnus peuvent rester si aucune correction n'est trouvée dont le coût soit inférieur à un certain seuil). Il s'est avéré qu'il n'est pas facile de gérer l'interaction entre la segmentation des multi-mots, les phénomènes de majuscule, et les fautes d'orthographe, en particulier lorsque l'on traite le premier mot d'une phrase. Cependant, nous avons défini de nombreuses heuristiques qui donnent des résultats très satisfaisants, dont un aperçu est donné dans les algorithmes 2 et 3.

[Dans cet algorithme, « correction » signifie « appel à l'heuristique intra-mot » décrite dans l'algorithme 3]
$p = pile de corrections en attente (initialement vide) ;
Tant que (Le mot courant est non-vide) **faire**
  $w = mot à traiter ;
  $c = commentaire associé à $w ;
  Si ($w = /^ETR.*/ **ET** $c non vide) **Alors**
    Essayer de corriger la concaténation des tokens originels ;
    [« {disp aru} _ETR » doit devenir « {disp aru} disparu »]
    Si (Cette correction est un succès) **Alors**
      Rendre le résultat et passer au mot suivant ;
    Fin Si
  Fin Si
  Essayer de corriger $w ;
  Si ($w commence par « _ » **ET** sa correction a réussi) **Alors**
    Vider $p en en rendant le contenu
    Rendre la correction de $w et passer au mot suivant
  Fin Si
  Si ($p est vide) **Alors**
    On empile la correction de $w dans $p
  Sinon
    Si (le mot précédent s'est corrigé à coût nul **ET** $w aussi) **Alors**
      Vider $p (qui ne contient que le mot précédent) en en rendant le contenu
      Empiler la correction de $w dans $p
    Sinon
      Essayer de corriger la concaténation du token originel précédent et de $w
      Si (Échec **OU** Succès de coût supérieur à la somme des coûts des corrections des 2 tokens) **Alors**
        Vider $p (qui ne contient que le mot précédent) en en rendant le contenu
        Empiler la correction de $w dans $p
      Sinon
        Vider $p (qui ne contient que le mot précédent)
        Empiler la correction de cette concaténation
      Fin Si
    Fin Si
  Fin Si
Fait
Vider $p en en rendant le contenu

Algorithme 2: Heuristique de correction orthographique inter-mots pour SXSpell.

```
$w = le mot à corriger ;
Si ($w est dans le lexique) Alors
    | Rendre $w et sortir
Fin Si
Si ($w peut être corrigé par correction « légère »  ET $w ne comporte aucune majuscule)
Alors
    | Rendre cette correction « légère » et sortir
Fin Si
Si ($w commence par une majuscule  ET l'équivalent en minuscules de $w est dans le
lexique) Alors
    | Rendre cette correction « légère » et sortir
Fin Si
Si ($w est le premier mot de la phrase  ET $n commence par une majuscule) Alors
    | Découper l'équivalent en minuscules de $w en pré-noyau/noyau/post-noyau (voir
    | algorithme 4)
Sinon
    | Découper $w en pré-noyau/noyau/post-noyau
Fin Si
Si (Le noyau est vide (succès)) Alors
    | Rendre le pré-noyau et le post-noyau, et sortir
Fin Si
[Le noyau $n est donc non-vide]
Si ($n commence au début de la phrase  ET $n commence par une majuscule) Alors
    | Découper l'équivalent en minuscules de $n en préfixes/cœur/suffixes (voir algo-
    | rithme 5)
Sinon
    | Découper $n en préfixes/cœur/suffixes
Fin Si
Si (Le cœur obtenu est dans le lexique) Alors
    | Rendre le pré-noyau, les préfixes, le cœur, les suffixes et le post-noyau, et sortir
Fin Si
Si ($n a au moins une majuscule) Alors
    | Rendre le pré-noyau, $n et le post-noyau, et sortir
Sinon
    | Essayer une correction « légère » de $n ;
    | Si (Si c'est un succès) Alors
    |     | Rendre le pré-noyau, cette correction et le post-noyau, et sortir
    | Sinon
    |     | Essayer une correction standard de $n ;
    |     | Si (Si c'est un succès) Alors
    |     |     | Rendre le pré-noyau, cette correction et le post-noyau, et sortir
    |     | Sinon
    |     |     | Si ($w comporte au moins une majuscule) Alors
    |     |     |     | Rendre _Uw et sortir
    |     |     | Sinon
    |     |     |     | Rendre _uw et sortir
    |     |     | Fin Si
    |     | Fin Si
    | Fin Si
Fin Si
```

Algorithme 3: Heuristique de correction orthographique intra-mot pour SxSpell

$w = le mot (inconnu du lexique) dont on cherche à détacher les mots agglutinés par des tirets ou des apostrophes ;
Si ($w commence par un tiret) **Alors**
  On supprime ce tiret ;
  Si (Le résultat est dans le lexique) **Alors**
    Rendre ce résultat
  Fin Si
Fin Si
Chercher de gauche à droite le premier tiret ou apostrophe, nommé $t ;
Si ($t est un tiret) **Alors**
  On coupe $w juste avant en $w1 et $w2 (découpage-gauche et découpage-droit)
Sinon
  On coupe $w juste après en $w1 et $w2 (idem)
Fin Si
Si ($w1 est dans le lexique en tant que mot mais pas en tant que préfixe) **Alors**
  Si (Tous les découpages ont conduit jusqu'à présent à des découpages-gauche corrects) **Alors**
    On empile $w1 dans une pile des découpages-gauches
    Si ($t est une apostrophe **ET** $w2 commence par une majuscule) **Alors**
      Rendre le contenu de la pile des découpages-gauches et $w2, et sortir
    Fin Si
  Sinon
    Ce découpage-gauche est incorrect
  Fin Si
Sinon
  Ce découpage-gauche est incorrect
Fin Si
Si ($w2 (qui commence donc par un tiret si $t est un tiret) est dans le lexique en tant que mot mais pas en tant que suffixe) **Alors**
  On empile $w2 dans une pile des découpages-droits
  Si ($w1 est dans le lexique en tant que mot mais pas en tant que préfixe **ET** $w1 n'a pas encore été empilé) **Alors**
    On empile $w1 dans une pile des découpages-gauches
  Fin Si
Fin Si
On appelle cet algorithme sur $w2 en se souvenant si les découpages ont conduit jusqu'à présent à des découpages-gauche corrects ;
Si (C'est un succès (le noyau rendu est vide)) **Alors**
  Si (Tous les découpages ont conduit jusqu'à présent à des découpages-gauche corrects **ET** $w1 n'a pas encore été empilé) **Alors**
    On empile $w1 dans la table des suffixes
  Fin Si
Fin Si
Le noyau $n est initialisé à la chaîne vide ;
Si ($w1 n'a pas été empilé) **Alors** Ajouter $w1 à $n **Fin Si**
Si ($w2 n'a pas été empilé) **Alors** Ajouter $w2 à $n **Fin Si**
Rendre le contenu de la pile des découpages-droits, le noyau $n et le contenu de la pile des découpages-gauches, et sortir.

Algorithme 4: Heuristique d'identification des mots agglutinés SxSpell.

```
$n = le noyau sur lequel on cherche à détacher préfixes et suffixes ;
Tant que ($n se termine par un suffixe présent dans le lexique des suffixes) faire
    Retirer ce suffixe de $n
    Effectuer sur ce suffixe l'opération qui lui est associée (typiquement, le faire précéder
    de « _ »)
    BOUCLE :
    Tant que ($n est non vide) faire
        Si ($n est dans le lexique) Alors
            Aller à SÉQUENCE FINALE
        Fin Si
        Si ($n commence par un préfixe présent dans le lexique des préfixes) Alors
            -
        Fin Si
        Pour i de 1 à 1 + Nombre de tirets dans $n faire
            Noter $p ce qui est à gauche du i-ième tiret le plus à droite (tout $n si$i
            = 1 + Nombre de tirets dans $n) ;
            Si ($n ne commence pas par un préfixe  ET $p ne comporte pas de tiret
            ni de majuscule  ET $n n'est pas en début de phrase) Alors
                Essayer une correction « légère » de $p ;
                Si (Cette correction à marché) Alors
                    Aller à DÉCOLLAGE
                Fin Si
            Fin Si
        Fin Pour
    Fait
    DÉCOLLAGE :
    Si ($p (éventuellement corrigé) est un préfixe) Alors
        Aller à SÉQUENCE FINALE
    Fin Si
    Essayer d'effectuer sur le préfixe $p l'opération qui lui est associée (rien, lui ajouter
    un tiret, ou autre) ;
    [Cela peut échouer, par exemple si on essaye de détacher un préfixe comme im-, qui
    spécifie qu'il doit précéder un m, un b ou un p, d'un radical qui commence par une
    autre lettre]
    Si (C'est un échec  OU le reste du mot commence par une majuscule  OU $p-tiret-
    reste du mot est un mot du lexique) Alors
        Aller à SÉQUENCE FINALE
    Sinon
        Empiler $p dans la pile des préfixes
    Fin Si
Fait
SÉQUENCE FINALE :
Rendre le contenu de la pile des préfixes, $n s'il est non-vide, et le contenu de la pile des
suffixes.
```

Algorithme 5: Heuristique d'identification des affixes.

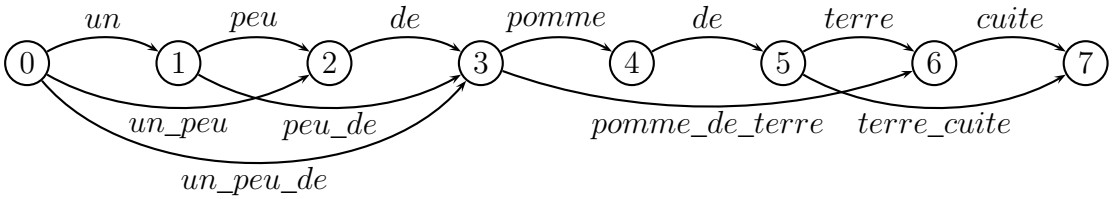


FIG. 2 – DAG associé à *un_peu_de_pomme_de_terre_cuite*.

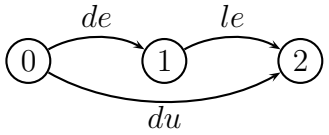


FIG. 3 – DAG associé à *du*.

1.4 Correction orthographique légère et construction du DAG de formes

Dans de nombreux cas, une simple énumérations linéaire de formes ne peut pas exprimer les subtilités et les ambiguïtés des langues naturelles : il n'y a pas bijection entre tokens et formes. Pour cette raison, la sortie de notre chaîne est un treillis (DAG) de formes, qui peut être donné en entrée de nos analyseurs syntaxiques[11]. De plus, nous ne produisons pas uniquement des DAG simples au sens de Barthélemy *et al.* (2001), car ils sont insuffisants (cf. par exemple la figure 2).

Considérons l'expression *un_peu_de_pomme_de_terre_cuite*. Chaque mot est une forme fléchie valide, mais *un_peu*, *peu_de*, *un_peu_de*, *pomme_de_terre* et *terre_cuite* également. Ainsi, elle est représentée par le DAG de la figure 2.

À l'inverse, le français (comme d'autres langues) comporte également des *agglutinées*. Par exemple, *du* peut être une forme fléchie (l'article partitif) ou l'agglutination de *de_le*. Il est ainsi représenté par le DAG de la figure 3.

Ces opérations sont réalisées comme suit. L'entrée de l'étape de « DAGage » est considérée comme un DAG (linéaire) $\mathcal{D}$. À chaque multi-mot et à chaque agglutinée du lexique est associé un transducteur. La composition de tous ces transducteurs est appliquée à $\mathcal{D}$, créant ainsi éventuellement de nouveaux chemins.

Le DAG résultant est alors passé à travers d'autres transducteurs qui créent d'autres alternatives. Par exemple, les mots à initiale majuscule pour lesquels la variante en minuscule est présente dans le lexique sont représentés comme une alternative entre les deux formes corres-

[11]La plupart des analyseurs syntaxiques classiques ne peuvent pas gérer les DAG en entrée, ce qui rend nécessaire une étape supplémentaire avant l'analyse, à savoir le (super-/hyper-)tagging (ou *étiquetage*), qui peut supprimer des possibilités correctes.

Classe d'entités nommées	Occurrences	Précision	Rappel
URL	174	100%	100%
adresses (physiques)	35	100%	100%
Expressions en langue étrangère[15]	42	83%	88%

TAB. 1 – Évaluation partielle de la reconnaissance d'entités nommées.

pondantes. Les mots inconnus restant à ce stade (y compris de nombreux mots à initiale majuscule) et pour lesquels l'ajout d'un diacritique sur certaines lettres conduit à une forme présente dans le lexique sont également représentés comme une alternative entre les deux formes concernées[12,13]. Enfin, les mots inconnus restant dans le DAG sont tous remplacés par une des deux entrées spéciales du lexique, *_Uw* et *_uw*, selon que leur caractère initial est une majuscule ou une minuscule. Le DAG de formes résultant est la sortie finale.

1.5 Évaluation

L'évaluation d'un tel système est difficile car nous ne disposons pas d'un corpus de référence approprié. Cependant, on peut en avoir un aperçu grâce à des tests que nous avons menés sur un corpus journalistique de 1,1 million de mots. Tout le processus prend 13 minutes 01 seconde[14], soit environ 1400 tokens/sec. Étant donnée la complexité des tâches réalisées, et en particulier les tailles des automates en jeu dans SXSpell, c'est une très bonne performance.

Nous avons également sélectionné quelques entités nommées pour lesquelles des détecteurs sur-générateurs peuvent être facilement écrits afin de permettre une validation manuelle. Les résultats sont dans le tableau 1.

L'évaluation de la segmentation en phrases nécessite une annotation manuelle. Nous l'avons effectuée sur les 400 premières phrases du corpus, ce qui donne un taux de précision de 100% et un taux de rappel de 100%. C'est très satisfaisant, compte tenu du fait que ce corpus journalistique est rempli de citations, de notes de bas de page, de références bibliographiques et de méta-informations qui rendent la détection des frontières de phrases assez difficile.

L'évaluation du correcteur orthographique n'est pas chose facile. De fait, comme indiqué plus haut, la phase de correction orthographique et de segmentation en mots est réalisée par un

[12]Nous essayons aussi de corriger les composants de multi-mots qui n'existent pas isolément mais qui ne prennent pas part à leur multi-mot. Par exemple, *brac* n'existe que comme composant du multi-mot *bric à brac*. Ainsi, *un brac* n'a pas été corrigé précédemment, mais est corrigé ici en *un bras*.

[13]Ceci pourrait être fait même sur des mots connus du lexique, afin de gérer les cas où l'oubli d'un diacritique ou d'une consonne double conduit par hasard à un autre mot connu. Toutefois, nous n'avons pas mis en œuvre cette idée, pour éviter une trop grande ambiguïté des DAG obtenus.

[14]Test réalisé sur une architecture AMD Athlon XP 2100+ (1.7 GHz) sous Mandrake Linux 10.1.

[15]Test réalisé seulement sur 2000 phrases, car une annotation manuelle est nécessaire.

composant qui fait appel au correcteur SXSPELL tout en gérant les phénomènes de segmentation et de majuscules. Ce qui signifie qu'il y a deux sous-composants à évaluer : le correcteur SXSPELL et le segmenteur-correcteur qui l'utilise. De plus, il faut isoler les performances de ce composant des qualités du lexique et du corpus considéré.

Pour ce faire, nous avons identifié automatiquement parmi les 1,1 million de tokens tous ceux qui ne sont pas reconnus par le correcteur-segmenteur comme mots connus ou combinaisons valides de mots connus. Nous avons alors identifié parmi ces tokens inconnus ceux qui devraient être corrigés en des mots ou combinaisons de mots présents dans le lexique, et nous les avons corrigés manuellement (en tenant compte de leur contexte d'apparition). Puis nous avons comparé la correction manuelle de ces tokens à celle fournie par notre système. 91% des 150 tokens concernés sont corrigés (et éventuellement segmentés) correctement[16]. Quelques exemples sont indiqués dans les tableaux 2 et 3.

Token d'entrée	Correction
arisienne	*parisienne*
barriére	*barrière*
celuici	*celui-ci*
l'intervent␣jonnisme	*l'␣interventionnisme*
n'aspire-til	*n'␣aspire␣-t-il*
monde-tel-qu'il-est	*monde␣tel␣qu'␣il␣est*
plrrase	*phrase*
redou-table	*redoutable*

TAB. 2 – Exemples de corrections réussies effectuées par le correcteur-segmenteur.

Token d'entrée	Correction automatique	Correction manuelle
argurnent	*arguèrent*	*argument*
lls	*las*	*ils*
de'investissement	*dé␣investissement*	*de␣l'␣investissement*

TAB. 3 – Exemples de corrections erronées effectuées par le correcteur-segmenteur.

Par ailleurs, 1846 tokens sont analysés comme combinaison de mots du lexique avec (au moins) un préfixe (1712 cas) ou un suffixe (54 cas, seuls *-né*, *-clef* et leurs variantes étant concernés) connu. Par exemple, la séquence *quasi-parti␣unique␣chrétien-libéral-conservateur* est transformée en *quasi-_␣parti␣unique␣chrétien-_␣libéral-_␣conservateur*, où « -_ » est par convention la marque des préfixes.

[16]De plus, ce pourcentage correspond à une correction déterministe. SXSPELL peut rendre plusieurs propositions de correction. Dans ce cas, le pourcentage de tokens dont une des corrections proposées est la bonne serait encore supérieur. Ainsi, la bonne correction pour *lls*, à savoir *ils*, est également trouvée, et elle est de même coût que la correction proposée (*las*).

Il nous faut préciser à ce stade deux faits. Tout d'abord, le corpus considéré est de très bonne qualité (150 mots du français standard mal orthographiés parmi 1,1 million de mots). D'autre part, cette évaluation du correcteur-segmenteur nous a permis de réaliser l'incomplétude du lexique, en particulier en ce qui concerne les mots d'emprunt à des langues étrangères. Mais ce n'est pas notre propos ici d'évaluer notre lexique.

2 Architecture informatique pour l'analyse automatique

2.1 Le gestionnaire d'analyseurs

Une fois un corpus brut transformé en ensemble de DAG de formes du lexiques, à l'aide de la chaîne de traitement pré-syntaxique SXPipe, l'étape suivante est d'envoyer ces treillis à un ou plusieurs systèmes d'analyse. C'est la tâche du *gestionnaire d'analyseurs*. Celui que nous avons développé pour SXLFG, nommé `esxanalyse`, dispose d'un certain nombre de fonctionnalités, que l'on peut regrouper en trois familles principales :

- options du processus global d'analyse, appelé *campagne* (ou *run*),
- stratégies globales d'analyse,
- gestion de l'analyse de chaque phrase,
- comportements spéciaux
- options de sortie (génération de journaux d'événements, affichage, construction de résultats complets,...),
- divers.

L'ensemble des options disponibles est donné figure 4 à titre indicatif.

2.2 Journaux d'événements et graphes statistiques

Une fois un corpus analysé, il est souvent indispensable de disposer d'outils de visualisation des résultats, ainsi que des propriétés statistiques du processus d'analyse lui-même. Si de plus le corpus que l'on a analysé a été par ailleurs manuellement annoté, il est souhaitable de pouvoir évaluer la précision et le rappel des analyses produites, ainsi que de visualiser les analyses automatiques et manuelles de façon à pouvoir les comparer. Tous ces graphes et pages `html` de visualisation sont obtenus à partir des résultats de l'analyseur, mais également à partir de journaux d'événements (fichiers de *log*) produits directement ou indirectement par le générateur d'analyseurs.

Nous avons donc développé, en collaboration Éric de La Clergerie, tout un ensemble de scripts permettant la construction automatique de résultats et d'évaluations des analyses pro-

```
Options :
1. Run characteristics
-s=[filename]                       parse sentences of file [filename]
-s=[filename] :[line_number]        parse sentence number [line_number] of file [filename]
-rn=[runname] -run_name=[]          specify the name given to the run (default is
                                    wday_month_day_time_year)
-e                                  parse sentences whose ids are in the file of previously
                                    failed sentences
-e=[filename]                       parse sentences whose ids are in the specified file
-d                                  die on first warning, instead of just printing it
                                    on STDERR

2. Global parsing strategies
-o=[ordering_method]                parse sentences in a given order.
                                    Possible values :
                                    [ordering_method]='l' : by increasing length
                                    [ordering_method]='i' : by decreasing length

3. Sentence-level parsing options
-t(=[number])                       timout on analysis time, 5 sec if not specified,
                                    [number] seconds otherwise
-T(=[number])                       timout on segment analysis time, 5 sec if not specified,
                                    [number] seconds otherwise (if not specified, this
                                    timeout is the same as the global timeout set by -t)
-r -robust                          if defined, call the robust parser if the standard
                                    parser failed to give an analysis
-ns -no_segmentation                if no global analysis is found, do not over-segment
                                    input sentence and try to parse these segments
-nsus -no_skip_uw_sentences         do parse sentences containing only _Uw and _uw
                                    (they are skipped by default)
-st -segment_on_timeout             if analysis timed out, sent the sentence by segments
                                    (over-segmentation)
-cfg -cfg                           perform only CFG parsing with the support grammar

4. Special behaviours
-np -no_parse                       do not parse
-ls -list_sentences                 do not parse, but print to STDOUT the list of sentences
                                    to parse in the computed order
-nao -no_analysis_output            do not save analyses

5. Output options
-infos                              print to STDERR sythetic information about parses
-stderr                             redirect to STDERR error messages of the parser
-stdout                             redirect to STDERR the raw output of the parser
-nw -no_warnings                    do not print warnings on STDERR
-l -log                             generate log file
-f -fail -failure                   generate file of failed sentences
-b                                  build result files by concatenating analysis for all
                                    sentences of the same file

6. Miscanellous
-h -help                            print this
```

FIG. 4 – Options du gestionnaire d'analyseurs `esxanalyse`.

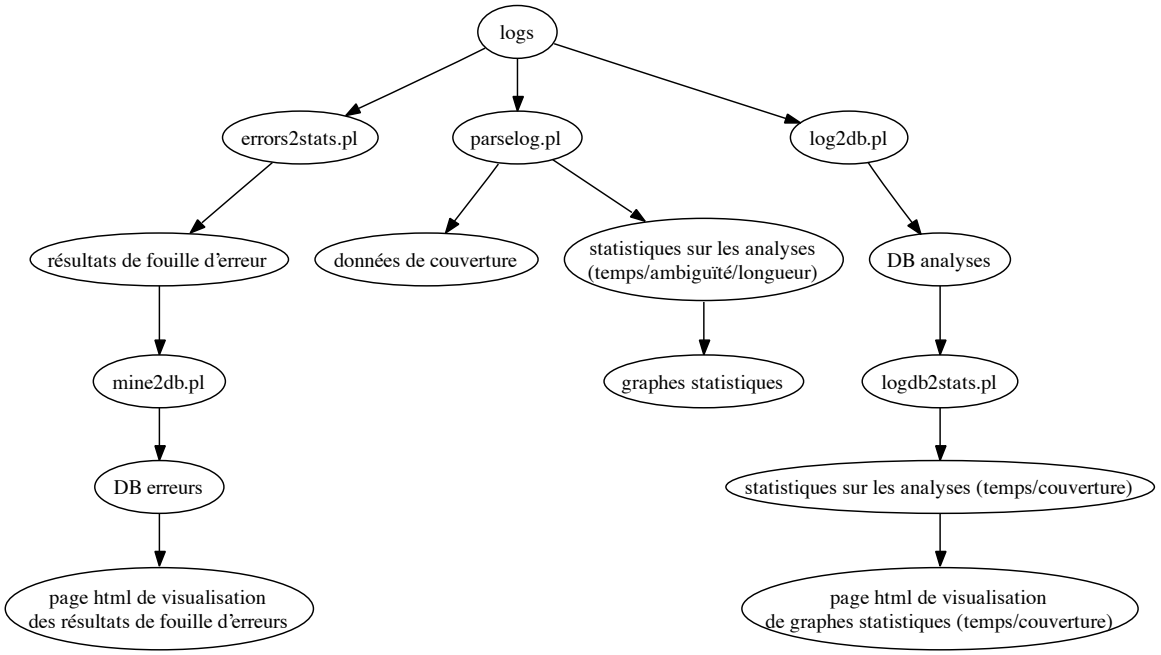


FIG. 5 – Exploitation des résultats d'une campagne d'analyse (journaux d'événements).

duites. Certains de ces scripts reposent sur le chargement des analyses obtenues dans des bases de données, via des requêtes à ces bases. Les figures 5 et 6 donne un aperçu des données que nous pouvons produire automatiquement, des scripts utilisés et des dépendances entre ces données.

3 Bilan

Nos expériences montrent l'importance cruciale des composants qui gravitent autour de l'analyseur proprement dit que sont la chaîne de traitement pré-syntaxique et le gestionnaire global du système d'analyse. C'est vrai en particulier dès que l'on cherche à analyser des corpus relativement éloignés du français écrit standard, comme les corpus de courrier électronique et les corpus de transcriptions d'oral, ou dès que l'on traite des corpus volumineux (plus que quelques centaines de phrases, et jusqu'à plusieurs millions). En effet, la robustesse qu'apportent pré-traitements et stratégies globales permet d'extraire des informations pertinentes de phrases dont la forme d'origine est parfois bien différente de ce que recouvrent le lexique et la grammaire.

Mais nos expériences montrent aussi que la mise en œuvre de systèmes d'analyse ne peut se faire sans une architecture informatique adaptée qui permet également l'étude des résultats et en particulier leur visualisation et leur exploitation. Ces problèmes, purement technologiques, ne

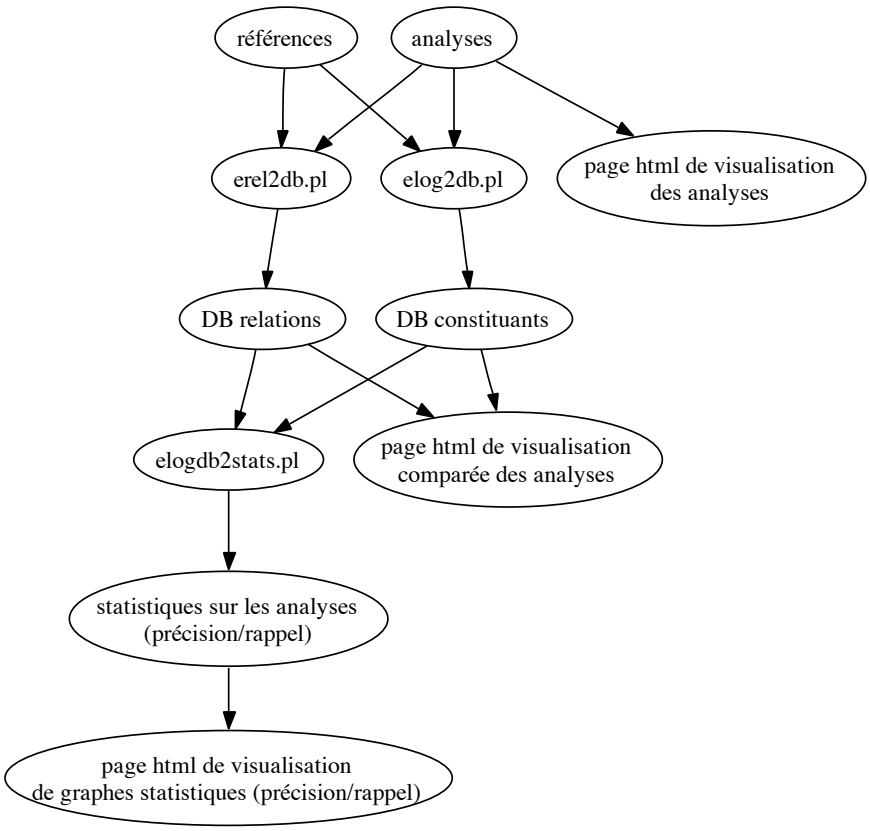


FIG. 6 – Exploitation des résultats d'une campagne d'analyse (analyses).

sont pas nécessairement les plus valorisants intellectuellement. Pourtant, leur prise en compte est indispensable.

Chapitre 12

Évaluation de systèmes d'analyse syntaxique

Le développement de formalismes, de grammaires, d'analyseurs, de lexiques, et de chaînes de traitements pré-syntaxiques n'a de sens que s'ils peuvent former un tout cohérent, avec une articulation aussi bonne que possible entre les différents composants, au sein de systèmes d'analyses complets. L'évaluation de ces composants passe dont *in fine* par l'évaluation des systèmes d'analyse dans lesquels ils prennent part. Cette évaluation est triple : elle doit refléter les performances du système en termes d'efficacité (temps d'analyse), de couverture (et donc de robustesse) et, si possible, de précision.

Nous avons effectué de telles validations à grande échelle avec diverses variantes de nos analyseurs. On peut les regrouper en cinq séries d'expériences :

- Analyse de gros corpus avec SXLFG et une grammaire LFG du français développée à partir de celle utilisée par Clément et Kinyon (2001) (nous appellerons SXLFG_L ce système d'analyse),
- Analyse d'un corpus moyen avec SXLFG_L avec et sans désambiguïsation interne,
- Campagne d'évaluation EASy avec SXLFG_L,
- Analyse de gros corpus (dont le corpus EASy) avec SXLFG et une nouvelle grammaire LFG du français (nous appellerons SXLFG_B ce système d'analyse),
- Évaluation du système-prototype Méta-RCG sur le corpus Eurotra Danlos et Laurens (1991).

Dans toutes ces expériences, les corpus sont traités par la chaîne de traitement pré-syntaxique SXPipe (voir chapitre 11) et le lexique utilisé est le Le*ffff* 2 décrit plus haut (voir chapitre 7).

1 Analyse de gros corpus (système SXLFG$_L$)

Le but principal de cette expérience est de montrer que la robustesse et l'efficacité de SXLFG permettent l'analyse de gros corpus à l'aide d'une grammaire à large couverture écrite dans un formalisme syntaxique profond linguistiquement motivé (Sagot et Boullier, 2006; Boullier et Sagot, 2005a). Pour ce faire, nous avons analysé 208 692 phrases de corpus journalistique brut (extrait du *Monde diplomatique*), soit environ 5 millions de tokens[1], à l'aide du système SXLFG$_L$ à large couverture produit par SXLFG. Le temps d'analyse total pour ces 4 millions et demi de mots n'est que de moins de 12 heures.

À ce stade, il est nécessaire d'expliciter une distinction importante sur laquelle nous reviendrons plus bas. SXLFG n'est pas un analyseur, c'est un générateur d'analyseurs. Il prend en entrée une grammaire et un lexique et produit en sortie un analyseur. Par conséquent, les performances de cet analyseur produit dépendent à la fois des performances du générateur SXLFG et des caractéristiques de la grammaire[2]. Ceci signifie qu'évaluer SXLFG n'est pas trivial, dans la mesure où l'on ne peut évaluer que les analyseurs qu'il génère pour des grammaires particulières. Le but de cette expérience étant de montrer que les performances de SXLFG permettent d'analyser de gros corpus avec des grammaires LFG à large couverture, nous évaluerons donc SXLFG en montrant simultanément :

- que le système SXLFG$_L$ produit par SXLFG à partir d'une grammaire à large couverture est suffisamment efficace et robuste pour analyser plusieurs millions de mots en quelques heures,
- que les performances de SXLFG$_L$ sont très satisfaisantes, étant données les caractéristiques de la grammaire dont il est issu.

En revanche, dans cette expérience, notre but n'est pas d'évaluer la grammaire elle-même (et les heuristiques de désambiguïsation qui l'accompagnent) ou le lexique. Nous les décrirons cependant, à la fois pour permettre la mise en rapport de leurs caractéristiques avec les performances de l'analyseur produit par SXLFG, mais également parce que nous utiliserons le système SXLFG$_L$ pour mener les deux expériences suivantes.

[1] Le nombre exact est de 5 508 467 transitions dans l'ensemble des DAG de mots produits par la phase de traitement pré-syntaxique, soit 1, 1 fois le nombre de tokens.

[2] Dans le cas général, une grammaire très ambiguë donnée à un générateur d'analyseurs performant peut induire un analyseur peu efficace. Mais une grammaire très simple donnée à un générateur d'analyseurs rudimentaire donnera également un analyseur peu efficace.

1.1 Méthodologie

1.2 Grammaire, règles de désambiguïsation, lexique

Le système SXLFG$_L$ repose sur une grammaire du français dont le développement s'est appuyé sur une grammaire LFG développée par L. Clément pour son système XLFG Clément et Kinyon (2001)[3]. Parmi les phénomènes complexes couverts par la grammaire, on peut citer les coordinations sans ellipse, les juxtapositions (de phrases ou de groupes), les interrogatives (non clivées), les sujets inversés ou doubles (*Pierre dort-il ?*), tous les types de noyaux verbaux (y compris les clitiques, les temps composés, le passif et la négation), les complétives (sous-catégorisées ou adjointes), les infinitives (y compris les verbes à montée et les trois types de verbes à contrôle) ou les relatives et les interrogatives indirectes (même enchâssées). En revanche, les comparatives, les clivées, les coordinations à ellipse ne sont pas couvertes (entre autres). D'autre part, nous avons pu constater que la grammaire squelette (cf. section 12.1.3) est excessivement ambiguë, bien que notre expérience montre que toute grammaire à large couverture reposant sur une grammaire squelette (non-contextuelle ou TAG, par exemple) est inévitablement très ambiguë. À titre indicatif, cette grammaire comporte 253 règles, 911 équations fonctionnelles, 40 symboles terminaux et 109 symboles non-terminaux, dont environ la moitié sont récursifs (la plus grande boucle de récursivité implique 30 symboles non-terminaux).

En complément de la grammaire proprement dite, nous avons défini un ensemble d'heuristiques de désambiguïsation[4], dont le rôle premier est de spécifier des préférences au sein d'un ensemble d'analyses qui sont toutes correctes (alors que la grammaire a pour rôle de spécifier quelles sont les analyses correctes et quelles sont celles qui ne le sont pas). Suivant sur ce point Clément et Kinyon (2001), nous utilisons un ensemble de règles qui est une adaptation et une extension des trois principes simples qu'ils décrivent et qui sont appliquées sur les f-structures, plutôt qu'un modèle stochastique[5]. Nos règles reposent sur des considérations linguistiques. Parmi elles, deux règles spéciales sont disponibles, qui éliminent les f-structures incohérentes et incomplètes soit dans tous les cas soit dans le seul cas où au moins une struc-

[3]Le projet initial XLFG remonte à 1996. Écrit en langage C, il comprend une interface graphique de développement de grammaires LFG et un analyseur qui repose sur un algorithme SLR non déterministe. Depuis le départ, cet analyseur est destiné plutôt à l'enseignement et à l'expérimentation de nouveaux concepts linguistiques, mais pas pour une utilisation sur de gros corpus ou de très longues phrases.

[4]Précisons bien ici que ces heuristiques ne sont pas définies dans SXLFG, mais sont fournies, avec la grammaire, en *entrée* à SXLFG.

[5]Comme évoqué plus haut, un modèle stochastique pourrait cependant facilement être intégré, en définissant une règle qui utiliserait un tel modèle pour pondérer les f-structures (cf. par exemple Miyao et Tsujii (2002)). Comme les autres règles, cette règle appliquée à un non-terminal éliminerait toutes les f-structures qui ne sont pas optimales au sens de cette règle, c'est-à-dire dont la pondération par le modèle stochastique ne serait pas maximale. Cependant, nos expériences montrent que des règles structurelles peuvent être suffisamment discriminantes pour permettre un parsing efficace, sans que soient nécessaires des données statistiques qui doivent être acquises sur des corpus annotés rares et coûteux, en particulier si la langue considérée n'est pas l'anglais.

ture cohérente et complète existe pour le même non-terminal instancié (ces règles ne sont pas appliquées lors de la seconde tentative, si celle-ci est nécessaire). Comme expliqué précédemment, nous avons attribué à certains non-terminaux du squelette CFG, qui correspondent à des groupes « saturés » linguistiquement, une liste (ordonnée) partielle de ces heuristiques, chaque non-terminal concerné ayant sa propre liste.

Nos règles heuristiques, dans leur ordre d'application lorsque l'on effectue la désambiguïsation globale, c'est-à-dire à la racine de la forêt, sont les suivantes (on rappelle encore une fois que sur les non-terminaux internes sur lesquels on effectue une désambiguïsation interne, certaines règles peuvent ne pas être appliquées, ou bien dans un ordre différent) :

Règle 1 : *Éliminer les structures incohérentes et incomplètes si toutefois il y en a au moins une qui est cohérente et complète*.

Règle 2 : *Préférer les analyses qui maximisent la somme des poids des lexèmes utilisés* ; le lexique utilisé associe en effet à certaines entrées un poids ; parmi les lexèmes qui ont un poids plus élevé que la normale se trouvent les mots composés,

Règle 3 : *Préférer les analyses qui minimisent le nombre de groupes nominaux sans déterminant.*

Règle 4 : *Préférer les analyses maximisant le nombre d'arguments par rapport au nombre de modifieurs, et le nombre de relations auxiliaire-participe par rapport au nombre d'arguments* (ce calcul est effectué récursivement sur toutes les sous-structures, une sous-structure contribuant d'autant moins à la mesure globale qu'elle est enchâssée).

Règle 5 : *Préférer les analyses minimisant la distance entre les prédicats (verbaux ou non) et leurs arguments.* (même remarque).

Règle 6 : *Préférer les structures les plus profondes*[6].

Règle 7 : *Ordonner les structures suivant le mode de leur tête verbale* (on préfère récursivement les structures à l'indicatif sur celles au subjonctif, et ainsi de suite).

Règle 8 : *Ordonner selon la catégorie des gouverneurs des adverbes.*

Règle 9 : *Choisir une analyse au hasard* (pour garantir que la sortie est une f-structure unique).

Le lexique que nous avons utilisé est la dernière version du Le*fff* (Lexique de formes fléchies du français, voir chapitre 5 ou Sagot *et al.* (2005)), qui contient des informations morphosyntaxiques et syntaxiques pour plus de 500 000 entrées correspondant à environ 400 000 tokens différents (mots simples ou composants de mots composés).

[6]On notera que cette règle est prévue pour être utilisée après la précédente, ce qui signifie que l'on a déjà éliminé toutes les analyses où les arguments sont en moyenne moins proches de leur prédicat. C'est la combinaison de cette règle et de la précédente qui a pour effet de sélectionner les constructions les plus « simples » structurellement.

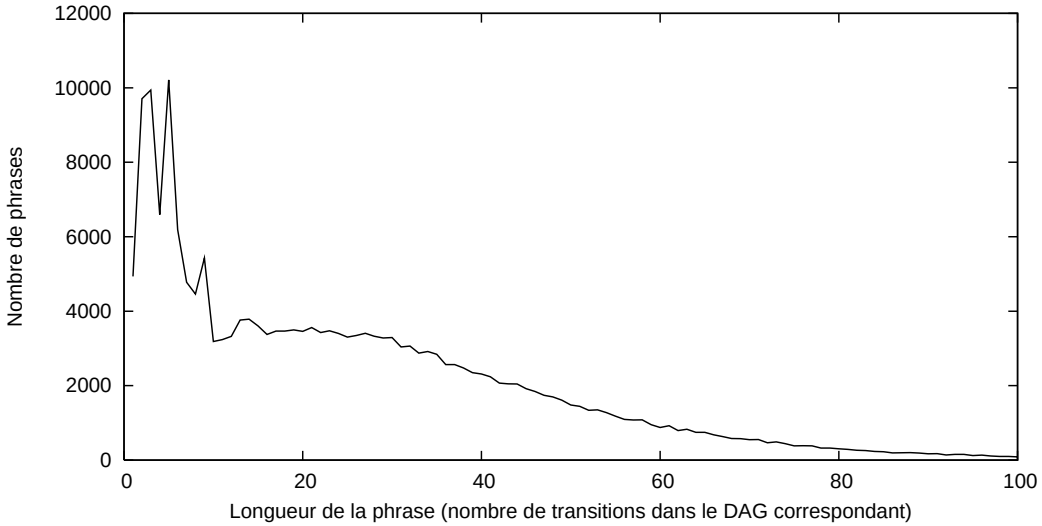


FIG. 1 – Répartition des phrases du corpus de test en fonction de leur longueur.

Mais le but de cette expérience n'est pas de valider la grammaire et les règles de désambiguïsation, la grammaire ayant pour seul rôle de permettre l'évaluation en situation réelle des techniques d'analyse développées pour SXLFG.

1.3 Résultats

Comme indiqué précédemment, nous avons analysé près de 5 millions de mots de corpus journalistique brut, après l'avoir passé à travers la chaîne SXPipe de traitement pré-syntaxique (voir 11), obtenant ainsi des DAG comportant au total plus de 5 millions et demi de transitions. La répartition des phrases en fonction de leur longueur est indiquée par la figure 1. Dans cette figure et celles qui suivent, l'abscisse est bornée pour ne montrer de résultats que sur des données statistiquement significatives, mais toutes les phrases sont analysées, la plus longue étant de longueur 384.

Cependant, pour évaluer les performances de notre générateur d'analyseurs SXLFG, il nous fallait isoler autant que possible l'influence de la grammaire dans nos résultats quantitatifs. En effet, et comme dit plus haut, l'efficacité de SXLFG est orthogonale à la qualité et à l'ambiguïté de la grammaire, qui est une donnée d'entrée pour SXLFG. Au contraire, notre but est de développer un générateur d'analyseurs qui produise un analyseur aussi efficace et robuste que

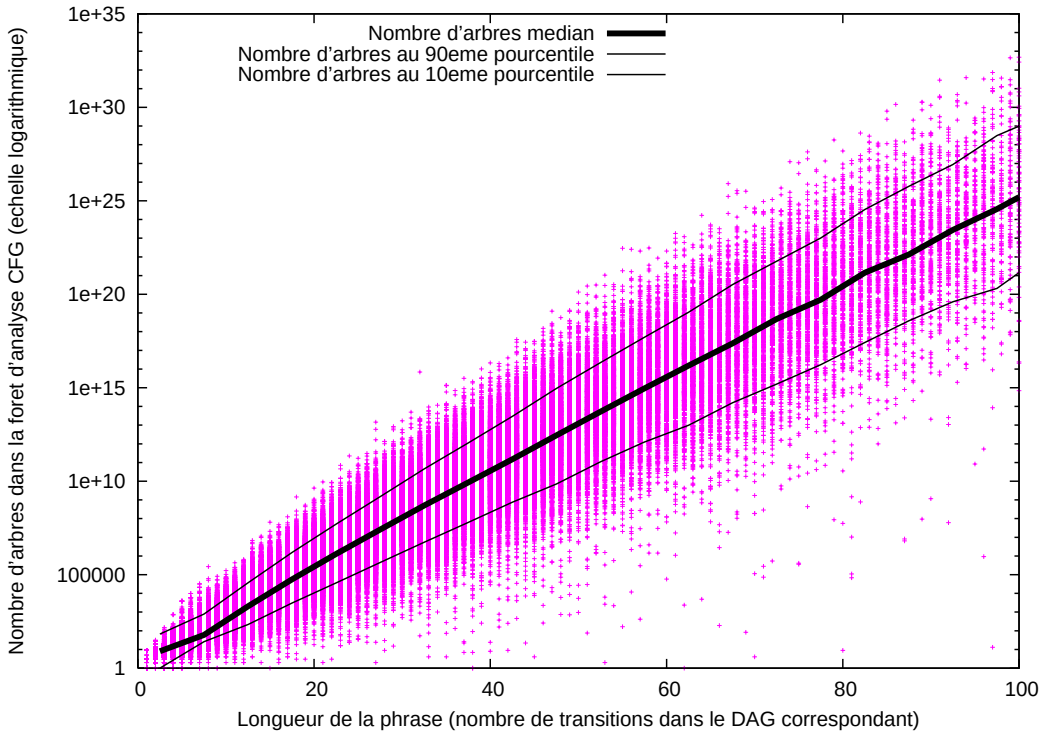


FIG. 2 – Ambiguïté selon le squelette non-contextuel en fonction de la longueur des phrases (les médianes sont calculées sur des classes de longueur allant de $5i$ à $5(i+1)-1$ et se voient attribuer une abscisse centrée $(5(i+1/2))$.

possible pour une grammaire donnée qu'on lui donne en entrée, y compris dans le cas où ladite grammaire est très ambiguë (au niveau du squelette CFG ou globalement) ou peu couvrante[7].

1.3.1 Évaluation de l'analyseur non-contextuel

Pour cette raison, la figure 2 indique le niveau d'ambiguïté du squelette non-contextuel en montrant le nombre médian d'analyses non-contextuelles en fonction du nombre de transitions dans le DAG de mots représentant une phrase donnée. Bien que le nombre d'arbres soit extrêmement élevé, la figure 3 montre l'efficacité de notre parseur CFG[8] (le nombre maximum d'arbres atteint pour une phrase de notre corpus ne nécessitant pas de rattrapage d'erreur est

[7]Nous n'avons pas pu comparer les résultats quantitatifs de SXLFG par rapport à d'autres systèmes. Le site internet de l'environnement XLE donne toutefois quelques indications (voir `http://www2.parc.com/istl/groups/nltt/xle/`), qui sont difficilement interprétables et ne semblent pas extrêmement récentes. SXLFG donne des résultats plus rapides de plusieurs ordres de grandeur, sans qu'il semble possible d'en tirer des conclusions précises.

[8]Ces expériences ont été réalisées sur une architecture AMD Athlon 2100+ (1,7 MHz) sous Linux Mandrake 10.2. Par ailleurs, la granularité de la mesure de temps d'exécution étant de 10 ms, nous avons ajouté 5 ms à tous les temps obtenus.

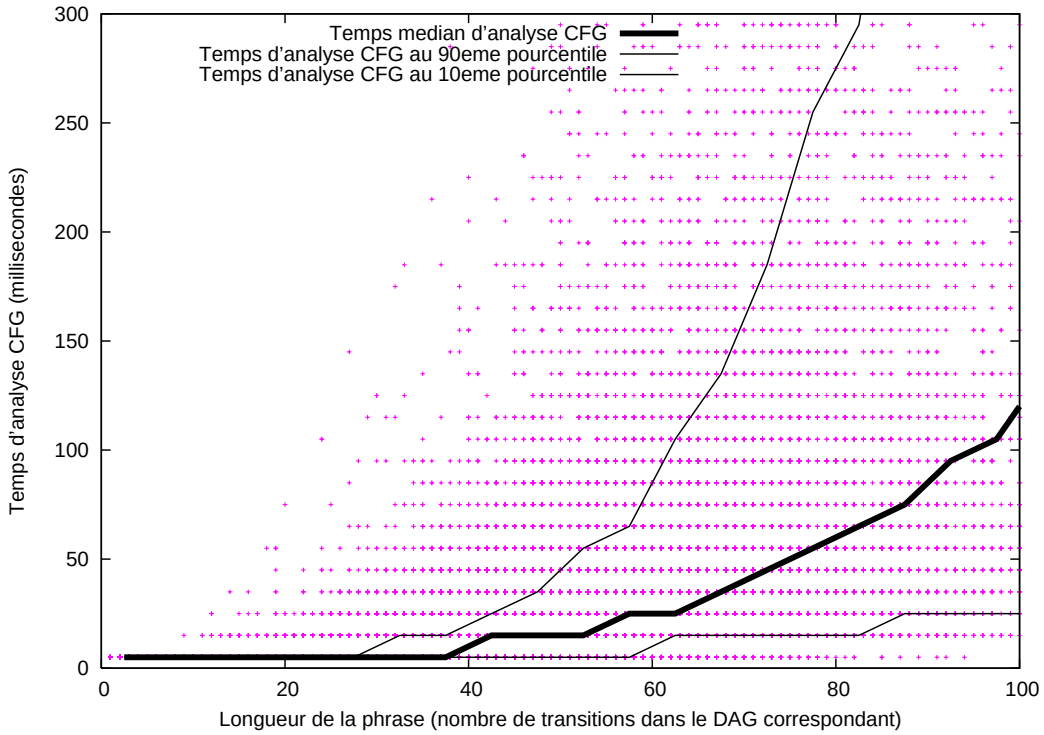


FIG. 3 – Temps d'analyse par l'analyseur non-contextuel (même remarque que pour la figure 2).

de $8,8\,10^{45}$ pour une phrase de longueur 143 dont l'analyse non-contextuelle prend seulement 0,05 s). De plus, les techniques de rattrapage d'erreur décrites dans la section 9.3.1 donnent de bons résultats dans la plupart des cas où le squelette CFG ne reconnaît pas la phrase d'entrée : sur les 208 692 phrases, seules 13 766 ne sont pas reconnues par le squelette ($6,60\%$).

1.3.2 Évaluation du calcul des structures fonctionnelles

Bien que le squelette non-contextuel soit massivement ambigu, nos résultats montrent que nos techniques d'évaluation des f-structures sont efficaces. En effet, avec un délai maximum (*timeout*) de 1 seconde seulement, il suffit de 42 797 secondes soit 11 heures et 53 minutes pour analyser l'ensemble du corpus de 5 millions et demi de transitions, et seules $12,3\%$ des phrases atteignent ce délai sans avoir produit une analyse. Cela correspond à une vitesse moyenne de 128 transitions par seconde. On notera qu'un délai de 0,2 seconde fait descendre à un peu plus de 4 heures le temps total d'analyse, en ne faisant monter qu'à $24,3\%$ le pourcentage de phrases atteignant le délai. Enfin, un délai très agressif de 0,1 seconde permet d'analyser les 5 millions et demi de transitions en environ 2 heures et demie, « seulement » $31,5\%$ des phrases atteignant alors ce délai.

Nombre total de phrases	208 692	
Reconnues par le squelette CFG	194 926	93,4%
Analyse CFG avec rattrapage	13 766	6,6%
f-structure cohérente et complète	122 188	58,5%
f-structure incohérente ou incomplète	37 979	18,2%
f-structures partielles	21 483	10,3%
Aucune f-structure (rattrapage CFG trivial)	1 245	0,6%
Erreur de l'analyseur (!)	31	0,01%
Délai expiré (1 s)	25 759	12,3%

TAB. 1 – Résultats de couverture sur un corpus journalistique brut de 5 millions de tokens environ (plus de 5,5 millions de transitions dans les DAG de mots correspondant).

La couverture de la grammaire sur notre corpus avec désambiguïsation interne est de $58,5\%$, la couverture étant définie comme la proportion de phrases pour lesquelles une f-structure principale cohérente et complète est fournie par l'analyseur[9]. Ceci inclut les cas où la phrase était non-grammaticale pour l'analyseur non-contextuel, mais où la forêt produite par le rattrapage d'erreur a permis de calculer une structure fonctionnelle principale cohérente et complète (cela concerne 1 782 phrases, c'est-à-dire $0,85\%$ de toutes les phrases et $12,9\%$ des phrases non-grammaticales pour le squelette CFG ; ceci montre que le rattrapage d'erreur dans l'analyseur non-contextuel donne des résultats assez pertinents).

Puisque nous ne voulons pas que l'ambiguïté du squelette non-contextuel influe sur notre évaluation, la figure 4 représente le temps total d'analyse, y compris l'évaluation des structures fonctionnelles, en fonction du nombre d'arbres produits par l'analyseur non-contextuel.

1.4 Conclusions

Cette expérience montre que des grammaires à large couverture reposant sur des mécanismes d'unification peuvent être utilisées dans des analyseurs capables d'analyser des volumes très importants de texte brut en des temps relativement courts, grâce à des techniques permettant de calculer des structures fonctionnelles sur une forêt partagée massivement ambiguë et même en l'absence de filtrage statistique.

Nous montrons également qu'il est pertinent d'utiliser des techniques de rattrapage d'erreur à la fois au niveau des constituants et au niveau des structures fonctionnelles. De plus, les différentes techniques de robustesse que nous appliquons au niveau fonctionnel nous permettent de construire de l'information pertinente, même si elle n'est parfois que partielle. On notera que ces techniques de robustesse, qui n'altèrent pas significativement les performances globales de

[9]Si on exclut les phrases qui atteignent le *timeout*, on obtient parmi les phrases analysées jusqu'au bout un taux de couverture de $66,8\%$.

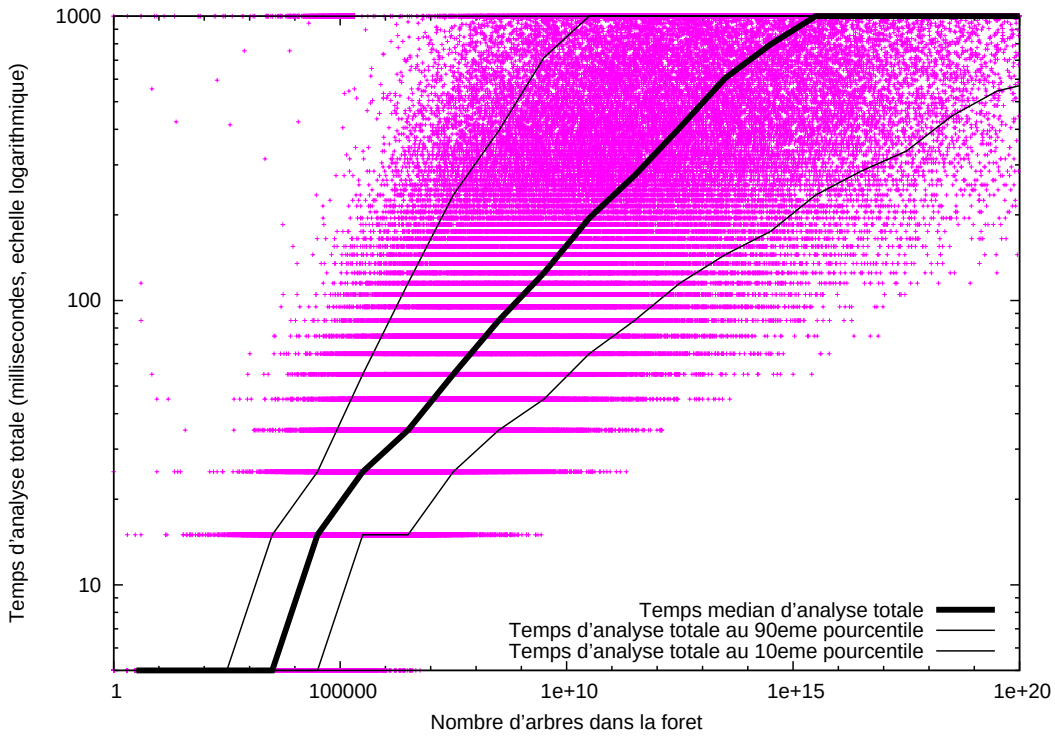


FIG. 4 – Temps total d'analyse en fonction du nombre d'arbres dans la forêt produite par l'analyseur non-contextuel (les médianes sont calculées sur des classes de longueur allant de 10^i à $10^{i+1} - 1$ et se voient attribuer une abscisse centrée ($10^{i+1/2}$)) sur une échelle logarithmique.

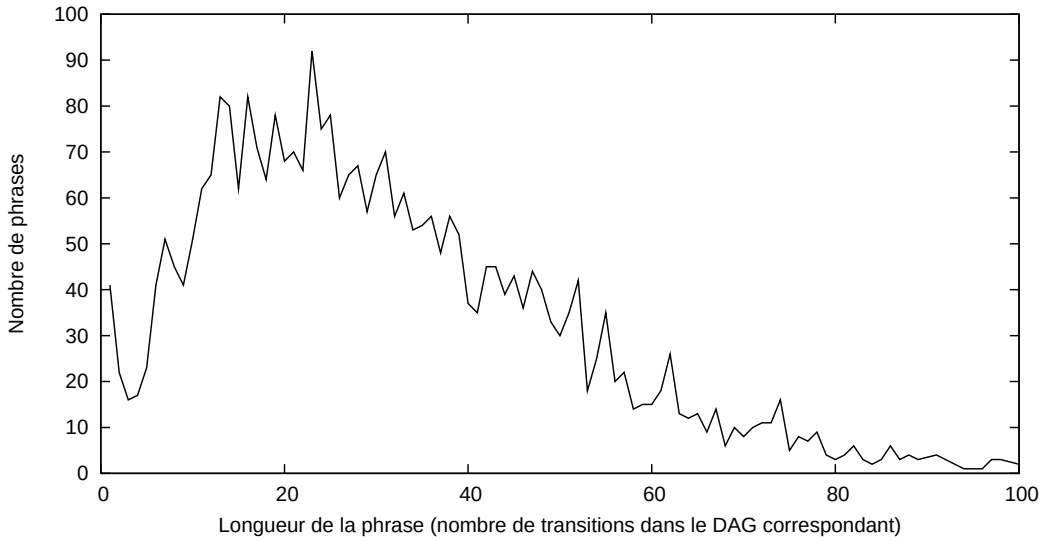


FIG. 5 – Répartition des phrases du corpus de test en fonction de leur longueur. La courbe donne le cardinal de classes de phrases de longueur comprise entre $10i$ et $10(i+1)-1$ avec une abscisse centrée (en $10(i+1/2)$).

SXLFG, s'appliquent à la fois dans le cas où la grammaire est incomplète (couverture insuffisante) et où la phrase est agrammaticale, bien qu'il semble que les résultats soient meilleurs dans le second cas.

2 Évaluation de la désambiguïsation interne (système SXLFG $_L$)

2.1 Méthodologie

Le but de cette expérience est d'évaluer l'intérêt de mettre en place des heuristiques de désambiguïsation interne (cf. chapitre 9), c'est-à-dire des heuristiques permettant de filtrer pendant l'analyse les structures fonctionnelles dont on est (presque) certain qu'elles ne peuvent que conduire à une structure principale incohérente ou incomplète. Pour ce faire, nous avons utilisé SXLFG$_L$ pour analyser, avec et sans désambiguïsation interne, un corpus journalistique de taille moyenne, dont la répartition des phrases en fonction de leur longueur est donnée figure 5 (Boullier et Sagot, 2005b). Le délai maximum a été augmenté à 20 secondes, pour permettre à l'analyseur sans désambiguïsation interne de donner des résultats dans un nombre raisonnable de cas.

Nous avons utilisé le système SXLFG$_L$ décrit dans la section précédente. Les figures 5 et 6 donnent quelques indications sur les longueurs des phrases et l'ambiguïté selon le squelette non-contextuel, à l'image des figures données dans la section précédente.

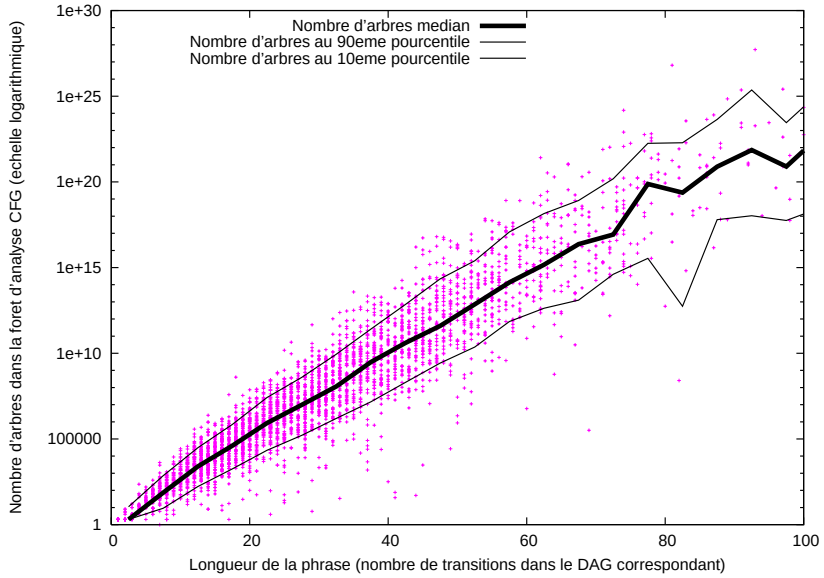


FIG. 6 – Ambiguïté selon le squelette non-contextuel en fonction de la longueur des phrases (les médianes sont calculées sur des classes de longueur allant de $10i$ à $10(i+1)-1$ et se voient attribuer une abscisse centrée $(10(i+1/2))$.

2.2 Résultats

Avec désambiguïsation, les résultats obtenus sont comparables à ceux de la section précédente : $59,2\%$ des phrases reçoivent une analyse cohérente et complète. Grâce au délai rallongé de 20 secondes, seulement $1,3\%$ des phrases ne sont pas analysées avant son expiration.

Sans désambiguïsation interne, la proportion importante de phrases qui dépassent le délai avant d'être analysées ($22,7\%$, à comparer aux $1,3\%$ précédents) conduit à une couverture bien inférieure de $42,8\%$. Parmi les phrases couvertes par un tel système, $94,7\%$ sont également couvertes par l'analyseur avec désambiguïsation interne, ce qui veut dire que seules 73 phrases couvertes par la grammaire sont perdues en raison de la désambiguïsation interne. Ce qui est à comparer avec les 600 phrases qui ne sont pas analysées à cause du délai lorsque la désambiguïsation interne est désactivée, mais qui font partie du langage défini par la grammaire et qui sont effectivement couvertes lorsque la désambiguïsation interne est utilisée : le risque qu'il y a à élaguer pendant l'analyse (dans l'état actuel des règles de désambiguïsation interne) est bien inférieur au bénéfice que l'on en tire, à la fois en termes de couverture et de temps d'analyse.

2.3 Conclusion

Cette expérience montre l'intérêt qu'il y a à mettre en œuvre le principe de désambiguïsation interne décrit au chapitre 9 : le risque pris est bien inférieur au bénéfice obtenu. Toutefois, la

Results	Avec désambiguïsation interne		Sans désambiguïsation interne	
Nombre total de phrases	3215			
Reconnues par le squelette CFG	2994		93,1%	
Analyse CFG avec rattrapage	221		6,9%	
f-structure principale cohérente et complète	1904	59,2%	1375	42,8%
f-structure principale incohérente ou incomplète	162	5,0%	273	8,5%
f-structures partielles	1005	31,3%	762	23,7%
Aucune f-structure trouvée (sur-segmentation)	103	3,2%	70	2,2%
Délai expiré (20 s)	41	1,3%	731	22,7%

TAB. 2 – Résultats de couverture avec et sans désambiguïsation interne, avec la même grammaire et le même lexique, sur un corpus journalistique brut.

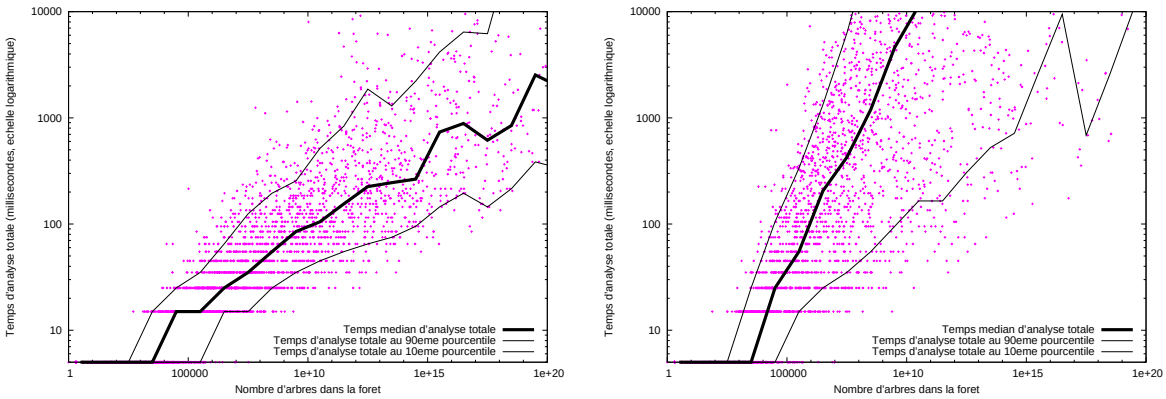


FIG. 7 – Temps total d'analyse en fonction du nombre d'arbres dans la forêt produite par l'analyseur non-contextuel. La figure de droite donne le résultat avec désambiguïsation interne, la figure de gauche donne le résultat sans désambiguïsation interne (les médianes sont calculées sur des classes de longueur allant de 10^{2i} à $10^{2i+2} - 1$ et se voient attribuer une abscisse centrée (10^{2i+1})) sur une échelle logarithmique.

solution optimale serait d'accompagner la mise en œuvre de ce principe de méthodologies de repli en cas d'échec de l'analyse, pour éliminer les pertes dues à des désambiguïsations internes par trop cavalières.

3 Campagne EASy (système SXLFG$_L$)

La campagne EASy est une campagne d'évaluation des analyseurs syntaxiques organisée à l'échelle nationale, qui a mis en compétition des participants à la fois publics et privés pendant le dernier trimestre 2004[10]. L'objectif était d'analyser automatiquement environ 35 000 phrases, dans un délai d'une semaine, et de rendre les résultats des analyses en un format défini dans le Guide d'annotation Gendner et Vilnat (2004). Ce format regroupe une annotation en constituants, qu'il fallait rendre obligatoirement, et une annotation en dépendances syntaxiques, qui était facultative et que l'on pouvait rendre sous une forme ambiguë ou désambiguïsée. Nous avons choisi de rendre à la fois les constituants et les dépendances désambiguïsées. Nos résultats font l'objet de Boullier *et al.* (2005).

3.1 Corpus et analyseurs

Les corpus de textes à analyser se regroupaient en différentes classes. Tous étaient des corpus réels, non retravaillés. Seule une segmentation en tokens et en phrases était proposée, principalement à des fins de compatibilité entre les résultats des différents participants. Il y avait environ 6 000 phrases de corpus généraux (journalistiques, législatifs), 8 000 phrases de corpus littéraires, près de 8 000 phrases de corpus de courrier électronique (avec tout le bruit que l'on peut imaginer dans un tel corpus), plus de 2 000 phrases de corpus médicaux, 7 000 phrases de corpus de transcription d'oral (avec les marques spécifiques à de tels corpus, comme les hésitations, les reprises, les répétitions, etc.), et 3 500 phrases de corpus de questions (issus de systèmes de questions-réponses).

Le développement de SXLFG, et en particulier du système SXLFG$_L$, ainsi que celui de SXPipe (cf. chapitre 11), ont été en partie motivés par cette campagne. Nous avons adapté la grammaire originale de SXLFG$_L$ pour permettre l'extraction des informations syntaxiques en fonction des desiderata des organisateurs d'EASy, mais également pour en augmenter la couverture tout en faisant diminuer l'ambiguïté de sa CFG support.

[10]Et donc à un stade très peu avancé du développement de SXLFG en général et de SXLFG$_L$ en particulier. Ainsi, le mécanisme de désambiguïsation interne n'était pas encore disponible. C'est la raison pour laquelle nous donnons également les résultats que nous aurions eu avec SXLFG$_L$ tel qu'il est aujourd'hui. Les progrès sont patents.

Au sein de l'équipe Atoll de l'INRIA, un autre analyseur, FRMG, a également participé à cette campagne (Thomasset et Éric Villemonte de la Clergerie, 2005)[11]. Il est principalement l'œuvre d'Éric de La Clergerie.

Le format et la nature des informations linguistiques attendus par les organisateurs de la campagne EASy (voir Gendner et Vilnat (2004)) ne correspondent pas nécessairement à nos propres formats et choix linguistiques. D'autre part, les techniques tabulaires de partage de calculs mises en œuvre dans nos analyseurs sont en partie motivées par le soucis d'obtenir l'ensemble des analyses pour une phrase, alors que la piste d'évaluation de base pour EASy concerne des analyses syntaxiques non ambiguës. Il a donc été nécessaire de mettre en place des algorithmes de désambiguïsation sur les structures partagées produites par nos analyseurs, telles que les heuristiques présentées précédemment et utilisées par SXLFG$_L$ (les choix effectués pour FRMG sont relativement différents), mais aussi des algorithmes de conversion pour extraire les informations requises (cette dernière phase, développée spécifiquement pour EASy et dans une relative précipitation, a d'ailleurs significativement dégradé nos résultats, ainsi que nous nous en sommes aperçus par la suite).

À titre d'illustration, les figure 8 et 9 montrent respectivement le DAG produit par SXPipe[12] ainsi que son analyse au format attendu par les organisateurs d'EASy pour la phrase (forgée) suivante : *Jean␣ abite␣en␣outre␣au␣1␣,␣rue␣de␣la␣Pompe* (une espace correspond à une frontière de tokens au sens de la segmentation fournie par EASy).

[11] L'analyseur FRMG s'appuie sur une grammaire d'arbres adjoints (TAG) avec décorations engendrée à partir d'un niveau plus abstrait de description, une *méta-grammaire* (MG) (Candito, 1999; Thomasset et Éric Villemonte de la Clergerie, 2005). La grammaire obtenue est très compacte avec seulement 133 arbres, car elle s'appuie sur des *arbres factorisés* grâce à l'emploi de disjonctions entre nœuds, de répétition de nœuds et surtout de nœuds optionnels contrôlés par des gardes. L'ancrage des arbres TAG par les entrées lexicales se fait par unification de structures de traits appelées *hypertags*. L'ensemble a été développé sous DyALog (voir chapitre 3.

Un analyseur syntaxique hybride TAG/TIG a été compilé à partir de la grammaire. Il peut prendre en entrée les DAG produits par la chaîne d'entrée SXPipe (chapitre 11) modulo quelques conversions pour construire les hypertags. Au démarrage de l'analyse, les arbres sont filtrés par rapport aux mots du DAG d'entrée pour ne garder que ceux compatibles pour les nœuds d'ancrages et les nœuds lexicaux. L'analyseur utilise une stratégie d'analyse tabulaire descendante gauche-droite en une seule passe : le traitement des décorations des nœuds n'est pas repoussé dans une seconde passe, contrairement à la stratégie SXLFG. Néanmoins, les décorations ne sont pas prises en compte pour les prédictions descendantes mais seulement dans les propagations de réponses. Le parcours des arbres factorisés se fait sans expansion de ceux-ci assurant une bonne efficacité. L'analyseur retourne soit une analyse complète du DAG d'entré, soit, en mode robuste, un ensemble d'analyses partiels couvrant au mieux ce DAG. Les analyses sont émises sous formes de forêts partagées de dérivation TAG indiquant les diverses opérations effectuées (substitution, adjonction, ancrage,...). Via une représentation intermédiaire sous forme XML, ces forêts sont ensuite converties en forêts partagées de dépendances. Ces forêts de dépendance servent de base pour les traitements post-syntaxiques.

[12] Les notations sont allégées, et seuls les cas où il n'y a pas correspondance exacte entre un token et une forme sont indiqués : le ou les tokens sont alors entre accolades, la forme associée étant indiqué derrière. On notera que *Jean*, en tant que premier token, peut aussi désigner une catégorie de pantalon, que la faute d'orthographe sur *abite* est corrigée, la reconnaissance de l'adresse et le traitement du multi-mot et de l'agglutinée.

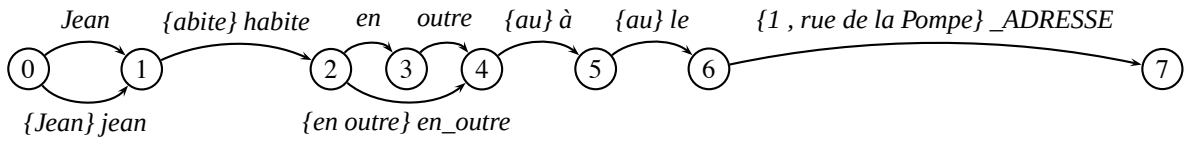


FIG. 8 – DAG associé à *Jean abite en outre au 1 , rue de la Pompe*.

GN 1	NV 2	GR 3		GP 4						
Jean	abite	en	outre	au	1	,	rue	de	la	Pompe
F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11

sujet	**verbe**
GN1	NV2

complément	**verbe**
GP4	NV2

modifieur	**verbe**
GR3	NV2

FIG. 9 – Sortie EASy fournie par SXLFG et FRMG pour la même phrase que précédemment.

3.2 Mise en œuvre et résultats expérimentaux

3.2.1 Couverture et temps d'analyse

Les tableaux 3 à 5 présentent divers résultats chiffrés concernant SXLFG$_L$ et FRMG. Trois remarques sont nécessaires. Tout d'abord, les nombres de phrases sont différents selon le système. Ceci est dû à une heuristique différente pour la segmentation en phrases[13]. Ainsi, les « phrases » de FRMG sont moins nombreuses pour un même corpus que les « phrases » de SXLFG. Par ailleurs, et par souci d'homogénéité, les temps d'analyse présentés dans le tableau 5 ont été calculés sur une machine unique[14] et non sur le cluster utilisé pour faire effectivement la campagne. Enfin, la notion d'ambiguïté du résultat final n'est pertinente que pour le système FRMG, puisque SXLFG utilise des heuristiques de désambiguïsation permettant de choisir une analyse unique dans tout ensemble non vide d'analyses. Le *taux d'ambiguïté moyen par mot* fourni pour FRMG est défini comme la moyenne privée de 1 du nombre moyen d'arcs de dépendance atteignant une forme[15].

[13]Pour SXLFG$_L$, SXPipe a été utilisé avec une option faisant du point-virgule une frontière de phrase, alors que ce n'était pas le cas pour FRMG.

[14]Architecture Athlon 2100+ (1.7 GHz), avec un analyseur compilé sous `gcc` 3.3.2.

[15]Pour une phrase non-ambiguë, chaque mot (sauf la « tête » de la phrase) est atteint par un seul arc, d'où un taux d'ambiguïté nul. Le nombre maximal d'analyses pour un taux α et une phrase de longueur n est en $O((1+\alpha)^{n})$.

[14]Cf. chapitre 9 pour plus de détails.

[15]Nous n'avons pas conservé les informations permettant de donner les résultats sur l'ensemble du corpus. Toutefois, la table 5 donne les temps d'analyse pour les 87.51% de phrases reconnues par la CFG support.

[16]On notera qu'un *timeout* plus élevé aurait augmenté les taux de couverture mais également les temps d'analyse.

[17]Cette valeur inférieure à 1 indique que dans les 14.34% de cas restant, aucune analyse n'a été trouvée en moins de 15s. C'est dans ces cas là que sont alors appliquées les heuristiques de sur-segmentation détaillées précédemment.

[18]Pour les données sur les corpus, n désigne un nombre de mots, et UW un nombre de mots inconnus.

Corpus	#phrases	% couv.	temps d'analyse				amb.
			moy.	méd.	$\geq$ 1s	$\geq$ 10s	
EASy	34438	42.45%	5.55s	1.61s	64.41%	9.32%	0.6

(a) Global

corpus	#phrases	% couverture	temps	amb.
general	6160	41.01%	10.31s	0.65
litteraire	7960	38.93%	5.59s	0.53
mail	7692	32.83%	4.37s	0.70
medical	2225	44.00%	5.47s	0.70
oral	6892	44.39%	3.58s	0.53
questions	3509	69.28%	3.47s	0.58
Total	34438	42.45%	5.55s	0.60

(b) Détail EASy

TAB. 3 – Résultats pour FRMG, avec un *timeout* de 100 secondes[18]

Corpus	#phrases	couverture de la CFG support	couverture	*timeout* (temps $\geq$ 15s)
EASy (jan. 2005)	40859	87.51%	41.95%	12.31%
EASy (jan. 2006)	39872	91.71	59.64% 2.04%	

(a) Global

corpus	#phrases	couverture de la CFG support	couverture	*timeout* (temps $\geq$ 15s)
general	6952	89.24%	32.42%	22.64%
litteraire	11408	89.92%	40.52%	13.03%
mail	9308	83.02%	40.18%	9.53%
medical	2553	87.34%	39.99%	12.10%
oral	7075	85.24%	46.47%	8.30%
questions	3563	92.73%	62.19%	5.22%
Total	40859	87.51%	41.95%	12.31%

(b) Détail EASy (janvier 2005)

corpus	#phrases	couverture de la CFG support	couverture	*timeout* (temps $\geq$ 15s)
general	7267	93.42%	58.88%	4.10%
litteraire	11707	94.40%	64.64%	1.78%
mail	9447	86.93%	57.27%	1.11%
medical	2622	86.04%	62.28%	1.45%
oral	11955	93.27%	76.50%	0.12%
questions	3565	94.73%	72.17%	0.34%
Total	46563	91.99%	65.73%	1.45%

(c) Détail EASy (mars 2006)

TAB. 4 – Résultats pour SXLFG, avec un *timeout* de 15 secondes[16].

		Corpus complet	Phrases valides pour la CFG support	
		Analyse CFG	Analyse CFG	Analyse complète
Corpus	#phrases	40859	35756	
	n_{max}	541	173	
	n_{moy}	20.95	19.06	
	n_{med}	16	15	
	UW_{max}	97	65	
	UW_{moy}	0.79	0.75	
Temps d'analyse	max	300s (timeout)	5.88s	15s (timeout)
	moy	0.05s	0.01s	3.35s
	med	0.00s	0.00s	0.03s
	$\geq$ 0.1s	1.79%	1.20%	42.2%
	$\geq$ 1s	0.24%	0.09%	29.0%
Nombre d'analyses	max	3.10^{73}	5.10^{52}	0.87^{17}
	med	32 028	29 582	1
	$\geq 10^6$	36.13%	35.28%	0%
	$\geq 10^{12}$	8.86%	7.84%	0%

TAB. 5 – Données sur les corpus[18], les temps et les nombres d'analyses pour SXLFG, avant application de l'heuristique de sur-segmentation (janvier 2005).

On constate des changements significatifs entre les résultats de janvier 2005 et ceux de mars 2006. La couvertire non-contextuelle semble évoluer de façon indéterminée. En réalité, la grammaire non-contextuelle a été complétée, d'où l'augmentation de la couverture. Mais le nombre de symboles terminaux utilisés a été récemment augmenté, ce qui a induit à la fois une légère baisse de précision mais aussi des analyses rattrapées de meilleure qualité, ainsi qu'une moindre ambiguïté. On constate par ailleurs une très nette amélioration de la couverture complète, et une diminution du pourcentage de phrases qui sont inanalysées à l'expiration du délai de 15 secondes. En réalité, un délai d'une seconde permettrait tout de même d'analyser 95,26% des phrases de l'ensemble des corpus EASy. Si l'on ne prend pas en compte la sur-segmentation et l'analyse des segments, ce délai permet d'analyser presque tout le corpus EASy avec SXLFG$_L$, analyseur profond et à large-couverture, en environ une heure et demie.

3.2.2 Précision et rappel

Nous sommes en mesure de donner un aperçu des résultats en termes de précision sur les constituants EASy (qui sont en réalité des chunks), obtenus par comparaison de nos analyses avec une référence construite manuellement[19], pour environ 4000 des phrases. Les résultats ne sont pas excellents (nous verrons que SXLFG$_B$ a des résultats bien supérieurs). Cela montre

[19]Cette annotation manuelle a été effectuée par différentes équipes, avec tous les problèmes de cohérence que l'on imagine. De fait, nous avons relevé des incohérences d'un sous-corpus à l'autre entre les annotations manuelles fournies.

que la grammaire utilisée dans SXLFG$_L$ nécessite encore un travail important de mise au point et d'augmentation de sa couverture. La désambiguïsation est également améliorable, et bien que la passage à des règles probabilistes soit une piste prometteuse, les résultats de SXLFG$_B$ montreront que cela n'est pas nécessairement indispensable, au moins dans un premier temps.

Les progrès accomplis par les diverses avancées effectuées au cours de 2005 sur SXLFG$_L$ (grammaire, analyseur, lexique) sont malgré tout significatifs. La précision comme le rappel sont augmentés (en restant assez loin des performances de SXLFG$_B$). Nous n'avons pourtant pas travaillé spécifiquement à l'amélioration de ces résultats, mais plutôt à la correction de la grammaire et du lexique. Un travail plus spécifique permettrait certainement d'améliorer encore les mesures de rappel et de précision.

3.3 Conclusion

La campagne d'évaluation EASy nous a permis de mettre en évidence l'importance de la différence qu'il y a entre le développement d'un analyseur syntaxique et le développement d'une chaîne complète d'analyse syntaxique. En effet, outre l'importance de la qualité de la grammaire et de l'analyseur, cette campagne a montré le rôle non moins déterminant de la couverture et de la richesse du lexique, de la qualité de la chaîne de traitement, de la précision des méthodes d'exploitation des sorties des analyseurs, ainsi que la très forte interaction entre les différents composants, et en particulier entre le lexique et la grammaire.

Cette forte complémentarité entre les différentes phases des chaînes d'analyse syntaxique a inévitablement élargi le champ de la campagne EASy. Ce n'est pas seulement l'analyse syntaxique elle-même qui a été évaluée lors de cette campagne, mais la capacité à mettre en place des chaînes d'analyse syntaxique complètes. En outre, la nécessaire harmonisation des résultats des différents participants passe par une segmentation commune en phrases et en mots. Cette segmentation, différente de celle produite et utilisée par nos outils[20], a dû être conservée tout au long de la chaîne pour pouvoir être utilisée dans la production des résultats finals.

Les résultats complets d'EASy seront riches d'enseignements. Mais nous comptons mettre à profit le fait que nous ayons déployé deux systèmes d'analyse que tout sépare, à l'exception du lexique et de la chaîne pré-syntaxique, ainsi qu'un troisième système, en cours de développement, mais dont les résultats sont déjà très prometteurs : SXLFG$_B$. Ceci nous permettra d'ef-

[20]Pour être franc, la tokenisation et la segmentation fournie par les organisateurs étaient de piètre qualité. Il nous a fallu introduire dans SXPipe des mécanismes permettant de mémoriser la tokenisation et la segmentation fournie, tout en utilisant nos propres outils, induisant ainsi des « recollages » et des « redécoupages » de phrases et de mots.

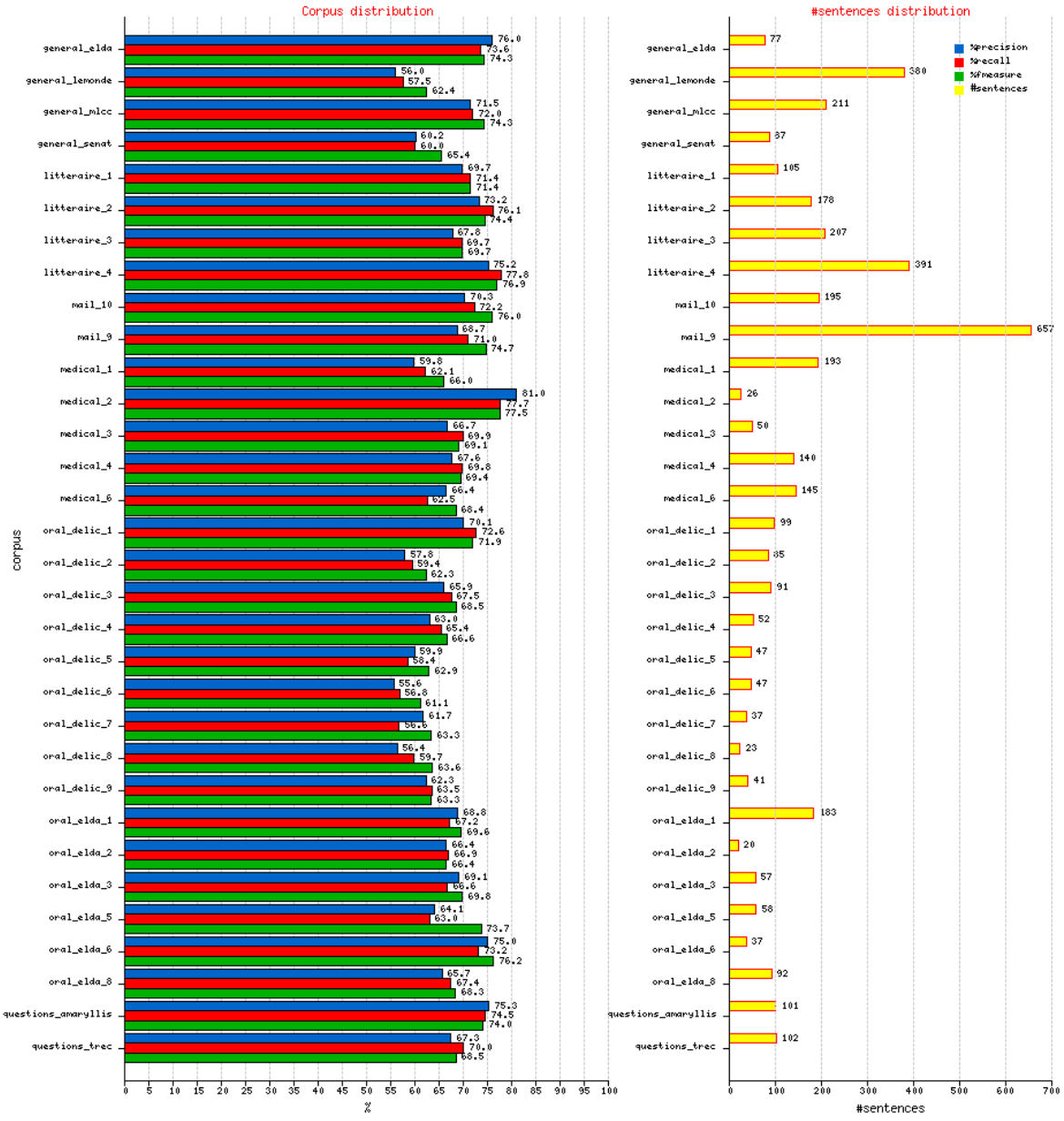


FIG. 10 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (janvier 2005) par corpus.

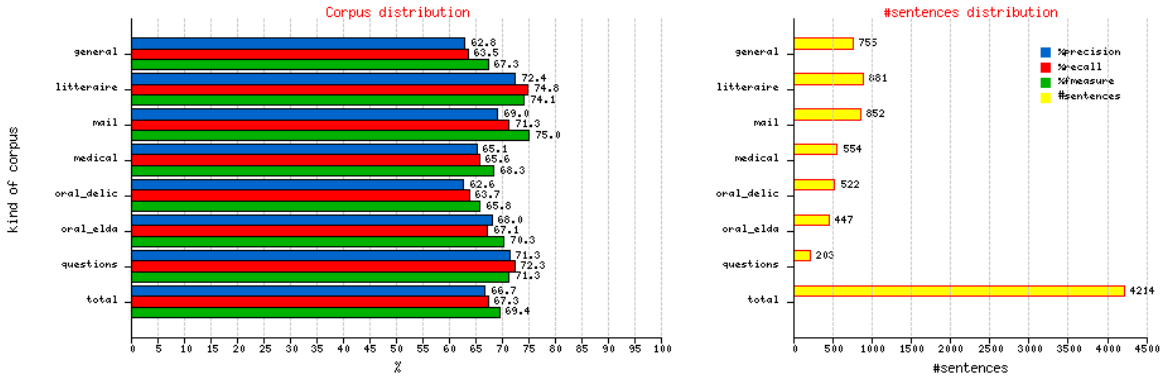


FIG. 11 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (janvier 2005) par type de corpus.

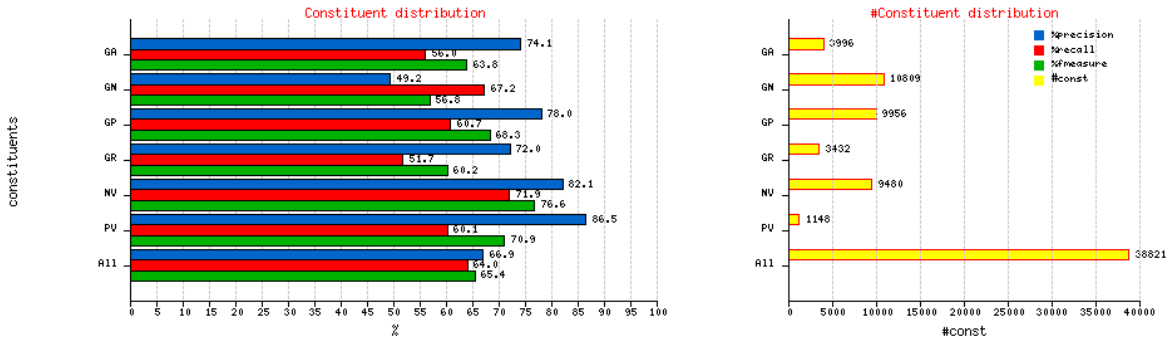


FIG. 12 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (janvier 2005) par type de constituants EASy.

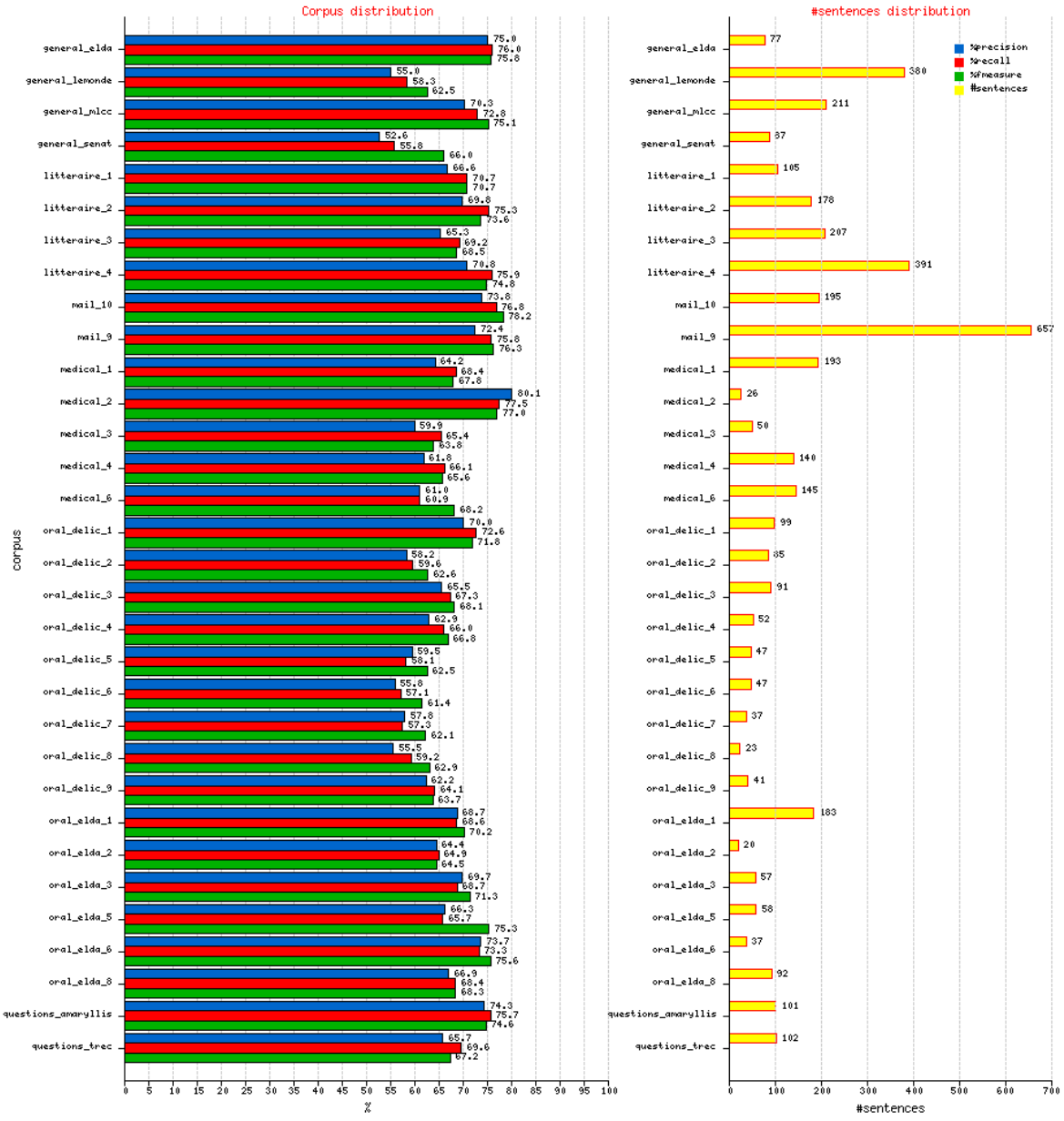


FIG. 13 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (mars 2006) par corpus.

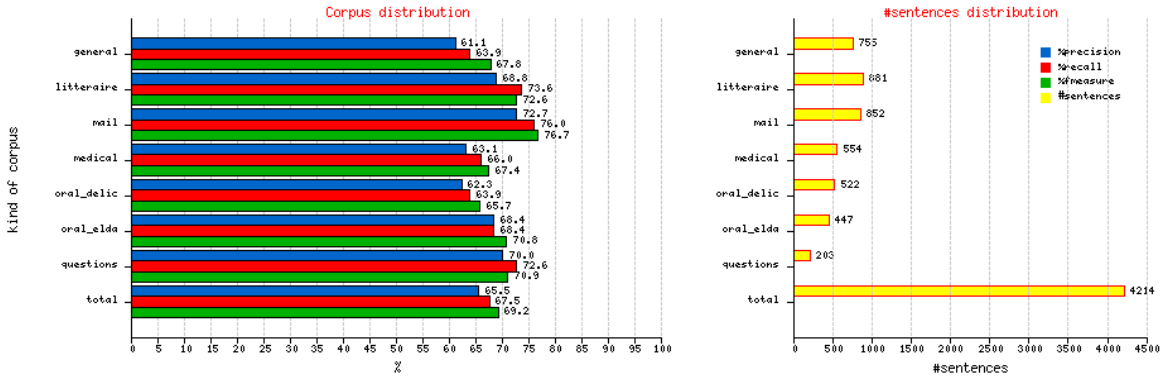


FIG. 14 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (mars 2006) par type de corpus.

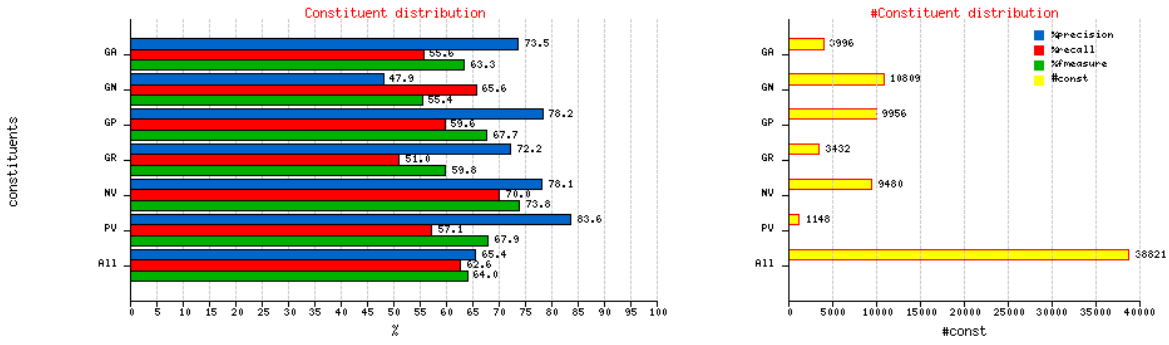


FIG. 15 – Précision, rappel et f-mesure sur le corpus EASy pour SXLFG$_L$ (mars 2006) par type de constituants EASy.

fectuer des comparaisons et d'améliorer ainsi les grammaires et les analyseurs (en étudiant les différences entre nos résultats), mais aussi le lexique et la chaîne de traitement pré-syntaxique[21].

4 Évaluation d'une nouvelle grammaire LFG du français (système SXLFG$_B$)

En étudiant les résultats obtenus par SXLFG$_L$, il nous a semblé que la grammaire utilisée dans ce système avait un certain nombre de défauts. Nous avons alors entrepris le développement d'une nouvelle grammaire LFG du français pour trois raisons principales :

- la grammaire utilisée dans le système SXLFG$_L$ a un support non-contextuel très ambigu,
- son développement ayant commencé bien avant l'élaboration du guide d'annotation pour la campagne EASy Gendner et Vilnat (2004), la mise en correspondance des non-terminaux de cette grammaire avec les constituants EASy est non-triviale et source d'erreurs évitables (et il en est de même pour les dépendances, appelées *relations* par EASy),
- son architecture est trop imbriquée pour permettre la mise en place de filtres (heuristiques ou probabilistes) à des niveaux intermédiaires entre les mots et les énoncés, et en particulier au niveau des *chunks*, c'est-à-dire en réalité ce qu'EASy appelle des *constituants*.

À son stade actuel de développement, nous disposons d'une grammaire de chunks munie de règles heuristiques de désambiguïsation. Le passage d'une grammaire de chunks à une grammaire complète est en cours, et donne des résultats prometteurs, mais la grammaire de chunks donne déjà des résultats tout à fait satisfaisants, y compris en termes de précision et de rappel, comme le montrent les figures 16 à 18. Ainsi, nous atteignons sur le corpus EASy une f-mesure (au sens des résultats provisoires d'EASy) de 78,5%, là où les meilleurs résultats obtenus parmi les participants de la campagne EASy sont de 79%. Ce résultat est pourtant atteint avec une grammaire très locale, et nous espérons donc qu'il sera encore amélioré une fois prises en compte des contraintes plus globales. À court terme, nous espérons pouvoir obtenir une grammaire aussi couvrante que celle utilisée dans SXLFG$_L$, mais de précision plus grande et avec un squelette moins ambigu. À moyen terme, nous souhaitons remplacer le squelette non-contextuel par un squelette Méta-RCG, fusionnant ainsi les deux pistes que nous avons suivies, et profitant à la fois de la non-linéarité des Méta-RCG (et donc de la prise en compte de façon satisfaisante des contraintes de constituance, de dépendance et de sémantique lexicale), de l'efficacité et de

[21] De tels travaux se rapprochent des travaux sur la fouille d'erreurs dans les résultats d'analyse présentés au chapitre 7. L'avantage des expériences envisagées ici est que la notion d'erreur voit sa qualité améliorée : une erreur serait une phrase ou un segment de phrase qui reçoit une analyse incorrecte (différente de l'analyse présente dans le corpus de référence annoté manuellement), et non une phrase dont l'analyse a échouée. L'inconvénient est naturellement le volume bien moindre des données traitées (4000 phrases annotées manuellement dans EASy contre plusieurs centaines de milliers).

la robustesse de SXLFG, et de notre expérience en termes de développement de grammaires et de lexiques à large couverture. Ce sera là une première tentative de mise en œuvre effective et complète de l'architecture proposée en fin de chapitre 8.

5 Système-prototype Méta-RCG

Comme indiqué au chapitre 10, notre système d'analyse Méta-RCG n'est pas encore un système à large couverture, capable d'analyser des corpus à grande échelle. Pour cette raison, nous ne sommes pas en mesure de donner d'évaluation quantitative de ce système-prototype. Pour une évaluation qualitative, on se reportera à la fin du chapitre 10.

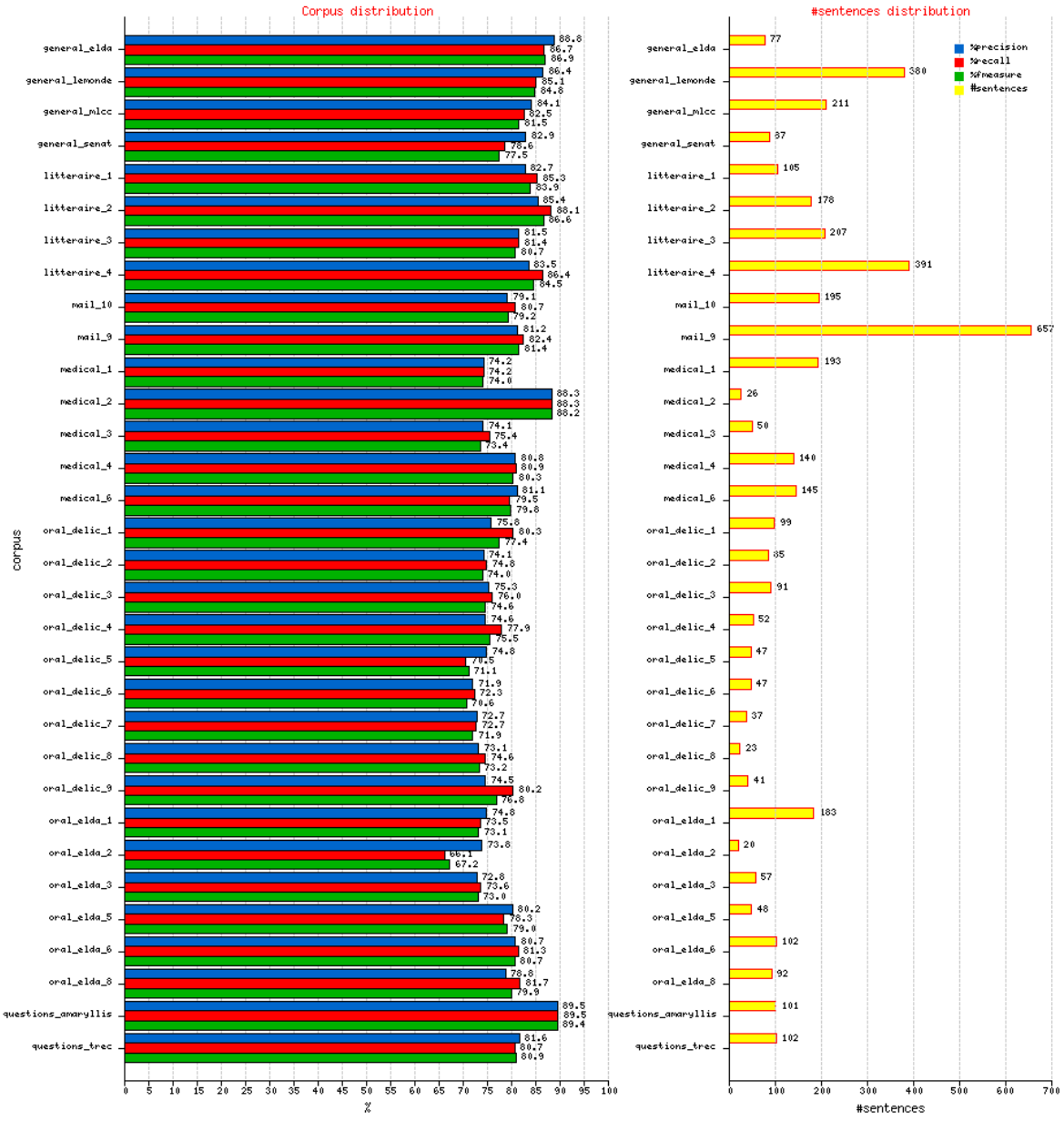


FIG. 16 – Précision, rappel et f-mesure sur le corpus EASy pour la partie chunker de SXLFG$_B$ par corpus.

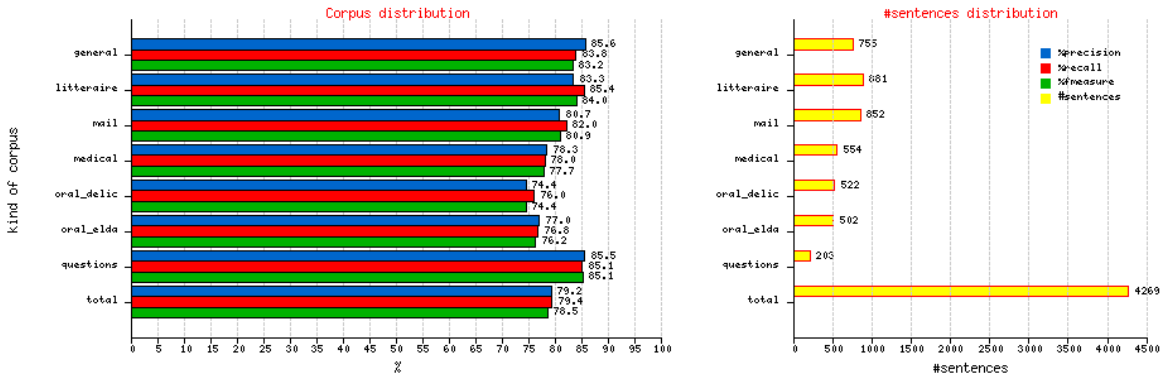


FIG. 17 – Précision, rappel et f-mesure sur le corpus EASy pour la partie chunker de SXLFG$_B$ par type de corpus.

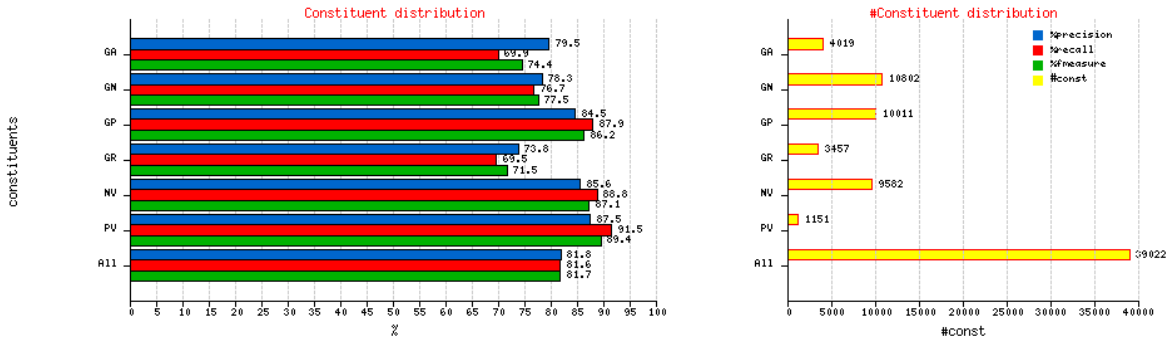


FIG. 18 – Précision, rappel et f-mesure sur le corpus EASy pour la partie chunker de SXLFG$_B$ par type de constituants EASy.

Conclusion

1 Bilan

Nous espérons avoir montré la pertinence de nos trois principe, que nous allons rappeler et détailler ici en guise de bilan.

1. Il est possible d'effectuer efficacement l'analyse non-générativiste d'énoncés avec des formalismes génératifs justifiés linguistiquement. C'est ce que nous avons montré en théorie au chapitre 8 puis en pratique avec SXLFG (chapitre 9) et lors de son évaluation avec diverses grammaires (chapitre 12).
2. La place des probabilités pour la modélisation des langues est dans le classement d'hypothèses obtenues par des mécanismes symboliques. C'est ainsi que certains mécanismes décrits dans la partie 2 tirent parti de modèles probabiliste pour l'apprentissage automatique d'informations ou la fouille d'erreur. C'est également ainsi que nous sommes en train de développer des mécanismes de désambiguïsation probabilistes pour SXLFG, qui reposeront sur des modèles appris sur des corpus annotés automatiquement par une version non probabilisée de SXLFG.
3. Il est pertinent et possible de coupler des contraintes morphologiques, syntaxiques et de sémantique lexicale, voire au-delà, à la fois dans les lexiques, les grammaires et les analyseurs. C'est ce que nous avons cherché à démontrer avec notre formalisme des Méta-RCG.

2 Perspectives

Le domaine du TAL est aujourd'hui à un moment charnière de son évolution. En effet, il semble que pour la première fois, construire des systèmes qui allient pertinence linguistique et efficacité pratique est un objectif à notre portée. Jusqu'à maintenant, les systèmes de traitement automatique se répartissaient en effet entre, d'une part, des outils opérationnels construisant des structures linguistiques assez pauvres et, d'autre part, des implémentations à petite échelle de

modèles formels linguistiquement motivés. C'est en particulier le cas dans le domaine qui est au cœur de cette thèse, celui de l'analyse automatique du français, où co-existent des analyseurs probabilistes ou surfaciques et des analyseurs-jouets pour des formalismes plus étoffés.

La suite naturelle de ce travail de thèse est la construction d'un modèle syntaxico-sémantique du français et d'un analyseur reposant sur ce modèle qui soit efficace, robuste et linguistiquement pertinent. Un tel système, qui n'existe pas aujourd'hui, aurait de nombreuses applications, par exemple dans les domaines de la recherche d'informations, de la traduction automatique, du résumé de textes, et plus généralement de toute application nécessitant que la machine puisse construire une représentation plus ou moins complexe du sens véhiculé par des énoncés en français. De plus, le développement du modèle sur lequel repose un tel système, s'il ne peut se faire qu'à l'aide de compétences en linguistique, apporte en retour de nombreux enseignements dans ce domaine.

Les résultats du chapitre 9 sur l'analyseur SXLFG permettent toutefois d'envisager la construction de systèmes qui soient non seulement efficaces et linguistiquement pertinents, mais qui disposent d'une propriété, la robustesse, qui permet de traiter toute phrase, quelles que soient ses imperfections ou celles du système. Mais SXLFG repose sur un formalisme linguistique classique, le formalisme LFG. Comme nous l'avons vu au chapitre 10, certaines propriétés de ce formalisme, propriétés qui sont partagées avec la plupart des formalismes linguistiques très répandus, ne sont pas satisfaisantes, à la fois pour des raisons computationnelles et des raisons linguistiques. C'est ce qui a motivé le développement des Méta-RCG, formalisme non-linéaire, ainsi que d'une grammaire et d'un analyseur du français efficace, à moyenne couverture, et qui permet la prise en compte d'informations syntaxiques et sémantiques (et en réalité de tous types de contraintes) au cours de l'analyse.

Il me semble qu'à eux deux, ces systèmes d'analyse regroupent l'ensemble des propriétés nécessaires et suffisantes pour construire un analyseur syntaxico-sémantique efficace, robuste, et linguistiquement pertinent. Il reste donc à développer un système qui tire le meilleur parti des avantages respectifs de SXLFG et des Méta-RCG. Pour cela, on peut imagier développer un système d'analyse qui associe la non-linéarité des Méta-RCG avec la robustesse et la grande efficacité de SXLFG. Un tel développement n'est naturellement intéressant qu'à la condition d'être mené de front avec la mise en œuvre de ce système dans un certain nombre d'applications, ainsi qu'avec le développement des ressources linguistiques sur lesquelles il repose (lexiques, grammaires, chaînes de traitements pré-syntaxiques). Considérons successivement plus en détails ces trois aspects.

2.1 Développement d'un système d'analyse non-linéaire, efficace, robuste et linguistiquement pertinent

Le développement d'un tel système passe tout d'abord par la définition d'un formalisme non-linéaire de description linguistique dont les propriétés algorithmiques doivent rester raisonnables (complexité polynomiale de l'analyse). Il semble possible d'étendre pour cela le formalisme des Méta-RCG, en étudiant en particulier à quelles conditions et de quelle façon certains traits pourraient être traités par des équations fonctionnelles à la LFG, sans pour autant subir les inconvénients des formalismes à décorations. Il serait intéressant de définir de façon théorique puis d'implémenter de façon efficace des techniques permettant l'obtention d'analyseurs Méta-RCG robustes, à l'image de SXLFG. La partie théorique de ce travail pourra s'appuyer entre autres sur une étude des rapports qu'entretiennent les RCG avec certaines classes d'automates à files (*thread automata*), d'automates de parcours d'arbres ou d'extensions des automates à piles (comme les iterative push-down automata), ainsi qu'avec la logique monadique du second ordre.

Il sera nécessaire également de poursuivre les travaux présentés au chapitre 9 sur la désambiguïsation, à la fois en cours d'analyse, avec la problématique de la remise en cause de décisions en cas d'élagage excessif, et en sortie d'analyse, à l'aide de modèles probabilistes. Ce travail, déjà commencé en collaboration avec Alexis Nasr et Pierre Boullier, doit être poursuivi. En particulier, le fait que les Méta-RCG soient un formalisme intégré, où interagissent des contraintes de toutes natures (morphologie, syntaxe, sémantique,...) n'interdit pas que l'on puisse définir différentes familles associées à différents types de techniques de désambiguïsation : heuristiques au niveau des *chunks*, modèles probabilistes au niveau des dépendances sémantiques, etc.

Il faudrait également développer une grammaire du français à large couverture dans la future version des Méta-RCG, en partant des grammaires utilisées dans SXLFG et de la grammaire Méta-RCG actuelle. À cet égard, une collaboration avec des spécialistes du TAL à forte compétence linguistique permettra d'allier justesse des descriptions linguistiques et rapidité du développement de la grammaire. En particulier, il semble prometteur de travailler sur les phénomènes de coordination, qui posent un certain nombre de problèmes aux formalismes usuels, et plus spécifiquement sur la coordination elliptique. À plus long terme, il serait intéressant de développer des systèmes d'analyse pour des langues réputées difficiles à traiter par les formalismes usuels, et en particulier des langues pour lesquelles l'ordre des mots est plus libre (allemand, langues slaves,...). Dans ces développements, des techniques d'identification automatique d'erreurs et de manques telles que ce qui est présenté au chapitre 7 seront utiles.

Enfin, le formalisme des Méta-RCG, et *a fortiori* sa version future, permet, outre les contraintes morphologiques, syntaxiques et sémantiques, d'inclure dans la grammaire une modélisation des

contraintes discursives (comme évoqué très rapidement au chapitre 10). C'est une piste à explorer, faisant ainsi passer les analyseurs de l'échelle de la phrase à celle du discours.

2.2 Représentation et acquisition automatique d'informations lexicales

Toutefois, le développement d'un tel système d'analyse passe par la disponibilité d'informations lexicales à la fois riches et précises. C'est d'autant plus vrai que l'on met en œuvre plus de contraintes d'ordres différents. En particulier, la prise en compte d'informations sémantiques ne peut se faire qu'à la condition de disposer de ces informations au niveau du lexique.

C'est pourquoi il sera indispensable de poursuivre les travaux présentés dans la deuxième partie de cette thèse sur la représentation et l'acquisition automatique d'informations lexicales, et ce dans plusieurs directions. Tout d'abord, il faudra faire un usage à grande échelle des modèles et des techniques déjà mises en place pour prendre en compte la morphologie dérivationnelle au sein du mécanisme d'acquisition automatique de lexiques morphologiques. Ceci permettra une forte réduction du nombre d'informations lexicales à acquérir : de nombreux mots sont des dérivés morphologiques d'autres mots, appelés bases, et leurs propriétés morphologiques, syntaxiques et même sémantiques (voir les travaux en cours de Fiammetta Namer) peuvent être obtenues directement à partir de celles de leurs bases.

Les mots qui ne sont pas des dérivés morphologiques sont malgré tout extrêmement nombreux. C'est pourquoi il est indispensable de disposer de techniques d'acquisition automatique d'informations lexicales, bien qu'il faille naturellement laisser en dernier recours au linguiste la tâche de valider ou non les informations extraites par la machine. Mais un certain nombre de travaux, tels que ceux présentés aux chapitres 6 et 7, montrent que des méthodes d'acquisition adaptées permettent à la phase de validation, par rapport à une spécification entièrement manuelle, d'être moins coûteuse en temps de plusieurs ordres de grandeur (de telles méthodes ont aussi l'avantage d'être plus reproductibles et donc plus complètement scientifiques que les techniques de développement entièrement manuelles). En particulier, il me semble nécessaire d'aller plus loin dans le développement de méthodologies d'acquisition automatique d'informations syntaxiques, soit à partir de corpus bruts (ou étiquetés), soit à partir du résultat de l'analyse syntaxique de gros corpus (par SXLFG, par exemple). Enfin, puisque notre futur système d'analyse prendra en compte les informations sémantiques, à la fois qualitatives et quantitatives (pour la désambiguïsation probabiliste, par exemple), il faudra mettre en place diverses techniques d'acquisition automatique d'informations sémantiques.

Il semble qu'il y ait là un véritable paradigme d'acquisition automatique d'informations lexicales de plus en plus riches : des informations lexicales d'un niveau donné permettent la construction d'outils d'analyse de corpus correspondant à ce niveau ; puis le résultat de l'ana-

lyse de gros corpus par ces outils permet l'acquisition automatique, avec validation manuelle par un linguiste, d'informations lexicales de niveau immédiatement supérieur. Ce paradigme de construction progressive et assistée par la machine de ressources lexicales permet un développement bien plus rapide de ces ressources.

En parallèle, il est indispensable de tout mettre en œuvre pour mettre à profit les travaux d'autres équipes dans le développement de ces ressources complexes que sont les lexiques. Des travaux sont en cours, par exemple, sur la normalisation des lexiques, et sur la comparaison et l'enrichissement mutuel entre lexiques (en particulier entre le Le*fff* et le lexique syntaxique construit par Claire Gardent et ses collaborateurs à partir des tables du lexique-grammaire.

2.3 Exploitation des systèmes d'analyse automatique

Le système d'analyse automatique du français esquissé ci-dessus permettra le développement de diverses applications, que l'on peut regrouper en trois grandes familles.

Tout d'abord, il est possible d'utiliser un tel système (ou, dans un premier temps, un système tel que SXLFG), pour analyser des corpus très volumineux Sagot et Boullier (2006). Le résultat de telles expériences, outre l'acquisition d'informations lexicales comme évoqué précédemment, permettra en effet d'identifier automatiquement les points faibles du système d'analyse, en particulier au niveau de la grammaire et du lexique (voir chapitre 7. Ces travaux, qui doivent être appronfondis, permettent un développement accéléré des ressources linguistiques (lexiques et grammaires), mais également de l'indispensable chaîne de traitements présyntaxiques (SXPipe) qui transforme les textes bruts en entrées valides pour les analyseurs.

Ensuite, l'évaluation qualitative des résultats d'analyse de corpus est inévitable. À cet égard, la campagne EASy d'évaluation des analyseurs syntaxiques (voir chapitre 12) a constitué un point de départ. En effet, il est indispensable de confronter les modèles et les outils linguistiques à la complexité de corpus réels, avec tous les aléas que cela comporte. Les applications des systèmes d'analyse ne seront en effet pertinentes qu'à la condition de ne pas se limiter aux phrases forgées par des linguistes.

Enfin, dans une perspective plus large, les systèmes d'analyse prennent tout leur sens dans un certain nombre d'applications. À court et moyen terme, il pourra s'agir d'acquisition automatique d'ontologies pour des domaines spécialisés (comme déjà ébauché dans des travaux au sein d'Atoll), ou de fouille de textes et de recherche d'informations. À plus long terme des travaux pourraient être menés sur la traduction automatique. Un tel domaine de recherche aurait le double avantage d'aller au-delà des seules techniques d'analyse (transfert de langue à langue, génération automatique, analyse de langues autres que le français) et d'être au cœur des besoins et des attentes du grand public.

Bibliographie

ABEILLÉ, A. (1993). *Les nouvelles syntaxes*. Armand Colin Éditeur, Paris, France. ISBN 2-200-21096-5.

ABEILLÉ, A., BARRIER, N., BARRIER, S. et GERDES, K. (2002). *La grammaire LTAG du français (FTAG)*. TALaNa, Univ. Paris 7.

ABNEY, S. P. (1991). Parsing by chunks. Dans BERWICK, R. C., ABNEY, S. P. et TENNY, C., rédacteurs, *Principle-Based Parsing : Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht.

ABU (1999). Association des bibliophiles universels, dictionnaire des mots communs. URL `http://abu.cnam.fr/DICO/mots-communs.html`

AJDUKIEWICZ, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, **1** : 1–27.

ANDREWS, A. (1990). Functional closure in LFG. Rapport technique, The Australian National University.

BAR-HILLEL, Y. (1953). A quasi arithmetical notation for syntactic description. *Language*, **29** : 47–58.

BARTHÉLEMY, F., BOULLIER, P., DESCHAMP, P. et ÉRIC DE LA CLERGERIE (2001). Guided parsing of range concatenation languages. Dans *Actes de ACL'01*, pages 42–49. Toulouse, France.

BASILI, R., PAZIENZA, M. T. et VELARDI, P. (1994). A "not-so-shallow" parser for collocational analysis. Dans *COLING*, pages 447–453.

BECKER, T., JOSHI, A. et RAMBOW, O. (1991). Long-distance scrambling and tree adjoining grammars. Dans *Actes de EACL-91*, pages 21–26.

BECKER, T., RAMBOW, O. et NIV, M. (1992). The derivational generative power of formal systems, or scrambling is beyond LCFRS. Rapport technique IRCS-92-38, University of Pennsylvania.

BENVENISTE, É. (1966). *Problèmes de linguistique générale*. TEL Gallimard.

BERGER, A., DELLA PIETRA, S. et DELLA PIETRA, V. (1996). A maximun entropy approach to natural language processing. *Computational Linguistics*, **22**(1) : pp. 39–71.

BESCHERELLE, L.-N. (1990). *La conjugaison — Dictionnaire de douze mille verbes*. Hatier, Paris, France. ISBN 2-218-01660-5.

BLACHE, P. (2001). *Les grammaires de propriétés*. Hermès Sciences Publications, Paris, France. ISBN 2-746-20236-0.

BLACK, E., JELINEK, F., LAFFERTY, J. D., MAGERMAN, D. M., MERCER, R. L. et ROUKOS, S. (1992). Towards history-based grammars : Using richer models for probabilistic parsing. Dans *Proceedings DARPA Speech and Natural Language Workshop*, pages 134–139.

BLANCHE-BENVENISTE, C., DELOFEU, J., STEFANINI, J. et VAN DEN EYNDE, K. (1984). *Pronom et syntaxe. L'approche pronominale et son application au français*. SELAF, Paris.

BOULLIER, P. (1999). Chinese numbers, mix, scrambling and range concatenation grammars. Dans *Actes de EACL 99*, pages 53–60.

BOULLIER, P. (2000). On tag parsing. *Traitement automatique des langues*, **41**(3) : 761–793.

BOULLIER, P. (2003a). Counting with range concatenation grammars. *Theoretical Computer Science*, **293** : 391–416.

BOULLIER, P. (2003b). Guided Earley parsing. Dans *Actes du International Workshop on Parsing Technologies (8ème IWPT)*, pages 43–54. Nancy, France.

BOULLIER, P. (2003c). Supertagging : a non-statistical parsing-based approach. Dans *Actes de IWPT'03*, pages 55–65. Nancy, France.

BOULLIER, P. (2004). Range concatenation grammars. Dans *Recent developments in parsing technology*, pages 269–289. Kluwer Academic Publishers.

BOULLIER, P., CLÉMENT, L., SAGOT, B. et VILLEMONTE DE LA CLERGERIE, E. (2005). « simple comme easy :-) ». Dans *Actes de TALN'05 EASy Workshop (poster)*, pages 57–60. ATALA, Dourdan, France.

BOULLIER, P. et DESCHAMP, P. (2004). Le système SYNTAX™ – manuel d'utilisation et de mise en œuvre sous UNIX™. En ligne sur `www.limsi.fr/Recherche/CORVAL/easy/PEAS_reference_annotations_v1.6.html`.

BOULLIER, P. et JOURDAN, M. (1987). A new error repair and recovery scheme for lexical and syntactic analysis. *Science of Computer Programming*, **9** : 271–286.

BOULLIER, P. et SAGOT, B. (2005a). Analyse syntaxique profonde à grande échelle : SXLFG. *Traitement Automatique des Langues (T.A.L.)*, **46**(2).

BOULLIER, P. et SAGOT, B. (2005b). Efficient and robust LFG parsing : SXLFG. Dans *Actes de IWPT'05*. Vancouver, Canada. 1-10.

BOURIGAULT, D. et FRÉROT, C. (2005). Acquisition et évaluation sur corpus de propriétés de sous-catégorisation syntaxique. Dans *Actes de TALN'05*. Dourdan, France.

BRENT, M. R. (1991). Automatic acquisition of subcategorization frames from untagged text. Dans *Actes de ACL'91*, pages 209–214.

BRIFFAULT, X., CHIBOUT, K., SABAH, G. et VAPILLON, J. (1997). An object-oriented linguistic engineering environment using LFG (Lexical-Functional Grammar) and CG (Conceptual Graphs). Dans *Actes du workshop Computational Environments for Grammar Development and Linguistic Engineering (ACL'97)*.

BRISCOE, T. et CARROLL, J. (1997). Automatic extraction of subcategorization from corpora. Dans *Actes de la 5ème Conference on Applied Natural Language Processing*. Washington, DC.

CANDITO, M.-H. (1996). A principle-based hierarchical representation of LTAGs. Dans *Proceedings of COLING-96*.

CANDITO, M.-H. (1999). *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées*. Thèse de doctorat, Université Paris 7.

CLÉMENT, L. et KINYON, A. (2001). XLFG – an LFG parsing scheme for French. Dans *Actes de LFG'01*. Hong Kong.

CLÉMENT, L. et KINYON, A. (2003). Generating parallel multilingual lfg-tag grammars from a MetaGrammar. Dans *Actes de ACL'03*. Sapporo, Japon.

CLÉMENT, L. et DE LA CLERGERIE, É. (2004). Terminology and other language resources – Morpho-Syntactic Annotation Framework (MAF). ISO TC37SC4 WG2 Working Draft. URL `http://atoll.inria.fr/RNIL/TC37SC4-docs/draft-MAF-en.pdf`

CLÉMENT, L. et SAGOT, B. (2004). Site internet du Le*fff* (Lexique des Formes Fléchies du Français). `www.lefff.net`.

CLÉMENT, L., SAGOT, B. et LANG, B. (2004). Morphology Based Automatic Acquisition of Large-coverage Lexica. Dans *Actes de LREC'04*, pages 1841–1844. Lisbonne, Portugal.

CRABBÉ, B., GAIFFE, B. et ROUSSANALY, A. (2004). Représentation et gestion du lexique d'une grammaire d'arbres adjoints. *Traitement Automatique des Langues*, **43**(3).

DAILLE, B. (2000). Morphological rule induction for terminology acquisition. Dans *Actes de COLING'00*, pages 215–221. Saarbrucken, Allemagne. ISBN 1-55860-717-X.

DAL, G., HATHOUT, N. et NAMER, F. (2005). Morphologie constructionnelle et traitement automatique des langues : le projet MorTAL. *Lexique*, **16**. à paraître.

DANLOS, L. et LAURENS, O. (1991). Présentation du projet Eurotra et des grammaires d'Eurotra-France. Rapport technique 1, Université Paris 7 - Talana/LISH.

DENDIEN, J. et PIERREL, J.-M. (2003). Le trésor de la langue française informatisé : un exemple d'informatisation d'un dictionnaire de langue de référence. *Traitement Automatique des Langues*, **44**(2).

DORR, B. J. et JONES, D. (1995). Automatic extraction of semantic classes from syntactic information in online resources. Rapport technique CS-TR-3481, University of Maryland.

DUNNING, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, **19**(1) : 61–74.

EARLEY, J. (1970). An efficient context-free parsing algorithm. *Communication of the ACM*, **13**(2) : 94–102.

ERJAVEC, T. et DŽEROSKI, S. (2004). Machine learning of morphosyntactic structure : lemmatizing unknown slovene words. *Applied Artificial Intelligence*, **18** : 17–41.

EYNDE, K. v. d. et MERTENS, P. (2003). La valence : l'approche pronominale et son application au lexique verbal. *French Language Studies*, **13**(1) : 63–104.

GARDENT, C., GUILLAUME, B., PERRIER, G. et FALK, I. (2005). Maurice gross' grammar lexicon and natural language processing. Dans *Actes de L&TC'O5*. Poznań, Pologne.

GENDNER, V. et VILNAT, A. (2004). Les annotations syntaxiques de référence PEAS. En ligne sur `www.limsi.fr/Recherche/CORVAL/easy/PEAS_reference_annotations_v1.6.html`.

GERDES, K. et KAHANE, S. (2001). Word order in german : A formal dependency grammar using a topological hierarchy. Dans *Actes de ACL'01*, pages 220–227. Toulouse, France.

GINSBURG, S. (1975). *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland/Elsevier, Amsterdam, Oxford/New-York.

GOFFIC, P. L. (1997). *Les formes conjuguées du verbe français, oral et écrit*. Ophrys (Coll. L'Essentiel français), Paris. ISBN 2-7080-0831-5.

GRABAR, N. et ZWEIGENBAUM, P. (1999). Acquisition automatique de connaissances morphologiques sur le vocabulaire médical. Dans *Actes de TALN 99*. Cargèse, France.

GREFENSTETTE, G. et TAPANAINEN, P. (1994). What is a word, what is a sentence ? Problems of tokenization. Dans *Actes de CCLTR*. Budapest, Hongrie.

GROENINK, A. (1995). Literal movement grammars. Dans *Actes de EACL 7*. Dublin, Irlande.

GROENINK, A. (1996). Mild context-sensitivity and tuple-based generalizations of context-free grammar. Dans JOHNSON, D. et MOSS, L., rédacteurs, *Actes de MoL 4*. Linguistics and Philosophy.

GROUP, T. X. R. (1995). A lexicalized tree adjoining grammar for english. *Technical Report IRCS Report 95-03, The Institute for Research in Cognitive Science, Univ. of Pennsylvania.*.

HINDLE, D. et ROOTH, M. (1991). Structural ambiguity and lexical relations. Dans *Meeting of the Association for Computational Linguistics*, pages 229–236.

HUET, G. (2002). The zen computational linguistics toolkit. Dans *Cours à ESSLLI 2002*. Vienne, Autriche.
URL `http://pauillac.inria.fr/~huet/PUBLIC/esslli.pdf`

HUET, G. (2004). Design of a lexical database for sanskrit. Dans *Actes du Workshop on Electronic Dictionaries de COLING'04*, pages 8–14. Genève, Suisse.

IDE, N. et VÉRONIS, J. (1994). MULTEXT : Multilingual text tools and corpora. Dans *Actes de COLING'94*, tome 1, pages 588–592. Kyoto, Japon.

JAZYKOVEDNÝ ÚSTAV L'. ŠTÚRA SAV (2004). Slovenský národný korpus (Slovak National Corpus). URL : http ://korpus.juls.savba.sk.

JOSHI, A. (1985). How much context-sensitivity is necessary for assigning structural descriptions : Tree adjoining grammars. Dans DOWTY, D., KARTTUNEN, L. et ZWICKY, A., rédacteurs, *Natural Language Processing*. Cambridge University Press, New-York.

JOSHI, A. K. (1987). An introduction to tree adjoining grammars. Dans MANASTER-RAMER, A., rédacteur, *Mathematics of Language*, pages 87–114. John Benjamins, Amsterdam.

KAHANE, S. (2004). Grammaires d'unification polarisées. Dans *Actes de TALN 04*. Fès, Maroc.

KAHANE, S. (2005). Grammaire d'unification sens-texte : modularité et polarisation. Dans *Actes de TALN'05*. Dourdan, France.

KAHANE, S. (2006). On the status of phrases in head-driven phrase structure grammar : Illustration by a totally lexical treatment of extraction. Benjamins.

KAJI, Y., NAKANISHI, R., SEKI, H. et KASAMI, T. (1992). The universal recognition problems for parallel multiple context-free grammars and for their subclasses. *IEICE*, **E75-D**(4) : 499–508.

KALLMEYER, L. (1997). Local tree description grammars. Dans BECKER, T. et KRIEGER, H.-U., rédacteurs, *Actes de MoL 5*, pages 77–84.

KAPLAN, R. (1989). The formal architecture of lexical functionnal grammar. *Journal of Informations Science and Engineering*.

KAPLAN, R. et BRESNAN, J. (1982). Lexical-functional grammar : a formal system for grammatical representation. Dans BRESNAN, J., rédacteur, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Massachusetts.

KAPLAN, R. et KAY, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, **20**(3) : 331–379.

KAPLAN, R., RIEZLER, S., KING, T., MAXWELL, J., VASSERMAN, A. et CROUCH, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. Dans *Actes de HLT/NAACL*. Boston, Massachusetts.

KAPLAN, R. M. et MAXWELL, J. T. (1994). Grammar writer's workbench, version 2.0. Rapport technique, Xerox Corporation.

KARTTUNEN, L. (2006). Twenty-five years of finite-state morphology. Dans SUOMINEN, M., rédacteur, *Studies in Honor of Kimmo Koskenniemi*. CSLI Publications, Stanford. To appear.

KORHONEN, A., GORRELL, G. et MCCARTHY, D. (2000). Statistical filtering and subcategorization frame acquisition. Dans *Actes de la Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 199–205. Hong Kong.

KORHONEN, A. et PREISS, J. (2003). Improving subcategorization acquisition using word sense disambiguation. Dans *Actes de ACL'03*, pages 48–55. Sapporo, Japan.

KOSKENNIEMI, K. (1983). Two-level model for morphological analysis. Dans BUNDY, A., rédacteur, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 683–685. William Kaufmann, Inc., Karlsruhe, Allemagne de l'Ouest.

KUKICH, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, **24**(4) : 377–439.

LAMBEK, J. (1958). The mathematics of sentence structure. *American mathematical monthly*, **65** : 154–169.

LANCELOT, C. et ARNAULD, A. (1660). Grammaire générale et raiſonnée. Pierre le Petit, Paris.

LEVIN, B. (1993). *English Verb Classes and Alternations : A Preliminary Investigation*. The University of Chicago Press, Chicago, Illinois. ISBN 0-226-47533-6.

MANASTER-RAMER, A. (1987). Dutch as a formal language. *Linguistics and Philosophy*, **10** : 221–246.

MANNING, C. D. (1993). Automatic acquisition of a large subcategorization dictionary from corpora. Dans *Actes de ACL'93*, pages 235–242.

MAYNARD, D., TABLAN, V., URSU, C., CUNNINGHAM, H. et WILKS, Y. (2001). Named entity recognition from diverse text types. Dans *Actes de RANLP 2001*. Tzigov Chark, Bulgarie.

MICHAELIS J., K. M. (1996). Semilinearity as a syntactic invariant. Dans RETORÉ, C., rédacteur, *Actes de LACL '96*, pages 329–345.

MILLER, G. A., BECKWITH, R., FELLBAUM, C., GROSS, D. et MILLER, K. (1990). Introduction to wordnet : An on-line lexical database. *International Journal of Lexicography*, **3(4)** : 235–312.

MIYAO, Y. et TSUJII, J. (2002). Maximum entropy estimation for feature forests. Dans *Actes de HLT*. San Diego, Californie.

NAMER, F. et SCHMIDT, P. (1995). Construction d'un dictionnaire : morphologie à deux niveaux pour le français à l'aide de contraintes basées sur les structures de traits typés. Dans *Actes des 4èmes Journées Scientifiques du Réseau LTT de l'AUPELF-UREF, Lyon.*.

NASR, A. (1995). A formalism and a parser for lexicalised dependency grammars.

NASR, A. (2005). Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement. Habilitation à diriger des recherches, Université Paris 7.

VAN NOORD, G. (2004). Error mining for wide-coverage grammar engineering. Dans *Proc. of ACL 2004*. Barcelone, Espagne.

OLIVER, A., CASTELLÓN, I. et MÀRQUEZ, L. (2003). Use of internet for augmenting coverage in a lexical acquisition system from raw corpora : application to russian. Dans *IESL Workshop of RANLP '03, Bulgaria*. Borovets, Bulgarie.

OLIVER, A. et TADIĆ, M. (2004). Enlarging the croatian morphological lexicon by automatic lexical acquisition from raw corpora. Dans *Actes de LREC'04*, pages 1259–1262. Lisbonne, Portugal.

PERRIER, G. (2000). Interaction grammars. Dans *Actes de ACL'00*, pages 600–606. Saarbrücken, Allemagne.

POLLARD, C. et SAG, I. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.

PUSTEJOVSKY, J. (1995). *The Generative Lexicon*. The MIT Press, Cambridge, Massachusetts.

RADZINSKI, D. (1991). Chinese number-names, tree adjoining grammars, and mild context-sensitivity. *Computational Linguistics*, **17**(3) : 277–299.

RAMBOW, O. et JOSHI, A. K. (1994a). *A Formal Look at Dependency Grammar and Phrase Structure Grammars, with Special consideration of Word Order Phenomena*. Leo Wanner, Pinter London, 94.

RAMBOW, O. et JOSHI, A. K. (1994b). A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. Dans WANNER, L., rédacteur, *Current Issues in Meaning-Text Theory*. Pinter, London, UK.

RIEZLER, S., KING, T., KAPLAN, R., CROUCH, R., MAXWELL, J. et JOHNSON, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. Dans *Actes de ACL'02*. University of Pennsylvania.

ROMARY, L., SALMON-ALT, S. et FRANCOPOULO, G. (2004). Standards going concrete : from lmf to morphalou. Dans *Actes du Workshop on Electronic Dictionaries de Coling 2004*. Genève, Suisse.

SAGOT, B. (2005a). Automatic acquisition of a slovak lexicon from a raw corpus. Dans *Lecture Notes in Artificial Intelligence 3658 (© Springer-Verlag), Actes de TSD'05*, pages 156–163. Karlovy Vary, Tchéquie.

SAGOT, B. (2005b). Les Méta-RCG : description et mise en oeuvre. Dans *Actes de TALN'05 (poster)*, pages 493–498. ATALA, Dourdan, France.

SAGOT, B. (2005c). Linguistic facts as predicates over ranges of the sentence. Dans *Lecture Notes in Computer Science 3492 (© Springer-Verlag), Actes de LACL'05*, pages 271–286. Bordeaux, France.

SAGOT, B. et BOULLIER, P. (2004). Les RCG comme formalisme grammatical pour la linguistique. Dans *Actes de TALN 04*, pages 403–412. Fès, Maroc.

SAGOT, B. et BOULLIER, P. (2005). From raw corpus to word lattices : robust pre-parsing processing. Dans *Actes de L&TC 2005*. Poznań, Pologne.

SAGOT, B. et BOULLIER, P. (2006). Deep non-probabilistic parsing of large corpora. Dans *Actes de LREC'06*. Gênes, Italie. à paraître.

SAGOT, B., CLÉMENT, L., ÉRIC VILLEMONTE DE LA CLERGERIE et BOULLIER, P. (2006). The L*efff* 2 syntactic lexicon for french : architecture, acquisition, use. Dans *Actes de LREC'06*. Gênes, Italie. à paraître.

SAGOT, B., CLÉMENT, L., VILLEMONTE DE LA CLERGERIE, E. et BOULLIER, P. (2005). Vers un méta-lexique pour le français : architecture, acquisition, utilisation. Journée d'étude de l'ATALA sur l'Interface lexique-grammaire et lexiques syntaxiques et sémantiques.

SAGOT, B. et ÉRIC DE LA CLERGERIE (2006). Trouver le coupable : Fouille d'erreurs sur des sorties d'analyseurs syntaxiques. Dans *Actes de TALN'06*. Louvain, Belgique. Soumis.

SAINT-DIZIER, P. (1998). Sense variation and lexical semantics generative operations. Dans POWERS, D. M. W., rédacteur, *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning : NeMLaP3/CoNLL98*, pages 121–130.

SCHIEBER, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, **8** : 333–343.

SCHONE, P. et JURAFSKY, D. (2001a). Is knowledge-free induction of multiword unit dictionary headwords a solved problem ? Dans *Actes de EMNLP 2001*, pages 100–108.

SCHONE, P. et JURAFSKY, D. (2001b). Knowledge-free induction of inflectional morphologies. Dans *Actes de NAACL'01*, pages 1–9. Pittsburgh, Pennsylvania.

SEDDAH, D. (2004). *Synchronisation des connaissances syntaxiques et sémantiques pour l'analyse d'énoncés en langage naturel à l'aide des grammaires d'arbres adjoints lexicalisées*. Thèse de doctorat, Université Nancy 1.

SEDDAH, D. et SAGOT, B. (2006). Modélisation et analyse des coordinations elliptiques par l'exploitation dynamique des forêts d'analyse. Dans *Actes de TALN'06*. Louvain, Belgique. Soumis.

SEKI, H., MATSUMURA, T., FUJII, M. et KASAMI, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, **88**(2) : 191–229.

STABLER, E. P. (2003). Varieties of crossing dependencies : Structure dependence and mild context sensitivity. *Cognitive Science*, **28**(5).

STEEDMAN, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, **261** : 523–568.

TANGUY, L. et HATHOUT, N. (2002). Webaffix : un outil d'acquisition morphologique dérivationnelle à partir du web. Dans *Actes de TALN'02*, pages 245–254. Nancy, France.

THOMASSET, F. et ÉRIC VILLEMONTE DE LA CLERGERIE (2005). Comment obtenir plus des métagrammaires. Dans *Actes de TALN'05*. Dourdan, France.

VIJAY-SHANKER, K., WEIR, D. J. et JOSHI, A. K. (1987). Characterizing structural descriptions produced by grammatical formalisms. Dans *Actes de ACL'87*, pages 104–111.

VILLEMONTE DE LA CLERGERIE, E. (2005). DyALog : a tabular logic programming based environment for NLP. Dans *Actes du 2ème International Workshop on Constraint Solving and Language Processing (CSLP'05)*. Barcelona, Spain.

VOSSEN, P., DIEZ-ORZAS, P. et PETERS, W. (1997). Multilingual design of EuroWordNet. Dans VOSSEN, P., ADRIAENS, G., CALZOLARI, N., SANFILIPPO, A. et WILKS, Y., rédacteurs, *Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 1–8.

ZWEIGENBAUM, P., HADOUCHE, F. et GRABAR, N. (2003). Apprentissage de relations morphologiques en corpus. Dans *Proc. of TALN'03*. Batz-sur-Mer, France.