Université Paris 7
UFR d'informatique

Habilitation à diriger des recherches

# Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement

Alexis Nasr

présentée le 14 décembre 2004

Jury :

| | |
|---|---|
| Mr Guy Cousineau | Rapporteur |
| Mme Laurence Danlos | Directeur |
| Mr Aravind Joshi | Rapporteur |
| Mr Sylvain Kahane | Examinateur |
| Mr Eric Laporte | Rapporteur |
| Mr Martin Rajman | Rapporteur |

Laboratoire d'accueil : Lattice CNRS UMR 8094

# Table des matières

# Conventions, sigles et notations

Nous allons être amenés à manipuler, dans ce document, de nombreux types différents d'objets. Afin de simplifier la lecture du document, nous allons avoir recours à un ensemble de conventions, sigles et notations, décrits ci-dessous.

## Conventions

Dans la mesure du possible, nous utiliserons les mêmes identifieurs pour désigner des objets d'un même type. Ces identifieurs sont représentés ci-dessous, avec le type des objets qu'ils désignent.

| Identifieur | Type de l'objet |
|---|---|
| $A$ | automate |
| $c$ | catégorie |
| $F$ | forêt de dépendances partagées |
| $G$ | grammaire |
| $m$ | mot |
| $e$ | état d'un automate |
| $\mathcal{R}$ | treillis de catégories |
| $S$ | phrase |
| $t$ | transition d'un automate |
| $T$ | arbre |
| $\mathcal{T}$ | table d'analyse |
| $X$ | sous-analyse |
| $Y$ | composante d'une sous-analyse |

## Sigles

| | |
|---|---|
| FDP | forêt de dépendances partagées |
| GD | grammaires de dépendances de Hays/Gaifman |
| GDE | grammaires de dépendances étendues |
| GDG | grammaires de dépendances génératives |
| GDGP | grammaires de dépendances génératives probabilistes |
| GHC | grammaires hors-contextes |
| GHCP | grammaires hors-contextes probabilistes |
| PTAG | grammaires d'arbres adjoints probabilistes |
| TAG | grammaires d'arbres adjoints (Tree Adjuction Grammars) |
| TIG | grammaires d'insertion d'arbres (Tree Insertion Grammars) |
| TST | théorie Sens-Texte |
| TAL | traitement automatique des langues |

## Notations principales

Un grand nombre de notations sera introduit au cours du document. Les principales sont reprises ci-dessous.

| | |
|---|---|
| $T_G(S)$ | ensemble des arbres que la grammaire $G$ associe à la phrase $S$ |
| $P_G(S)$ | probabilité que la grammaire $G$ associe à la phrase $S$ |
| $\hat{T}_G(S)$ | arbre de probabilité maximale que la grammaire $G$ associe à la phrase $S$ |
| $\hat{P}_G(S)$ | probabilité de $\hat{T}_G(S)$ |
| $\textsc{fdp}_G(S)$ | forêt de dépendances partagées que la grammaire $G$ associe à la phrase $S$ |
| | |
| $\mathcal{Q}_a(A)$ | ensemble des états d'acceptation de l'automate $A$ |
| $e_0(A)$ | état initial de l'automate $A$ |
| $\delta(A)$ | ensemble des transitions de l'automate $A$ |
| $\delta^A(e, f, c)$ | ensemble des transitions émanant de l'état $e$ de l'automate $A$ étiquetées par le couple $\langle f, c\rangle$ |
| $\delta^A(e, c)$ | ensemble des transitions émanant de l'état $e$ de l'automate $A$ étiquetées par un couple de la forme $\langle f, c\rangle$, avec $f$ quelconque |
| $accept(A, e)$ | prédicat valant *vrai* si $e$ est un état d'acceptation de l'automate $A$ |
| $dest(t)$ | état destination de la transition $t$ |
| $orig(t)$ | état origine de la transition $t$ |
| $fct(t)$ | étiquette fonctionnelle de la transition $t$ |
| $cat(t)$ | étiquette catégorielle de la transition $t$ |
| $p(t)$ | probabilité de la transition $t$ |
| | |
| $Auto(X)$ | automate de la sous-analyse $X$ |
| $e(X)$ | état courant de la sous-analyse $X$ |
| $i(X)$ | indice du début de segment de phrase correspondant à la sous-analyse $X$ |
| $j(X)$ | indice de fin de segment de phrase correspondant à la sous-analyse $X$ |
| $\mathcal{Y}(X)$ | ensemble des composantes de la sous-analyse $X$ |
| $\mathcal{E}(X)$ | extension de la sous-analyse $X$ |
| | |
| $t(Y)$ | transition de la composante $Y$ |
| $X_G(Y)$ | sous-analyse gauche de la composante $Y$ |
| $X_D(Y)$ | sous-analyse droite de la composante $Y$ |
| $\mathcal{E}(Y)$ | extension de la composante $Y$ |

# 1 Introduction

Ce document résume les travaux effectués dans le cadre de ma principale activité de recherche des quatre dernières années. Ces travaux portent sur la réalisation d'un système d'analyse syntaxique probabiliste pour grammaires de dépendances dont la grammaire a été extraite automatiquement d'un corpus. Les raisons qui m'ont poussé à m'intéresser à ce problème résultent d'un constat effectué à l'issue de ma thèse. La tâche de reformulation, qui composait ma problématique de thèse, supposait, dans le modèle que j'ai proposé, une phase d'analyse syntaxique des phrases à reformuler. A l'issue de cette étape, une structure syntaxique de dépendances[1] est construite, qui sert d'entrée aux étapes suivantes du traitement. Il s'agit d'une architecture assez classique en traitement automatique des langues naturelles (noté TAL dorénavant), que l'on retrouve dans la plupart des applications nécessitant une représentation explicite de la structure syntaxique des énoncés, tels que la traduction automatique ou, dans certains cas, les interfaces en langage naturel.

Cette architecture souffre de deux problèmes majeurs. Le premier concerne la couverture de la grammaire employée pour mener à bien l'analyse syntaxique. La grammaire développée durant ma thèse avait pour seul objectif de valider le formalisme grammatical (montrer que les phénomènes les plus courants du français pouvaient être raisonnablement bien représentés dans le formalisme) et les processus de traitement décrits dans la thèse. Sa couverture était par conséquent modeste. Cette façon de procéder est classique dans le domaine du TAL : on décrit dans un premier temps un formalisme que l'on valide avec une grammaire de faible couverture. Le développement d'une grammaire à large couverture constitue en général un autre travail qui est rarement mené à terme pour diverses raisons, en particulier le coût important de développement de la grammaire, la complexité de la tâche qui consiste à maintenir la cohérence d'une telle quantité de connaissances et la mise en lumière de certaines limites du formalisme, apparaissant au fur et à

[1] Nous reviendrons dans le chapitre 2 sur les structures de dépendances. Néanmoins, pour faciliter la lecture de cette introduction aux non initiés, en voici une définition rapide : les structures de dépendances constituent un outil de description de la syntaxe des phrases, fondé sur la notion de dépendance entre des couples de mots de ces dernières. Un des mots du couple est appelé gouverneur, et l'autre dépendant. La structure syntaxique d'une phrase se résume à indiquer, pour chaque mot de la phrase, son gouverneur. L'ensemble des dépendances de la phrase constitue un arbre, appelé arbre de dépendances, dont les mots sont représentés par les nœuds de l'arbre et les dépendances, par ses branches. Les grammaires de dépendances sont généralement opposées aux grammaires syntagmatiques, qui reposent sur la notion de constituant, ou syntagme.

mesure qu'avance la tâche de développement de la grammaire et qui jette un doute sur l'opportunité de poursuivre l'entreprise.

Le second problème est celui de l'ambiguïté, apparaissant de façon massive dans les systèmes de TAL, qui ne prennent en compte qu'une partie des informations habituellement mobilisées par les êtres humains pour l'analyse des énoncés. La conséquence est la construction d'analyses souvent aberrantes mais inévitables tant que des connaissances sémantiques, conceptuelles et pragmatiques ne sont pas prises en compte dans les traitements. Le problème se pose concrètement, dans le cadre d'un système comportant une étape d'analyse syntaxique, lorsqu'à l'issue de l'analyse d'une phrase, un grand nombre de structures syntaxiques différentes est produit, parmi lesquelles le système n'est pas en mesure de faire un choix.

Des solutions à ces deux problèmes ont commencé à apparaître grâce à deux évolutions dans le domaine du TAL.

La première est le développement de corpus annotés syntaxiquement, tel que le corpus du Penn Treebank ([Marcus et al., 1993]) ou, plus tard, le corpus LE MONDE ([Abeillé et al., 2003]) qui associent à chaque phrase du corpus, sa structure syntaxique. L'existence de tels corpus permet d'extraire automatiquement des grammaires à partir des structures syntaxiques contenues dans le corpus. Les grammaires produites permettent de couvrir en général, au minimum, le corpus duquel elles ont été extraites. Le premier problème, celui de la couverture de la grammaire, peut alors se ramener aux problèmes de la conception de procédures d'extraction de grammaires, ainsi qu'à celui de la représentativité du corpus duquel a été extrait la grammaire.

La seconde évolution est l'essor des méthodes probabilistes dans le domaine du TAL. Appliquées à l'analyse syntaxique, ces méthodes permettent d'associer une probabilité à chacune des analyses produites par un analyseur syntaxique. Moyennant un bon modèle probabiliste, de telles méthodes offrent une solution au problème du choix d'une analyse parmi les différentes analyses produites. Il suffit en effet de choisir, à l'issue de l'analyse syntaxique, l'analyse à laquelle le modèle probabiliste attribue la probabilité la plus élevée. Le second problème se ramène alors au problème, non trivial, de la conception d'un bon modèle probabiliste et à l'estimation de ses paramètres.

Le présent document décrit comment ces principes ont été mis en œuvre dans un système d'analyse syntaxique automatique pour grammaires de dépendances. Dans les quatre sections qui vont suivre, nous allons tenter de mieux définir différents aspects de ce travail. Dans la section 1.1, nous décrirons deux choix fondamentaux sur lesquels reposent notre travail. Le positionnement de ce dernier par rapport aux domaines de la linguistique, de l'informatique et du TAL sera précisé dans la section 1.2. La forme de ce document, qui n'est pas classique pour une habilitation à diriger des recherches, sera justifiée dans la section 1.3. Le plan du document sera décrit en 1.4 et nous terminerons cette introduction, en 1.5, par la présentation des personnes ayant contribué à ce projet.

## 1.1 Deux choix fondamentaux

L'objectif du travail décrit dans ce document est de construire un analyseur syntaxique probabiliste pour grammaires de dépendances dont la grammaire a été extraite de manière automatique d'un corpus annoté syntaxiquement. Les résultats de l'entreprise ne seront décrits que dans la troisième et dernière partie de ce document, dédiée aux expériences effectuées et à l'évaluation des résultats obtenus. Les deux premières parties vont décrire dans le détail les outils que nous avons développés pour réaliser les expériences. Ces outils reposent sur deux choix importants que nous allons résumer ici.

Le premier choix est celui des structures de dépendances comme moyen de représentation de la structure syntaxique des phrases. Mon intérêt pour les grammaires de dépendances provient de la théorie Sens-Texte ([Mel'čuk, 1988]) que j'avais choisie comme cadre théorique de mon travail de thèse, et qui a adopté les structures de dépendances comme mode de représentation de la syntaxe. Ma volonté de continuer à travailler sur le traitement automatique des grammaires de dépendances a été confortée par une recrudescence d'intérêt pour ces dernières, qui avaient reçu relativement peu d'attention dans la communauté du TAL au moment de ma thèse. Elles ont ensuite connu un certain succès à l'occasion de deux évolutions dans le domaine du TAL et dans certains travaux de linguistique formelle.

La première évolution, qui a en réalité commencé bien avant mes travaux de thèse, est le phénomène de lexicalisation des grammaires syntagmatiques, dont les grammaires d'arbres adjoints ([Joshi, 1985]) offrent un exemple. Le phénomène de lexicalisation, qui intègre la syntaxe dans le lexique en associant à toute entrée de ce dernier une structure syntaxique, a permis de mettre à jour des liens entre les grammaires syntagmatiques et les grammaires de dépendances ([Rambow & Joshi, 1997]). Ce mouvement a aussi contribué à remettre en avant certains avantages des grammaires de dépendances sur les grammaires syntagmatiques, tels que la possibilité de représenter plus facilement la syntaxe des langues à ordre libre ainsi que la proximité des structures de dépendances et des structures prédicat-argument souvent employées pour représenter le sens d'un énoncé. Cette proximité simplifie les processus de construction de représentations sémantiques à partir de structures syntaxiques.

La seconde évolution est plus récente. Elle concerne toute une série de travaux influents en analyse syntaxique probabiliste ([Magerman, 1995], [Eisner, 1996] [Charniak, 1997] et [Collins, 2003]). Ces travaux ont mis en avant l'importance de la relation entre les mots d'une phrase pour l'analyse probabiliste, redécouvrant ainsi la notion de dépendance syntaxique, tout en restant dans un cadre syntagmatique. Il en résulte des modèles hybrides, combinant grammaires syntagmatiques et grammaires de dépendances, qui gagneraient en simplicité s'ils étaient directement représentés sous la forme de grammaires de dépendances.

Ces événements nous poussent à croire que le développement d'outils de représentation et de traitement pour les grammaires de dépendances constitue un objectif utile pour le domaine du TAL. Nous nous intéresserons, en particulier, à la représentation des différentes structures syntaxiques d'une phrase sous la forme d'un objet complexe, appelé *forêt de dépendances partagées*, qui permet une représentation économique d'un grand nombre d'arbres de dépendances. Le développement de tels outils de représentation constitue, à nos yeux, une condition importante pour le développement du traitement automatique des grammaires de dépendances. En effet, l'ambiguïté à laquelle est confrontée l'analyse syntaxique - et, plus généralement, les applications de TAL- est telle, que la représentation d'un grand nombre d'analyses sous une forme compacte est nécessaire dans toute application comportant une étape d'analyse syntaxique. De tels outils de représentation ont fait l'objet de nombreuses études dans le cadre des grammaires syntagmatiques, mais ont été très peu étudiés pour les grammaires de dépendances. Nous espérons que notre travail contribuera à combler cette lacune.

Le second choix, plus technique, sur lequel repose ce travail est celui d'une analyse syntaxique en deux étapes : une étape d'*étiquetage grammatical* (*supertagging* en anglais), suivie de l'étape d'analyse syntaxique proprement dite. L'étiquetage grammatical, initialement proposé par [Bangalore & Joshi, 1999a], consiste à associer, à chaque mot de la phrase à analyser, un objet grammatical représenté par un identifieur que nous appellerons catégorie (*supertag* en anglais). Dans le cas du *supertagging*, l'objet grammatical correspondant à la catégorie est un arbre élémentaire TAG. Dans notre cas, il s'agit d'un *automate lexicalisé*, objet décrit en détail dans ce document. Le processus d'étiquetage grammatical est très proche de l'étiquetage morpho-syntaxique, mais ses conséquences sont plus importantes pour l'analyse syntaxique. En effet, dans le cas de l'étiquetage grammatical, l'analyse syntaxique est menée avec les seuls objets grammaticaux sélectionnés à l'issue de l'étiquetage. L'étiquetage grammatical a ainsi permis de déterminer un sous-ensemble de la grammaire qui sera utilisé pour mener à bien l'analyse syntaxique. Cette propriété découle de la lexicalisation de la grammaire, qui associe un élément lexical à tout objet syntaxique qu'elle définit.

Cette décomposition du processus d'analyse syntaxique nous a semblé constituer un cadre intéressant et original par rapport aux travaux actuels sur l'analyse syntaxique probabiliste, qui ne distinguent pas cette phase préalable de réduction de la grammaire. L'idée maîtresse de l'étiquetage grammatical est que cette étape préalable permet de simplifier grandement l'analyse syntaxique (pour reprendre les termes d'Aravind Joshi et Srinivas Bangalore : « *Supertagging is almost parsing* »). La tâche d'analyse syntaxique pour grammaires lexicalisées peut en effet être vue comme la réalisation d'un certain nombre de choix qui peuvent être de deux natures différentes : associer une catégorie à chaque mot de la phrase (sélectionnant ce faisant la grammaire utilisée par l'analyseur) et établir des dépendances entre les mots à l'aide de la grammaire sélectionnée. Dans un modèle composé d'une phase d'étiquetage grammatical suivie d'une analyse syntaxique, ces deux types de choix sont effectués par des processus différents : l'étiqueteur et l'analyseur. En extrayant d'un corpus des grammaires plus ou moins ambiguës, on peut répartir différemment le travail entre les deux modules. Une grammaire très ambiguë définira peu de catégories. La tâche de l'étiqueteur s'en trouvera simplifiée, celle de l'analyseur en deviendra plus ardue. A l'inverse, une grammaire moins ambiguë donnera plus de travail à l'étiqueteur et moins à l'analyseur. La recherche d'un bon équilibre entre le travail effectué par l'étiqueteur et l'analyseur en jouant sur les caractéristiques de la grammaire constitue à nos yeux un programme de recherche intéressant et original que nous avons eu envie d'explorer.

## 1.2 Positionnement dans le domaine du TAL

Le TAL se situe à cheval sur deux disciplines : la linguistique et l'informatique. De plus, il hérite de la première deux courants : un courant empirique, représenté par la linguistique de corpus (voir par exemple [McEnery & Wilson, 2001]) et un courant qualifié quelquefois de rationaliste, représenté principalement par l'école générativiste, qui voit la linguistique comme une branche des sciences cognitives (on trouvera, par exemple dans [Pollock, 1997], un résumé des thèses générativistes). Cette situation complexe permet de concevoir le TAL de différentes manières, selon la position que l'on adopte vis-à-vis de cette double dichotomie : linguistique/informatique et rationalisme/empirisme.

Le travail présenté ici se situe dans une perspective informatique et empirique. Son but est la réalisation d'un système d'analyse syntaxique et son évaluation sur des corpus. Le formalisme syntaxique sur lequel il repose est, comme on le verra, assez fruste. Il s'agit d'un formalisme conçu spécifiquement pour le TAL. Son objectif n'est pas d'offrir un outil de description permettant de traiter des phénomènes syntaxiques réputés difficiles. Il est de permettre l'écriture de grammaires de couverture suffisante pour analyser les corpus que nous avons à notre disposition, tout en possédant les caractéristiques nécessaires pour un traitement efficace. Ce positionnement pose le problème de la validation d'un tel travail. Cette dernière sera faite par une évaluation quantitative des sorties de l'analyseur syntaxique sur un corpus de test, d'où la part relativement importante de ce document dédiée aux expérimentations et à l'évaluation des résultats.

## 1.3 Forme du document

La forme que nous avons choisi de donner à ce document n'est pas classique pour une habilitation à diriger des recherches. Nous avons en effet opté pour une sorte de mémoire, plus proche d'une thèse que d'une compilation d'articles. Deux raisons nous ont dicté ce choix. La première est, tout simplement, que les travaux et résultats présentés ici n'ont pas tous été publiés, en particulier les résultats obtenus sur le corpus LE MONDE. Ces résultats sont en effet très récents et seront prochainement soumis à publication. D'autre part, les parties de ce travail déjà publiées ont souvent été profondément remaniées afin de constituer un ensemble cohérent ou d'intégrer des améliorations postérieures à leur publication. Nous indiquerons néanmoins explicitement en tête de chaque chapitre les parties de celui-ci qui ont fait l'objet d'une publication.

La forme du document relève aussi d'un parti pris, celui d'une formalisation poussée des objets et des processus mis en œuvre. Cette formalisation nous est apparue nécessaire pour arriver à décrire de manière précise les différents aspects de ce travail. Ce processus conduit à la définition d'un grand nombre de types différents d'objets. Une telle profusion peut sembler *a priori* rebutante. Nous l'avons néanmoins choisie car, une fois établie, elle permet une description économique et précise de processus relativement complexes. Nous espérons que cette profusion d'objets formels ne nuira pas à la lecture du texte et nous tenterons de donner une description intuitive des objets avant leur définition formelle.

## 1.4 Plan du document

Ce document est composé de trois parties, intitulées respectivement « formalisme », « traitements » et « expériences ». La première décrit un formalisme grammatical, appelé grammaires de dépendances génératives (ou GDG), ainsi qu'une version probabiliste du formalisme, appelée grammaires de dépendances génératives probabilistes (ou GDGP). La seconde partie décrit un algorithme d'analyse pour grammaires GDG ainsi qu'un algorithme de recherche des arbres syntaxiques les plus probables d'une phrase, étant donné une GDGP. L'algorithme d'analyse prend, en entrée, une phrase et une grammaire pour produire une représentation compacte de l'ensemble des arbres syntaxiques de la phrase, appelé forêt de dépendances partagée (ou FDP). L'algorithme de recherche permet d'extraire d'une FDP les $n$ arbres les plus probables, étant donné une GDGP. La troisième partie décrit deux séries d'expériences effectuées sur le corpus LE MONDE et sur le Penn Treebank.

Nous avons représenté, dans la figure 1, un schéma général du système d'analyse syntaxique. Les éléments situés à l'intérieur du rectangle en pointillé sont décrits dans ce document. Les numéros des chapitres qui leurs sont consacrés sont indiqués à leurs côtés, dans des cercles[2]. Les autres composants du système ne font pas partie du travail présenté ici. Seuls les détails nécessaires à la bonne compréhension de l'ensemble seront brièvement décrits.

## 1.5 Contributions

Plusieurs personnes ont contribué à l'élaboration du système décrit ici. Mon collaborateur principal est Owen Rambow de l'université de Columbia, à New York. C'est avec lui que j'ai développé le formalisme des grammaires de dépendances probabilistes, ainsi que certains aspects de l'algorithme d'analyse. L'extraction des grammaires à partir du Penn Treebank a été réalisée par John Chen et Owen Rambow à l'université de Columbia. L'extraction des grammaires à partir du corpus LE MONDE a été réalisé par Ane Dybro Johansen à l'université Paris 7, à l'occasion de son stage de DEA. L'étiquetage grammatical du français et de l'anglais ont été effectués par François Tous-

[2]Les chapitres n'apparaissant pas dans la figure ne décrivent pas directement des parties du système, ils concernent des états de l'art ou des expériences.

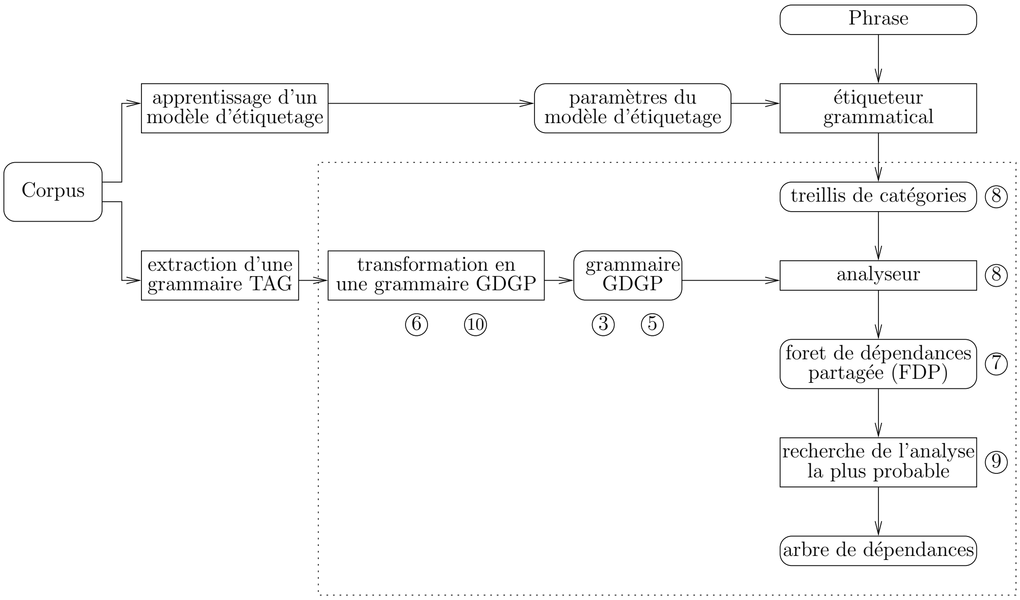



Fig. 1 – Schéma général du système

senel, dans le cadre de sa thèse de doctorat à Paris 7. L'étiqueteur utilisé lors des expériences a été réalisé par Pierre-Yves Strub dans le cadre de son projet de maîtrise d'informatique à Paris 7. Le développement de l'étiqueteur a été partiellement financé par le projet Technolangue AGILE du ministère de la culture. Ce travail à d'autre part bénéficié d'une bourse CNRS-NSF.

# Première partie

# Formalisme

Les cinq chapitres qui constituent cette partie ont pour objectif de définir le formalisme grammatical qui sera utilisé par les algorithmes de traitement de la partie II et qui permettra de représenter les grammaires utilisées pour les expériences de la partie III. Dans la suite de ce document, nous distinguerons deux composantes d'une grammaire probabiliste. D'une part le *modèle algébrique*, qui est le formalisme servant de base à la grammaire probabiliste - par exemple les grammaires hors-contexte, les grammaires d'arbres adjoints ou les grammaires de dépendances génératives - et, d'autre part, le *modèle probabiliste* qui est l'ensemble des paramètres probabilistes définis par la grammaire, lesquels permettent d'attribuer une probabilité aux arbres et aux phrases générés.

Les deux premiers chapitres ne concernent que le modèle algébrique du formalisme. Ils vont introduire les grammaires de dépendances génératives, notées dorénavant GDG. Le premier chapitre retrace rapidement l'histoire de certains courants de formalisation des grammaires de dépendance. Il permettra d'introduire les travaux qui nous ont influencé et de nous situer par rapport à eux. Le formalisme lui-même est décrit dans le chapitre 3, on introduira en particulier la notion d'*automates lexicalisés* qui constitue notre mode de représentation des règles grammaticales.

Les deux chapitres 4 et 5 concernent le modèle probabiliste du formalisme, ils introduiront les grammaires de dépendances génératives probabilistes, notées dorénavant GDGP. Ces deux chapitres reprennent l'organisation des deux premiers : le chapitre 4 introduit le cadre général des *grammaires fondées sur l'historique*, dans lequel s'inscrivent de nombreux travaux en analyse syntaxique probabiliste, en particulier les grammaires hors-contexte probabilistes, qui ont servi de base à toute une série de travaux récents sur les grammaires probabilistes. Le chapitre 5 définit le formalisme des GDGP. Il reprend le modèle algébrique des GDG pour y ajouter une composante probabiliste.

Le dernier chapitre de cette partie décrit les relations existant entre les grammaires d'arbres adjoints et les GDG. Ce chapitre a surtout un intérêt pratique. La procédure de transformation d'une grammaire d'arbres adjoints en GDG qu'il définit sera en effet utilisée dans le cadre des expériences de la partie III.

# 2 Formalismes syntaxiques pour grammaires de dépendances

Le travail décrit dans ce mémoire repose sur les grammaires de dépendances. La genèse et l'évolution de ces dernières sont moins connues que celle de leurs illustres cousines, les grammaires syntagmatiques. Nous allons présenter dans ce chapitre certains travaux portant sur la formalisation des grammaires de dépendances. Cette présentation est partiale, l'objectif du chapitre n'est pas de dresser un inventaire exhaustif de ces recherches mais de tenter de situer notre travail par rapport à des travaux dont nous nous sommes inspirés. Nous commencerons par évoquer dans la section 2.1 les textes pionniers de Lucien Tesnière lesquels seront comparés aux premiers travaux de Chomsky sur la syntaxe. Nous dégagerons en 2.2 et 2.3 deux filières qu'ont suivi les grammaires de dépendances, une filière *transductive* représentée en particulier par la théorie Sens-Texte et une filière *générative* inspirée des premiers travaux de Chomsky sur les grammaires hors-contexte. Nous nous demanderons en 2.4 quelles sont les implications de cette distinction et nous terminerons en évoquant rapidement, dans la section 2.5, le problème de la projectivité qui a été l'objet de nombreux travaux dans le cadre de la formalisation des grammaires de dépendances.

## 2.1 Des structures de dépendances aux grammaires de dépendances

On fait traditionnellement remonter les grammaires de dépendances aux travaux de Lucien Tesnière, dans les années trente. Ceux-ci ont été publiés en 1959, à titre posthume, dans l'ouvrage *Eléments de syntaxe structurale* [Tesnière, 1959]. Dans cet ouvrage, Tesnière propose un mode de représentation de la structure syntaxique des phrases fondé sur la notion de *connexions* entre les mots de ces dernières, connexions auxquelles il donne le nom de *dépendance*, indiquant la nature asymétrique de la connexion. Un des mots mis en jeu dans une dépendance est appelé *régissant* ou *gouverneur* tandis que l'autre est appelé *subordonné* ou *dépendant*. Au sein d'une phrase, tout mot, à l'exception d'un seul, possède un gouverneur unique, mais chaque mot peut gouverner un nombre variable de dépendants. L'ensemble des dépendances que les mots d'une phrase entretiennent entre eux constituent un arbre, que l'on appelle généralement arbre de dépendance, *stemma* dans la terminologie de Tesnière, dont un exemple, emprunté à Tesnière, est reproduit dans la figure 2. On pourra remarquer dans cet exemple que l'arbre de dépendances n'encode pas l'ordre linéaire des mots de la phrase, mais uniquement leur *ordre structurel*, toujours dans la terminologie de Tesnière.

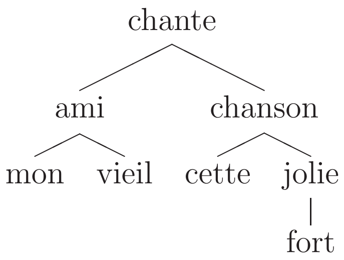



FIG. 2 – Un exemple d'arbre de dépendances tiré de [Tesnière, 1959]

Ce que Tesnière décrit dans son livre est un mode de représentation de la structure syntaxique des phrases. Il ne s'intéresse pas aux moyens de construire cette structure à partir d'éléments plus simples en suivant une procédure particulière. On ne peut par conséquent parler de grammaire dans le sens que l'école générativiste a donné à ce terme (un mécanisme engendrant toutes les suites grammaticales d'une langue et aucunes des suites agrammaticales ([Chomsky, 1957])). L'idée d'un processus produisant des structures de dépendances est néanmoins esquissée dans l'ouvrage de Tesnière, lors de la description du rapport existant entre l'ordre structural de la phrase et son ordre linéaire. Tesnière parle en effet d'établissement (ou de construction) d'un stemma à partir d'un ordre linéaire et, inversement, de *relevage* d'un stemma en une séquence de mots. L'idée d'un processus semble sous-jacente, mais ce dernier n'est pas explicité, rien n'est dit de sa mise en œuvre. Il est néanmoins intéressant de relever ici que le processus évoqué met en relation deux structures : un arbre et une chaîne, le processus semble donc de nature équative (construction d'un objet à partir d'un autre) et non générative. Principe qui sera largement repris par la Théorie Sens-Texte dont nous parlerons un peu plus loin.

Il est tentant, ne serait-ce que pour des raisons de proximité chronologique, de faire le lien entre cet ouvrage et *Structures syntaxiques* de Noam Chomsky [Chomsky, 1957], paru deux ans auparavant. Les deux ouvrages traitent de syntaxe mais se distinguent d'une part, par le mode de représentation de la structure syntaxique (arbre de dépendances dans un cas et arbre syntagmatique dans l'autre) et d'autre part, par l'importance que revêt, dans l'ouvrage de Chomsky, le processus de génération de phrases et les moyens permettant de réaliser un tel processus : les règles de réécriture. Alors que la structure syntaxique est présentée dans l'œuvre de Tesnière comme un objet implicite, mis au jour par le travail d'analyse du linguiste, dans l'ouvrage de Chomsky et plus généralement dans la tradition générativiste, la structure syntaxique de la phrase est le résultat d'un processus parfaitement spécifié manipulant des règles de réécriture.

L'émergence de grammaires de dépendance, comme moyen de production de structures de dépendances, va suivre deux filières, une filière transductive qui vit le jour dans ce qui était à l'époque l'Union Soviétique et une filière générative, née aux Etats-Unis.

## 2.2 La filière transductive

Les grammaires de dépendances sont souvent associées aux travaux gravitant autour de la Théorie Sens-Texte (TST dorénavant) développée dans les années soixantes par Igor Mel'čuk et Alexandre Zholkovsky [Mel'čuk & Zholkovsky, 1970, Mel'čuk, 1988] et qui ont adopté les arbres de dépendances comme moyen de représentation de la structure syntaxique des phrases. De manière plus générale, la TST s'est fixée comme objectif la modélisation de la relation existant entre un sens (plus précisément, la représentation formelle de ce dernier) et les différentes phrases véhiculant ce sens. Cette modélisation définit différents niveaux de représentation des énoncés, allant de la sémantique à la phonétique, ainsi que des moyens de construire une représentation $R_i$ d'un énoncé à un niveau de représentation donné à partir du même énoncé à un niveau de représentation adjacent : $R_{i-1}$ ou $R_{i+1}$. Cette construction est réalisée à l'aide de *règles de correspondance* qui mettent en relation une partie de la structure $R_i$ avec une partie des structures de $R_{i-1}$ ou de $R_{i+1}$. Des formats de règles assurant la correspondance entre les niveaux morphologiques et syntaxiques sont proposés dans [Mel'čuk, 1988] et dans [Mel'čuk & Pertsov, 1987]. Un format de règles de correspondance entre syntaxe de surface et syntaxe profonde est esquissé dans [Gladkij & Mel'čuk, 1975]. Ces règles, qui permettent de construire des structures de dépendances, peuvent être rapprochées des règles de réécriture des grammaires syntagmatiques, en ce sens qu'elles permettent de produire des structures syntaxiques. Cependant, alors que les règles génératives permettent de générer des structures à partir d'un symbole (l'axiome de la grammaire), les règles de correspondance construisent une (ou plusieurs) structures de dépendances à partir d'une structure préexistante qui, elle, n'est pas le fruit d'un processus décrit dans la théorie. Nous reviendrons en 2.4 sur cette distinction.

Le choix d'un cadre transductif sous la forme des règles de correspondances n'est en rien incompatible avec la formalisation des processus de manipulation de ces règles. On aurait pu voir apparaître, dans le cadre de la TST, une problématique procédurale dont l'objet aurait été l'étude des mécanismes permettant de réaliser la correspondance Sens-Texte, de la même façon qu'ont émergé, dans le cadre génératif, certains problèmes procéduraux spécifiques à ce cadre telles que la reconnaissance de phrases par une grammaire et l'analyse syntaxique automatique à l'aide de machines abstraites. Pour différentes raisons, un tel courant n'a pas émergé et l'étude des processus de mise en œuvre des transitions ne fut jamais au cœur des préoccupations de la communauté Sens-Texte. Ces raisons sont sans doute de natures diverses et leur étude dépasse de loin le cadre de ce travail. Il en est une qui mérite néanmoins d'être évoquée pour avoir été explicitement énoncée. Il s'agit de la distinction opérée dans la TST entre les règles de correspondance et le processus de manipulation de ces règles, et l'idée que seules les règles sont du ressort de la linguistique, les aspects procéduraux étant supposés généraux et hors du champ de cette dernière, sans pouvoir déterminer de quelle discipline ils relèvent. Le résultat de ce choix est la définition d'outils de description (les règles de correspondance) dont les propriétés procédurales n'ont pas fait l'objet d'un véritable travail de recherche. On pourra remarquer qu'une telle dichotomie n'a pas été effectuée dans le paradigme génératif où les relations entre des dispositifs statiques (la hiérarchie des grammaires de Chomsky) et des dispositifs procéduraux (les reconnaisseurs associés aux différentes grammaires de la hiérarchie), tels que les automates finis ([Chomsky & Miller, 1958]) et les automates à pile ([Chomsky, 1962]) ont été établis très tôt.

Malgré l'absence d'une tradition procédurale dans le cadre de la TST, certaines études ont été menées dans cette direction. La plupart d'entre elles se sont intéressées à la mise en œuvre de la transition Sens → Texte qui est le sens le plus étudié de la correspondance Sens-Texte, dans la TST. On pourra citer en particulier [Boyer & Lapalme, 1985], [Polguère, 1990], [Kahane & Mel'čuk, 1999]

Malgré la prééminence de l'étude de la direction Sens → Texte, certains chercheurs se sont intéressés aux aspects procéduraux de la direction Texte → Sens, parmi lesquelles on peut citer [Kahane, 2000] qui s'est penché sur le processus de construction d'un arbre de dépendances à partir d'une phrase, à l'aide de règles de correspondance.

## 2.3 La filière générative

Parallèlement, et indépendamment des travaux sur la TST, s'est développée aux Etats-Unis une filière générative des grammaires de dépendances ([Hays, 1964, Gaifman, 1965, Robinson, 1970]). Ces travaux ont abouti à l'élaboration de formalismes grammaticaux fortement inspirés des grammaires de réécriture permettant de générer des arbres de dépendances.

### 2.3.1 Les grammaires de réécriture de chaînes

On peut faire remonter l'origine des grammaires de dépendance génératives aux travaux de David Hays [Hays, 1964] et de Haim Gaifman [Gaifman, 1965]. Ces travaux, publiés après *Eléments de syntaxe structurale* de Tesnière et *Structures syntaxiques* de Chomsky, visent à établir une relation entre les grammaires génératives hors-contexte (notées dorénavant GHC) et les structures de dépendances. Le fruit de ce travail est la définition d'un formalisme génératif pour grammaires de dépendances (que nous appellerons GD) qui suit d'assez près le formalisme des GHC.

Contrairement aux arbres de dépendances de Tesnière et des arbres de dépendances de la TST, le produit d'une dérivation dans le cadre des GD est un *arbre de dépendances ordonné*. Ce dernier représente au sein d'une même structure les deux ordres linéaire et structural. Dans les cas projectifs, que nous définirons en 2.5, et qui sont les seuls pris en compte dans les GD, la représentation simultanée des deux ordres se réduit à indiquer pour chaque nœud de l'arbre de dépendances sa position vis-à-vis de ses frères et de son gouverneur.

Le principal obstacle dont il faut tenir compte pour la définition d'un formalisme génératif pour arbres de dépendances ordonnés, inspiré des GHC, concerne le positionnement d'un nœud vis-à-vis de ses nœuds fils. Cette information est nécessaire dans un arbre de dépendance ordonné, comme nous l'avons vu ci-dessus. Or dans un arbre syntagmatique, le positionnement d'un nœud vis-à-vis de ses constituants immédiats n'a pas de sens, puisque ces derniers ne coexistent pas avec le premier dans la chaîne. Ce problème est résolu dans les GD en matérialisant, dans la partie droite des règles hors-contexte, la position du père (le gouverneur) parmi ses fils (les dépendants du gouverneur) à l'aide d'un fils « fictif », comme on le verra ci-dessous.

Les GD sont composées, à l'instar des grammaires syntagmatiques de symboles terminaux et de symboles non terminaux, notés $X_k$ dans le reste de cette section. Les symboles terminaux correspondent aux mots du lexique et les symboles non terminaux correspondent à des catégories morpho-syntaxiques. Les règles de la grammaire décrivent la valence d'une catégorie : les différents dépendants qu'elle peut gouverner. Elles sont composées d'un symbole non terminal, la catégorie du gouverneur, et d'une suite finie de symboles non terminaux, les catégories des dépendants du gouverneur. Trois types de règles sont définis :

1. Des règles de la forme :

$$X_i(X_{j_1}, \ldots, X_{j_{k-1}}, *, X_{j_k}, \ldots, X_{j_n})$$

   où $X_i$ est le gouverneur et $X_{j_1}, \ldots, X_{j_n}$ sont les $n$ dépendants de $X_i$. La règle ordonne les différents dépendants les uns par rapport aux autres. L'astérisque indique la position du gouverneur vis-à-vis de ses dépendants. La règle présentée ci-dessus indique qu'un mot de catégorie $X_i$ peut gouverner simultanément des mots appartenant aux catégories $X_{j_1}, \ldots, X_{j_n}$. L'ordre des dépendants entre eux est représenté par leur ordre d'occurrence dans la règle : $X_{j_1} < X_{j_2} < \ldots < X_{j_{n-1}} < X_{j_n}$). Les dépendants $X_{j_1}, \ldots, X_{j_{k-1}}$ sont situés à gauche du gouverneur tandis que les dépendants $X_{j_k}, \ldots, X_{j_n}$ sont situés à sa droite.
2. Des règles de la forme $X_i(*)$ indiquent que les mots de catégorie $X_i$ peuvent apparaître sans dépendants. Formellement, il s'agit d'un cas particulier des règles de type 1.
3. Des règles de la forme $*(X_i)$ indiquent que les mots de la catégorie $X_i$ peuvent apparaître dans la phrase sans gouverneur.

De plus, une fonction d'assignation associe à toute catégorie $X_i$ les mots qui appartiennent à cette dernière.

Outre les différences de notation, sur lesquelles nous reviendrons ci-dessous, deux distinctions importantes entre les GD et les GHC méritent d'être évoquées. La première est que dans les règles de type 1 des GD (qui constituent en fin de compte les véritables règles structurelles), chaque règle possède, entre les deux parenthèses, l'astérisque qui permet de positionner un nœud vis-à-vis de ses fils. C'est le moyen choisi en GD pour résoudre le problème du positionnement du gouverneur vis-à-vis de ses dépendants, que nous avons évoqué ci-dessus. La seconde est que tous les symboles non terminaux sont en fait des catégories pré-terminales qui peuvent être réécrites comme des symboles terminaux, grâce à la fonction d'assignation.

Etant donnée une grammaire $G$, le processus de dérivation consiste à choisir une catégorie $c$ telle qu'il existe une règle de type 3 de la forme $*(c)$. Ensuite, une règle de type 1 ou 2 ayant la catégorie $c$ comme gouverneur est utilisée pour réécrire $c$. Si la règle choisie est de type 2 alors la dérivation est terminée, sinon, les symboles introduits lors de la réécriture sont à leur tour réécrits grâce aux règles de la grammaire jusqu'à ce que chaque symbole introduit ait été réécrit à l'aide d'une règle de type 2. A cette étape de la dérivation, une expression parenthésée a été construite, composée, outre les parenthèses, de symboles non terminaux et d'astérisques. La seconde étape de la dérivation consiste alors à remplacer tout symbole non terminal par un symbole terminal grâce à la fonction d'assignation. A l'issue de la dérivation, une expression parenthésée constituée de symboles non terminaux est construite, le parenthésage représentant la structure de l'arbre de dépendance. L'arbre de la figure 2 serait représenté dans un tel formalisme de la manière suivante :

$$\textit{chante}\ (\textit{ami}\ (\textit{mon}\ (*), *, \textit{vieil}\ (*)), *, \textit{chanson}\ (\textit{cette}\ (*), *, \textit{jolie}\ (\textit{fort}\ (*), *)))$$

On pourra noter ici une différence entre le processus de dérivation dans les GD et dans les GHC. Les premières dérivent directement des arbres, sous la forme d'une expression parenthésée et non des chaînes. Dans les secondes, en revanche, le processus de dérivation produit des chaînes et la structure syntaxique de ces chaînes est représentée par l'arbre de dérivation, qui garde la mémoire des règles ayant servi lors des différentes étapes de la dérivation.

Gaifman a établi ([Gaifman, 1965]) que les GD sont fortement équivalentes à un sous-ensemble des GHC[3]. Plus précisément, tout arbre généré par une GD

[3]Le lien avec les GHC est évident. Etant donnée une GD $G$, on peut construire une GHC $G'$ de même capacité générative faible de la façon suivante :

– Tous les symboles de l'alphabet terminal de $G$ sont ajoutés à l'alphabet terminal de $G'$ et tous les symboles de l'alphabet non terminal de $G$ sont ajoutés à l'alphabet non terminal de $G'$.
– Pour toute règle de $G$ de la forme :

$$X_i(X_{j_1}, X_{j_2}, \ldots, *, \ldots, X_{j_n})$$

une règle est ajoutée à $G'$ de la forme :

$$X_i \rightarrow X_{j_1} X_{j_2} \ldots X'_i \ldots X_{j_n}$$

où $X'_i$ est un nouveau symbole non terminal de $G'$ appelé le symbole pré-terminal associé à $X_i$.
– Toute règle de la forme $*(X_i)$ donne naissance à la règle RACINE $\rightarrow X_i$ où RACINE est l'axiome de $G'$.
– Pour tout couple $(X_i, l_j)$ de la fonction d'assignation, où $X_i$ est un symbole non terminal et $l_j$ un symbole terminal, une règle de la forme $X'_i \rightarrow l_j$ est ajoutée à $G'$, où $X'_i$ est la catégorie pré-terminale associée à $X_i$.

est isomorphe à un arbre syntagmatique pouvant être généré par une GHC. Inversement, certaines GHC peuvent générer des arbres syntagmatiques qui ne correspondent pas à un arbre de dépendance. Une conséquence importante pour nous est que certains résultats établis pour les GHC tiennent aussi pour les GD. En particulier, le problème de la reconnaissance pour les GD peut être résolut en $O(n^3)$ grâce à des algorithmes de programmation dynamique tels que l'algorithme CYK ([Younger, 1967, Kasami, 1965]) et l'algorithme de Earley ([Earley, 1970]).

On pourra remarquer que dans un tel formalisme, un mot ne peut avoir un nombre indéterminé de dépendants. Le nombre de dépendants de chaque catégorie est en effet fixé dans la grammaire. Cette limite pose un problème pour la représentation des dépendants répétables d'un mot (tels que les adverbes ou les groupes prépositionnels circonstanciels) qui ne peuvent être introduits, comme dans les grammaires syntagmatiques, par des règles récursives, du fait que dans un arbre de dépendance, tous les dépendants d'un mot sont directement reliés à ce dernier. La solution que propose [Abney, 1996a] à ce problème, et qui est reprise dans [Lombardo, 1996] ainsi que dans [Kahane et al., 1998] consiste à remplacer la séquence de catégories qui constitue les parties droites des règles par une expression régulière sur l'alphabet des catégories et à autoriser ainsi la répétition de certains dépendants. Cette nouvelle famille de grammaire sera appelée la famille des grammaires de dépendances étendues (notées GDE), à l'instar des grammaires hors-contexte étendues ([Albert et al., 2001]) qui sont des grammaires hors-contexte dans lesquelles la partie droite des règles de production sont des expressions régulières. Cette augmentation du formalisme ne modifie pas son pouvoir génératif faible (les expressions rationnelles peuvent en effet être réécrites comme des grammaires régulières) et ne change pas, par conséquent, les résultats de reconnaissance des GD.

### 2.3.2 Les grammaires de composition d'arbres

Une autre famille de grammaires génératives de dépendances trouve sa source dans les grammaires d'arbres adjoints [Joshi et al., 1975] et plus particulièrement dans les grammaires d'arbres adjoints lexicalisées [Schabes, 1990] (dorénavant TAG). Ces dernières associent à une entrée lexicale $m$ des arbres syntagmatiques, représentant les éléments de la phrase sous-catégorisés par $X$. Ces arbres sont appelés *arbres élémentaires* et l'entrée lexicale leur correspondant est appelé l'*ancre lexicale* de l'arbre élémentaire. Deux opérations, appelées *substitution* et *adjonction*, permettent de combiner entre eux les arbres élémentaires pour construire des structures syntaxiques plus importantes. Ces grammaires appartiennent à la famille des grammaires de réécriture d'arbres, qui se distinguent des grammaires de réécriture de chaînes, telles que les GHC, par le fait que le processus de dérivation conduit à la construction d'un arbre syntagmatique (et non d'une chaîne), appelé *arbre dérivé*. L'histoire de la dérivation, qui indique quels arbres élémentaires ont été combinés entre eux lors de la dérivation, est représentée par un arbre appelé *arbre de dérivation*. Le lien entre les grammaires d'arbres adjoints et les grammaires de dépendances a été établi très tôt du fait que l'arbre de dérivation peut être vu comme un arbre de dépendances : il est composé de nœuds lexicaux (les ancres des arbres élémentaires) reliés entre eux par des arcs, représentant l'opération de combinaison de deux arbres élémentaires. De tels arcs peuvent être interprétés comme des dépendances sémantiques ([Candito & Kahane, 1998]). Comme le montre [Rambow & Joshi, 1997], cette coïncidence des deux structures provient du rôle fondamental que joue le lexique dans les TAG. C'est en effet lui qui est au centre de la grammaire, et une opération de réécriture revient à établir une dépendance entre deux entrées lexicales.

Les grammaires d'arbres adjoints ont été à l'origine du formalisme développé dans [Nasr, 1995, Nasr, 1996]. Ce dernier reprend la notion d'arbre élémentaire pour proposer celle d'arbres élémentaires de dépendances. Ces derniers reposent sur les deux principes fondamentaux des arbres élémentaires des TAG : la distinction entre arguments et circonstants d'une part et, d'autre part, le domaine de localité étendu, qui consiste à représenter au sein d'une même structure élémentaire tous les arguments d'une ancre lexicale. La différence entre les arbres élémentaires des TAG et ceux de [Nasr, 1995, Nasr, 1996] provient essentiellement du fait que les seconds sont des arbres de dépendances ordonnés. Une opération de combinaison d'arbres permet de combiner des arbres élémentaires pour construire des arbres de dépendances. A la différence des TAG, l'objet construit lors d'une dérivation est directement un arbre de dépendance. La notion d'arbres élémentaires de dépendances et d'opérations permettant de les combiner on été reprises dans [Mertens, 2002] , [Kahane, 2002, Kahane, 2004] et [Joshi & Rambow, 2003].

## 2.4 Un faux débat ?

La distinction que nous avons effectuée entre les deux approches générative et transductive renvoie l'image d'une séparation franche entre ces deux mondes. La réalité n'est pas si tranchée. On sait depuis longtemps que certains systèmes transductifs, tels que les transducteurs finis ([Eilenberg, 1974, Eilenberg, 1976]) construits sur les alphabets $\Sigma_1$ et $\Sigma_2$, peuvent aussi bien être vus comme des mécanismes génératifs, produisant des couples de mots $(u, v) \in \Sigma_1^* \times \Sigma_2^*$, que comme des transducteurs produisant pour un mot $u \in \Sigma_1^*$ un mot $v \in \Sigma_2^*$. La relation entre les deux perspectives a été étudiée dans [Kahane, 2000]. Celui-ci à montré qu'une grammaire de dépendance transductive pouvait être vue comme une grammaire générative produisant des couples composés d'une phrase et d'un arbre de dépendance[4].

S'il est vrai que des systèmes transductifs peuvent être interprétés comme des systèmes génératifs et vice-versa, il reste qu'une différence profonde distingue les deux approches. Un mécanisme transductif $TR$ peut établir une correspondance entre deux structures $R$ et $R'$, sans que ces dernières ne répondent à certains critères de bonne formation. Illustrons cela sur le problème de l'analyse syntaxique ou la construction d'un arbre syntaxique correspondant à une phrase $S$. Un système génératif tel qu'une GHC n'associe une structure à $S$ que si cette dernière est bien formée (si elle appartient au langage généré par la grammaire). De ce point de vue, l'analyse syntaxique de $S$ n'est rien d'autre que la preuve de sa bonne formation. Dans une approche transductive (dans le formalisme simple proposé par [Kahane, 2000], par exemple), une grammaire transductive $TR$ peut très bien associer une structure à $S$ bien que celle-ci soit mal formée. Pour reprendre les termes de [Kahane, 2000], « ...une grammaire transductive n'a pas pour mission d'assurer la bonne formation des structures qu'elle met en correspondance ».

On peut illustrer le même point à l'aide d'un exemple trivial, qui consiste à écrire un transducteur fini permettant de transformer, dans une expression parenthésée, les parenthèses ouvrantes (respectivement fermantes) par des crochets ouvrants (respectivement fermants). Ce transducteur effectuera sa transformation que la chaîne d'entrée soit bien formée (parenthésage équilibré) ou pas. Dans cet exemple, le transducteur ne peut garantir la bonne formation de la chaîne. Il faudrait pour cela un transducteur à pile.

Il est difficile de mesurer précisément les conséquences d'une telle différence. Si l'on n'attend pas d'un mécanisme transductif qu'il (1) ne définisse des correspondances qu'entre structures bien formées, il est raisonnable d'attendre

[4]Comme le montre [Kahane, 2000], l'objet produit est plus riche qu'un simple couple de structures. Il s'agit en réalité de ce que [Kahane, 2000] appelle une *supercorrespondance* : un couple de deux structures plus une partition de ces deux dernières et une fonction associant les parties des deux structures.

de lui, en revanche, que (2) pour une structure d'entrée[5] bien formée, il ne produise que des structures bien formées. Pour reprendre l'exemple de l'analyse syntaxique, on peut admettre que le processus produise, pour une phrase agrammaticale, des structures syntaxiques aberrantes. En revanche, pour une phrase bien formée en entrée, ne devraient être produites que des structures syntaxiques correctes. La question est de savoir si la propriété 2 peut être vérifiée sans que la propriété 1 le soit. Il n'existe pas à notre connaissance de réponse générale à cette question. Mais, pour reprendre l'exemple de l'analyse syntaxique, il semble difficile d'attendre d'un tel processus qu'il ne construise que des structures syntaxiques bien formées, s'il n'intègre pas la notion de bonne formation syntaxique telle que représentée dans une grammaire générative. Dans le cas du transducteur remplaçant des parenthèses par des crochets, en revanche, le processus transductif permet de vérifier 2 sans pour autant vérifier 1.

Une solution consiste à distinguer deux types de règles : des règles transductives, prenant en charge uniquement la construction de structures à partir d'une structure initiale, et des règles de bonne formation, dont le but est de ne garder, parmi les structures produites par le processus transductif, celles qui répondent à certains critères de bonne formation. Une telle solution est séduisante d'un point de vue déclaratif, car elle permet d'effectuer une distinction, qui semble fondée, entre différents types de règles. Mais un tel processus risque d'être inefficace du fait qu'à l'issue du processus transductif, des structures sont créées qui seront ensuite rejetées par le processus de contrôle de bonne formation. La tentation est alors grande d'intégrer les règles de bonne formation et les règles transductives et d'aboutir à un système génératif !

## 2.5 La projectivité

Il est difficile de parler de grammaires de dépendances sans évoquer le sujet de la projectivité. Cette propriété, découverte par Yves Lecerf ([Lecerf, 1960]) définit une relation entre les deux ordres structural et linéaire de la phrase. Une dépendance $m_1 \rightarrow m_2$ est projective si les mots séparant $m_1$ et $m_2$ dans l'ordre linéaire sont tous des descendants[6] de $m_1$. Un arbre de dépendances ordonné est projectif si toutes ses dépendances le sont. En d'autres termes, un arbre de dépendances ordonné est projectif si la projection de tout nœud $n$ de l'arbre (l'ensemble des descendants de $n$, $n$ compris) forme un segment

[5] On considère ici, pour faciliter l'exposé, que le mécanisme est directionnel.

[6] La descendance est définie comme la clôture transitive de la dépendance.

continu de la phrase. Les formalismes des GD et des GDE sont projectifs : ils ne permettent de générer que des arbres projectifs.

Si la majorité des structures syntaxiques des langues naturelles est projective ([Mel'čuk, 1988]), il existe néanmoins dans la langue certains cas de non projectivité. Par conséquent, de telles structures ne peuvent pas être produites par une GD ou une GDE.

Bien que le problème de l'analyse syntaxique des grammaires de dépendances non projective soit NP-complet ([Bröker & Neuhaus, 1997]), plusieurs types de grammaires génératives admettant une forme limitée de non projectivité ont été proposés dans la littérature ([Kahane et al., 1998], [Lombardo & Lesmo, 2000], [Bröker, 2000]). Ces travaux ont montré en particulier que le problème de la reconnaissance de certaines classes de grammaires non projectives, tout en restant plus coûteux en temps que celui de la reconnaissance des grammaires projectives, restait polynomial. Malgré ces résultats, nous avons décidé de nous limiter à un formalisme pour grammaires projectives. La raison principale de ce choix est que les phénomènes de non projectivité sont assez rares en français et en anglais, qui sont les deux langues que nous avons pris en compte ici, pour pouvoir être négligés dans ce travail.

# 3 Grammaires de dépendances génératives

Le formalisme des grammaires de dépendances génératives (GDG), décrit dans ce chapitre, s'inscrit dans la lignée des travaux de la filière générative que nous avons identifiée en 2.3. Il reprend d'idée proposée par [Hays, 1964] de représenter une grammaire de dépendances à l'aide de règles hors-contexte dans lesquelles la position du père est définie vis-à-vis de ses fils. Il reprend à [Abney, 1996b] l'idée de règles dont les parties droites sont composées d'expressions régulières pour résoudre le problème des dépendances répétables. Il se distingue néanmoins de ces formalismes en représentant de façon explicite la notion de rôle fonctionnel, et en représentant les règles génératives sous la forme d'automates finis d'un type particulier, appelés *automates lexicalisés*. Le remplacement des expressions régulières proposées par Abney par des automates finis ne représente pas, à ce stade de notre exposé, une différence notoire. L'intérêt de de mode de représentation apparaîtra plus tard, en particulier au chapitre 5, lors de l'introduction de probabilités dans les GDG. On verra que l'existence de plusieurs automates différents, reconnaissant un même langage, pourra être mis à profit pour mieux articuler le modèle probabiliste et le modèle algébrique d'une grammaire probabiliste. Nous verrons d'autre part, au chapitre 7, que les automates permettront de représenter des ensembles d'arbres de dépendances correspondant à une même phrase sous une forme compacte.

Ce chapitre est composé de deux sections. Nous décrirons dans la première le formalisme des GDG, puis, dans la seconde, le processus de génération d'arbres de dépendances. Cette section va nous permettre d'introduire un mode de représentation des arbres de dépendances appelé *représentation linéaire* qui constitue la représentation des arbres de dépendances manipulée par les différents algorithmes.

*Les travaux décrits dans ce chapitre ont déjà été partiellement publiés. L'ébauche des* GDG *apparaît dans [Kahane et al., 1998]. Le formalisme syntaxique proposé dans cet article est néanmoins plus puissant que celui des* GDG*, il permet en particulier la génération de certaines structures non projectives. La notion d'automate lexicalisé apparaît, proche de sa forme présente, dans [Nasr & Rambow, 2004a].*

## 3.1 Définition

Une GDG se présente comme un ensemble d'automates d'un type particulier, appelés *automates lexicalisés*. Un automate lexicalisé dont $m$ est une *ancre* décrit tous les dépendants possibles du mot $m$. Chaque automate possède un *nom*, qui définit une catégorie morpho-syntaxique. Cette dernière spécifie, outre la partie de discours de l'ancre de l'automate, la valence de cette dernière. Dans la suite de ce document, nous emploierons indifféremment les termes *automate* ou *catégorie* dans le contexte des GDG. Un exemple d'automate lexicalisé, de nom $V$, est représenté dans la partie supérieure de la figure 3[7]. Un tel automate indique que le verbe *mange* possède un dépendant sujet, obligatoire et non répétable, dont la catégorie est *nom* ou *pronom*, un dépendant objet qui est optionnel et non répétable, et des dépendants circonstanciels, optionnels et répétables introduits par une préposition. Les transitions de l'automate sont étiquetées par des couples $\langle f, c\rangle$, où $f$ est un rôle fonctionnel et $c$ une catégorie (le nom d'un automate), ou par des couples $\langle \texttt{LEX}, m\rangle$, où $m$ est une ancre de l'automate. L'ordre des dépendants entre eux et vis-à-vis de leur gouverneur est représenté par la structure de l'automate. Le dépendant le plus à gauche est le sujet ; il se trouve à gauche du gouverneur. L'objet direct éventuel se trouve à droite du gouverneur, et peut être séparé de lui par des compléments prépositionnels. D'autres automates lexicalisés sont représentés dans la figure 4.

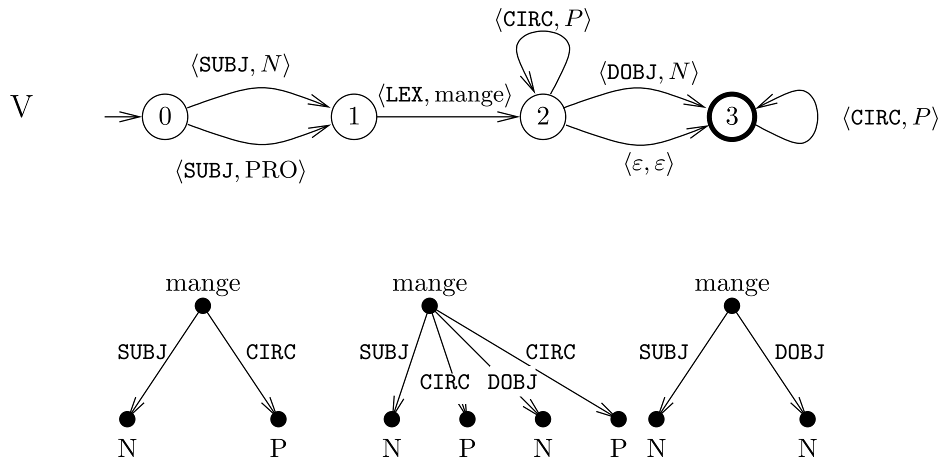



FIG. 3 – Un automate lexicalisé et trois arbres élémentaires

[7]L'état initial d'un automate est identifié par une flèche pointant sur ce dernier. Ses états d'acceptation sont représentés en gras. Les transitions vides sont, soit représentées en pointillés, soit étiquetées par le couple $\langle \varepsilon, \varepsilon\rangle$.

Chaque mot (au sens de la théorie des langages) reconnu par un automate est une séquence de couples $\langle f, c\rangle$. Cette séquence correspond à un arbre de dépendances de profondeur 1, que l'on appellera *arbre élémentaire* de la grammaire. Trois exemples de tels arbres sont représentés dans la partie inférieure de la figure 3. Le mot correspondant à l'arbre de gauche est : $\langle \texttt{SUBJ}, N\rangle$ $\langle \texttt{LEX}, \text{mange}\rangle$ $\langle \texttt{CIRC}, P\rangle$. On verra en détail en 3.2.3 comment construire un arbre de dépendances à partir d'une représentation linéaire de ce dernier. L'ensemble des arbres élémentaires pouvant être générés par un automate $A$ est noté $T_E(A)$.

On peut ici faire le lien et noter une différence importante entre les grammaires GDG et des grammaires de manipulation d'arbre, telles que les TAG et les grammaires de [Nasr, 1996] ou de [Joshi & Rambow, 2003]. Le lien entre les deux familles de grammaires est la notion d'arbre élémentaire, qui représente la valence des mots. Cependant, les GDG, contrairement aux autres, ne considèrent pas les arbres élémentaires comme des objets atomiques. Ils sont eux-mêmes produits par des automates, dont les transitions correspondent à des dépendances syntaxiques, à l'exception des transitions lexicales et des transitions vides. Les GDG peuvent être vues comme un double système génératif : un premier processus génératif permet de générer des arbres élémentaires à partir des automates, et un second processus permet de combiner les arbres élémentaires pour former des arbres de dépendances.

### 3.1.1 Définition formelle

D'un point de vue formel, une grammaire GDG est définie par un sextuplet $\langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I}\rangle$ où $\mathcal{C}$, $\Sigma$ et $\mathcal{F}$ sont des ensembles de symboles. $\mathcal{C}$ est l'ensemble des catégories, $\Sigma$ est l'ensemble des éléments lexicaux, et $\mathcal{F}$ est l'ensemble des étiquettes fonctionnelles. $\mathcal{A}$ est un ensemble d'automates lexicalisés, décrits plus en détails ci-dessous. $\theta$ est une fonction bijective, qui associe à tout élément de $\mathcal{A}$ un élément de $\mathcal{C}$. C'est cette fonction qui définit une relation bi-univoque entre les automates et les catégories. $\mathcal{I}$ est un sous-ensemble de $\mathcal{C}$ appelé ensemble des symboles *initiaux*, lesquels jouent le rôle de l'axiome dans une grammaire générative : c'est par l'un de ces symboles que débute une dérivation.

Etant donné les trois ensembles de symboles $\mathcal{C}$, $\Sigma$ et $\mathcal{F}$, un automate lexicalisé est un automate fini construit sur l'alphabet $(\mathcal{F} \times \mathcal{C}) \cup (\texttt{LEX} \times \Sigma)$. En d'autres termes, chaque transition de l'automate est étiquetée par un couple $\langle f, c\rangle$ où $f$ est une étiquette fonctionnelle et $c$ une catégorie, ou par un couple $\langle \texttt{LEX}, m\rangle$ où LEX est un symbole particulier n'appartenant pas à l'ensemble

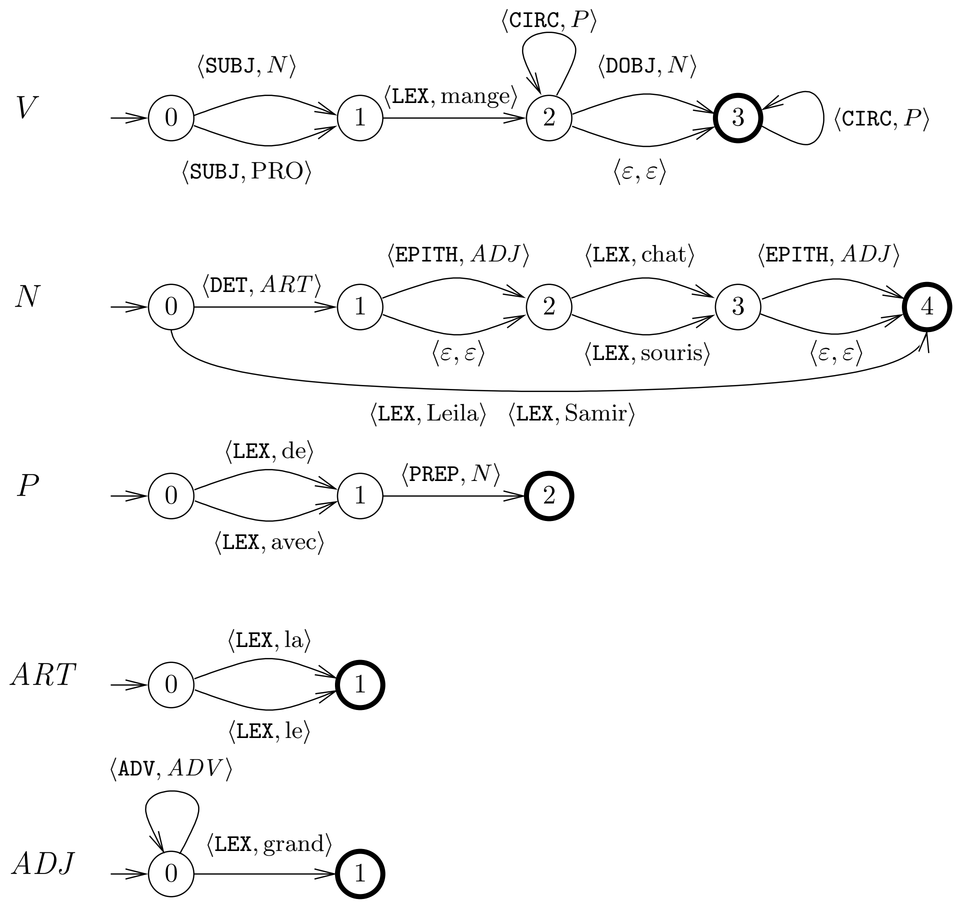



FIG. 4 – Un exemple de grammaire GDG

des étiquettes fonctionnelles, $\mathcal{F}$ et $m$ est un élément lexical ($m \in \Sigma$). Une telle transition est appelée *transition lexicale*. De plus, tout chemin d'un automate lexicalisé menant de l'état initial à un état d'acceptation possède une transition lexicale et une seule. Les éléments lexicaux d'un automate lexicalisé sont appelés les *ancres* de l'automate. Lorsque l'on remplace par la chaîne vide la partie lexicale de toutes les transitions lexicales, on obtient un *schéma d'automate*.

De plus, les automates lexicalisés ne possèdent pas de transitions vers leur état initial. Il s'agit d'un détail technique dont la raison apparaîtra en 3.2.3. Les transitions émanant de l'état initial d'un automate seront appelées les *transitions initiales* de ce dernier.

D'un point de vue formel, un automate lexicalisé $A$ est défini comme un septuplet $\langle \mathcal{Q}, e_0, \mathcal{F}, \mathcal{C}, \Sigma, \delta, \mathcal{Q}_a \rangle$, où $\mathcal{Q}$ est l'ensemble des états de l'automate, $e_0 \in \mathcal{Q}$ est son état initial, $(\texttt{LEX} \times \Sigma) \cup (\mathcal{F} \times \mathcal{C})$ est son alphabet, $\delta$ est sa fonction de transition ($\delta \subseteq \mathcal{Q} \times ((\texttt{LEX} \times \Sigma) \cup (\mathcal{F} \times \mathcal{C})) \times \mathcal{Q}$), et $\mathcal{Q}_a \subseteq \mathcal{Q}$ sont ses états d'acceptation. Etant donné un automate lexicalisé $A$, on notera $\mathcal{Q}(A)$, $e_0(A)$, $\mathit{fct}(A)$, $\mathit{cat}(A)$, $\delta(A)$ et $\mathcal{Q}_a(A)$ chacun de ses membres.

Dans la suite de ce document, nous manipulerons souvent des transitions d'automates. Chaque transition d'un automate est identifiée de façon non ambiguë par un couple $\langle c, t \rangle$ où $c$ est le nom d'un automate (une catégorie) et $t$ est une transition représentée par un triplet $\langle e_o, \langle f, c \rangle, e_d \rangle$ ($e_o$ et $e_d$ sont respectivement l'état origine et destination, tandis que $f$ et $c$ sont les symboles fonctionnels et catégoriels ou le symbole `LEX` et un élément lexical). Lorsque le contexte n'est pas ambigu (on sait de quel automate il est question), une transition sera juste représentée par le triplet $\langle e_o, \langle f, c \rangle, e_d \rangle$. Etant donné une transition $t$, on notera successivement $\mathit{Auto}(t)$, $\mathit{orig}(t)$, $\mathit{dest}(t)$, $\mathit{fct}(t)$ et $\mathit{cat}(t)$, l'automate auquel elle appartient, son état origine, son état destination, son symbole fonctionnel et son symbole catégoriel.

On notera $\delta^A(e, f, c)$ ($\delta(e, f, c)$ lorsque le contexte n'est pas ambigu) l'ensemble des transitions émanant de l'état $e$ de l'automate $A$ étiquetées par le couple $\langle f, c \rangle$. On notera $\delta^A(e, c)$ ($\delta(e, c)$) l'ensemble des transitions émanant de l'état $e$ de l'automate $A$ étiquetées par un couple de la forme $\langle f, c \rangle$ avec $f \in \mathcal{F} \cup \{\texttt{LEX}\}$ (on spécifie la catégorie mais pas l'étiquette fonctionnelle).

La clôture-$\varepsilon$ d'un état $e$ d'un automate $A$, (l'ensemble des états accessibles depuis $e$ en n'empruntant que des transitions étiquetées $\langle \varepsilon, \varepsilon \rangle$) sera notée $\varepsilon^A(e)$ ($\varepsilon(e)$ lorsque le contexte le permet).

On définit de plus le prédicat $\mathit{accept}(A, e)$ qui est vrai s'il existe un état d'acceptation de l'automate $A$ dans la clôture-$\varepsilon$ de l'état $e$.

### 3.1.2 Sites d'un automate

Les transitions des automates lexicalisés ne jouent pas toutes le même rôle. Certaines décrivent des dépendances que les ancres de l'automate peuvent établir, d'autres permettent de sélectionner une ancre de l'automate tandis que les transitions vides n'ont aucune influence sur les arbres que l'automate permet de générer. Nous allons établir une classification plus précise des différents types de transitions, et regrouper certaines transitions d'un automate au sein d'ensembles de transitions appelés *sites* de l'automate. Un automate lexicalisé se présente alors comme un ensemble de sites reliés entre eux par des transitions appelées *transitions inter-sites*, comme l'illustre l'automate de la figure 5. L'introduction de la notion de site va permettre de faire émerger une organisation interne des automates qui n'apparaît pas lorsque ces derniers sont vus comme des ensembles de transitions. L'émergence de ce niveau intermédiaire de structuration va permettre une description plus simple de certaines opérations sur les automates. On dira en particulier qu'un site est *pourvu* lorsque, lors de la traversée de l'automate, une des transitions du site est empruntée. D'autre part, la notion de site permettra de décrire plus simplement différents types d'automates qui seront définis par la suite.

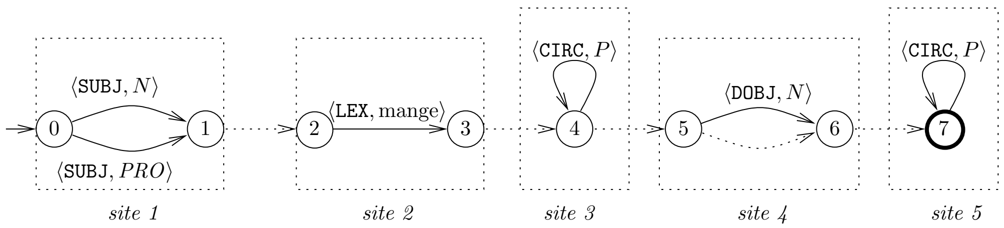



FIG. 5 – Sites d'un automate

Les transitions d'un automate qui décrivent des actants des ancres de l'automate sont appelées *transitions actancielles*. Les transitions actancielles qui partagent la même étiquette fonctionnelle et le même état origine constituent un *site actanciel*, à l'image des transitions *sujet* et *objet* de l'automate de la figure 5, dont elles constituent les sites 1 et 4. Les sites actanciels se déclinent en deux types, les *sites obligatoires* et les *sites optionnels*. Les seconds se distinguent des premiers par la présence d'une transition vide permettant de ne pas pourvoir le site, à l'image du site 4 de l'automate de la figure 5.

Les transitions qui décrivent des modifieurs sont appelées *transitions modificatrices*. Elles sont regroupées au sein de *sites modifieurs*, comme les sites 3 et 5 de l'automate de la figure 5. Les sites modifieurs se déclinent eux aussi en deux types : des *sites répétables* et des *sites non répétables*. Les deux sites modifieurs de la figure 5 sont répétables, tandis que le site comprenant la dépendance *épithète* de l'automate $N$ de la figure 4 offre un exemple de site non répétable. On remarquera que les sites modifieurs non répétables et les sites actanciels optionnels sont structurellement identiques (un faisceau de transitions entre deux mêmes états dont l'une et une transition vide), ils ne se distinguent que par la nature des étiquettes fonctionnelles de leurs transitions. Lorsque plusieurs dépendances sont générées par un même site répétable, lors de la génération d'un arbre élémentaire, nous les distinguerons en disant que la première est la dépendance de *position* 1, la seconde est de position 2, et ainsi de suite. L'ensemble des transitions actantielles et modificatrices d'un automate sont appelées ses transitions *d'attachement*. Les sites actanciels et modifieurs constituent, eux, les *sites d'attachements* de l'automate.

Les transitions lexicales qui relient deux mêmes états forment un *site lexical*. Un automate peut avoir plusieurs sites lexicaux, à l'image de l'automate $N$ de la figure 4.

Un automate lexicalisé dont les sites répétables sont composés de boucles sur un état unique est sous *forme canonique*. L'automate de la figure 5 offre un exemple d'automate sous forme canonique. On verra dans le chapitre 5 des automates lexicalisés qui ne sont pas sous forme canonique. Ces différents types d'automates se distinguent par la structure de leurs sites répétables.

Les sites de l'automate de la figure 3 ont été explicitement représentés dans la figure 5. L'automate se présente comme une séquence de sites reliés entre eux par des transitions inter-sites, qui sont des transitions vides. Les sites se trouvant à la gauche (à la droite) du site lexical sont appelés *sites gauches* (*sites droits*) de l'automate. Les sites 1 et 4 sont des sites actanciels, les sites 3 et 5 sont des sites modifieurs et le site 2 est un site lexical. L'automate de la figure 5 reconnaît le même langage que l'automate $V$ de la figure 3. La seule différence entre les deux réside dans la présence de transitions inter-sites.

### 3.1.3 Comparaison avec d'autres approches

L'utilisation d'automates finis pour la représentation de grammaires non régulières n'est pas une nouveauté. L'exemple le plus illustre est probablement le modèle des réseaux de transition récursifs (Recursive Transition Networks) de [Woods, 1970], formellement équivalent au modèle des grammaires hors-contexte.

D'autres travaux plus récents ont repris l'idée de représenter les règles d'une grammaire de dépendances sous la forme d'automates. Le modèle des *head automata* de [Alshawi, 1996] définit aussi des automates associés à des entrées lexicales, appelées têtes de l'automate. L'automate décrit aussi les différents dépendants de sa tête, plus précisément les catégories de ces derniers. La principale différence avec les automates lexicalisés se situe au niveau de la structure des deux types d'automates. Les automates lexicalisés génèrent les dépendants de ses ancres de gauche à droite alors que les *head automata* génèrent d'abord les dépendants les plus éloignés de la tête puis, au fur et à mesure de la génération, les dépendants s'en rapprochant. Les dépendants gauches et droits de la tête sont générés sur deux bandes d'écriture différentes, permettant ainsi d'entremêler génération de dépendants gauches et génération de dépendants droits.

Le modèle des *automates bilexicaux* de [Eisner, 2000] se distingue des GDG et des *head automata* en ne manipulant pas des catégories mais uniquement des mots. Il s'agit là de la principale caractéristique de ce modèle. C'est de là que provient le terme *bilexical* : seules des dépendances entre paires de mots (gouverneur et dépendant) sont définies. Un automate associé au mot $m$ décrit tous les mots que $m$ peut gouverner. A l'instar des *head automata*, les dépendants gauches et droits sont générés par deux automates différents, un automate gauche et un automate droit.

## 3.2 Génération d'un arbre

Nous avons évoqué en 3.1 qu'une GDG peut être vue comme un double système génératif (génération d'arbres élémentaires et combinaison de ces derniers). Etant donné cette particularité, le processus de génération à partir d'une GDG $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I} \rangle$ peut être réalisé de différentes façons, selon que l'on distingue ces deux étapes ou pas.

Une première façon d'envisager le processus consiste à choisir une catégorie $c_i \in \mathcal{I}$, puis un arbre élémentaire $T \in T_E(c_i)$[8]. Chaque feuille de $T$ ayant pour catégorie $c$ est ensuite remplacée par un arbre élémentaire $T' \in T_E(c)$, et ainsi de suite jusqu'à obtenir un arbre de dépendances ne contenant plus de nœuds non lexicaux. Cette méthode de génération ressemble, dans son esprit, à la génération dans le cadre des grammaires de composition d'arbres, telles que les TAG, les grammaires de [Nasr, 1996] et de [Joshi & Rambow, 2003],

[8]Rappelons que $T_E(c)$ désigne l'ensemble des arbres élémentaires que l'automate correspondant à la catégorie $c$ permet de générer.

évoquées dans le chapitre 2.

Dans le cadre des grammaires de composition d'arbres, le choix d'un arbre élémentaire lors de la génération constitue une seule opération. Or ce choix peut lui-même être décomposé en une séquence de choix qui correspondent à la traversée d'un automate de la grammaire. De ce point de vue, une opération élémentaire consiste à franchir une transition d'un automate. Ce franchissement correspond à l'établissement d'une dépendance dans l'arbre de dépendances en train d'être généré ou à la sélection d'une ancre lexicale. C'est cette perspective que nous adopterons ici. La raison de ce choix apparaîtra plus clairement lorsque nous aborderons, dans le chapitre 7, le problème de la représentation compacte d'une forêt d'arbres de dépendances. Nous verrons en particulier que pour obtenir une bonne compaction d'une forêt, il est nécessaire de pouvoir manipuler des structures qui ne correspondent pas à des arbres élémentaires, mais à des parties de ces derniers. Ces parties correspondent précisément aux structures produites aux différentes étapes de la génération d'un arbre élémentaire par un automate.

Dans la section suivante, nous allons définir un mode de représentation particulier des arbres de dépendances, appelé *représentation linéaire*, dans lequel un arbre est représenté sous la forme d'une séquence de transitions d'automates de la grammaire. C'est ce mode de représentation qui sera utilisé dans les différents traitements décrits dans la suite de ce document. Ce sont en particulier des représentations linéaires d'arbres qui sont produites à l'issue du processus de génération décrit en 3.2.2. La construction d'une représentation arborescente conventionnelle d'un arbre de dépendances à partir de sa représentation linéaire est exposée en 3.2.3.

### 3.2.1 Représentation linéaire d'un arbre de dépendances

Le produit d'une dérivation se présente comme une séquence de transitions de différents automates de la grammaire : les transitions ayant été franchies lors de la dérivation. Cette séquence de transitions constitue la représentation linéaire d'un arbre de dépendances. Elle correspond à un parcours de ce dernier. Selon la nature d'une transition de la séquence, une opération particulière est effectuée dans le parcours de l'arbre. Une transition d'attachement correspond au franchissement d'une dépendance dans l'arbre (passage d'un nœud $n$ à un fils de $n$). Une transition lexicale, étiquetée $\langle \texttt{LEX}, m \rangle$, correspond à l'étiquetage, du nœud visité, par $m$. Une transition vide ne correspond à aucune opération.

Plusieurs parcours d'un arbre sont possibles. Ils définissent des ordres différents sur un même ensemble de transitions. L'ordre que nous avons choisi d'adopter pour la représentation linéaire d'un arbre de dépendances est un parcours gauche-racine-droite, en profondeur d'abord. On trouvera, dans la partie gauche de la figure 6, la représentation linéaire d'un arbre de dépendances produite à l'aide des automates de la figure 4, et, dans la partie droite, la représentation conventionnelle de l'arbre. Les onze transitions qui constituent la représentation linéaire sont disposés verticalement, sur deux colonnes. Le premier élément est une transition émanant de l'état 0 de l'automate $V$. Il correspond à l'établissement de la dépendance *sujet*. Le second élément correspond à l'établissement de la dépendance *déterminant* du sujet. Le troisième correspond au choix du déterminant *le*, et ainsi de suite. On notera que la représentation linéaire est plus riche que la représentation conventionnelle, car elle indique quels automates ont été utilisés pour générer l'arbre.

$$\begin{array}{ll}
\langle\langle V, \langle 0, \langle \texttt{SUBJ}, N\rangle, 1\rangle\rangle, & \langle V, \langle 2, \langle \texttt{DOBJ}, N\rangle, 3\rangle\rangle, \\
\langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle, & \langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle, \\
\langle ART, \langle 0, \langle \texttt{LEX}, le\rangle, 1\rangle\rangle, & \langle ART, \langle 0, \langle \texttt{LEX}, la\rangle, 1\rangle\rangle, \\
\langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle, & \langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle, \\
\langle N, \langle 2, \langle \texttt{LEX}, chat\rangle, 3\rangle\rangle, & \langle N, \langle 2, \langle \texttt{LEX}, souris\rangle, 3\rangle\rangle\rangle \\
\langle V, \langle 1, \langle \texttt{LEX}, mange\rangle, 2\rangle\rangle, &
\end{array}$$

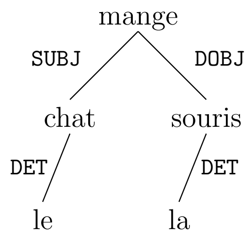



FIG. 6 – Un arbre de dépendances et sa représentation linéaire

La représentation linéaire d'un arbre de dépendances permet de construire ou de décomposer ce dernier par simple concaténation ou césure. Etant donné un arbre $T$, représenté sous la forme d'une séquence $t_{1,n} = \langle t_1, \ldots, t_n\rangle$, $T$ peut être décomposé d'autant de façons qu'il y a de décomposer une séquence de $n$ éléments en sous-séquences, et peut être recomposé par simple concaténation de sous-séquences de $t_{1,n}$. Chacune de ces sous-séquences ne correspond pas nécessairement à un sous-arbre de $T$. En séparant la séquence de transitions de la figure 6 en deux, entre $\langle V, \langle 1, \langle \texttt{LEX}, mange\rangle, 2\rangle\rangle$ et $\langle V, \langle 2, \langle \texttt{DOBJ}, N\rangle, 3\rangle\rangle$ par exemple, on obtient des morceaux d'arbres qui ne sont pas des sous-arbres de $T$. On reviendra plus en détail sur cette propriété importante dans le chapitre 7.

### 3.2.2 Le processus de génération

Le processus de génération d'un arbre de dépendances (plus précisément d'une représentation linéaire de ce dernier) possède beaucoup de points communs avec la traversée d'un réseau récursif de transitions. Il fait appel, en particulier, à une pile. Cette dernière contient des couples $\langle c, e \rangle$ où $c$ est le nom d'un automate de la grammaire ($c \in \mathcal{C}$) et $e$ est un état de l'automate de $c$. Le recours à une pile s'explique par le fait que, durant la génération, plusieurs automates sont parcourus en parallèle. Ces derniers sont stockés dans la pile.

Etant donné une GDG $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I} \rangle$, le processus commence par le choix d'une catégorie $c_i \in \mathcal{I}$ et l'empilement du couple $\langle c_i, e_0(c_i) \rangle$. Une transition $t = \langle e_0(c_i), \langle f, c \rangle, e_1 \rangle$ émanant de $e_0(c_i)$ est alors choisie, et émise sur la bande de sortie. La pile est dépilée, puis les deux couples $\langle c_i, e_1 \rangle$ et $\langle c, e_0(c) \rangle$ sont successivement empilés. $\langle c_i, e_1 \rangle$ représente l'état courant dans le parcours de l'automate $c_i$ : c'est l'état atteint à l'issue du franchissement de la transition $t$. Le couple $\langle c, e_0(c) \rangle$ correspond à l'état initial de l'automate associé à la transition $t$. Le parcours de l'automate $c_i$ est « suspendu » en attendant d'achever le parcours de $c$. La suite du processus va consister à traverser l'automate $c$ : le couple $\langle c, e_0(c) \rangle$ est dépilé, puis une transition émanant de l'état $e_0(c)$ est choisie, et l'automate lui correspondant est à son tour parcouru. A l'issue du parcours de $c$, la pile est dépilée et le parcours de $c_i$ est poursuivi. Le processus prend fin lorsque la pile est vide. Ce processus non déterministe est décrit de façon plus précise dans l'algorithme de la figure 7.

On pourra remarquer qu'à l'instar du processus des GD de Hays, notre processus de dérivation ne produit pas une phrase, comme dans le cas des grammaires syntagmatiques, mais directement un arbre. La différence est que, dans notre cas, l'arbre est représenté par un parcours de ce dernier : sa représentation linéaire, alors que dans le cas des GD l'objet produit est une expression parenthésée, représentant la structure de l'arbre.

Les différentes étapes de la génération de l'arbre de la figure 6 sont représentées par la figure 8.

Etant donnée une GDG $G$, l'ensemble des arbres qu'elle permet de générer est noté $T(G)$. La phrase correspondant à l'arbre de dépendances $T$ est notée $S(T)$. Celle-ci est unique, car l'arbre $T$ est ordonné. De façon symétrique, étant donné la phrase $S$ et une grammaire $G$ on note par $T_G(S)$ ($T(S)$ lorsque le contexte ne prête pas à confusion) l'ensemble des arbres de dépendances que la grammaire $G$ associe à $S$.

**Entrée :**
une grammaire GDG $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I} \rangle$

**Sortie :**
la représentation linéaire d'un arbre de dépendances

**Initialisation :**
choisir $A_i \in \mathcal{I}$
empiler $\langle A_i, e_0(A_i) \rangle$ dans la pile $P$

```
1 tant que  P n'est pas vide
  ⟨A, e⟩← dépiler
  si e ∈ Q_a A alors revenir en 1 ou aller en 2. sinon aller en 2.
  2 choisir une transition t = ⟨e, ⟨f, c⟩, e'⟩ ∈ δ^A(e)
    émettre t sur la bande de sortie
    si f = c = ε alors
      empiler ⟨A, e'⟩
    si f = LEX alors
      empiler ⟨A, e'⟩
    sinon
      empiler ⟨A, e'⟩
      empiler ⟨c, e_0(c)⟩
```

FIG. 7 – Procédure de génération d'arbres de dépendances sous forme linéaire

| Pile | Transition émise |
|---|---|
| $\langle V, 0\rangle$ | $\langle V, \langle 0, \langle \texttt{SUBJ}, N\rangle, 1\rangle\rangle$ |
| $\langle V, 1\rangle\langle N, 0\rangle$ | $\langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle$ |
| $\langle V, 1\rangle\langle N, 1\rangle\langle ART, 0\rangle$ | $\langle ART, \langle 0, \langle \texttt{LEX}, le\rangle, 1\rangle\rangle$ |
| $\langle V, 1\rangle\langle N, 1\rangle\langle ART, 1\rangle$ | $\langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle$ |
| $\langle V, 1\rangle\langle N, 2\rangle$ | $\langle N, \langle 2, \langle \texttt{LEX}, chat\rangle, 3\rangle\rangle$ |
| $\langle V, 1\rangle$ | $\langle V, \langle 1, \langle \texttt{LEX}, mange\rangle, 3\rangle\rangle$ |
| $\langle V, 2\rangle$ | $\langle V, \langle 2, \langle \texttt{DOBJ}, N\rangle, 3\rangle\rangle$, |
| $\langle V, 3\rangle\langle N, 0\rangle$ | $\langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle$, |
| $\langle V, 3\rangle\langle N, 1\rangle\langle ART, 0\rangle$ | $\langle ART, \langle 0, \langle \texttt{LEX}, la\rangle, 1\rangle\rangle$, |
| $\langle V, 3\rangle\langle N, 1\rangle\langle ART, 1\rangle$ | $\langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle$, |
| $\langle V, 3\rangle\langle N, 2\rangle$ | $\langle N, \langle 2, \langle \texttt{LEX}, souris\rangle, 3\rangle\rangle$, |
| $\langle V, 3\rangle$ | |

FIG. 8 – Exemple de génération d'un arbre de dépendances

### 3.2.3 Construction d'un arbre à partir de sa représentation linéaire

Un arbre de dépendances, sous sa forme conventionnelle, peut être automatiquement construit à partir de sa représentation linéaire $t_{1,n}$. L'arbre est construit à partir de sa racine de gauche à droite (le dépendant le plus à gauche d'un nœud est créé d'abord, puis son frère droit ...) et en profondeur d'abord (le sous-arbre d'un nœud est créé avant la création de son frère droit). L'algorithme utilise une pile, dans laquelle sont stockés les automates correspondant aux nœuds déjà créés mais dont tous les dépendants n'ont pas encore été créés. Un nouvel automate est ajouté à la pile, pour chaque dépendance initiale présente dans la séquence $t_{1,n}$. C'est la raison pour laquelle, comme nous l'avons précisé lors de la définition des automates dans la section 3.1.1, un automate ne possède pas de transitions aboutissant à son état initial. Il est ainsi impossible de passer deux fois par une transition initiale lors du parcours d'un automate. L'algorithme de construction est décrit plus en détails dans la figure 9.

La construction de l'arbre correspondant à l'exemple de la figure 8 est représentée par la figure 10. Dans cette dernière, les transitions constituant la représentation linéaire de l'arbre on été associées, dans la partie supérieure de la figure, aux identifieurs $t_1, \ldots, t_{11}$. La partie inférieure de la figure reproduit les étapes de la construction de l'arbre et l'état de la pile pour chacune de ces étapes. Le nœud courant est indiqué en noir.

**Entrée :**
la représentation linéaire $t_{1,n} = \langle t_1 \dots t_n \rangle$ d'un arbre de dépendances

**Sortie :**
un arbre de dépendances dont la racine est
la valeur de la variable $NC$ (pour nœud courant)

**Initialisation :**
créer un nœud $n$
$NC \leftarrow n$

`Pour` $i$ `allant de` 1 `à` $n$
  `si` $t_i$ est une transition initiale de $Auto(t_i)$ `alors`
    empiler $Auto(t_i)$
  `sinon`
    `tant que` sommet$(P) \neq Auto(t_i)$
      dépiler $P$
      $NC \leftarrow gouverneur(NC)$
  `si` $fct(t_i) =$ `LEX alors`
    étiqueter $NC$ par $cat(t_i)$
  `sinon`
    `si` $fct(t_i) \neq \varepsilon$ `alors`
      créer un nouveau nœud $N$
      établir une dépendance entre $NC$ et $N$ étiquetée par $fct(t_i)$
      $NC \leftarrow N$

FIG. 9 – Procédure de construction d'un arbre de dépendances à partir de sa représentation linéaire

$$
\begin{array}{ll}
t_1 = \langle V, \langle 0, \langle \texttt{SUBJ}, N\rangle, 1\rangle\rangle & t_7 = \langle V, \langle 2, \langle \texttt{DOBJ}, N\rangle, 3\rangle\rangle \\
t_2 = \langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle & t_8 = \langle N, \langle 0, \langle \texttt{DET}, ART\rangle, 1\rangle\rangle \\
t_3 = \langle ART, \langle 0, \langle \texttt{LEX}, le\rangle, 1\rangle\rangle & t_9 = \langle ART, \langle 0, \langle \texttt{LEX}, la\rangle, 1\rangle\rangle \\
t_4 = \langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle & t_{10} = \langle N, \langle 1, \langle \varepsilon, \varepsilon\rangle, 2\rangle\rangle \\
t_5 = \langle N, \langle 2, \langle \texttt{LEX}, chat\rangle, 3\rangle\rangle & t_{11} = \langle N, \langle 2, \langle \texttt{LEX}, souris\rangle, 3\rangle\rangle \\
t_6 = \langle V, \langle 1, \langle \texttt{LEX}, mange\rangle, 2\rangle\rangle &
\end{array}
$$

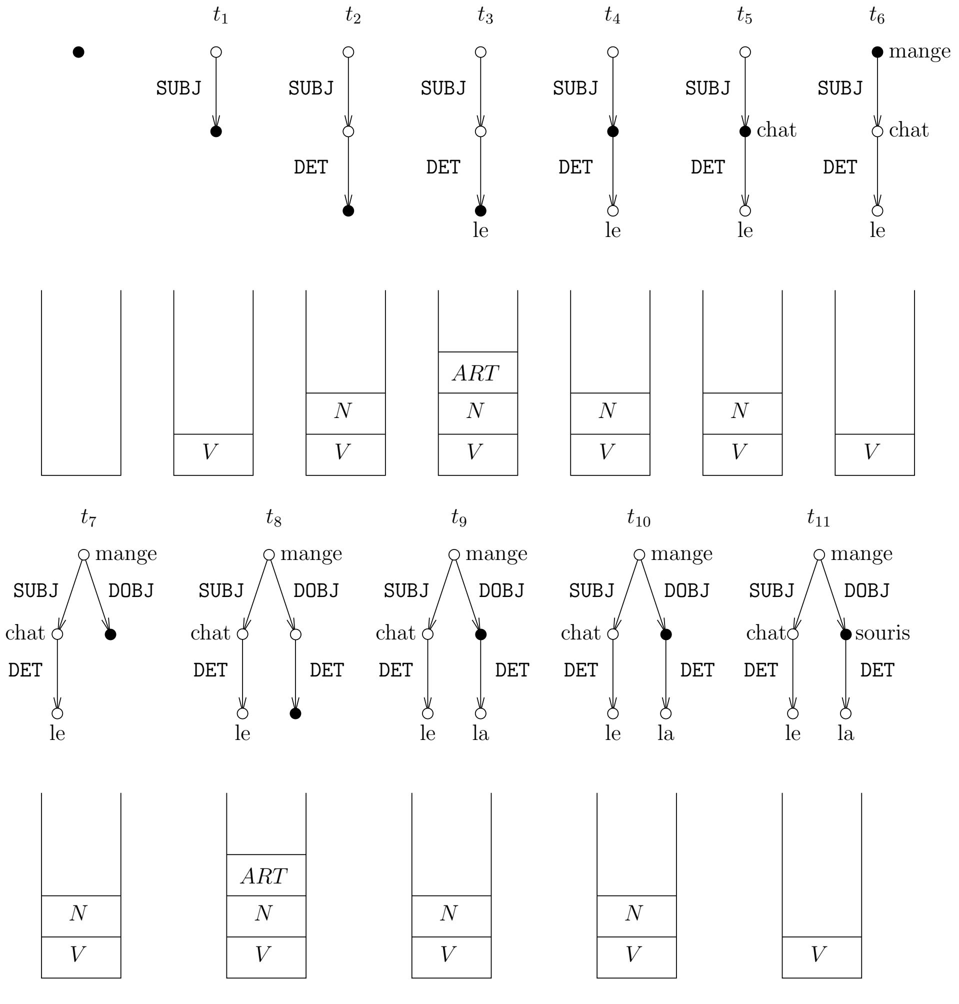



FIG. 10 – Exemple de construction d'un arbre de dépendances à partir de sa représentation linéaire

# 4 Les grammaires probabilistes

Le processus de dérivation, dans le cadre d'une grammaire générative, est généralement non déterministe. A de nombreuses étapes de la dérivation (souvent à chacune d'entre elles) un choix doit être effectué parmi plusieurs issues possibles. Dans une grammaire générative probabiliste, une probabilité est associée à chacune de ces décisions. Le produit de ces probabilités constitue la probabilité de la dérivation, ou de l'arbre produit.

Les GHC constituent le modèle algébrique d'une série récente de travaux sur les grammaires probabilistes ayant eu un grand retentissement dans le domaine du TAL. Nous désignerons ce courant sous le nom de *grammaires probabilistes contemporaines*. Il comprend en particulier les travaux de [Magerman, 1995], [Eisner, 1996], [Charniak, 1997],[Collins, 1999] et [Collins, 2003]. Nous donnerons un aperçu de ces grammaires dans la section 4.1. Cet aperçu n'est pas exhaustif. Son rôle est de décrire les principales caractéristiques des grammaires probabilistes contemporaines, et de situer le modèle que nous proposerons au chapitre 5 par rapport à elles. Nous décrirons ensuite, dans la section 4.2, les grammaires probabilistes reposant sur le formalisme des grammaires d'arbres adjoints, qui présente de nombreux points communs avec notre modèle. Nous terminerons par décrire, dans la section 4.3, l'étiquetage grammatical, ou *supertagging*. Celui-ci ne constitue pas à proprement parler une grammaire probabiliste mais il peut constituer une étape d'un processus génératif probabiliste.

## 4.1 Les grammaires fondées sur l'historique

La majorité des travaux actuels sur les grammaires probabilistes syntagmatiques s'inscrit dans le cadre général des *grammaires fondées sur l'historique* proposé par [Black et al., 1992]. Dans ce cadre, les décisions à prendre aux différentes étapes d'une dérivation concernent le choix du symbole non terminal à réécrire, ainsi que la règle permettant de réécrire ce symbole. En se restreignant au cas des dérivations gauches, le choix du symbole à réécrire est évacué car ce dernier est le symbole non terminal le plus à gauche dans la proto-phrase courante. La seule décision à prendre concerne la règle qui servira à réécrire ce non terminal. Etant donné une GHC $G$ d'axiome $S$ et un mot $m$ (dans le sens de la théorie des langages), la dérivation gauche d'un arbre $T$ correspondant à $m$ peut être représentée par une suite de $n$ arbres syntagmatiques $T_1 \dots T_n$, obtenue par l'application successive des règles $R_1, \dots, R_{n-1}$ sur la feuille non terminale la plus à gauche de la frontière de l'arbre courant :

$$S = T_1 \stackrel{R_1}{\Longrightarrow} T_2 \stackrel{R_2}{\Longrightarrow} \ldots \stackrel{R_{n-2}}{\Longrightarrow} T_{n-1} \stackrel{R_{n-1}}{\Longrightarrow} T_n = T \quad (1)$$

La probabilité associée à cette dérivation et, par conséquent, à l'arbre $T$, est le produit des probabilités associées à la décision de réécrire le symbole le plus à gauche de $T_i$ par la règle $R_i$, étant donné l'arbre $T_i$. On obtient donc[9] :

$$P(T) = \prod_{i=1}^{n-1} P(R_i|T_i) \quad (2)$$

Le modèle décrit en (2) n'est pas réaliste, car la partie conditionnelle des probabilités qu'il met en jeu[10] devient, au fur et à mesure de la dérivation, de plus en plus importante. Elle est en effet constituée de l'intégralité de l'arbre généré à cette étape de la dérivation. La variété que peuvent présenter les historiques (les arbres générés) est telle, que l'estimation des probabilités $P(R_i|T_i)$, étant donné un corpus, est quasiment impossible pour une grammaire réaliste. En effet, la majorité des événements aura une probabilité nulle d'avoir été observée dans le corpus. C'est la raison pour laquelle les historiques sont regroupés en classes d'équivalence, lesquelles consistent à ne garder que certains éléments de ces historiques. En d'autres termes, la probabilité $P(R_i|T_i)$ sera modélisée par $P(R_i|E[T_i])$, où $E[T_i]$ est la classe d'équivalence de l'historique $T_i$. Divers types de GHC probabilistes peuvent être définis dans ce cadre. Ils se distinguent par la définition qu'ils donnent des classes d'équivalences des historiques ou, en d'autres termes, par les hypothèses d'indépendances sur lesquelles ils reposent. Ce sont ces hypothèses qui permettent d'enlever certains événements de la partie conditionnelle, et d'aboutir à des modèles possédant moins de paramètres et qui sont, par conséquent, plus faciles à estimer.

On pourra remarquer que dans les grammaires fondées sur l'historique, le modèle probabiliste et le modèle algébrique (les GHC) ne reposent pas sur les mêmes hypothèses. Le modèle algébrique est hors-contexte. Il suppose

[9]Dans le cadre des grammaires fondées sur l'historique, tel que défini dans [Black et al., 1992], la dérivation est représentée de manière plus classique par une séquence de $n$ proto-phrases $\alpha_1 \ldots \alpha_n$. La probabilité $P(R_i|\alpha_i)$ est alors la probabilité de réécrire le symbole non terminal le plus à gauche de $\alpha_i$ à l'aide de la règle $R_i$, étant donné la proto-phrase $\alpha_i$. Nous avons préféré une définition dans laquelle l'application d'une règle ne dépend pas que de la proto-phrase courante, mais de l'intégralité de l'arbre généré (donc de toutes les règles utilisées jusque là dans la dérivation). Il s'agit en quelque sorte d'une généralisation de l'idée de [Black et al., 1992].

[10]Dans le contexte des processus stochastiques, les événements constituant la partie conditionnelle sont souvent appelés « l'historique » de l'événement qu'ils conditionnent.

que la règle utilisée pour réécrire un symbole non terminal ne dépend que de ce dernier et pas de son contexte, alors que le modèle probabiliste peut conditionner l'application d'une règle au contexte du symbole non terminal.

### 4.1.1 Les GHC probabilistes

Le modèle le plus simple pouvant être défini dans le cadre des grammaires fondées sur l'historique est celui des grammaires hors-contextes probabilistes (notées GHCP) introduites par [Booth, 1969]. Les GHCP associent une probabilité à chaque règle d'une GHC. Avec la contrainte que les probabilités de toutes les règles possédant le même symbole pour partie gauche somment à 1 :

$$\sum_{\alpha_i} P(A \rightarrow \alpha_i) = 1 \ , \ \forall A \tag{3}$$

Dans un tel modèle, les probabilités sont associées directement aux règles, indépendamment d'un historique. On peut voir ce modèle comme le plus simple dans le paradigme des grammaires fondées sur l'historique, celui pour lequel le seul élément de l'historique pris en compte est le symbole non terminal à réécrire. Il s'agit en quelque sorte du modèle probabiliste le plus proche de l'esprit des grammaires hors-contexte dans la mesure où les probabilités sont aussi des probabilités hors-contexte. Les modèles algébriques et probabilistes sont homogènes.

La probabilité d'un arbre $T$, obtenu par application successive des règles $R_1 \dots R_{n-1}$, s'écrit donc tout simplement[11] :

$$P(T) = \prod_{i=1}^{n-1} P(R_i) \tag{4}$$

Les GHCP et les nombreuses hypothèses d'indépendance sur lesquelles elles reposent constituent un bon point de départ pour introduire diverses grammaires probabilistes, et préciser les différentes hypothèses d'indépendance que ces dernières permettent d'atténuer. Parmi ces hypothèses, deux sont régulièrement évoquées par les promoteurs de modèles alternatifs. Nous les présenterons dans les deux sous-sections suivantes.

[11] La probabilité $P(R)$, où $R$ est la règle $A \rightarrow \alpha$, peut aussi être vue comme une probabilité conditionnelle : $P(\alpha|A)$. Les deux notations coexistent dans la littérature des grammaires probabilistes.

### 4.1.2 L'hypothèse d'indépendance structurale

La première hypothèse mise en cause est l'hypothèse d'indépendance structurale. Celle-ci stipule, comme nous l'avons vu ci-dessus, que le choix d'une règle pour la réécriture d'un symbole est indépendant du contexte de ce dernier. Cette hypothèse est facilement mise en défaut, comme l'ont montré [Jurafsky & Martin, 2000] et [Resnik, 1992], en remarquant qu'en anglais, la probabilité qu'un $GN$ se réalise comme un pronom dépend de façon cruciale de la fonction syntaxique qu'il occupe dans la phrase . En particulier, en position sujet, cette probabilité sera élevée alors qu'en position d'objet direct, elle sera faible. Dans une GHCP comportant les deux règles $GN \rightarrow PRO$ et $GN \rightarrow DET\ N$, chacune de ces dernières est associée à une probabilité qui est indépendante du contexte. Il est par conséquent impossible de conditionner par le contexte du $GN$ le fait que ce dernier se réécrive $DET\ N$ ou $PRO$.

Diverses solutions ont été apportées à ce problème par les grammaires probabilistes contemporaines. Elles consistent en général à choisir dans l'historique de la probabilité d'une règle un certain nombre d'éléments importants à prendre en compte pour l'estimation de sa probabilité. Ce choix peut être réalisé de manière manuelle par le concepteur du modèle, comme dans [Collins, 1999] ou [Black et al., 1992], ou automatiquement, par des algorithmes de classification, comme dans [Magerman, 1995]. Dans le modèle de [Collins, 1999], par exemple, la probabilité d'une règle $P \rightarrow HR_1R_2$ est conditionnée par la longueur de la chaîne dérivée à partir du non terminal $R_1$. Dans le modèle de [Magerman, 1995], l'application d'une règle $A \rightarrow \alpha$, par exemple, pourra dépendre du non terminal ayant introduit $A$ à une étape précédente de la dérivation, si l'algorithme de classification considère que cet élément d'information est pertinent.

### 4.1.3 L'hypothèse d'indépendance lexicale

La seconde hypothèse généralement mise en cause est l'hypothèse d'indépendance lexicale. Dans une GHC, l'introduction d'un mot dans une dérivation passe par la réécriture d'un symbole pré-terminal à l'aide d'une règle lexicale (règle ayant un pré-terminal pour partie gauche et un élément lexical pour partie droite). Ainsi, dans le cas d'une phrase comportant un verbe et un syntagme prépositionnel, . . . *penser à* $DET\ N$ . . . par exemple, la génération de la préposition *à* est réalisée indépendamment de la réalisation du verbe *penser*. Il est par conséquent impossible de représenter dans le modèle la dépendance lexicale pouvant exister entre ces deux éléments.

La solution classique, pour la prise en compte de telles dépendances dans les GHCP, passe par l'introduction, dans les règles de la grammaire, de la notion de *tête lexicale*. Une manière simple de prendre en compte ces dernières consiste à définir les symboles non terminaux de la grammaire comme des couples $\langle c, m \rangle$ où $c$ est une catégorie syntagmatique, et $m$ est la tête lexicale du syntagme. Ainsi, la grammaire distinguera par exemple, les non terminaux $\langle GV, penser \rangle$ et $\langle GV, manger \rangle$. Une règle de la forme :

$GV \rightarrow \ V \ GP$

donnera lieu, entre autres, à la règle :

$\langle GV, penser \rangle \rightarrow \ \langle V, penser \rangle \ \langle GP, à \rangle$

La probabilité de cette dernière dépend donc de la nature des têtes lexicales des syntagmes $V$ et $GP$. Cette solution, ou des variantes, ont été mises en œuvre dans les grammaires probabilistes contemporaines. On pourra remarquer que les événements élémentaires pris en compte dans ce genre d'approches sont précisément l'établissement de dépendances. Une règle de la forme :

$\langle GV, penser \rangle \rightarrow \ \langle V, penser \rangle \ \langle GP, à \rangle$

correspond à l'établissement d'une dépendance entre le verbe *penser* et la préposition *à*. Sa probabilité correspond à celle d'établir la dépendance citée. La prise en compte de l'importance des relations entre mots, et la représentation explicite de ces relations dans les grammaires probabilistes contemporaines - pourtant fondées sur le modèle syntagmatique - a donné naissance à des modèle hybrides, curieux mélange de grammaires syntagmatiques et de grammaires de dépendances. Ces grammaires sont aussi appelées grammaires *bilexicales* du fait que leur modèle probabiliste prend en compte la nature lexicale du gouverneur et du dépendant.

Les approches bilexicales ont connu succès important, et c'est sur elles que reposent, jusqu'à ce jour, les meilleurs analyseurs syntaxiques probabilistes. Néanmoins, des études récentes ([Bikel, 2004], [Gildea, 2001]) ont montré que les probabilités bilexicales n'étaient pas à l'origine des performances de l'analyseur de Michael Collins ([Collins, 1999]) qui est l'un des analyseurs les plus performants à cette date[12]. La faible influence des probabilités bilexicales

[12]L'analyseur de Collins repose sur les GHCP, auxquelles il ajoute un grand nombre de dépendances, dont les dépendances lexicales ne sont qu'un aspect. C'est en enlevant les dépendances lexicales du modèle et en répétant les expériences de Collins que la faible influence des dépendances lexicales a été révélée par [Gildea, 2001].

provient du fait que les événements qu'elles modélisent (des co-occurrences lexicales) sont assez rares pour que la majeure partie des co-occurrences apparaissant dans une phrase à analyser n'ait pas été observée dans un corpus d'apprentissage et que, par conséquent, leur probabilité est nulle. Dans de tels cas, l'analyseur utilise d'autres probabilités, plus grossières, qui ne prennent pas en compte la nature lexicale des dépendants.

La situation pourrait être résumée de la façon suivante. D'une part, les grammaires ne prenant en compte que la partie du discours semblent trop grossières pour modéliser certaines distinctions. D'autre part, les grammaires bilexicales semblent trop fines. Lors de l'analyse, les événements qui se présentent à l'analyseur n'ont souvent pas été observés dans le corpus d'apprentissage. Nous verrons en 4.3 une solution possible à ce problème.

## 4.2 Les TAG probabilistes

Les réponses apportées aux problèmes que posent les hypothèses d'indépendance structurale et lexicale des GHCP par différentes approches fondées sur les GHC ne constituent pas un cadre formel homogène. Elles s'inscrivent, en général, dans le paradigme des grammaires fondées sur l'historique. Mais ce dernier n'est pas un formalisme. Le formalisme utilisé reste celui des GHC et, comme nous l'avons noté ci-dessus, il repose sur des hypothèses différentes de celles des modèles probabilistes. Ces derniers mettent en jeu des dépendances qui ne sont pas représentées dans les règles de réécriture hors-contexte. Il est en revanche toute une série de travaux qui s'appuient sur le formalisme des TAG et qui apportent une réponse plus homogène à ces problèmes d'indépendance. L'avantage que présentent les TAG, par rapport aux GHC, est que le modèle algébrique qu'elles définissent est plus riche : les arbres élémentaires constituent des objets plus complexes que les règles hors-contexte. Cette richesse va permettre d'articuler plus facilement les modèles probabiliste et algébrique.

Les deux avantages des TAG vis-à-vis des deux hypothèses d'indépendance tiennent d'une part, dans le caractère lexical du formalisme (chaque arbre élémentaire est associé à une entrée lexicale) et d'autre part, au domaine de localité étendu, qui offre une réponse simple et élégante au problème de l'hypothèse structurale. Dans une TAG probabiliste ([Schabes, 1992, Resnik, 1992, Carroll & Weir, 1997, Chiang, 2000]) (dorénavant PTAG), comme dans une TAG, une dérivation est une suite d'opérations d'adjonctions et de substitutions. Chacune de ces opérations met en jeu quatre éléments : la nature de l'opération (adjonction ou substitution), un arbre élémentaire dans lequel s'effectue l'opération, l'arbre élémentaire qui est substitué ou adjoint et le nœud sur lequel est réalisée l'opération. La probabilité associée à une dérivation est le produit des probabilités des opérations.

Il est facile de voir, sur l'exemple trivial de la figure 11, les réponses qu'apportent naturellement les PTAG aux deux problèmes mentionnés ci-dessus. Les deux opérations de substitution des arbres $\alpha_1$ et $\alpha_3$, en positions sujet et objet de l'arbre $\alpha_2$, correspondent à deux événements distincts, auxquels sont associées des probabilités différentes. Ceci permet de résoudre le problème structural soulevé ci-dessus, sans mettre en jeu des éléments extérieurs aux arbres élémentaires participant à l'opération. D'autre part, la probabilité associée à la substitution de $\alpha_3$ dans $\alpha_2$ prend en compte l'ancre lexicale de ces deux arbres, apportant ainsi une réponse aux problèmes posés par l'hypothèse d'indépendance lexicale des GHCP, sans toutefois apporter de réponse au problème d'estimation des paramètres des modèles bilexicaux !

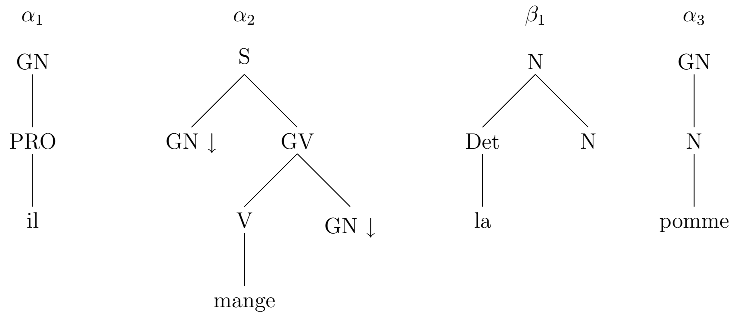



FIG. 11 – Exemple de TAG

## 4.3 Le supertagging

Le *supertagging* ou étiquetage grammatical ([Bangalore, 1997, Bangalore & Joshi, 1999b]) combine un modèle algébrique (une TAG) et un modèle probabiliste (une chaîne de Markov cachée) d'une manière originale. Il ne s'agit pas à proprement parler d'une grammaire probabiliste, car elle n'associe pas une probabilité à des structures syntaxiques mais à des couples $\langle T_{1,n}, m_{1,n}\rangle$. Ceux-ci sont composés d'une séquence $T_{1,n}$ d'arbres élémentaires de la grammaire[13] et d'une séquence de mots $m_{1,n}$, où chaque mot $m_i$ est une ancre possible du schéma d'arbre élémentaire $T_i$. Les *supertags* ne sont rien d'autre que les noms des

[13]Il s'agit plus précisément d'une séquence de *schémas* d'arbres élémentaires.

schémas d'arbres élémentaires, vus comme des parties de discours d'un type particulier, intégrant un nombre important des propriétés lexico-syntaxiques de leur ancre (d'où le préfixe *super*). En ce sens, les *supertags* sont très proches des catégories des GDG.

Le modèle sous-jacent au *supertagging* est celui des chaînes de Markov cachées. Les probabilités mises en jeu dans un tel modèle sont très différentes de celles évoquées jusque là. Il s'agit des probabilités classiques des chaînes de Markov cachées : probabilité d'occurrence d'un mot étant donné un *supertag* (les probabilités d'émission du modèle de Markov caché) et probabilité d'occurrence d'un *supertag* étant donné les $n$ *supertags* précédents dans la chaîne linéaire (les probabilités de transition). Le *supertagging* est utilisé comme moyen de désambiguïsation. Il permet de déterminer, étant donné une séquence de mots, la séquence de *supertags* la plus probable. A l'issue de ce traitement, tout mot de la phrase auquel peuvent correspondre plusieurs *supertags* s'en voit attribuer un seul. Le *supertagging* est en cela très proche de l'étiquetage morpho-syntaxique ([Church, 1988]). Il s'en distingue principalement par la taille du jeu d'étiquettes.

L'idée sous-jacente au *supertagging* est que, s'il est possible de mener à bien cette tâche, alors le problème de l'analyse syntaxique est grandement facilité. Ceci s'explique par le fait qu'à l'issue du *supertagging*, un sous-ensemble de la grammaire initiale à été déterminé (l'ensemble des arbres élémentaires sélectionnés durant le *supertagging*). C'est à l'aide de ce seul sous-ensemble que l'analyse syntaxique sera réalisée. Cette dernière se résume à combiner entre eux les arbres élémentaires correspondant aux *supertags*. Les expériences de *supertagging* menées avec la grammaire XTAG ([Doran et al., 1994]) sur le Penn Treebank ont montré que cette approche était viable. Le taux de précision pour un modèle tri-gramme est de l'ordre de 92%, il atteint 97% en prenant les trois meilleures solutions de l'étiqueteur.

La notion de *supertag* peut constituer une solution au problème de l'inadéquation des parties de discours et des mots, évoqué en 4.1.3, pour résoudre le problème de l'hypothèse d'indépendance lexicale. En effet, les *supertags* constituent un niveau de description intermédiaire entre le mot et la partie de discours, qui va permettre de décomposer le processus génératif en deux étapes. Dans une première étape, un processus génératif proche de celui des TAG permet de générer un arbre dont les éléments lexicaux n'ont pas été sélectionnés. L'avantage de ce modèle probabiliste est qu'il possède moins de paramètres qu'un modèle lexical : les événements qu'il modélise mettent en jeu des *supertags*, dont le nombre est plus réduit que celui des mots. Dans un deuxième temps, une séquence de mots est générée à partir des *supertags* produits lors de la première étape.

# 5 Grammaires de dépendances génératives probabilistes

Nous avons rappelé, dans le chapitre 4, que la probabilité de générer un arbre étant donnée une grammaire probabiliste, se décomposait en un produit de probabilités d'événements élémentaires, qui dépendaient de la nature du modèle algébrique sous-jacent. Dans le cas des GHCP, ces événements correspondent à la réécriture d'un symbole non terminal par une règle. Dans le cas des PTAG, ils correspondent à la combinaison de deux arbres élémentaires. Dans le cas des GDG probabilistes (GDGP), le processus de génération, tel qu'il a été défini au chapitre 3, se décompose en une séquence de franchissements de transitions des automates de la grammaire. Ces opérations de franchissement constituent les événements élémentaires sur lesquels vont être construits les modèles probabilistes. Ces derniers sont intimement liés à la structure des automates. Différents modèles probabilistes peuvent être définis, qui induiront des automates de structures différentes. Pour reprendre la distinction entre modèle algébrique et modèle probabiliste, on peut dire que dans le cas des GDGP, le modèle algébrique peut s'adapter à des modèles probabilistes différents, tout en restant dans le cadre des GDGP. La principale conséquence pratique de ce fait est qu'un seul algorithme d'analyse sera défini pour les GDGP et pourra être utilisé pour des grammaires implémentant des modèles probabilistes différents.

Nous donnerons, dans la section 5.1, la définition des GDGP, puis nous verrons en 5.2 les hypothèses d'indépendance sur lesquelles elles reposent. Trois types de GDGP, implémentant des modèles probabilistes différents, seront décrits en 5.3. Nous terminerons le chapitre par les définitions, dans la section 5.4, de la probabilité d'un arbre et d'une phrase.

*Les résultats décrits dans ce chapitre ont partiellement été publiés dans [Nasr & Rambow, 2004b]. On y trouve en particulier l'idée d'adapter la structure des automates aux modèles probabilistes implémentés. Cette idée trouve, en partie, son origine dans des travaux plus anciens sur les modèles de langage en reconnaissance de la parole décrits dans [Nasr et al., 1999].*

## 5.1 Définition

Une grammaire de dépendances générative probabiliste (GDGP) est une GDG dans les automates de laquelle des probabilités sont affectées aux transitions. De plus, chaque élément $c_i$ de l'ensemble des symboles initiaux de la grammaire est associé à une probabilité, qui est notée $\pi(c_i)$ et appelée *probabilité initiale.*

Formellement, une GDGP est définie comme un sextuplet $\langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I}, \pi \rangle$, dans lequel $\mathcal{C}$, $\Sigma$, $\mathcal{F}$ et $\mathcal{I}$ gardent la même définition que dans une GDG. $\mathcal{A}$ est un ensemble d'automates lexicalisés *probabilistes*, et $\pi$ est la distribution de probabilités initiales, appelée elle-même *distribution initiale.*

Un automate lexicalisé probabiliste $A$ est défini comme un sextuplet $\langle \mathcal{Q}, e_o, \mathcal{F}, \mathcal{C}, \Sigma, \delta, \mathcal{Q}_a \rangle$, où $\mathcal{Q}$, $e_0$, $\mathcal{F}$, $\mathcal{C}$ et $\mathcal{Q}_a$ gardent la même définition que dans un automate lexicalisé non probabiliste. Les deux types d'automates se distinguent par le fait que chaque transition d'un automate probabiliste est affectée d'une probabilité $p$ ($\delta \subseteq \mathcal{Q} \times ((\mathtt{LEX} \times \Sigma) \cup (\mathcal{F} \times \mathcal{C})) \times [0,1] \times \mathcal{Q}$). La probabilité d'une transition $t$ est notée $p(t)$.

## 5.2 Hypothèses d'indépendance

Nous avons évoqué, en 4.1.3 et en 4.1.2, les deux hypothèses d'indépendances structurale et lexicale sur lesquelles reposaient les GHCP ainsi que les réponses qu'apportaient diverses grammaires probabilistes aux problèmes posés par de telles hypothèses. Nous allons voir dans les deux sous-sections suivantes les réponses apportées à ces problèmes par les GDGP.

### 5.2.1 L'hypothèse d'indépendance structurale

La principale hypothèse d'indépendance intrinsèque au modèle des GDGP est l'hypothèse markovienne, provenant de l'utilisation d'automates probabilistes. Cette hypothèse stipule que la probabilité d'une transition dépend de l'événement dénoté par l'étiquette de la transition, ainsi que de l'état duquel émane la transition, et pas du chemin ayant été suivi dans l'automate pour accéder à cet état. La probabilité d'une transition étiquetée $c$, émanant de l'état $e_o$ et aboutissant à l'état $e_d$ ($\langle e_o, c, e_d \rangle$) correspond à la probabilité conditionnelle $P(c|e_o)$. La correspondance entre le modèle probabiliste et le modèle algébrique s'effectue par l'intermédiaire de la structure des automates de la grammaire, plus précisément par la signification des états des automates. Cette dernière sera différente pour des modèles probabilistes différents.

Cette idée peut être illustrée simplement sur l'exemple de la figure 12. Les trois automates de cette figure définissent le même langage ($a^*$). Cependant, ils possèdent des structures différentes, et ils associent des significations différentes aux états. Ces derniers représentent les $n$ derniers symboles ayant été générés dans le chemin menant à l'état. La valeur de $n$ est différente pour les trois automates : 0 pour le premier, 1 pour le second et 2 pour le troisième[14]. Les trois automates peuvent associer des probabilités différentes aux mêmes mots. La probabilité du mot $aaa$, par exemple, vaudra dans le premier cas (automate de gauche) :

$P(aaa) = P(a)^3$,

dans le deuxième cas :

$P(aaa) = P(a) \times P(a|a)^2$

et dans le troisième :

$P(aaa) = P(a) \times P(a|a) \times P(a|aa)$.

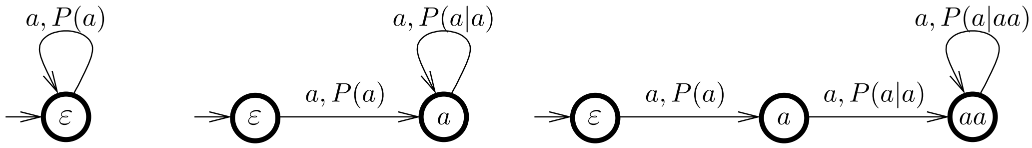



FIG. 12 – Trois automates reconnaissant le même langage mais implémentant des modèles probabilistes différents

On peut établir ici le lien entre le cadre général des grammaires fondées sur l'historique évoqué en 4.1 et les GDGP. Dans une GDGP, un événement est matérialisé par l'étiquette d'une transition, et l'historique est représenté par l'état duquel émane la transition. Cet historique dépend, comme nous l'avons vu ci-dessus, de la structure des automates. Mais, quelle que soit cette dernière, l'historique est entièrement représenté dans l'automate : la probabilité d'un événement représenté par une transition d'un automate ne peut être conditionnée par un événement extérieur à ce dernier. En ce sens, les GDGP sont proches des GHCP et des PTAG : la probabilité d'une règle ne dépend pas d'événements extérieurs à la règle.

Nous verrons, en 5.3, trois types de GDGP qui définissent des automates de structures différentes, implémentant des modèles probabilistes différents. En plus de l'hypothèse de Markov, propre aux cadre général des GDGP, ces trois types de grammaires imposent une hypothèse d'indépendance structurale plus contraignante, qui est l'*hypothèse d'indépendance des sites*. Cette

[14]Nous avons choisi de donner aux états des noms correspondant aux $n$ derniers symboles générés, afin que les probabilités $P(a|\textit{état})$ et $P(a|\textit{n derniers symboles})$ s'écrivent de la même manière.

dernière stipule que le choix d'une transition dans un site est indépendant des choix effectués dans les autres sites de l'automate. Cette contrainte n'est pas intrinsèque aux GDGP. On peut envisager des GDGP ne la respectant pas.

### 5.2.2 L'hypothèse d'indépendance lexicale

Plusieurs réponses peuvent être apportées à la question de l'indépendance lexicale, selon le nombre d'ancres associées à un automate ; en d'autres termes, le nombre de transitions lexicales qu'il comprend. Lorsqu'un automate comporte une seule ancre, alors les probabilités associées aux transitions de l'automate correspondent aux probabilités d'attachement de cet item lexical particulier. On se trouve alors dans le cas des probabilités bilexicales, du fait que chaque transition d'attachement correspond à l'établissement d'une dépendance entre deux items lexicaux. Comme nous l'avons évoqué dans le chapitre 4, un tel modèle est confronté à de sévères problèmes d'estimation. En augmentant le nombre d'ancres par automate, on décide de ne plus distinguer ces items lexicaux du point de vue des probabilités des dépendances qu'ils peuvent établir. On diminue ce faisant le nombre de catégories et, par conséquent, le nombre de paramètres du modèle probabiliste. En poussant à l'extrême ce processus de diminution du nombre de catégories, on peut aboutir à un seul automate (une seule catégorie) qui agrège toutes les règles de la grammaire. Nous adopterons une position intermédiaire, qui consiste à distinguer des catégories à partir de critères syntaxiques, en particulier leur valence active (le nombre et la nature de leurs dépendants). Ainsi, tous les éléments lexicaux partageant la même valence active définissent une catégorie et le modèle probabiliste ne les distinguera pas les uns des autres : les probabilités seront définies sur l'espace des catégories.

## 5.3 Types de GDGP

Les trois types de GDGP définis ci-dessous respectent l'hypothèse d'indépendance des sites évoquée dans la section précédente. Ils se distinguent par la structure interne des sites, plus précisément des sites répétables. Le premier modèle décrit, appelé modèle initial et noté GDGP-INIT, définit des automates dont la structure est celle des automates canoniques. Les deux modèles suivants se distinguent de GDGP-INIT en proposant des sites répétables plus complexes, permettant de modéliser plus finement le phénomène de l'attachement multiple à un site, principale source d'ambiguïté. C'est en particulier par ce moyen que l'on pourra contrôler les rattachements prépositionnels.

### 5.3.1 Le modèle initial : GDGP-INIT

Comme nous l'avons vu en 3.1.2, un automate canonique se présente sous la forme d'une séquence de sites reliés entre eux par des transitions inter-sites. Les sites lexicaux, les sites obligatoires et les sites optionnels se présentent comme des faisceaux de transitions reliant deux états. Les sites répétables se présentent, eux, comme des ensembles de transitions modificatrices bouclant sur un même état. C'est cette caractéristique qui permet de modéliser des dépendances répétables. Les différents types de transitions d'un automate correspondent à des événements de nature différente, dont la probabilité est représentée par la probabilité de la transition.

Les probabilités d'attachement s'interprètent assez facilement. La probabilité d'une transition étiquetée $\langle f, c_2 \rangle$ sur un site $i$ de l'automate $c_1$ correspond à la probabilité de l'attachement de l'automate $c_2$ sur le site $i$ de l'automate $c_1$.

Les probabilités des transitions d'un site obligatoire correspondent à la probabilité de choisir un automate donné pour pourvoir le site. Chaque site obligatoire devant être pourvu au plus par un seul automate, la somme des probabilités des transitions du site est égale à 1. L'intégralité de la masse des probabilités est par conséquent répartie sur les transitions du site. Lorsque le site est optionnel, alors ce dernier comporte une transition vide. La probabilité de cette transition est celle que le site ne soit pas pourvu.

Dans le cas d'un site répétable, la situation est plus délicate, dans la mesure où, contrairement aux autres sites, un site répétable peut décrire plusieurs attachements successifs. On ne sait à l'avance combien d'attachements différents s'effectueront sur ce site lors d'une dérivation, ni, par conséquent, sur quel nombre d'événements répartir la masse de probabilité. Lorsque plusieurs attachements d'un même automate se produisent sur un même site, ils possèdent tous la même probabilité. Intuitivement, cette modélisation semble bien pauvre, car la probabilité d'occurrence d'un premier attachement à un site devrait être différente de la probabilité de deuxième occurrence ...Nous verrons dans les sections 5.3.2 et 5.3.3 comment modéliser plus finement ces phénomènes.

La probabilité associée à une transition lexicale étiquetée $\langle \texttt{LEX}, m \rangle$, dans un automate $c$ correspond à la probabilité conditionnelle $P(m|c)$. En d'autres termes, c'est la probabilité de choisir le mot $m$ comme ancre de l'automate $c$[15].

[15] Il ne s'agit pas de la probabilité, plus intuitive, que le mot $m$ ancre l'automate $c$, qui s'écrit $P(c|m)$.

Tout chemin de l'automate passant par une transition lexicale et une seule, la somme des probabilités lexicales de l'automate est égale à 1. Dans le cas des schémas d'automates, les sites lexicaux sont réduits à une seule transition et la masse de probabilité est répartie uniformément entre les différents sites lexicaux de l'automate.

Les probabilités de transition décrivent la probabilité de passer d'un site à un autre. Lorsque les automates sont sous forme canonique, ces transitions ne présentent pas d'intérêt, dans la mesure où les automates se présentent comme une suite linéaire de sites. D'un site donné, on ne peut aller, au plus, que vers un seul autre site. La probabilité de chacune de ces transitions est par conséquent égale à 1.

### 5.3.2 Le modèle positionnel : GDGP-POS

Le modèle positionnel, que nous noterons GDGP-POS, se distingue de GDGP-INIT par la solution qu'il apporte au problème d'attachements multiples sur un site répétable. Cette solution se traduit par une transformation de la structure des sites répétables. Lorsque plusieurs attachements successifs sont effectués sur un tel site, chacun dispose d'une distribution de probabilité qui lui est propre. Pour distinguer les différents attachements, nous dirons que le premier correspond à la position 1, le second à la position 2 ... D'un point de vue structurel, un site répétable distingue maintenant plusieurs positions. Chacune est représentée par un état duquel émane autant de transitions qu'il peut y avoir d'attachements différents à cette position. Le nombre de positions distinguées est un paramètre du modèle. Nous avons représenté, dans la partie droite de la figure 13, un site répétable sur lequel peuvent s'attacher les deux automates $c_1$ et $c_2$. Ce site distingue deux positions, représentées par les états 1 et 2. La transition vide reliant l'état 1 à l'état 4 correspond à la probabilité qu'aucun attachement n'ait été réalisé sur le site. La transition vide reliant 2 à 4 correspond à la probabilité qu'un seul attachement ait été réalisé et la transition vide reliant 3 à 4 correspond à la probabilité qu'au moins deux attachements aient été réalisés. Les autres transitions émanant de l'état 3 correspondent aux attachements d'ordre supérieur à deux. On ne distingue plus, à ce niveau, la position de l'attachement.

Le site répétable sous forme canonique, tel qu'il est représenté dans la partie gauche de la figure 13, et le modèle positionnel, de la partie droite, vus comme de simples automates pondérés, reconnaissent tous deux le même langage ($L = \{c_1, c_2\}*$). Ils se distinguent par la probabilité qu'ils associent aux différents mots de $L$. Dans le cas du modèle initial :

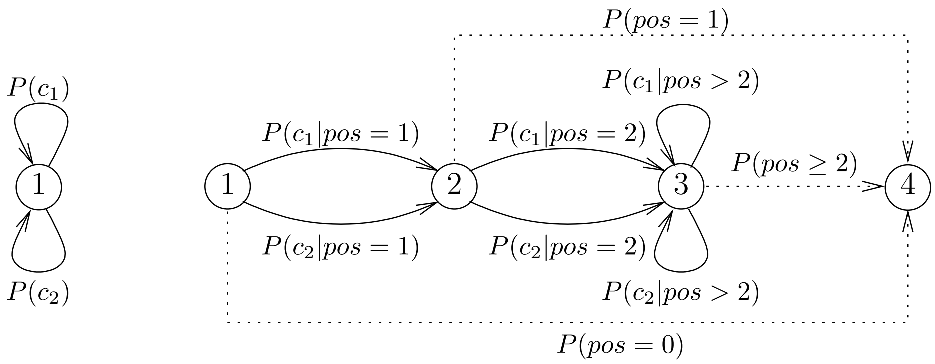



FIG. 13 – Site répétable canonique et site répétable à 2 positions

$$P(c_1\ c_2\ c_1) = P(c_1) \times P(c_2) \times P(c_1)$$

alors que pour le modèle positionnel :

$$P(c_1\ c_2\ c_1) = P(c_1|pos = 1) \times P(c_2|pos = 2) \times P(c_1|pos > 2) \times P(pos \geq 2)$$

### 5.3.3 Le modèle bigramme : GDGP-2G

Le modèle positionnel présenté ci-dessus distinguait, dans le cas d'attachements multiples à un site répétable, le premier attachement, du second ... mais ne prenait pas en compte la nature des attachement précédents (ou suivants) pour modéliser la probabilité d'un attachement particulier. Cette information est prise en compte dans le modèle bigramme, que nous noterons GDGP-2G. Celui-ci prend en compte, dans la modélisation d'un attachement, la nature de l'attachement précédent, comme l'illustre la figure 14.

Formellement, ces automates correspondent à des chaînes de Markov d'ordre 1, représentées sous la forme d'automates pondérés ([Nasr et al., 1999], [Allauzen et al., 2003]). Dans l'automate de la figure 14,

L'état 2 (respectivement 3) de l'automate correspond au fait que l'attachement précédent soit l'automate $c_1$ (respectivement $c_2$). La transition reliant l'état 1 à l'état 2 (respectivement 3) correspond à la probabilité que le premier attachement à ce site soit l'automate $c_1$ (respectivement $c_2$). Cette probabilité s'écrit $P(c_1|\texttt{BEGIN})$ (respectivement $P(c_2|\texttt{BEGIN})$. Enfin, la transition vide menant de l'état 2 (respectivement 3) vers l'état 4 correspond au fait que le dernier attachement sur le site soit l'automate $c_1$ (respectivement $c_2$). Cette probabilité s'écrit $P(\texttt{END}|c_1)$ (respectivement $P(\texttt{END}|c_2)$. Le modèle bigramme peut être étendu à un modèle d'ordre supérieur, au prix d'une augmentation de la complexité des automates et du nombre de paramètres à estimer.

Les deux modèles GDGP-POS et GDGP-2G sont complémentaires dans la mesure où ils modélisent des phénomènes différents.

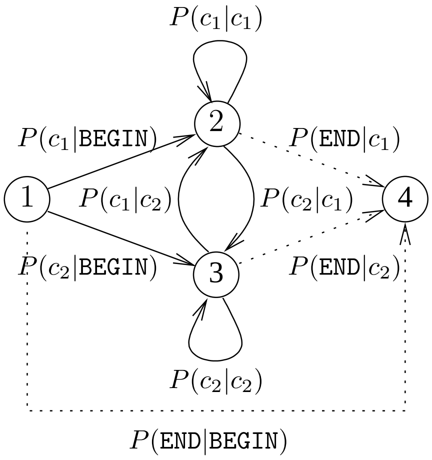



FIG. 14 – Site répétable bigramme

Le modèle de la figure 14, vu comme un automate, reconnaît le même langage que le modèle de la figure 13. Il associe au mot $c_1\ c_2\ c_1$ la probabilité suivante :

$$P(c_1\ c_2\ c_1) = P(c_1|\texttt{BEGIN}) \times P(c_2|c_1) \times P(c_1|c_2) \times P(c_1|\texttt{END})$$

## 5.4 Probabilité d'un arbre et probabilité d'une phrase

Comme dans toutes les grammaires génératives probabilistes, la probabilité d'un arbre $T$, étant donné une GDGP $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I}, \pi \rangle$ est le produit des probabilités des choix effectués lors de la génération de $T$. Le premier choix porte sur la sélection d'un élément de $\mathcal{I}$ comme premier automate de la dérivation, puis chacun des autres choix correspond au franchissement d'une transition dans un automate de la grammaire, lors du processus décrit dans l'algorithme de la section 3.2. La probabilité d'un arbre $T$, dont la représentation linéaire est $t_{1,n}$, est représentée en (5).

$$P(T = t_{1,n}) = \pi(Auto(t_1)) \times P(t_{1,n}) = \pi(Auto(t_1)) \times \prod_{i=1\ldots n} P(t_i) \qquad (5)$$

Comme nous l'avons expliqué ci-dessus, la probabilité d'une transition $t$ est en réalité la probabilité conditionnelle de l'étiquette de $t$, étant donné l'état duquel émane $t$ ($P(\langle fct(t), cat(t)\rangle | orig(t))$). L'équation (5) peut donc se réécrire :

$$P(T = t_{1,n}) = \pi(Auto(t_1)) \times \prod_{i=1\ldots n} P(\langle fct(t_i), cat(t_i)\rangle | orig(t_i)) \qquad (6)$$

L'équation (6) est valable pour les trois types de GDGP vus ci-dessus et, de façon plus générale, pour tout type de GDGP. De la même manière que la représentation linéaire $t_{1,n}$ d'un arbre de dépendances permet de construire l'arbre par concaténation de sous-séquences de $t_{1,n}$, cette représentation permet de calculer la probabilité de $t_{1,n}$ par simple multiplication des probabilités de sous-séquences de ce dernier.

La probabilité associée à une phrase $S$ par une grammaire générative probabiliste $G$ est définie de façon standard, comme la somme des probabilités des différents arbres syntaxiques que $G$ permet d'associer à $S$, notés $T_G(S)$, comme l'illustre l'équation (7). La probabilité jointe $P(T, S)$ peut se réécrire grâce à la règle des probabilités conditionelles (équation (8)). Elle peut ensuite être simplifiée en remarquant que les arbres produits par le processus génératif sont ordonnés et que, par conséquent, une seule suite de mots correspond à un arbre, d'où $P(S|T) = 1$ et l'équation (9).

$$\begin{aligned} P(S) &\stackrel{def}{=} \sum_{T \in T_G(S)} P(T, S) && (7)\\ &= \sum_{T \in T_G(S)} P(S|T) \times P(T) && (8)\\ &= \sum_{T \in T_G(S)} P(T) && (9) \end{aligned}$$

# 6 Relation avec les grammaires d'arbres adjoints

Les GDG possèdent une relation privilégiée avec les TAG, et plus particulièrement un sous-ensemble de ces dernières : les grammaires d'insertion d'arbres (dorénavant TIG). Nous allons décrire dans ce chapitre une procédure de transformation des TIG en GDG. La mise en œuvre de cette transformation revêt une importance pratique, du fait qu'elle sera utilisée dans les expériences décrites dans la partie III. En effet, les grammaires utilisées lors de ces expériences ont été extraites, sous la forme de TIG, à partir de corpus arborés. Elles ont ensuite été transformées en GDG pour mener à bien les expériences. Nous rappellerons dans la section 6.1 la définition des TIG et leur relation avec les TAG, puis nous décrirons en 6.2 la construction de grammaires GDG à partir de TIG.

*L'algorithme de transformation des* TIG *en* GDG *est partiellement décrit dans [Nasr et al., 2002]. Il a aussi été utilisé dans un cadre différent pour le projet WordsEye de AT&T [Rambow et al., 2002].*

## 6.1 Grammaires d'insertion d'arbres

Les grammaires d'insertion d'arbres sont une variante des grammaires d'arbres adjoints, introduite dans [Schabes & Waters, 1995]. A l'instar des TAG, les TIG définissent des arbres élémentaires initiaux et auxiliaires, et deux opérations de réécriture d'arbres : l'adjonction et la substitution. Cependant, les TIG introduisent des restrictions sur les arbres élémentaires, dont le résultat est de réduire la puissance générative du formalisme : on passe d'un système permettant de reconnaître des langages faiblement dépendants du contexte ([Joshi et al., 1975]) à un système ne permettant de reconnaître que les langages hors-contexte. La contrepartie de cette perte de pouvoir génératif est l'existence d'algorithmes d'analyse en $O(n^3)$.

Les TIG se distinguent néanmoins des grammaires hors contexte, en ce qu'elles lexicalisent fortement (selon la définition de [Schabes, 1990]) ces dernières. Selon [Schabes, 1990], un formalisme $L$ lexicalise fortement un formalisme $F$ si toutes les structures élémentaires de $L$ (règles de réécriture ou arbres élémentaires) contiennent un élément lexical, appelé *ancre lexicale*. De plus, pour toute grammaire écrite dans le formalisme $F$, il existe une version lexicalisée de cette dernière, écrite dans le formalisme $L$, telle que les deux grammaires permettent de reconnaître le même langage, et d'associer aux mots du langages les mêmes arbres de dérivation.

Une TIG est définie, à l'instar des TAG, comme un quintuplet $\langle \Sigma, N, I, A, S \rangle$, où $\Sigma$ est un ensemble de symboles terminaux, $N$ un ensemble de symboles non terminaux, $I$ un ensemble fini d'arbres initiaux, $A$ un ensemble fini d'arbres auxiliaires, et $S$ un élément de $N$.

La racine, ainsi que les nœuds internes des arbres auxiliaires d'une TIG (les nœuds qui ne sont ni racine ni feuille), tout comme ceux d'une TAG, sont étiquetés par des symboles non terminaux. Les feuilles sont étiquetées par des symboles terminaux, des symboles non terminaux ou la chaîne vide ($\varepsilon$). Les feuilles étiquetées par un non terminal sont des nœuds de substitution, à l'exception d'un seul, qui constitue le nœud pied d'un arbre auxiliaire. Le nœud pied est étiqueté par le même symbole que la racine.

Les arbres auxiliaires tels que toutes les feuilles non vides se trouvent à la *gauche* du nœud pied, sont appelés arbres auxiliaires gauches (voir l'arbre auxiliaire de la partie supérieure de la figure 15). De la même manière, les arbres auxiliaires tels que toutes les feuilles non vides se trouvent à la *droite* du nœud pied, sont appelés arbres auxiliaires droits(voir l'arbre auxiliaire de la partie inférieure de la figure 15). Les autres arbres auxiliaires sont appelés arbres auxiliaires *enveloppants*. Un arbre auxiliaire dont toutes les feuilles sont étiquetées par la chaîne vide, à l'exception du nœud pied, est dit arbre auxiliaire *vide*.

La principale distinction entre les TIG et les TAG est que les premières n'autorisent pas les arbres élémentaires auxiliaires enveloppants ni les arbres élémentaires auxiliaires vides. Les arbres élémentaires auxiliaires ne peuvent par conséquent qu'être des arbres auxiliaires gauches ou droits. C'est cette restriction qui limite la puissance générative des TIG : les langages contextuels nécessitent la présence d'arbres élémentaires auxiliaires enveloppants.

Les opérations d'adjonction et de substitution reprennent la définition des TAG. La distinction effectuée entre arbres auxiliaires gauches et droits permet de distinguer deux opérations d'adjonction, l'adjonction gauche, qui consiste à adjoindre un arbre auxiliaire gauche et l'adjonction droite. Les deux opérations sont représentées schématiquement dans la figure 15.

## 6.2 Transformation d'une TIG en une GDG

La transformation d'une TIG $G$ en une GDG $G'$ sous forme canonique consiste à construire un automate lexicalisé pour chaque arbre élémentaire de $G$. La

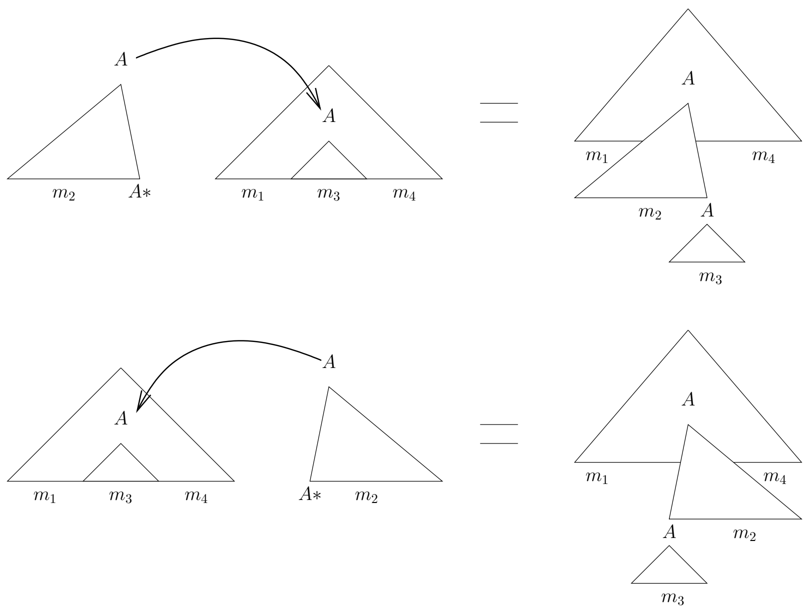



FIG. 15 – Adjonction gauche et adjonction droite

construction de l'automate $A$, correspondant à l'arbre élémentaire $T$, est réalisée par un parcours de ce dernier. Pour chaque nœud parcouru de $T$, toutes les opérations de substitution ou d'adjonction qu'il est possible d'effectuer à ce nœud sont envisagées. Pour chacune d'entre elles, une transition est construite dans $A$. Ces transitions constituent un site de l'automate. $A$ représente ainsi toutes les opérations d'attachement potentielles pouvant être réalisées sur $T$. La GDG produite distingue deux types de dépendances : les actants, étiquetés par le symbole `A`, et les circonstants étiquetés `C`. Le sens de parcours de $T$ détermine l'ordre dans lequel les transitions apparaîtront dans l'automate. Ce dernier décrivant les dépendants d'un élément lexical de gauche à droite, le parcours de l'arbre doit s'effectuer de manière à respecter cet ordre. Pour cela, les arbres sont parcourus de gauche à droite, en profondeur d'abord. Lors du parcours d'un arbre, chaque nœud est visité deux fois : une fois à la descente, et une fois à la montée, comme l'illustre la partie gauche de la figure 16. A la descente, on dira que l'on se trouve à la gauche du nœud visité et à la montée, que l'on se trouve à sa droite. Cette information est représentée par une variable appelée $côté$ qui peut avoir pour valeur *gauche* ou *droite*. On définit la fonction $Suivant(n, côté)$ où $n$ est un nœud et $côté$ indique si l'on se trouve à gauche ou à droite du nœud. Cette fonction renvoie le nœud suivant dans le parcours et modifie la valeur de la variable $côté$.

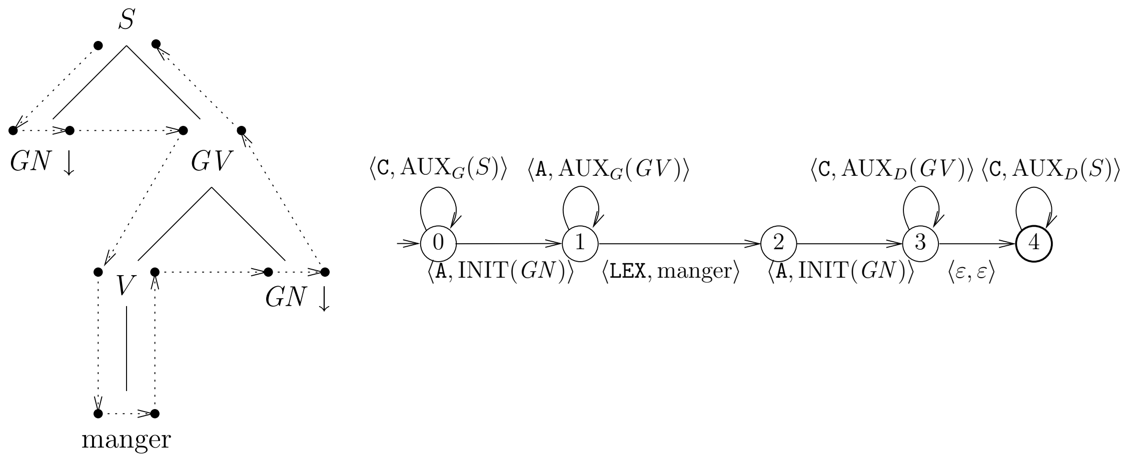



FIG. 16 – Transformation d'un arbre élémentaire TIG en un automate GDG

Soit une TIG $G = \langle \Sigma, N, I, A, S \rangle$, on considère que tous les arbres élémentaires de $G$ possèdent un nom qui permet de les identifier de façon non ambiguë. On représente par $\mathrm{AUX}_G(X)$ (respectivement $\mathrm{AUX}_D(X)$) l'ensemble de tous les noms d'arbres auxiliaires gauches (respectivement droits) de $G$ ayant le non terminal $X$ pour racine, et par $\mathrm{INIT}(X)$ l'ensemble de tous les arbres initiaux de $G$ ayant le non terminal $X$ pour racine.

L'ordre de parcours d'un arbre élémentaire $T$ ayant été déterminé, l'algorithme de transformation de $T$ en $A$ est entièrement spécifié par les opérations effectuées sur l'automate, pour chaque nœud $N$ de $T$ parcouru.

– si $N$ est un nœud interne de catégorie $c$ ($c$ n'est pas un pré-terminal) et $côté = gauche$, alors un site modifieur gauche est créé. Ce site est constitué d'autant de transitions bouclant sur l'état courant qu'il existe d'arbres auxiliaires gauches ayant $c$ pour racine.

– si $N$ est un nœud interne de catégorie $c$ ($c$ n'est pas un pré-terminal), et $côté = droit$, alors un site modifieur droit est créé. Ce site est constitué d'autant de transitions bouclant sur l'état courant qu'il existe d'arbres auxiliaires droits ayant $c$ pour racine.

– si $N$ est un nœud de substitution de catégorie $c$ alors un site actanciel est créé. Il est composé d'autant de transitions qu'il existe d'arbres initiaux ayant $c$ pour racine.

– si $N$ est une ancre $m$, alors un site lexical est créé. Celui-ci est composé d'une transition lexicale.

L'algorithme complet est décrit dans la figure 17.

**Entrée :**
une TIG $G$
un arbre élémentaire $T$ de $G$

**Sortie :**
un automate lexicalisé $A = \langle \mathcal{Q}, e_o, \mathcal{F}, \mathcal{C}, \Sigma, \delta, \mathcal{Q}_a \rangle$ au format canonique

**Variables :**
$e_c$ est l'état courant de l'automate en construction
NC est le nœud courant de $T$

**Initialisation :**
NC $\leftarrow$ racine($T$)
$côté \leftarrow$ gauche
créer l'automate $A$
$e_c \leftarrow e_0(A)$

`tant que` NC $\neq \perp$
  $c \leftarrow$ l'étiquette de NC
  `si` NC est un nœud interne `et` $côté = gauche$ `alors`
    `pour tout` $e \in \text{AUX}_G(c)$ `faire`
      $\delta \leftarrow \delta \cup \langle e_c, \langle \mathtt{C}, e \rangle, e_c \rangle$
  `sinon si` NC est un nœud interne `et` $côté = droite$ `alors`
    `pour tout` $e \in \text{AUX}_D(c)$ `faire`
      $\delta \leftarrow \delta \cup \langle e_c, \langle \mathtt{C}, e \rangle, e_c \rangle$
  `sinon si` NC est un nœud de substitution `et` $côté = gauche$ `alors`
    créer un nouvel état $e_n$
    `pour tout` $e \in \text{INIT}(c)$ `faire`
      $\delta \leftarrow \delta \cup \langle e_c, \langle \mathtt{A}, e \rangle, e_n \rangle$
    $e_c \leftarrow e_n$
  `sinon si` NC est un nœud terminal `et` $côté = gauche$ `alors`
    créer un nouvel état $e_n$
    $\delta \leftarrow \delta \cup \langle e_c, \langle \mathtt{LEX}, c \rangle, e_n \rangle$
    $e_c \leftarrow e_n$
  créer un nouvel état $e_n$
  créer la transition $\langle e_c, \langle \varepsilon, \varepsilon \rangle, e_n \rangle$
  $e_c \leftarrow e_n$
  NC $\leftarrow$ $Suivant$(NC, $côté$)
$Q_a \leftarrow \{e_c\}$

FIG. 17 – Construction d'un arbre automate lexicalisé à partir d'un arbre élémentaire TIG

## Deuxième partie

# Traitements

Les trois chapitres qui constituent cette partie ont pour objectif d'introduire les deux algorithmes principaux de ce travail, l'algorithme d'analyse syntaxique et l'algorithme de recherche de l'analyse la plus probable d'une phrase.

Le chapitre 7 va introduire la notion de forêt de dépendances partagées, notée FDP, qui représente, sous une forme compacte, un ensemble d'arbres de dépendances. Cet objet est fondamental pour les deux modules de traitements car il en constitue l'interface : à l'issue de l'analyse syntaxique une FDP est construite, qui constitue l'entrée du module de recherche de l'analyse la plus probable. Hormis son intérêt pratique dans le cadre de notre système, la notion de partage dans une forêt d'arbres de dépendances présente un intérêt plus général. En effet, peu d'attention a été apportée à ce problème qui nous semble pourtant fondamental pour le traitement automatique des grammaires de dépendances. C'est la raison pour laquelle nous allons commencer, dans le chapitre 7, par définir les principes de partage de structures communes à plusieurs arbres de dépendances, indépendamment du cadre particulier des GDG, pour montrer ensuite comment nous l'avons mise en œuvre au sein des FDP.

Le chapitre 8 décrit l'algorithme d'analyse syntaxique. Ce dernier prend en entrée une phrase étiquetée grammaticalement et une grammaire pour produire une FDP contenant tous les arbres syntaxiques de la phrase analysée. L'algorithme est d'une conception assez classique et repose, comme la plupart des algorithmes d'analyses syntaxiques pour la langue naturelle, sur les principes de la programmation dynamique ([Bellman, 1957]). La partie la plus originale de l'algorithme provient de sa capacité à prendre en entrée de multiples solutions d'un étiqueteur grammatical représenté sous la forme d'un treillis de catégories.

Le chapitre 9 clôt cette partie par la présentation de l'algorithme de recherche des analyses les plus probables d'une phrase. L'algorithme prend en entrée une FDP et produit des arbres de dépendances. Sa conception est, comme pour l'analyseur syntaxique, assez conventionnelle. Il est assez proche de l'algorithme de Viterbi [Viterbi, 1967]. Il s'en distingue principalement par le fait que son espace de recherche à été préalablement réduit aux seuls arbres que la grammaire associe à la phrase, représentés sous la forme d'une FDP.

# 7 Représentation compacte de l'ambiguïté

L'analyse syntaxique de certaines phrases peut mener à la construction d'un nombre exponentiel d'arbres, comme l'ont montré [Church & Patil, 1982]. Il ne s'agit pas d'un problème uniquement théorique, n'apparaissant que pour des phrases « artificielles ». Nous verrons, en effet, dans la partie III, que certaines phrases extraites de corpus vont effectivement mener à la construction de millions d'analyses différentes. L'immense majorité des structures construites est bien entendu incorrecte. Mais, étant donné la nature partielle des connaissances linguistiques prises en compte lors de l'analyse syntaxique, la prolifération des structures produites est inévitable.

La représentation d'un nombre potentiellement exponentiel d'analyses d'une phrase au sein d'une structure plus économique (généralement cubique par rapport à la longueur de la phrase), est un problème majeur du TAL. En effet, la majorité des applications de TAL nécessitant de représenter explicitement la structure syntaxique des phrases, telles que la traduction automatique ou la compréhension de texte, comprend une étape d'analyse syntaxique autonome, qui produit des structures syntaxiques. Celles-ci seront l'objet d'autres traitements dans des étapes ultérieures de l'application. Le nombre d'analyses produites par l'analyseur syntaxique pouvant être exponentiel, il ne peut être question de les traiter une par une. La solution à ce problème réside dans le fait que les différentes analyses d'une phrase possèdent en général un certain nombre de sous-structures identiques. L'idée est alors de tirer parti de cette caractéristique pour représenter l'ensemble des arbres sous une forme compacte, généralement appelée *forêt partagée*. Le principe de cette dernière est de ne représenter qu'une fois des sous-structures communes à plusieurs arbres.

L'étude des forêts partagées dans le cadre des grammaires hors-contexte (ou des grammaires d'unifications reposant sur ces dernières) a fait l'objet d'un grand nombre de travaux, remontant aux moins aux travaux de William Woods sur les réseaux de transition récursifs, dans les années soixante-dix. On ne peut donner ici un aperçu complet du recours à de tels objets dans différents domaines du TAL ou de la linguistique formelle. On trouvera, par exemple, dans [Maxwell & Kaplan, 1993], l'utilisation de forêts partagées dans le cadre des grammaires fonctionnelles lexicales (LFG) et, dans [C.Emele & Dorna, 1998], leur utilisation en traduction automatique. [Billot & Lang, 1989] proposent, quant à eux, une étude formelle de la structure et de la représentation des forêts d'arbres partagés pour grammaires hors-contexte. Ces auteurs proposent en particulier deux modes de représentation des forêts : une représentation, assez classique, sous forme de graphe, et une représentation sous forme de grammaire hors-contexte.

Le problème de représentation de forêts partagées pour structures de dépendances n'a pas été étudié, à notre connaissance, en dehors de [Nasr, 2003]. Le travail décrit ici repose sur ce travail. A l'instar de [Billot & Lang, 1989], nous définirons deux modes de représentations pour forêts d'arbres de dépendances : une représentation sous la forme d'un type particulier de graphes, que nous avons appelé *graphes états-transitions*, notés graphes $ET$, et une représentation algébrique proche des grammaires hors-contexte de [Billot & Lang, 1989]. La première est plus intuitive, car on y retrouve (avec un peu d'effort) la structure des arbres qu'elle représente. En revanche, la seconde est plus facile à manipuler pour des traitements formels, en particulier pour l'analyseur syntaxique décrit dans le chapitre 8, et pour la recherche des arbres les plus probables de la forêt, décrite dans le chapitre 9.

Nous commencerons, dans la section 7.1 par exposer les principes de la factorisation. Nous verrons ensuite, en 7.2, les problèmes spécifiques que posent la factorisation pour les arbres de dépendances, avant de décrire, en 7.3, les forêts de dépendances partagées et leurs deux modes de représentation. Nous terminerons le chapitre en montrant, en 7.4, comment les arbres peuvent être extraits de la forêt partagée.

*Une première version des* FDP *a été publiée dans [Nasr, 2003]. Elle reposait sur la représentation d'un arbre de dépendances sous la forme d'un ensemble de dépendances. La version des* FDP *présentée ici est assez éloignée des travaux de [Nasr, 2003]. La notion d'ensemble de dépendances a en particulier été abandonnée, au profit de la notion de séquence de transitions, qui constitue la représentation linéaire des arbres de dépendances. De plus, la nouvelle présentation permet de mieux voir les existants avec les travaux de factorisation des grammaires syntagmatiques, en définissant la notion d'arbre de dépendances binaires.*

## 7.1 Principe

La factorisation d'un ensemble d'arbres sous la forme d'une forêt partagée est réalisée par la mise en commun des parties identiques de ces derniers. Pour que deux arbres puissent partager une partie qu'ils ont en commun, il faut s'être donné les moyens de l'identifier, de la circonscrire au sein des deux arbres. En théorie, les parties communes peuvent prendre n'importe quelle forme, mais l'identification d'une partie quelconque d'un arbre nécessite d'enrichir la structure de cet arbre en lui superposant un découpage en parties. Il existe cependant des parties « naturelles » d'un arbre : celles que l'on peut peut identifier sans ajouter de structure. Ces parties sont les sous-arbres d'un arbre : les arbres ayant pour racine un nœud de l'arbre initial. Une manière économique - car ne nécessitant pas d'enrichir la structure - de factoriser un ensemble d'arbres consiste à se reposer sur ce découpage naturel et gratuit.

Etant donné deux arbres $T_1$ et $T_2$, deux situations peuvent se présenter pour la factorisation, selon que les deux arbres possèdent un sous-arbre commun ou, au contraire, se distinguent par un sous-arbre. Ces deux situations sont illustrées schématiquement dans la figure 18. Dans la partie supérieure de la figure, $T_1$ et $T_2$ ont en commun le sous-arbre $T'$. Dans la représentation factorisée, ce dernier n'est représentée qu'une fois, et les deux arbres $T_1$ et $T_2$ pointent sur une occurrence unique de $T'$. On parlera dans ce cas de *partage de sous-arbre*. Dans la partie inférieure, la différence entre les deux arbres $T_1$ et $T_2$ se limite aux deux sous-arbres $T'$ et $T''$. Dans ce cas, la partie commune à $T_1$ et $T_2$, appelée ici $C$, n'est représentée qu'une fois, et pointe sur une structure qui correspond à l'ensemble $\{T', T''\}$, indiquant que le sous-arbre $T'$ ou le sous-arbre $T''$ peuvent s'insérer dans le contexte $C$. On parlera dans ce cas de *partage de contexte*.

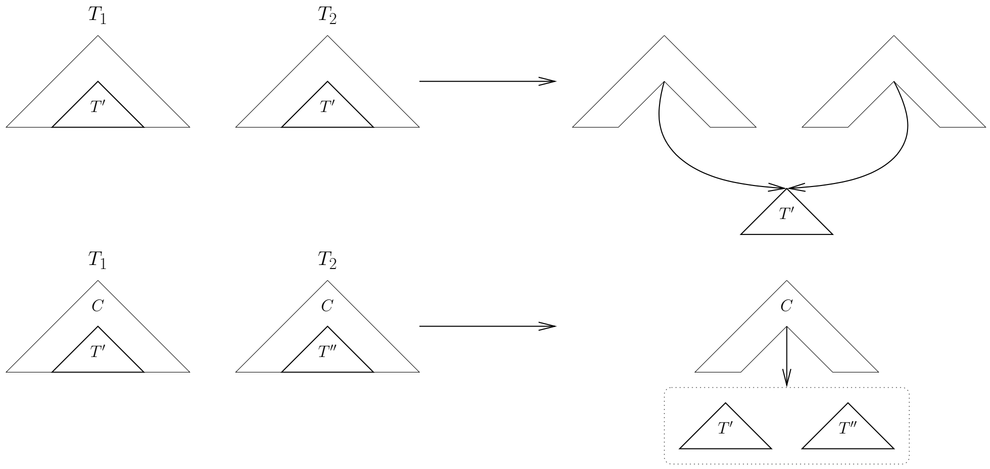



FIG. 18 – Factorisation d'arbres par partage de sous-arbre, et par partage de contexte

Ces deux principes peuvent être illustrés par l'exemple de la figure 19. Celle-ci représente les cinq arbres de dépendances $T_0 \dots T_4$ associés à la chaîne $n_0\ v_1\ n_2\ p_3\ n_4\ p_5\ n_6$, où $n$, $v$ et $p$ dénotent respectivement les catégories morpho-syntaxiques *nom*, *verbe* et *préposition*, tandis que l'indice représente la position dans la phrase. On pourra remarquer que les deux arbres $T_0$ et $T_1$ ont en commun le sous-arbre de racine $p_3$, et que les contextes du sous-arbre de racine $n_2$ dans $T_0$ et $T_3$ sont identiques. Les deux principes de factorisation peuvent être mis en œuvre dans ces deux cas.

On peut remarquer aussi que, dans certains cas, le partage de structures communes est impossible à l'aide des deux principes énoncés ci-dessus. La dépendance *sujet*, qui se retrouve à l'identique dans les cinq arbres, en offre un exemple. Bien que présente dans tous les arbres, cette dépendance ne constitue ni un sous-arbre, ni un contexte. Par conséquent, elle ne peut être mise en facteur, ce qui empêche une factorisation plus poussée des arbres. Nous allons proposer une solution à ce problème dans la section suivante, en modifiant la structure des arbres de dépendances, afin d'en faire émerger des sous-arbres plus fins, et de faire apparaître ainsi, sous la forme de sous-arbres, des parties qui n'en étaient pas dans la représentation initiale. Ce mode de représentation des arbres de dépendances, que nous appellerons *arbre de dépendances binaire*, augmentera les possibilités de factorisation.

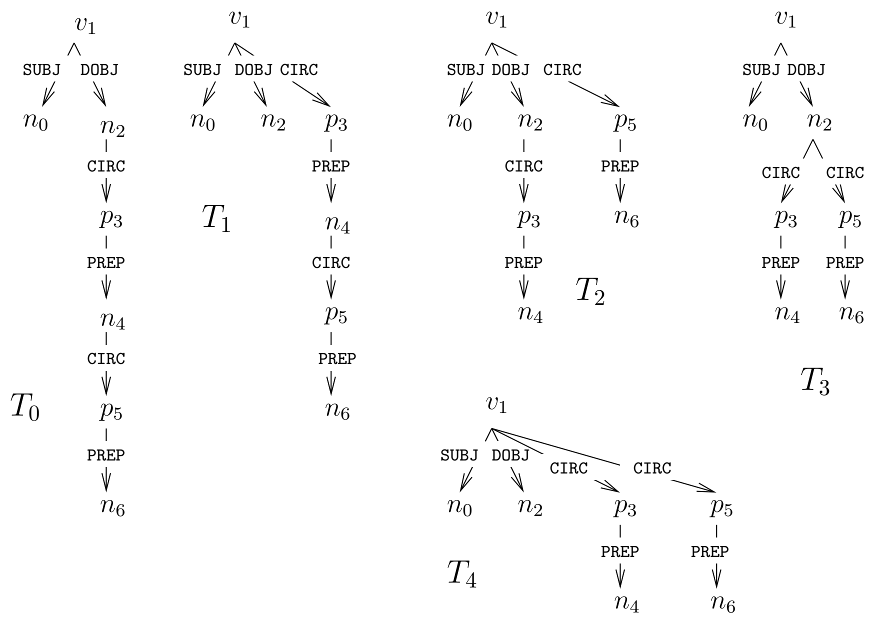



FIG. 19 – Forêt de cinq arbres de dépendances

## 7.2 Arbres de dépendances binaires

L'idée des arbres de dépendances binaires peut être rapprochée de celle de la décomposition des arbres syntagmatiques sous forme normale de Chomsky ([Chomsky, 1963]), qui est à l'origine de la factorisation mise en œuvre dans l'algorithme CYK.

Soit la règle $X \rightarrow a\ b\ c\ d$, et sa décomposition sous la forme de sept règles en forme normale de Chomsky :

$$\begin{array}{lll} X_0 \rightarrow A\ B & \qquad & A \rightarrow a \\ X_1 \rightarrow X_0\ C & & B \rightarrow b \\ X_2 \rightarrow X_1\ D & & C \rightarrow c \\ & & D \rightarrow d \end{array}$$

Les deux « grammaires » permettent d'associer à la chaîne $a\ b\ c\ d$ les deux arbres de la figure 20. La structure de gauche correspond à la structure initiale, alors que celle de droite correspond à la structure obtenue après décomposition de la grammaire sous forme normale de Chomsky. Cette structure sera appelée la structure binaire.

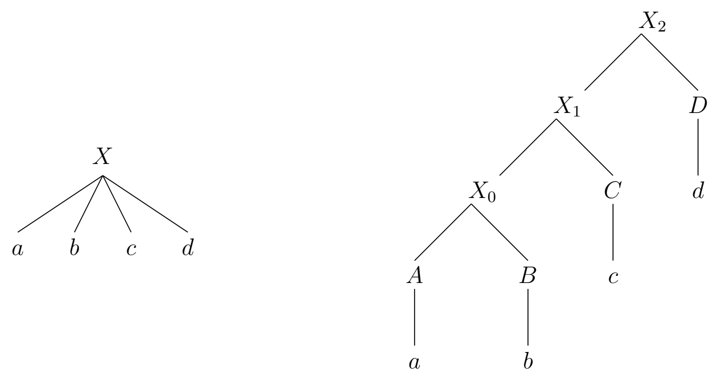



FIG. 20 – Forme normale de Chomsky

L'intérêt de la structure binaire, vis-à-vis de la factorisation, est qu'elle permet d'identifier les sous-arbres de racines $X_0$ et $X_1$, qui correspondent à des sous-chaînes de la chaîne $a\ b\ c\ d$. De tels sous-arbres pourront éventuellement être utilisés lors de la factorisation de plusieurs analyses de la chaîne $a\ b\ c\ d$. Ces sous-arbres n'ont pas d'équivalent dans la structure initiale.

Bien que les deux arbres de la figure 20 ne soient pas identiques, on peut noter toutefois qu'ils constituent tous deux des arbres syntagmatiques. Il n'a donc pas été nécessaire de changer de mode de représentation des arbres mais uniquement la structure de ces derniers.

L'idée d'une décomposition binaire d'un arbre, telle que réalisée grâce à la forme normale de Chomsky, peut être mise en œuvre pour les structures de dépendances. Un exemple de décomposition binaire d'un arbre de dépendances est présenté dans la figure 21. Une telle représentation sera appelée *arbre de dépendances binaire*, bien qu'il ne s'agisse pas à proprement parler d'un arbre de dépendances, dans la mesure où il introduit un certain nombre de symboles non terminaux ($A_2, \ldots, A_6, B_2, C_2, D_2$ et $E_2$) qui ne correspondent pas à des mots et dans la mesure où ses arcs ne représentent pas tous des dépendances entre mots (par exemple $A_6 \rightarrow A_5$ ou $A_5 \rightarrow A_4$).

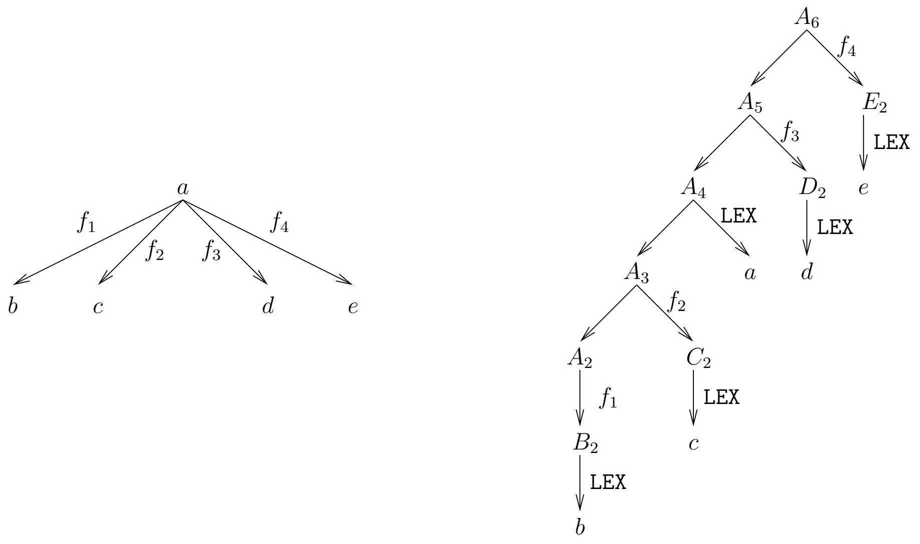



FIG. 21 – Arbre de dépendances binaire

La décomposition d'un arbre de dépendances en arbre de dépendances binaire consiste à décomposer tout nœud $X$ de l'arbre de dépendances en $n+1$ nœuds, notés $X_1 \ldots X_{n+1}$, où $n$ est le nombre de fils de $X$. Tout nœud $X_i$ de cet ensemble a pour fils droit un dépendant de $X$, et pour fils gauche le nœud $X_{i-1}$. Le nœud surnuméraire a pour fils droit un nœud étiqueté par l'élément lexical de $X$. Les $n+1$ nœuds sont donc liés entre eux par des arcs non étiquetés, représentant le fait qu'ils proviennent tous d'un même nœud dans l'arbre de dépendances.

Illustrons cela sur la figure 21 : le nœud étiqueté $a$, dans l'arbre de dépendances, est décomposé en cinq nœuds $A_2, \ldots, A_6$, reliés entre eux par des arcs gauches ($\swarrow$). Chaque arc droit ($\searrow$) de ces nœuds correspond à une dépendance de l'arbre de dépendances, à l'exception du nœud $A_4$ dont le fils droit est l'élément lexical $a$.

On peut ici faire le lien entre la représentation des règles de grammaire sous la forme d'automates, telle que proposée dans le chapitre 3, et les arbres de dépendances binaires. Ce lien est réalisé en identifiant tout nœud d'un arbre de dépendances binaire à un couple constitué d'un automate de la grammaire, et d'un état de ce dernier. Afin d'illustrer cette idée, nous avons représenté, dans la figure 22, une grammaire GDG permettant de générer l'arbre de dépendances situé dans la partie gauche de la figure 21. Les nœuds $A_2, \ldots, A_6, B_2, C_2, D_2, E_2$ de l'arbre de dépendances binaire de la même figure correspondent chacun à un état d'un automate de la grammaire de la figure 22. La séquence d'états $A_2$, ..., $A_6$ correspond à un chemin dans l'automate $A$, le chemin ayant permis de produire l'arbre de dépendances. Un arbre de dépendances binaire peut, par conséquent, être vu comme un arbre de dérivation dans le cadre des GDG, dans lequel chaque étape de la dérivation - chaque franchissement d'une transition d'un automate - est représentée. En ce sens, un arbre de dépendances binaire est très proche de la représentation linéaire d'un arbre de dépendances, telle que définie dans le chapitre 3.

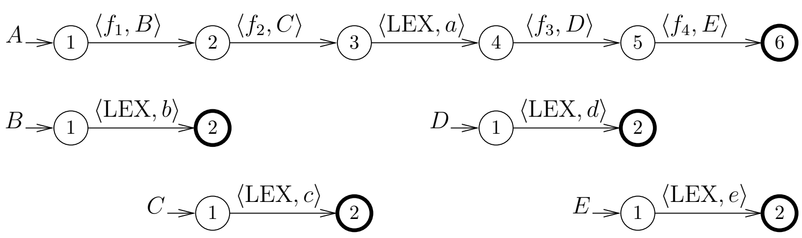



FIG. 22 – Exemple de grammaire GDG

On pourra remarquer que les états initiaux des automates de la grammaire ($A_1, B_1, C_1, D_1$ et $E_1$) ne sont pas représentés dans l'arbre de la figure 21. Ils devraient apparaître comme fils gauche des nœuds $A_2, B_2, C_2, D_2$ et $E_2$. La raison est que leur présence n'est pas nécessaire, puisque tout chemin dans un automate commence obligatoirement par l'état initial de ce dernier. Nous avons par conséquent décidé de ne pas les représenter, pour simplifier la structure des arbres. D'autre part, Les chemins dans les automates sont représentés à l'envers. La raison de cette bizarrerie apparaîtra lors de la description de l'algorithme d'analyse (chapitre 8) qui permet de créer ce type d'arbre.

Les avantages des arbres de dépendances binaires sont exactement les mêmes que ceux décrits pour les arbres syntagmatiques : la possibilité d'identifier dans l'arbre binaire des sous-arbres plus fins que les sous-arbres de la structure initiale, et celle d'augmenter, par conséquent, les possibilités de mise en facteur.

Les arbres de dépendances binaires vont permettre de représenter des forêts d'arbres de dépendances, telle que celle de la figure 19, sous une forme compacte que nous appellerons, *forêt de dépendances partagée*, décrite dans la section suivante.

## 7.3 Forêts de dépendances partagées

Le principe des forêts de dépendances partagées (notées dorénavant FDP) consiste à représenter les arbres de la forêt sous forme binaire, puis à les factoriser grâce aux deux principes de partage de sous-arbres et de partage de contextes que nous avons évoqués en 7.1. Dans les faits, une FDP n'est pas construite de cette manière, par factorisation d'une forêt d'arbres existante. Elle est produite directement par l'analyseur syntaxique, sans passer par une forêt d'arbres de dépendances.

Etant donné une GDG $G$ et une phrase $S$, une FDP générée par $G$ pour $S$, notée $\text{FDP}_G(S)$, est la représentation compacte de tous les arbres générés par $G$ correspondant à $S$ : l'ensemble $T_G(S)$. Cet ensemble est appelé l'*extension* de la FDP. Nous allons présenter, dans les deux sections suivantes, deux modes de représentation des FDP. Nous commencerons par décrire, en 7.3.1, une représentation des FDP sous la forme de graphes d'un type particulier, que nous avons appelé graphes $ET$, puis, en 7.3.2, une représentation algébrique.

### 7.3.1 Graphes $ET$

Un graphe $ET$ est un graphe orienté hétérogène, ayant deux types de sommets : des *sommets états* (notés sommets-$E$), correspondant à des états des automates de la grammaire, et des *sommets transitions* (notés sommets-$T$), correspondant à des transitions des automates de la grammaire. Nous avons disposé, côte à côte, dans la figure 23, l'arbre binaire de la figure 21 et sa représentation sous la forme d'un graphe $ET$. Les sommets-$E$ sont représentés graphiquement par des cercles, et les sommets-$T$, par des rectangles[16].

[16]On remarquera qu'à l'instar des arbres de dépendances binaires, les graphes $ET$ n'associent pas de sommets-$E$ aux états initiaux des automates.

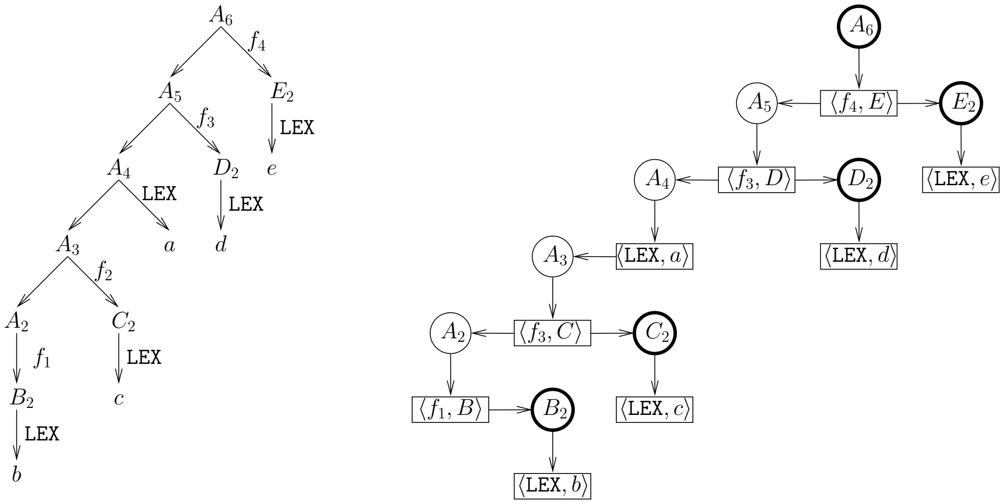



FIG. 23 – Un arbre de dépendances binaire (à gauche) et sa représentation sous la forme d'un graphe $ET$ (à droite)

Chaque nœud interne de l'arbre de dépendances binaire correspond à un sommet-$E$ du graphe $ET$. Les feuilles de l'arbre sont représentées par des sommets-$T$ étiquetés $\langle \texttt{LEX}, m\rangle$. Lorsqu'un sommet-$E$ correspond à un état d'acceptation d'un automate, il est représenté en gras, comme dans la représentation graphique des automates. Un sommet-$T$ permet de relier entre eux trois sommets-$E$. Le sommet-$T$ étiqueté $\langle f_4, E\rangle$, par exemple, correspondant à la transition $\langle A, \langle 5, \langle f_4, E\rangle, 6\rangle\rangle$ (la transition de l'automate $A$ reliant l'état 5 à l'état 6), permet de relier les trois sommets-$E$ $A_5, A_6$ et $E_2$. Le premier correspond à l'état origine de la transition, le second à son état destination, et le troisième à un état d'acceptation de l'automate $E$ : l'automate correspondant à l'étiquette catégorielle de la transition. La correspondance entre les transitions des automate, les arbres de dépendances binaires et les graphes-$ET$ apparaît explicitement dans la figure 24. Nous y avons représenté une transition de l'automate $A$ (à gauche), la partie de l'arbre de dépendances binaire de la figure 21, correspondant à cette transition (au centre), et la partie correspondante du graphe-$ET$ de la figure 23 (à droite).

Etant donné un sommet-$T$ correspondant à une transition $t$, le sommet-$E$ correspondant à l'état origine de $t$ est atteignable à partir du sommet-$T$, en suivant l'arête gauche de ce dernier. Le sommet-$E$ correspondant à l'état d'acceptation est atteignable à partir du sommet-$T$, en suivant son arête droite. Les arêtes droites correspondent donc aux branches droites des arbres de dépendance binaires et les arêtes gauches, aux branches gauches.

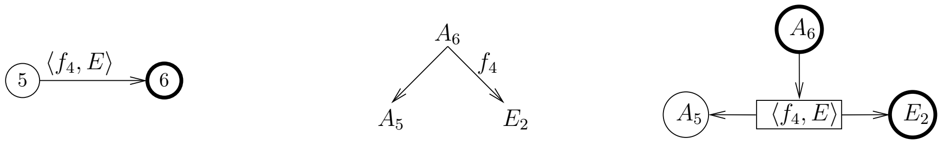



FIG. 24 – Une transition d'automate, une partie d'arbre de dépendances binaire et une partie de graphe-$ET$

Les sommets-$T$ correspondant à des transitions lexicales n'ont pas d'arête droite, traduisant le fait qu'elles ne sont pas étiquetées par le nom d'un automate, mais par un élément lexical. Les sommets-$T$ correspondant à des transitions initiales d'un automate n'ont pas d'arête gauche, conséquence du fait que les états initiaux des automates ne sont pas représentés dans le graphe $ET$.

La mise en œuvre des principes de partage de sous-arbre et de partage de contexte dans un graphe $ET$ est illustrée dans la figure 25. La partie gauche de la figure correspond à un partage de sous-arbre, et la partie droite à un partage de contexte. Chaque partie présente, de haut en bas, une grammaire, deux arbres de dépendances générés par la grammaire, les arbres de dépendances binaires qui leur sont associés, et le graphe $ET$ correspondant aux deux arbres. Le partage de sous-arbre se traduit graphiquement par le fait que les arêtes droites de plusieurs sommets-$T$ aboutissent à un même sommet-$E$. Dans la partie gauche de la figure 25, il s'agit du sommet $C_2$, racine du sous-arbre commun aux deux arbres binaires, qui est atteignable à partir des deux sommets-$T$ $\langle f_1, C\rangle$ et $\langle f_2, C\rangle$. Le partage de contexte se traduit par le fait que deux sommets-$T$ sont accessibles à partir d'un même sommet-$E$. Dans la partie droite de la figure 25, il s'agit des deux sommets-$T$ $\langle f_1, B\rangle$ et $\langle f_2, C\rangle$, atteignables depuis le sommet-$E$ $A_2$[17].

Les grammaires de la figure 25 permettent de mieux comprendre dans quels cas le partage de sous-arbre et le partage de contexte peuvent avoir lieu. Le partage de sous-arbre peut intervenir lorsque deux automates distincts possèdent une transition ayant le même symbole catégoriel, par exemple les deux transitions $\langle f_1, C\rangle$ et $\langle f_2, C\rangle$ de la figure 25. Dans ce cas, les deux contextes ne sont pas identiques, car il s'agit d'automates différents, mais les

[17]Dans la représentation graphique des graphes $ET$, les sommets-$T$ accessibles depuis un même sommet-$E$, tels que les deux sommets-$T$ $\langle f_1, B\rangle$ et $\langle f_2, C\rangle$ de la figure 25, sont liés entre eux par des arêtes verticales, au lieu d'être directement reliés au sommet-$E$ dont ils dépendent. Ce choix a été fait dans le but de simplifier la représentation graphique des graphes $ET$.

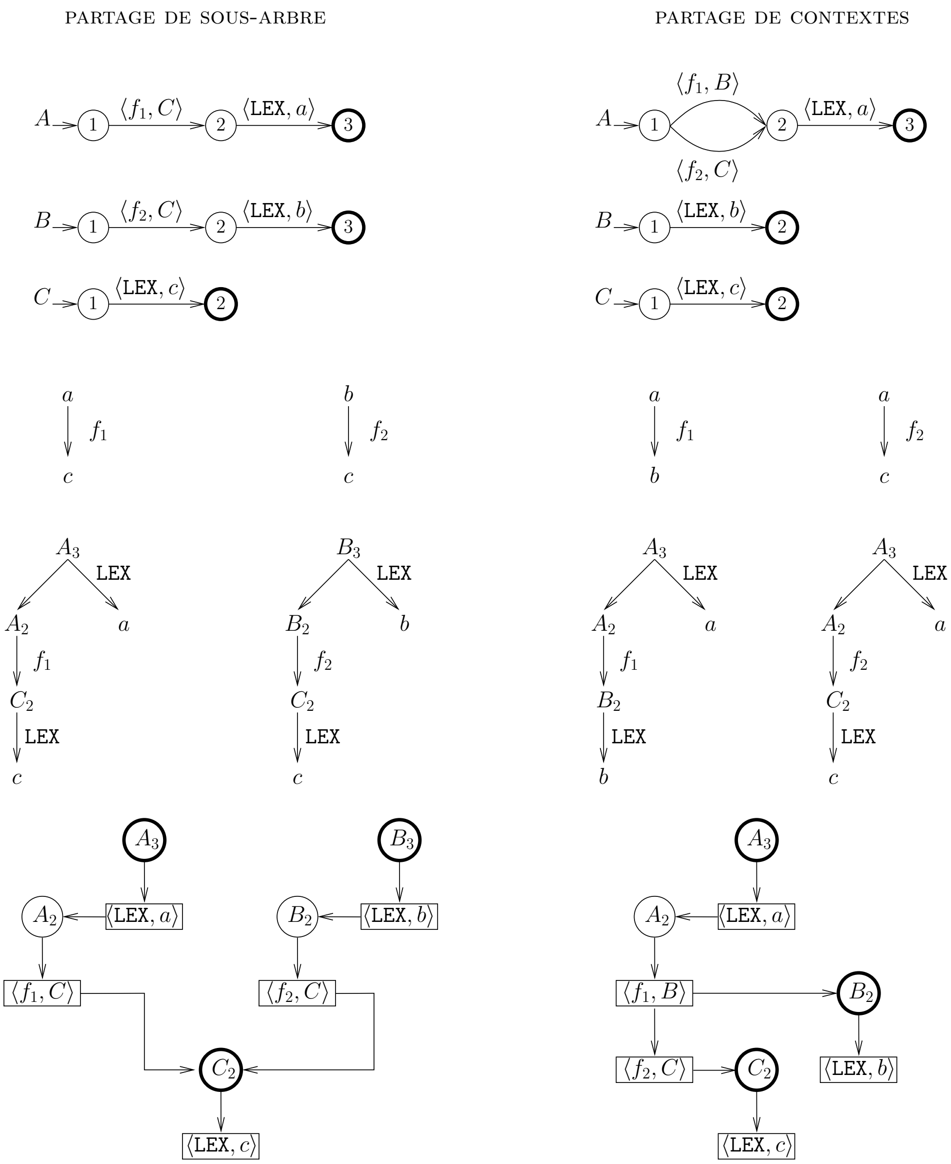



FIG. 25 – Partage de sous-arbre et partage de contexte sous la forme de graphes *ET*

sous-arbres sont communs. Le partage de contexte intervient lorsque d'un sommet émanent deux transitions ayant des symboles catégoriels différents, par exemple les deux transitions $\langle f_1, B\rangle$ et $\langle f_2, C\rangle$ émanant de l'état 1 de l'automate $A$ de la figure 25. Dans ce cas, le contexte est unique : l'automate $A$. Mais les sous-arbres sont différents : un sous-arbre correspondant à l'automate $B$ pour le premier, et à l'automate $C$ pour le second.

### 7.3.2 Représentation algébrique

Les graphes $ET$ de la section précédente ont permis de voir graphiquement la relation existant entre les arbres de dépendances binaires et les FDP. Cependant, pour décrire des traitements sur les FDP, en particulier leur construction lors de l'analyse syntaxique, nous allons recourir à une autre représentation de ces dernières, appelée *représentation algébrique*. La représentation algébrique d'une FDP distingue aussi deux types d'entités : des *sous-analyses*, qui correspondent aux sommets-$E$ d'un graphe $ET$, et des *composantes*, qui correspondent aux sommets-$T$. Une sous-analyse correspond à un ensemble d'arbres qu'une grammaire peut associer à un segment de phrase. Cet ensemble est appelé l'*extension* de la sous-analyse. L'extension d'une sous-analyse $X$ est notée $\mathcal{E}(X)$.

Une sous-analyse est représentée formellement par un quintuplet $X = \langle A, e, i, j, \mathcal{Y}\rangle$, où $A$ est un automate, $e$ est un état de $A$, appelé l'*état courant* de la sous-analyse (on retrouve là l'automate et l'état d'un sommet-$E$). $i$ et $j$ sont deux entiers délimitant le segment de phrase correspondant à la sous-analyse. Finalement, $\mathcal{Y}$ est l'ensemble des composantes de $X$ (l'ensemble des sommets-$T$ atteignables depuis un sommet-$E$). On notera $Auto(X)$, $e(X)$, $i(X)$, $j(X)$ et $\mathcal{Y}(X)$ les différents éléments de $X$. De plus, une sous-analyse est associée à un identifiant, appelé le *nom* de la sous-analyse.

Une composante $Y$, appartenant à l'ensemble $\mathcal{Y}$ des composantes d'une sous-analyse $X = \langle A, e, i, j, \mathcal{Y}\rangle$, est elle-même représentée par un triplet $\langle X_G, t, X_D\rangle$ où $X_G$ et $X_D$ sont des sous-analyses (plus précisément des noms de sous-analyses), et $t$ est une transition de l'automate $A$ menant à l'état courant $e$ de $X$. Comme nous l'avons précisé ci-dessus, une composante correspond à un sommet-$T$ et les sous-analyses $X_G$ et $X_D$ correspondent aux deux sommets-$E$ droit et gauche d'un sommet-$T$. On notera $X_G(Y)$, $t(Y)$ et $X_D(Y)$ les différents éléments de $Y$. Les composantes correspondant à des transitions initiales ne possèdent pas de sous-analyses gauches et les composantes correspondant à des transitions lexicales ne possèdent pas de sous-analyses droites. Cette absence sera matérialisée par la présence du symbole

$\perp$ à la place de la sous-analyse manquante. Une composante correspond aussi à un certain nombre d'arbres, appelé l'extension de la sous-analyse, notée $\mathcal{E}(Y)$.

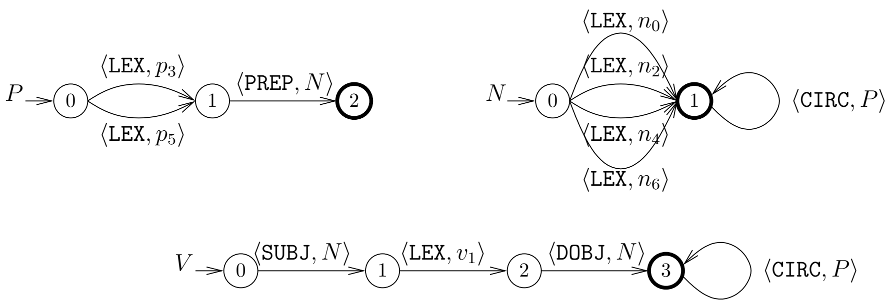
Etant donné une phrase $S$, une grammaire $G$ et une FDP $F = \text{FDP}_G(S)$ (la forêt de dépendances partagées contenant les arbres que $G$ associe à $S$), la sous-analyse $X = \langle A, e, i, j, \mathcal{Y}\rangle$ de $F$ s'interprète de la façon suivante : il existe au moins un arbre de dépendances dans $\mathcal{E}(X)$ (l'extension de $X$), qui correspond au segment $m_{i,j}$ de $S$. La racine de cet arbre est une ancre de l'automate $A$. L'état $e$ indique la progression dans la reconnaissance des dépendants de la racine. En d'autres termes, lorsque deux arbres de dépendances appartiennent à l'extension d'une même sous-analyse, ils correspondent au même segment de phrase : le segment $m_{i,j}$. De plus, leurs racines sont de même catégorie : la catégorie correspondant à l'automate $A$. Finalement, les deux racines en sont au même point dans la reconnaissance de leur dépendances, c'est-à-dire que toute nouvelle dépendance pouvant être ajoutée à la racine de l'un, peut aussi l'être à la racine de l'autre.

Lorsque l'état courant de la sous-analyse est un état d'acceptation de l'automate $A$, c'est-à-dire lorsque tous les dépendants obligatoires d'une ancre lexicale de $A$ ont été reconnus, la sous-analyse est dite *saturée*. Lorsque son état courant est l'état initial, la sous-analyse est dite *vierge* : aucun dépendant n'a encore été reconnu.

Une FDP correspond à un ensemble de sous-analyses. Plus précisément, une FDP est définie comme un couple $\langle \mathcal{X}, \mathcal{F}\rangle$, où $\mathcal{X}$ est l'ensemble des sous-analyses qui composent la FDP, et $\mathcal{F}$ est un sous-ensemble de $\mathcal{X}$, appelé l'ensemble des *sous-analyses finales*. Ce dernier est l'ensemble des sous-analyses saturées de $\mathcal{X}$ qui n'apparaissent dans aucune composante des sous-analyses de $\mathcal{X}$. Ces sous-analyses correspondent aux racines des arbres de la forêt.

Un exemple de FDP, correspondant aux cinq arbres de la figure 19, est représenté sous forme d'un graphe $ET$, et sous forme algébrique, dans la figure 27. La grammaire ayant permis de générer cette FDP, composée de 3 automates apparaît dans la figure 26. Un identifieur a été associé à chaque transition des automates de la grammaire, dans le but d'alléger la figure 27. La transition $\langle 0, \langle \texttt{LEX}, n_0\rangle, 1\rangle$, par exemple, est associée, dans la figure 26, à l'identifieur $t_{N_1}$.

La représentation algébrique des FDP est proche de la représentation des forêts d'arbres sous la forme d'une GHC, telle que proposée par [Billot & Lang, 1989]. Une sous-analyse peut en effet être vue comme un ensemble de règles hors-contexte partageant la même partie gauche, qui n'est




| transitions de N | transitions de P | transitions de V |
|---|---|---|
| $t_{N_1} = \langle 0, \langle \texttt{LEX}, n_0\rangle, 1\rangle$ | $t_{P_1} = \langle 0, \langle \texttt{LEX}, p_3\rangle, 1\rangle$ | $t_{V_1} = \langle 0, \langle \texttt{SUBJ}, N\rangle, 1\rangle$ |
| $t_{N_2} = \langle 0, \langle \texttt{LEX}, n_2\rangle, 1\rangle$ | $t_{P_2} = \langle 0, \langle \texttt{LEX}, p_5\rangle, 1\rangle$ | $t_{V_2} = \langle 1, \langle \texttt{LEX}, v_1\rangle, 2\rangle$ |
| $t_{N_3} = \langle 0, \langle \texttt{LEX}, n_4\rangle, 1\rangle$ | $t_{P_3} = \langle 0, \langle \texttt{PREP}, N\rangle, 1\rangle$ | $t_{V_3} = \langle 2, \langle \texttt{DOBJ}, N\rangle, 3\rangle$ |
| $t_{N_4} = \langle 0, \langle \texttt{LEX}, n_6\rangle, 1\rangle$ | | $t_{V_4} = \langle 3, \langle \texttt{CIRC}, P\rangle, 3\rangle$ |
| $t_{N_5} = \langle 1, \langle \texttt{CIRC}, P\rangle, 1\rangle$ | | |

FIG. 26 – Exemple de grammaire GDG pouvant générer la forêt de la figure 19

autre que le nom de la sous-analyse. Chaque composante d'une sous-analyse donne naissance à une partie droite de règle hors-contexte.

La sous-analyse $A = \langle N, 1, 1, 1, \{\langle \perp, t_{N_1}, \perp\rangle\}\rangle$, donnera naissance à la règle de réécriture :

$A \to t_{N_1}$

La sous-analyse $Q = \langle V, 3, 1, 5, \{\langle J, t_{V_3}, N\rangle, \langle M, t_{V_4}, K\rangle\}\rangle$ donnera naissance aux deux règles :

$Q \to J\ t_{V_3}\ N$

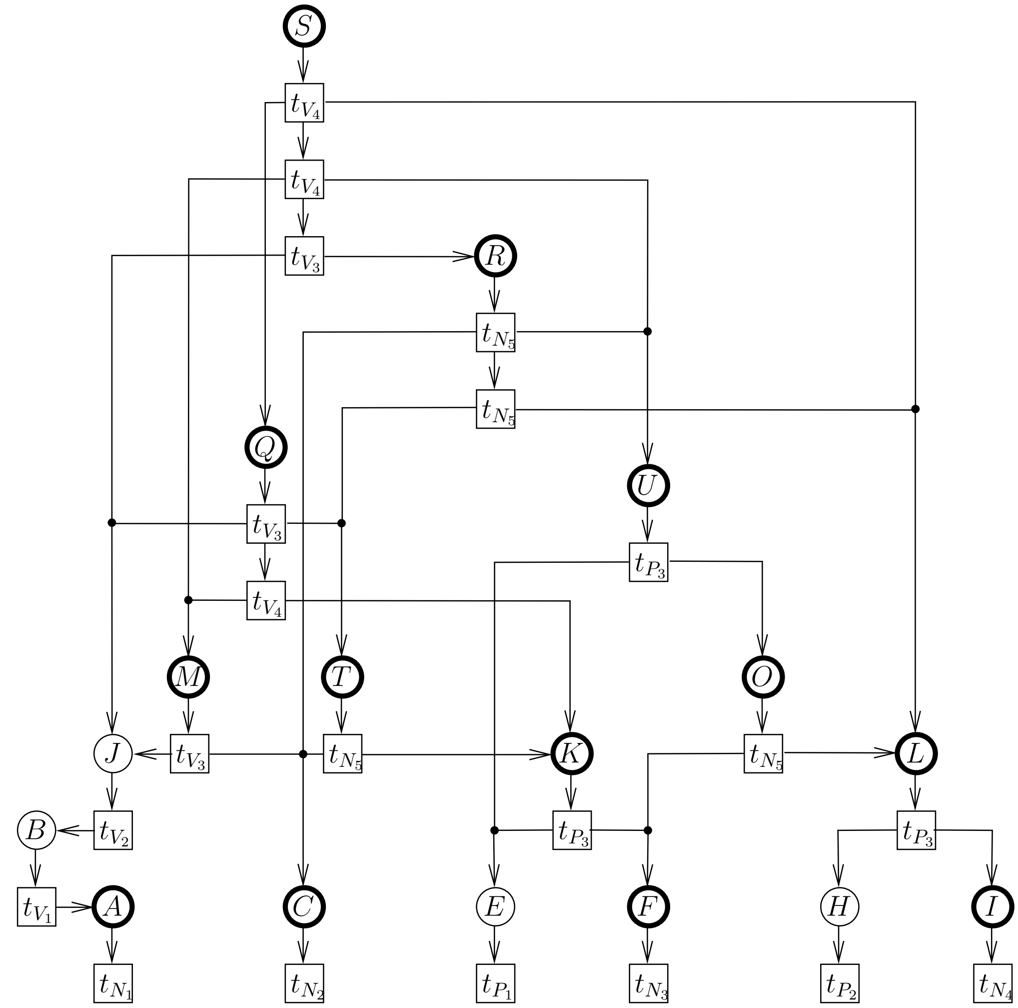
et

$Q \to M\ t_{V_4}\ K$

La FDP de la figure 27 peut alors être vue comme une GHC, dont le langage est l'ensemble des arbres de l'extension de la FDP.

## 7.4 Calcul de l'extension d'une FDP

L'ensemble des arbres contenus dans une FDP (appelé l'extension de la FDP) peut être construit à l'aide d'une procédure récursive que nous allons décrire



$\mathcal{F} = \langle \{A, B, C, E, F, H, I, J, K, L, M, T, O, U, Q, R, S\}, \{S\}\rangle$ avec

$A = \langle N, 1, 1, 1, \{\langle \perp, t_{N_1}, \perp\rangle\}\rangle$ $B = \langle V, 1, 1, 1, \{\langle \perp, t_{V_1}, A\rangle\}\rangle$
$C = \langle N, 1, 3, 3, \{\langle \perp, t_{N_2}, \perp\rangle\}\rangle$ $E = \langle P, 1, 4, 4, \{\langle \perp, t_{P_1}, \perp\rangle\}\rangle$
$F = \langle N, 1, 5, 5, \{\langle \perp, t_{N_3}, \perp\rangle\}\rangle$ $H = \langle P, 1, 6, 6, \{\langle \perp, t_{P_2}, \perp\rangle\}\rangle$
$I = \langle N, 1, 7, 7, \{\langle \perp, t_{N_4}, \perp\rangle\}\rangle$ $J = \langle V, 2, 1, 2, \{\langle B, t_{V_2}, \perp\rangle\}\rangle$
$K = \langle P, 2, 4, 5, \{\langle E, t_{P_3}, F\rangle\}\rangle$ $L = \langle P, 2, 6, 7, \{\langle H, t_{P_3}, I\rangle\}\rangle$
$M = \langle V, 3, 1, 3, \{\langle J, t_{V_3}, C\rangle\}\rangle$ $T = \langle N, 1, 3, 5, \{\langle C, t_{N_5}, K\rangle\}\rangle$
$O = \langle N, 1, 5, 7, \{\langle F, t_{N_5}, L\rangle\}\rangle$ $U = \langle P, 2, 4, 7, \{\langle E, t_{P_3}, O\rangle\}\rangle$
$Q = \langle V, 3, 1, 5, \{\langle J, t_{V_3}, T\rangle, \langle M, t_{V_4}, K\rangle\}\rangle$
$R = \langle N, 1, 3, 7, \{\langle C, t_{N_5}, U\rangle, \langle T, t_{N_5}, L\rangle\}\rangle$
$S = \langle V, 3, 1, 7, \{\langle J, t_{V_3}, R\rangle, \langle M, t_{V_4}, U\rangle, \langle Q, t_{V_4}, L\rangle\}\rangle$

FIG. 27 – Représentations d'une FDP sous la forme d'un graphe $ET$ (en haut) et sous la forme algébrique (en bas)

ici. Cette procédure peut créer un nombre exponentiel d'arbres, c'est pourquoi elle n'est pas réalisée en pratique. On verra dans le chapitre 9 que seuls les arbres les plus probables seront extraits de la FDP. Nous allons néanmoins décrire la procédure d'extraction de tous les arbres, car elle permet de mieux voir le lien existant entre la structure des FDP et la représentation linéaire des arbres de dépendances.

Nous avons vu ci-dessus qu'une sous-analyse possède une structure récursive : une sous-analyse est constituée d'un ensemble de composantes et ces dernières sont, à leur tour, constituées de deux sous-analyses et d'une transition. Cette structure récursive va être mise à profit pour définir un processus récursif de construction des arbres de l'extension d'une sous-analyse. Les arbres produits par le processus se présentent sous forme linéaire, introduite dans le chapitre 3. Rappelons que la représentation linéaire d'un arbre de dépendances $T$, étant donné une GDG $G$, est une séquence de transitions des automates de $G$. Une telle séquence correspond à un parcours de $T$. Cette représentation des arbres de dépendances sous la forme de séquences de transitions permet de construire des arbres par simple concaténation de séquences de transitions correspondant à des sous-arbres.

L'opération de concaténation est utilisée pour construire l'extension d'une composante $Y$. Cette construction consiste à concaténer les arbres de l'extension de la sous-analyse gauche de $Y$ avec la transition de $Y$ et avec les arbres de l'extension de sa sous-analyse droite. Tout sous-arbre de la sous-analyse gauche peut donc se combiner avec un sous-arbre de la sous-analyse droite pour constituer un nouveau sous-arbre. C'est la raison pour laquelle le nombre de sous-arbres créés peut croître de manière exponentielle. La construction de l'extension d'une sous-analyse repose sur les extensions des sous-analyses de ses composantes, elle consiste à en faire l'union.

Ce processus est défini formellement par les deux équations 10 et 11 où $Y$ est une composante et $X$ une sous-analyse.

$$\mathcal{E}(Y) = \mathcal{E}(X_G(Y)) \cdot t(Y) \cdot \mathcal{E}(X_D(Y)) \tag{10}$$

$$\mathcal{E}(X) = \begin{cases} \varepsilon & si\ X = \perp \\ \bigcup_{Y \in \mathcal{Y}(X)} \mathcal{E}(Y) & sinon \end{cases} \tag{11}$$

L'extension d'une FDP $F = \langle \mathcal{X}, \mathcal{F} \rangle$ est l'union des extensions des sous-analyses de $\mathcal{F}$.

$$\mathcal{E}(\langle \mathcal{X}, \mathcal{F} \rangle) = \bigcup_{X \in \mathcal{F}} \mathcal{E}(X) \tag{12}$$

Les étapes du calcul de l'extension de la FDP de la figure 27 sont représentées ci-dessous :

$$\begin{array}{rcl}
\mathcal{E}(A) & = & t_{N_1} \\
\mathcal{E}(B) & = & t_{V_1} \cdot \mathcal{E}(A) = \langle t_{V_1}, t_{N_1} \rangle \\
\mathcal{E}(C) & = & t_{N_2} \\
\mathcal{E}(E) & = & t_{P_1} \\
\mathcal{E}(F) & = & t_{N_3} \\
\mathcal{E}(H) & = & t_{P_2} \\
\mathcal{E}(I) & = & t_{N_4} \\
\mathcal{E}(J) & = & \mathcal{E}(B) \cdot t_{V_2} = \langle t_{V_1}, t_{N_1}, t_{V_2} \rangle \\
\mathcal{E}(K) & = & \mathcal{E}(E) \cdot t_{P_3} \cdot \mathcal{E}(F) = \langle t_{P_1}, t_{P_3}, t_{N_3} \rangle \\
\mathcal{E}(L) & = & \mathcal{E}(H) \cdot t_{P_3} \cdot \mathcal{E}(I) = \langle t_{P_2}, t_{P_3}, t_{N_4} \rangle \\
\mathcal{E}(M) & = & \mathcal{E}(J) \cdot t_{V_3} \cdot \mathcal{E}(C) = \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2} \rangle \\
\mathcal{E}(T) & = & \mathcal{E}(C) \cdot t_{N_5} \cdot \mathcal{E}(K) = \langle t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3} \rangle \\
\mathcal{E}(O) & = & \mathcal{E}(F) \cdot t_{N_5} \cdot \mathcal{E}(L) = \langle t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle \\
\mathcal{E}(U) & = & \mathcal{E}(E) \cdot t_{P_3} \cdot \mathcal{E}(O) = \langle t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle \\
\mathcal{E}(Q) & = & (\mathcal{E}(J) \cdot t_{V_3} \cdot \mathcal{E}(T)) \cup (\mathcal{E}(M) \cdot t_{V_4} \cdot \mathcal{E}(K)) \\
& = & \{\langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3} \rangle, \\
& & \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{V_4}, t_{P_1}, t_{P_3}, t_{N_3} \rangle\} \\
\mathcal{E}(R) & = & (\mathcal{E}(C) \cdot t_{N_5} \cdot \mathcal{E}(U)) \cup (\mathcal{E}(T) \cdot t_{N_5} \cdot \mathcal{E}(L)) \\
& = & \{\langle t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle, \\
& & \langle t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle\} \\
\mathcal{E}(S) & = & (\mathcal{E}(J) \cdot t_{V_3} \cdot \mathcal{E}(R)) \cup (\mathcal{E}(M) \cdot t_{V_4} \cdot \mathcal{E}(U)) \cup (\mathcal{E}(Q) \cdot t_{V_4} \cdot \mathcal{E}(L)) \\
& = & \{\langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle, \\
& & \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle, \\
& & \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{V_4}, t_{P_1}, t_{P_3}, t_{N_3}, t_{N_5}, t_{P_2}, t_{P_3}, t_{N_4} \rangle, \\
& & \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{N_5}, t_{P_1}, t_{P_3}, t_{N_3}, t_{V_4}, t_{P_2}, t_{P_3}, t_{N_4} \rangle, \\
& & \langle t_{V_1}, t_{N_1}, t_{V_2}, t_{V_3}, t_{N_2}, t_{V_4}, t_{P_1}, t_{P_3}, t_{N_3}, t_{V_4}, t_{P_2}, t_{P_3}, t_{N_4} \rangle\}
\end{array}$$

L'extension de $S$ contient les cinq arbres de dépendances de la figure 19, sous forme linéaire. Les arbres de dépendances peuvent être construits à partir de ces représentations en suivant la procédure présentée au chapitre 3.

# 8 Analyse

Une partie importante des analyseurs syntaxiques pour grammaires ambiguës, utilisées pour l'analyse syntaxique de la langue naturelle, reposent sur l'algorithme ascendant CYK [Younger, 1967], sur l'algorithme descendant de Earley [Earley, 1968] ou sur des variantes de ces derniers. Ces deux algorithmes appartiennent au paradigme de la programmation dynamique ([Bellman, 1957],[Cormen et al., 2001]) qui leur confère une complexité en temps en $\mathcal{O}(n^3)$.

Ces deux algorithmes ont aussi servi de base à des analyseurs pour grammaires de dépendances. [Lombardo, 1996] propose un analyseur descendant pour grammaires de dépendances, fondé sur l'algorithme de Earley, tandis que [Kahane et al., 1998] définissent une adaptation de l'algorithme CYK pour ces grammaires. Le format des grammaires que ces deux analyseurs permettent de traiter s'inspire du formalisme des grammaires de réécriture de David Hays [Hays, 1964]. La principale différence entre les deux algorithmes de [Lombardo, 1996] et [Kahane et al., 1998] d'une part, et l'algorithme décrit ici, d'autre part, est que les premiers sont vus avant tout comme des reconnaisseurs : l'accent est mis sur la reconnaissance de la phrase, et pas sur les structures produites à l'issue de l'analyse. Dans notre cas, l'accent est mis sur le résultat de l'analyse qui n'est autre qu'une FDP, objet décrit dans le chapitre précédent. Chaque opération de l'analyseur se traduira par un enrichissement de la FDP en cours de construction : création de nouvelles sous-analyses ou de nouvelles composantes.

L'analyseur décrit dans ce chapitre reprend les principes de l'algorithme de [Kahane et al., 1998]. A l'instar de l'algorithme CYK, il s'agit d'un analyseur ascendant, qui combine entre elles les analyses de deux segments contigus $m_{i,k}$ et $m_{k+1,j}$ de la phrase, pour former une analyse du segment $m_{i,j}$. Il emprunte la notion de règle hors-contexte pointée (*dotted rule*) à l'analyseur de Earley. Celle-ci permet d'indiquer une progression dans la reconnaissance de la partie droite d'une règle : la partie droite d'une règle n'est pas reconnue en une seule étape, comme dans l'algorithme CYK. Elle est reconnue symbole par symbole. Cette caractéristique permet de s'affranchir de la contrainte qui consiste à représenter la grammaire sous forme normale de Chomsky avant d'effectuer l'analyse. L'analyseur effectue ce que [Chappelier & Rajman, 1998] appellent une décomposition binaire dynamique de la grammaire (*dynamic binarization of the grammar*) : lors de la reconnaissance d'un symbole dans une partie droite de règle, on combine les analyses de la partie de la règle déjà analysée avec les analyses correspondant au nouveau symbole reconnu. Il s'agit bien d'une opération binaire.

Cette combinaison de l'algorithme d'Earley et de l'algorithme CYK n'est pas nouvelle. On la retrouve dans [Graham et al., 1980], [Erbach, 1994] et [Chappelier & Rajman, 1998] pour les grammaires hors-contexte. La principale différence entre ces approches et la notre, outre le fait que nous produisons des arbres de dépendances, est que nous ne manipulons pas des règles pointées, mais un couple formé d'un automate et d'un état de ce dernier. C'est cet état, appelé l'état courant, qui indique la progression de la reconnaissance d'une règle. La notion d'un couple composé d'un automate et d'un état courant est plus générale que celle d'une règle pointée. Cette dernière n'est autre qu'un cas particulier de la première[18].

L'algorithme prend en entrée une grammaire $G$ et une phrase $S$ de longueur $n$, ($S = m_{1,n}$). Il produit en sortie une FDP $F = \text{FDP}_G(S)$, dont l'extension est l'ensemble $T_G(S)$ de tous les arbres de dépendances que la grammaire $G$ associe à la phrase $S$. L'idée clef de l'algorithme consiste à construire les sous-analyses du segment $m_{i,j}$ de $S$, en combinant les sous-analyses $X_1$ et $X_2$, correspondant respectivement aux segments contigus $m_{i,k}$ et $m_{k+1,j}$ pour $i \leq k < j$. La combinaison des deux sous-analyses $X_1$ et $X_2$ revient à établir une dépendance entre la racine de chaque arbre de l'extension de $X_1$, et la racine de chaque arbre de l'extension de $X_2$.

L'efficacité de l'algorithme provient d'une part, du fait qu'on manipule uniquement des sous-analyses et jamais les extensions de ces dernières (les dépendances évoquées ci-dessus ne sont pas effectivement construites). D'autre part, les analyses des segments de phrases sont effectuées une seule fois, et stockées dans une table bi-dimensionnelle $\mathcal{T}$. La case $\mathcal{T}_{i,j}$ de cette table contient toutes les sous-analyses du segment $m_{i,j}$. L'ordre dans lequel les sous-analyses sont construites est par conséquent important : avant d'analyser le segment $m_{i,j}$, les deux segments $m_{i,k}$ et $m_{k+1,j}$ doivent avoir été analysés pour toutes les valeurs possibles de $k$.

Bien que son principe soit simple, la description de l'algorithme d'analyse est rendue complexe par deux éléments : le non-déterminisme des automates de la grammaire, et la possibilité de prendre en entrée un sous-ensemble de la grammaire, qui aura été délimité par un processus préalable à l'analyse syntaxique.

Dans un souci de lisibilité, nous commencerons par décrire, dans la section 8.1, la version la plus simple de l'algorithme. Nous introduirons ensuite

[18]Une partie droite de règle hors-contexte peut être vue comme un automate linéaire et le point comme un moyen de désigner un état de ce dernier.

successivement, dans les parties 8.2 et 8.4, des modifications permettant de prendre en entrée une grammaire dont les automates sont non-déterministes (cela pour des raisons que nous présenterons alors), puis une version permettant de ne prendre en entrée qu'une partie de la grammaire sous la forme d'un treillis de catégories.

*La version la plus simple de l'analyseur (analyseur pour grammaires déterministes) a déjà été publiée, sous une forme assez différente dans [Kahane et al., 1998]. Elle a ensuite été reprise dans [Nasr et al., 2002] et [Nasr & Rambow, 2004a]. L'adaptation de l'analyseur aux grammaires non-déterministes ainsi que l'adaptation à la prise en compte d'une partie de la grammaire n'ont jamais été publiées.*

## 8.1 Analyseur pour grammaires déterministes

L'algorithme d'analyse consiste à construire des sous-analyses correspondant à des segments de plus en plus longs de la phrase d'entrée et, comme nous l'avons mentionné ci-dessus, à stocker les sous-analyses correspondant au segment $m_{i,j}$ dans la case $\mathcal{T}_{i,j}$ de la table $\mathcal{T}$. Les sous-analyses sont construites à l'aide des opérations de construction de sous-analyses décrites ci-dessous. Ces opérations reposent sur l'hypothèse que les automates de la grammaire sont déterministes : ils ne possèdent pas de transitions vides, et il ne peut y avoir deux transitions émanant d'un même état étiquetées par le même couple $\langle f, c \rangle$.

### 8.1.1 Opérations de construction de sous-analyses

Les opérations de construction de sous-analyses correspondent à des franchissements de transition dans les automates de la grammaire. Lors d'un tel franchissement, une nouvelle dépendance est créée, ou une ancre lexicale est reconnue.

La principale opération de construction de sous-analyses est l'opération d'*attachement*, qui permet d'établir une dépendance entre les racines de deux sous-analyses. Cette opération correspond au franchissement d'une transition dans un automate de la grammaire : la transition correspondant à la dépendance mentionnée ci-dessus. Trois autres opérations sont définies : l'opération de *reconnaissance de l'ancre*, qui correspond au franchissement d'une transition lexicale ; l'opération d'*initialisation*, qui consiste à construire une sous-analyse vierge (une sous-analyse dont l'état courant de l'automate est son état initial) à partir d'un automate ; et l'opération de *factorisation*, qui permet de regrouper, au sein d'une sous-analyse plusieurs sous-analyses partageant le même automate et le même état courant. Ces opérations sont décrites plus précisément ci-dessous :

**Initialisation :** Cette opération permet de créer une sous-analyse à partir d'un automate de la grammaire. Etant donné l'automate $A$, l'opération d'initialisation crée une sous-analyse, dont l'automate est $A$, et dont l'état courant est l'état initial de $A$. Cette sous-analyse ne correspond à aucun segment de la phrase à analyser car pour le moment, ni son ancre, ni aucun de ses dépendants n'ont été reconnus. Ceci est matérialisé, ci-dessous, par la présence du symbole $\perp$ à la place des indices de début et de fin de segment :

$$Init(A) = \langle A, e_0(A), \perp, \perp, \emptyset \rangle$$

La sous-analyse produite à l'issue de l'opération d'initialisation est dite *vierge*. Comme nous l'avons mentionné lors de la description des graphes-$ET$, en 7.3.1, de telles sous-analyses ne sont pas représentées par des sommets-$E$ dans ces graphes.

On étend l'opération d'initialisation à un ensemble $\mathcal{A}$ d'automates :

$$Init(\mathcal{A}) = \bigcup_{A \in \mathcal{A}} Init(A)$$

**Attachement :** Cette opération correspond au franchissement d'une transition d'attachement dans un automate $A_1$. Elle combine deux sous-analyses $X_1 = \langle A_1, e_1, i, k, \mathcal{Y}_1 \rangle$ et $X_2 = \langle A_2, e_2, k+1, j, \mathcal{Y}_2 \rangle$ pour former une sous-analyse $X_3 = \langle A_1, e_3, i, j, \{\langle X_1, t, X_2 \rangle\} \rangle$.

Cette opération peut être paraphrasée de la manière suivante :

- $X_1$ correspond à une sous-analyse du segment $m_{i,k}$ et $X_2$ correspond à une sous-analyse du segment $m_{k+1,j}$.
- $X_2$ est saturée. En d'autres termes, tous les dépendants obligatoires de $X_2$ ont été identifiés dans la phrase.
- $A_1$ attend un dépendant de type $A_2$. En d'autres termes, il existe une transition émanant de l'état courant de $A_1$, dont l'étiquette catégorielle est $A_2$.

L'opération consiste à franchir cette transition, ce qui revient à établir une dépendance entre l'ancre de $A_1$ et l'ancre de $A_2$. Cette opération est représentée formellement ci-dessous[19] :

[19]Rappelons que $\delta^{A_1}(e_1, A_2)$ est est l'ensemble des transitions émanant de l'état $e_1$

$$\begin{array}{rcl} X_1 & = & \langle A_1, e_1, i, k, \mathcal{Y}_1 \rangle \\ X_2 & = & \langle A_2, e_2, k+1, j, \mathcal{Y}_2 \rangle \ \textit{avec}\ e_2 \in \mathcal{Q}_a(A_2) \\ \textit{Attach}(X_1, X_2) & = & \left\{ \begin{array}{ll} \langle A_1, dest(t), i, j, \{\langle X_1, t, X_2 \rangle\} \rangle & \textit{si}\ t = \delta^{A_1}(e_1, A_2) \neq \emptyset \\ \emptyset & \textit{sinon} \end{array} \right. \end{array}$$

L'opération d'attachement vue comme opération de construction d'un graphe $ET$ apparaît dans la figure 28. Elle consiste à construire un nouveau sommet-$E$, correspondant à la sous-analyse $X_3 = \textit{Attach}(X_1, X_2)$, un nouveau sommet-$T$, correspondant à la transition $t$, et à lier le sommet-$T$ aux trois sommets-$E$.

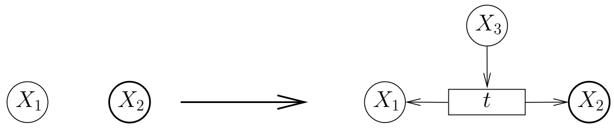



FIG. 28 – L'opération d'attachement

On étend l'opération d'attachement à l'attachement de deux ensembles de sous-analyses $\mathcal{X}_1$ et $\mathcal{X}_2$. Cette opération consiste à tenter l'attachement des sous-analyses de $\mathcal{X}_1$ avec les sous-analyses de $\mathcal{X}_2$. Le résultat de l'opération est un ensemble de sous-analyses :

$$\textit{Attach}(\mathcal{X}_1, \mathcal{X}_2) = \bigcup_{\langle X_1, X_2 \rangle \in \mathcal{X}_1 \times \mathcal{X}_2} \textit{Attach}(X_1, X_2)$$

**Premier attachement :** L'opération de *premier attachement* est un cas particulier de l'opération d'attachement. Elle consiste à établir un attachement entre une sous-analyse vierge et une sous-analyse saturée. La dépendance établie lors de cette opération est la première dépendance (la dépendance la plus à gauche) de l'ancre de la sous-analyse vierge. Cette opération est représentée formellement ci-dessous :

$$\begin{array}{rcl} X_1 & = & \langle A_1, e_0(A_1), \perp, \perp, \emptyset \rangle \\ X_2 & = & \langle A_2, e_2, i, j, \mathcal{Y}_2 \rangle \ \textit{avec}\ e_2 \in \mathcal{Q}_a(A_2) \\ \textit{Attach}1(X_1, X_2) & = & \left\{ \begin{array}{ll} \langle A_1, dest(t), i, j, \{\langle \perp, t, X_2 \rangle\} \rangle & \textit{si}\ t = \delta^{A_1}(e_1, A_2) \\ \emptyset & \textit{sinon} \end{array} \right. \end{array}$$

de l'automate $A_1$ étiqueté par un couple $\langle f, A_2 \rangle$, où $f$ est une étiquette fonctionnelle quelconque.

Vue comme opération de construction d'un graphe $ET$ (figure 29), l'opération de premier attachement consiste à construire un sommet-$E$ correspondant à $X_3$, un sommet-$T$, correspondant à la transition $t$ et à relier le sommet-$T$ aux deux sommets-$E$. Comme nous l'avions précisé, la sous-analyse vierge, correspondant à l'état initial de l'automate n'est pas représentée dans le graphe.


FIG. 29 – L'opération de premier attachement

L'opération de premier attachement peut être étendue à deux ensembles de sous-analyses, de la même manière que l'opération d'attachement.

**Reconnaissance de l'ancre** L'opération de reconnaissance de l'ancre correspond au franchissement d'une transition lexicale dans un automate $A_1$. Il peut sembler étrange de parler d'opération de reconnaissance de l'ancre dans une grammaire lexicalisée. L'existence de cette opération s'explique par le fait qu'un automate possède plusieurs ancres possibles et exclusives. L'opération de reconnaissance de l'ancre consiste à choisir parmi ces dernières une ancre correspondant à un mot de la phrase. C'est durant cette opération qu'une catégorie est associée à un mot.

Considérons une sous-analyse $X_1 = \langle A_1, e_1, i, j, \mathcal{Y}_1 \rangle$ et un mot $m$. Cette opération aboutit si l'automate $A_1$ est tel qu'il existe une transition $t$ (émanant de $e_1$ et aboutissant à un état $e_2$) étiquetée $\langle \texttt{LEX}, m \rangle$. Une telle opération correspond à la reconnaissance d'une ancre de l'automate $A_1$. Elle consiste à franchir la transition lexicale :

$$\begin{array}{rcl} X_1 & = & \langle A_1, e_1, i, j, \mathcal{Y}_1 \rangle \\ \mathit{Reco}(X_1) & = & \begin{cases} \langle A_1, \mathit{dest}(t), i, j, \{\langle X_1, t, \perp \rangle\}\rangle & \mathit{si}\ t = \delta^{A_1}(e_1, \texttt{LEX}, m) \\ \emptyset & \mathit{sinon} \end{cases} \end{array}$$

Du point de vue du graphe $ET$, l'opération consiste à construire un sommet-$E$ correspondant à la sous-analyse $X_2$, un sommet-$T$ correspondant à la transition lexicale, et de relier les deux sommets-$E$ à l'aide du sommet-$T$, comme l'illustre la figure 30.

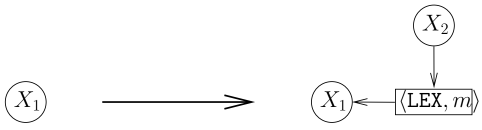



FIG. 30 – L'opération de reconnaissance de l'ancre

On étend cette opération à la reconnaissance d'une ancre $m$par un ensemble de sous-analyses $\mathcal{X}$ :

$$Reco(\mathcal{X}, m) = \bigcup_{X \in \mathcal{X}} Reco(X, m)$$

**Factorisation :** L'opération de factorisation consiste à regrouper, au sein d'une sous-analyse, deux sous-analyses qui partagent le même automate et le même état courant, et qui correspondent au même segment de phrase. Cette factorisation revient à regrouper les composantes de $X_1$ et de $X_2$ au sein d'un même ensemble de composantes. L'opération de factorisation est illustrée graphiquement dans la figure 31 et décrite formellement ci-dessous :

$$\begin{array}{rcl} X_1 & = & \langle A, e, i, j, \mathcal{Y}_1 \rangle \\ X_2 & = & \langle A, e, i, j, \mathcal{Y}_2 \rangle \\ Factor(X_1, X_2) & = & \langle A, e, i, j, \mathcal{Y}_1 \cup \mathcal{Y}_2 \rangle \end{array}$$

On étend cette opération à la factorisation d'une sous-analyse $X$ et d'un ensemble de sous-analyses $\mathcal{X}$. Cette opération consiste à tenter la factorisation de $X$ avec chacune des sous-analyses de $\mathcal{X}$. Si aucune de ces opérations n'aboutit, $X$ est ajoutée à l'ensemble $\mathcal{X}$.

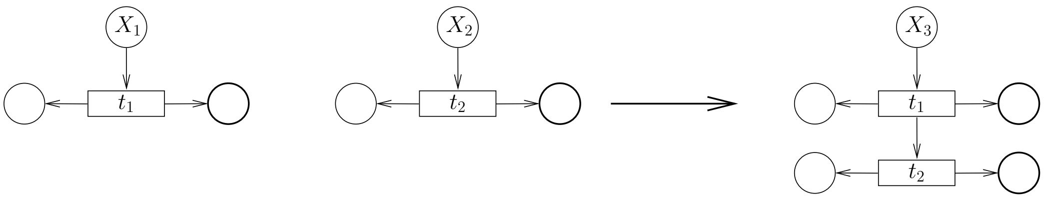



FIG. 31 – Factorisation de deux sous-analyses

### 8.1.2 L'algorithme d'analyse

L'algorithme d'analyse consiste à construire itérativement des sous-analyses correspondant à des segments de phrase de taille croissante en utilisant les quatre opérations vues ci-dessus.

Dans un but de lisibilité, nous désignerons par $X_{i,j}$ une sous-analyse correspondant au segment $m_{i,j}$ de la phrase à analyser. Lorsqu'une sous-analyse $X_{i,j}$ est créée, elle est stockée dans la case $\mathcal{T}_{i,j}$ de la table $\mathcal{T}$. $\mathcal{T}$ est remplie de telle manière que $\mathcal{T}_{i,j}$ soit remplie après que $\mathcal{T}_{i,k}$ et $\mathcal{T}_{k+1,j}$ (avec $i \leq k < j$) l'aient été. La figure 32 montre l'ordre de remplissage de la table correspondant à une phrase de longueur 5. La première case remplie est $\mathcal{T}_{1,1}$, suivie de $\mathcal{T}_{2,2}$, suivie de $\mathcal{T}_{1,2}$ et ainsi de suite.

| 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|
| 1 | 3 | 6 | 10 | 15 | 1 |
| | 2 | 5 | 9 | 14 | 2 |
| | | 4 | 8 | 13 | 3 |
| | | | 7 | 12 | 4 |
| | | | | 11 | 5 |

FIG. 32 – Ordre de remplissage de la table d'analyse

L'analyse aboutit si à la fin du remplissage de la table, la case $\mathcal{T}_{1,n}$ contient au moins une sous-analyse saturée, ce qui revient à dire qu'une analyse couvrant la totalité de la phrase a été construite.

L'algorithme commence par construire l'ensemble de sous-analyses $I_0$ dont les éléments sont les sous-analyses vierges correspondant aux automates de la grammaire. Cet ensemble est construit à l'aide de l'opération d'initialisation, appliquée à l'ensemble $\mathcal{A}$ des automates de la grammaire :

$$I_0 \leftarrow Init(\mathcal{A})$$

Le but de cette étape est de ne plus manipuler directement des automates dans le reste de l'algorithme, mais uniquement des sous-analyses.

La table est ensuite remplie colonne par colonne, en remontant dans chacune d'entre elles à partir de la diagonale, comme dans la figure 32.

Le remplissage des cases de la diagonale ($\mathcal{T}_{i,i}$) s'effectue différemment du remplissage des autres cases. Il consiste à inspecter toutes les sous-analyses de $I_0$. Si l'une d'entre elles est dans un état duquel émane une transition lexicale étiquetée $m_i$, alors l'opération de reconnaissance de l'ancre s'applique, et une nouvelle sous-analyse est créée :

$$\mathcal{T}_{i,i} \leftarrow Reco(I_0, m_i)$$

Ceci correspond soit au cas où tous les dépendants obligatoires d'un mot sont situés à sa droite, soit au cas où il n'en possède pas. Pour chacun de ces deux cas, le premier élément reconnu par l'automate est l'ancre. A l'issue de la reconnaissance de l'ancre, l'opération de premier attachement est tentée entre cette nouvelle sous-analyse et les analyses vierges de $I_0$ :

$$\mathcal{T}_{i,i} \leftarrow Attach1(I_0, \mathcal{T}_{i,i})$$

Les autres cases $\mathcal{T}_{i,j}$ de la colonne $j$ sont remplies en combinant les éléments des cases $\mathcal{T}_{i,k}$ et $\mathcal{T}_{k+1,j}$ grâce à l'opération d'attachement :

$$\mathcal{T}_{i,j} \leftarrow Attach(\mathcal{T}_{i,k}, \mathcal{T}_{k+1,j})$$

Cette opération peut mener à la création de sous-analyses dans la case $\mathcal{T}_{i,j}$. Ces nouvelles sous-analyses sont alors soumises aux deux opérations de reconnaissance de l'ancre :

$$\mathcal{T}_{i,j} \leftarrow Reco(\mathcal{T}_{i,j}, m_j)$$

et de premier attachement, avec les sous-analyses de $I_0$ :

$$\mathcal{T}_{i,j} \leftarrow Attach1(I_0, \mathcal{T}_{i,j})$$

Ces deux opérations sont réitérées tant qu'elles donnent naissance à de nouvelles sous-analyses.

Lors des différentes étapes de l'algorithme, des sous-analyses sont créées et ajoutées aux cases de la table. Il est important de vérifier, lors de l'ajout de la sous-analyse $X$ à une case $\mathcal{T}_{i,j}$, que $X$ ne peut pas être factorisée avec une autre sous-analyse déjà présente dans $\mathcal{T}_{i,j}$. Il s'agit là d'un point essentiel. C'est en effet par ce moyen qu'est mis en œuvre le partage de sous-arbres, et que l'on limite le nombre de sous-analyses dans une case de la table. L'algorithme est décrit sous forme de pseudo-code dans la figure 33.

Lors de l'analyse syntaxique, certaines sous-analyses qui ne rentrent dans la composition d'aucune analyse globale de la phrase peuvent avoir été créées. Ces sous-analyses peuvent par conséquent être éliminées de la forêt. Cette élimination se fait par marquage des sous-analyses d'une FDP $\langle \mathcal{X}, \mathcal{F} \rangle$ atteignable à partir d'une sous-analyse de $\mathcal{F}$. A l'issue de cette étape, toute sous-analyse non marquée est éliminée de la forêt.

**Entrée :**

une grammaire GDG $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I} \rangle$
une phrase à analyser $m_{1,n}$

**Sortie :** une FDP$\langle \mathcal{X}, \mathcal{F} \rangle$

**Initialisation**

$I_0 \leftarrow Init(\mathcal{A})$
`Pour` $i$ `allant de` $1$ `à` $n$
  $\mathcal{T}_{i,i} \leftarrow Reco(I_0, m_i)$
  `tant que` de nouvelles sous-analyses sont créées `faire`
    $\mathcal{T}_{i,i} \leftarrow Factor(\mathcal{T}_{i,i}, Attach1(I_0, \mathcal{T}_{i,i}))$

**Remplissage de la table**

`Pour` $i$ `allant de` $1$ `à` $n$
  `Pour` $j$ `allant de` $i-1$ `à` $1$
    `Pour` $k$ `allant de` $i$ `à` $j-1$
      $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Attach(\mathcal{T}_{i,k}, \mathcal{T}_{k+1,j}))$
    `tant que` de nouvelles sous-analyses sont créées `faire`
      $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Reco(\mathcal{T}_{i,j}, m_j))$
      $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Attach1(I_0, \mathcal{T}_{i,j}))$

$\mathcal{X} = \bigcup_{i,j} \mathcal{T}_{i,j}$
$\mathcal{F} = \bigcup_{X \in \mathcal{T}_{1,n}} X$ si $q(X) \in \mathcal{Q}_a(Auto(X))$

Fig. 33 – L'algorithme d'analyse pour grammaires déterministes

Le résultat de l'analyse d'une séquence de la forme $n_0$ $v_1$ $n_2$ $p_3$ $n_4$ $p_5$ $n_6$ par la grammaire de la figure 26 est représenté dans la figure 34. Dans la partie supérieure de la figure est représentée la table issue de l'analyse. Chaque sous-analyse est représentée par une lettre majuscule. Dans la partie centrale de la figure, la structure interne de chaque sous-analyse est dévoilée, ainsi que l'opération qui lui a donné naissance. La représentation graphique de la FDP créée se trouve dans la figure 27.

## 8.2 Analyseur pour grammaires non-déterministes

La version de l'analyseur proposée en 8.1 supposait que les automates de la grammaire étaient déterministes. Cette hypothèse n'est en rien restrictive car, pour tout automate non-déterministe, on sait construire un automate déterministe qui lui est équivalent. Il suffit donc, dans notre cas, de déterminiser les automates de la grammaire avant d'effectuer l'analyse. Cette solution n'est malheureusement plus viable dès lors que les automates sont pondérés (les transitions sont affectées d'un poids), car certains automates non-déterministes pondérés ne sont pas déterminisables ([Buchsbaum et al., 2000]). Or, de tels automates sont utilisés dans les GDG probabilistes définies en 5. C'est la raison pour laquelle nous allons, dans cette section, modifier l'analyseur, afin qu'il puisse accepter en entrée une grammaire dont les automates ne sont pas déterministes.

### 8.2.1 Modification des opérations de construction

Les modifications vont consister à ajouter une opération, qui correspond au franchissement d'une transition vide, et à modifier les opérations d'initialisation, d'attachement et de reconnaissance de l'ancre, afin de prendre en compte le non-déterminisme. Le résultat de ces modifications est que ces opérations pourront produire plus d'une sous-analyse, précisément dans les cas de non-déterminisme. Le reste de l'algorithme est inchangé. Les nouvelles définitions des opérations sont décrites ci-dessous :

**Franchissement d'une transition vide :** Comme son nom l'indique, cette opération correspond au franchissement d'une transition vide dans un automate $A$. Cette opération prend comme seul argument une sous-analyse $X = \langle A, e, i, j, \mathcal{Y} \rangle$ et produit un ensemble d'analyses noté $Eps(X)$.

L'opération aboutit s'il existe au moins une transition vide à partir

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|
| $A, B$ | $J$ | $M$ | | $Q$ | | $S$ | 1 |
| | | | | | | | 2 |
| | | $C$ | | $T$ | | $R$ | 3 |
| | | | $E$ | $K$ | | $U$ | 4 |
| | | | | $F$ | | $O$ | 5 |
| | | | | | $H$ | $L$ | 6 |
| | | | | | | $I$ | 7 |

| | |
|---|---|
| $A = \langle N, 1, 1, 1, \{\langle \perp, t_{N_1}, \perp \rangle\}\rangle$ | *Reco* |
| $B = \langle V, 1, 1, 1, \{\langle \perp, t_{V_1}, A \rangle\}\rangle$ | *Attach1* |
| $C = \langle N, 1, 3, 3, \{\langle \perp, t_{N_2}, \perp \rangle\}\rangle$ | *Reco* |
| $E = \langle P, 1, 4, 4, \{\langle \perp, t_{P_1}, \perp \rangle\}\rangle$ | *Reco* |
| $F = \langle N, 1, 5, 5, \{\langle \perp, t_{N_3}, \perp \rangle\}\rangle$ | *Reco* |
| $H = \langle P, 1, 6, 6, \{\langle \perp, t_{P_2}, \perp \rangle\}\rangle$ | *Reco* |
| $I = \langle N, 1, 7, 7, \{\langle \perp, t_{N_4}, \perp \rangle\}\rangle$ | *Reco* |
| $J = \langle V, 2, 1, 2, \{\langle B, t_{V_2}, \perp \rangle\}\rangle$ | *Reco* |
| $K = \langle P, 2, 4, 5, \{\langle E, t_{P_3}, F \rangle\}\rangle$ | *Attach* |
| $L = \langle P, 2, 6, 7, \{\langle H, t_{P_3}, I \rangle\}\rangle$ | *Attach* |
| $M = \langle V, 3, 1, 3, \{\langle J, t_{V_3}, C \rangle\}\rangle$ | *Attach* |
| $T = \langle N, 1, 3, 5, \{\langle C, t_{N_5}, K \rangle\}\rangle$ | *Attach* |
| $O = \langle N, 1, 5, 7, \{\langle F, t_{N_5}, L \rangle\}\rangle$ | *Attach* |
| $U = \langle P, 2, 4, 7, \{\langle E, t_{P_3}, O \rangle\}\rangle$ | *Attach* |
| $Q = \langle V, 3, 1, 5, \{\langle J, t_{V_3}, T \rangle, \langle M, t_{V_4}, K \rangle\}\rangle$ | *Attach* |
| $R = \langle N, 1, 3, 7, \{\langle C, t_{N_5}, U \rangle, \langle T, t_{N_5}, L \rangle\}\rangle$ | *Attach* |
| $S = \langle V, 3, 1, 7, \{\langle J, t_{V_3}, R \rangle, \langle M, t_{V_4}, U \rangle, \langle Q, t_{V_4}, L \rangle\}\rangle$ | *Attach* |

| transitions de N | transitions de P | transitions de V |
|---|---|---|
| $t_{N_1} = \langle 0, \langle \texttt{LEX}, n_0 \rangle, 1 \rangle$ | $t_{P_1} = \langle 0, \langle \texttt{LEX}, p_3 \rangle, 1 \rangle$ | $t_{V_1} = \langle 0, \langle \texttt{SUBJ}, N \rangle, 1 \rangle$ |
| $t_{N_2} = \langle 0, \langle \texttt{LEX}, n_2 \rangle, 1 \rangle$ | $t_{P_2} = \langle 0, \langle \texttt{LEX}, p_5 \rangle, 1 \rangle$ | $t_{V_2} = \langle 1, \langle \texttt{LEX}, v_1 \rangle, 2 \rangle$ |
| $t_{N_3} = \langle 0, \langle \texttt{LEX}, n_4 \rangle, 1 \rangle$ | $t_{P_3} = \langle 0, \langle \texttt{PREP}, N \rangle, 1 \rangle$ | $t_{V_3} = \langle 2, \langle \texttt{DOBJ}, N \rangle, 3 \rangle$ |
| $t_{N_4} = \langle 0, \langle \texttt{LEX}, n_6 \rangle, 1 \rangle$ | | $t_{V_4} = \langle 3, \langle \texttt{CIRC}, P \rangle, 3 \rangle$ |
| $t_{N_5} = \langle 1, \langle \texttt{CIRC}, P \rangle, 1 \rangle$ | | |

FIG. 34 – Résultat de l'analyse de la séquence $n_0\ v_1\ n_2\ p_3\ n_4\ p_5\ n_6$

de l'état $e$ de $A$. Le produit de l'opération est une ou plusieurs sous-analyses, de la forme $\langle A, e', i, j, \langle X, t, \perp\rangle\}\rangle$, car il peut y avoir plusieurs transitions vides émanant d'un état. L'opération est définie formellement ci-dessous, et représentée graphiquement dans la figure 35.

$$\begin{array}{rcl} X & = & \langle A, e, i, j, \mathcal{Y}\rangle \\ Eps(X) & = & \displaystyle\bigcup_{t \in \delta^A(e,\varepsilon)} \langle A, dest(t), i, j, \{\langle X, t, \perp\rangle\}\rangle \end{array}$$

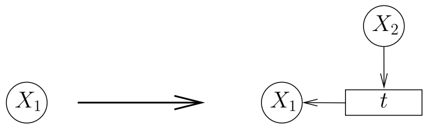



FIG. 35 – Franchissement d'une transition vide

On note $Eps^*(X)$ la clôture de $Eps(X)$. Cette opération consiste à former l'ensemble de tous les états que l'on peut atteindre à partir d'un état donné en ne suivant que des transitions vides. Cet ensemble est défini récursivement de la manière suivante :

$$Eps^*(X) = Eps(X) \cup \bigcup_{X' \in Eps(X)} Eps^*(X')$$

L'extension de ces opérations à un ensemble $\mathcal{X}$ de sous-analyses est évidente :

$$Eps(\mathcal{X}) = \bigcup_{X \in \mathcal{X}} Eps(X) \quad \text{et} \quad Eps^*(\mathcal{X}) = \bigcup_{X \in \mathcal{X}} Eps^*(X)$$

**Initialisation :** L'opération d'initialisation, dans le cas des grammaires déterministes, consistait à produire, à partir d'un automate $A$, une sous-analyse dont l'état courant était l'état initial de $A$. La nouvelle version de l'opération d'initialisation consiste à créer une sous-analyse pour tout état appartenant à la clôture de l'état initial de $A$ :

$$Init(A) = Eps^*(\langle A, e_0(A), \perp, \perp, \emptyset\rangle)$$

Comme nous l'avions fait dans le cas déterministe, on étend l'opération d'initialisation à un ensemble d'automates.

**Attachement :** Du fait du non-déterminisme des automates, il peut maintenant exister plusieurs transitions émanant d'un seul état, qui sont étiquetées par le même couple $\langle f, c\rangle$. Dans ce cas, l'opération d'attachement produira autant de sous-analyses qu'il existe de transitions. De plus, à l'issue du franchissement de ces transitions, les éventuelles $\varepsilon$-transitions sont à leur tour franchies, grâce à l'opération de clôture :

$$\begin{array}{rcl} X_1 & = & \langle A_1, e_1, i, k, \mathcal{Y}_1\rangle \\ X_2 & = & \langle A_2, e_2, k+1, j, \mathcal{Y}_2\rangle \\ Attach(X_1, X_2) & = & Eps^*(\bigcup_{t \in \delta^{A_1}(e_1, A_2)} \langle A_1, dest(t), i, j, \{\langle X_1, t, X_2\rangle\}\rangle) \end{array}$$

Comme dans le cas déterministe, on étend l'opération d'attachement à l'attachement de deux ensembles de sous-analyses.

**Reconnaissance de l'ancre :** Lorsque plusieurs transitions lexicales correspondant à la même entrée lexicale émanent d'un même état[20], l'opération de reconnaissance de l'ancre donnera naissance à plusieurs sous-analyses. De plus, à l'issue du franchissement de ces transitions, les éventuelles $\varepsilon$-transitions seront à leur tour franchies, grâce à l'opération de clôture :

$$Reco(X_1, m) = \bigcup_{t \in \delta^{A_1}(e_0, \mathtt{LEX}, m)} \langle A_1, dest(t), i, j, \{\langle X_1, t, \perp\rangle\}\rangle$$

Comme dans le cas déterministe, on étend l'opération de reconnaissance de l'ancre à la reconnaissance d'une ancre $m$ par un ensemble de sous-analyses $\mathcal{X}$.

## 8.3 Analyses partielles

Lorsque la grammaire utilisée pour effectuer l'analyse ne possède pas une couverture suffisante, ou lorsque la phrase à analyser est agrammaticale, l'analyse n'aboutit pas. A l'issue du remplissage de la table d'analyse, la case $\mathcal{T}_{1,n}$ est vide, ou ne contient que des sous-analyses non saturées. Il est néanmoins possible, dans ces cas, de produire une *analyse partielle* de la phrase : une séquence d'arbres de dépendances, telle que la concaténation des segments de phrases correspondant aux différents arbres constituent la phrase.

[20] Dans la pratique, cette situation n'a pas lieu de se produire. Nous l'avons néanmoins prise en compte, dans un souci de généralité.

La procédure de production des analyses partielles que nous proposons est assez rudimentaire. Elle consiste à compléter le remplissage de la table par des sous-analyses *factices*. Ces dernières ne correspondent pas à des automates de la grammaire. Elles ne sont là que pour regrouper des sous-analyses de segments de phrases adjacents. Elles ne donnent pas lieu à l'établissement de dépendances. La procédure consiste à parcourir la table : pour toute case $\mathcal{T}_{i,j}$ vide, ou ne contenant pas de sous-analyses saturées, une sous-analyse factice est créée, qui combine les sous-analyses des cases $\mathcal{T}_{i,k}$ et $\mathcal{T}_{k+1,j}$. La procédure est décrite sous la forme de pseudo-code dans la figure 36, où les sous-analyses factices sont de la forme $\langle \perp, \perp, i, j, \{\langle X_1, \perp, X_2\rangle\}\rangle$ : des signes $\perp$ à la place de l'automate, de l'état courant et de la transition de la composante.

**Entrée :**
une grammaire table d'analyse $\mathcal{T}$

**Sortie :** La table $\mathcal{T}$ complétée

`Pour` $i$ `allant de` $1$ `à` $n$
  `Pour` $j$ `allant de` $i-1$ `à` $1$
    `si` $\mathcal{T}_{i,j}$ est vide `alors`
    `Pour` $k$ `allant de` $i$ `à` $j-1$
      `Pour tout` $X_1 \in \mathcal{T}_{i,k}$ `faire`
        `Pour tout` $X_2 \in \mathcal{T}_{k+1,j}$ `faire`
          $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, \langle \perp, \perp, i, j, \{\langle X_1, \perp, X_2\rangle\}\rangle)$

Fig. 36 – Complétion de la table d'analyse par des sous-analyses factices

Comme nous l'avons mentionné ci-dessus, cette méthode de complétion de la table est très sommaire. Elle construit toutes les analyses partielles possibles sans exploiter le fait que, lorsqu'un segment de phrase a déjà été créé, il n'est pas toujours nécessaire de construire des analyses partielles de ce segment. Il existe diverses manières de limiter le nombre d'analyses partielles créées, en tirant profit des analyses existantes, ou en utilisant des heuristiques de rattachement. Nous n'avons pas exploré ces pistes, parce que c'est le modèle probabiliste qui sera chargé de retrouver, parmi les différentes analyses partielles possibles de la phrase, l'analyse partielle la plus probable.

## 8.4 Analyseur prenant en entrée un treillis de catégories

On peut remarquer que dans les deux premières versions de l'analyseur, une partie importante des automates de la grammaire ne sera jamais utilisée lors de l'analyse d'une phrase $S$ : il s'agit de tous les automates dont les ancres ne correspondent à aucun mot de $S$. Ces automates n'entreront dans aucune analyse complète de la phrase, car aucune de leurs transitions lexicales ne pourra être franchie. Cependant, tous ces automates seront pris en compte lors de l'analyse et pourront être partiellement parcourus lors de cette dernière, dégradant ainsi les performances de l'analyseur.

Il est possible, avant d'effectuer l'analyse, de réduire la grammaire aux seuls automates dont au moins une ancre apparaît dans la phrase. Il suffit pour cela d'associer à chaque mot $m$ du lexique, un ensemble d'automates, précisément les automates dont $m$ constitue une ancre. Cet ensemble sera noté $\mathcal{A}(m)$. Avant l'analyse de la phrase $m_{1,n}$, un ensemble, composé de l'union des ensembles $\mathcal{A}(m_i)$ des automates que peuvent ancrer les mots $m_i$, est créé. C'est avec ce sous-ensemble de la grammaire initiale que l'analyse sera menée. Nous appellerons cette méthode *réduction de la grammaire par accès au lexique*, car c'est le lexique qui permet de réduire le nombre d'automates à prendre en compte lors de l'analyse.

La réduction de la grammaire ne sélectionne pas, pour un mot $m_i$, l'automate de $\mathcal{A}(m_i)$ qui sera utilisé lors de l'analyse syntaxique. Ce choix est effectué lors de l'analyse. Il est aussi possible de choisir, avant l'analyse, pour chaque mot $m_i$, l'automate de $\mathcal{A}(m_i)$ qui sera utilisé lors de l'analyse. Dans une telle perspective, ce choix n'est plus effectué par l'analyseur, mais par un autre processus, déjà évoqué en 4.3, et connu sous le nom d'étiquetage grammatical ou supertagging [Bangalore, 1997, Bangalore & Joshi, 1999b]. Le résultat de ce processus est d'associer à chaque mot $m_i$ un automate unique $A_i \in \mathcal{A}(m_i)$. La grammaire se réduit alors à l'ensemble constitué des automates $A_i$. Cette réduction drastique de la grammaire va nettement améliorer les performances de l'analyseur.

La situation peut être comparée, dans une certaine mesure, à l'étiquetage morpho-syntaxique d'une phrase préalablement à son analyse à l'aide d'une GHC. Dans les deux cas, une partie de l'ambiguïté - l'ambiguïté lexicale - est résolue lors de la phase d'étiquetage. Cependant, l'influence de l'étiquetage grammatical sur l'analyse est plus importante que celle de l'étiquetage morpho-syntaxique classique. En effet, en plus d'éliminer l'ambiguïté lexicale, l'étiquetage grammatical réduit la grammaire utilisée pour l'analyse syntaxique : seuls les automates correspondant aux catégories attribuées, lors de l'étiquetage, aux mots de la phrase sont utilisées pour l'analyse. C'est là une conséquence directe de la lexicalisation de la grammaire.

Malheureusement, comme nous le verrons dans la partie III, cette perspective n'est pas réaliste. En effet, les techniques actuelles d'étiquetage grammatical fondées sur les modèles de Markov cachés, ne permettent pas, lorsqu'elles sont appliquées aux grammaires que nous utiliserons dans la partie III, d'associer, à chaque mot de la phrase, l'automate qui entrera dans l'analyse correcte de la phrase, avec un taux d'erreur raisonnable. Or, une erreur lors de l'étape d'étiquetage grammatical provoque généralement l'échec de l'analyse syntaxique.

Si l'on ne peut se baser sur l'étiquetage grammatical pour sélectionner un automate pour chaque mot de la phrase, il est néanmoins possible de s'en servir pour effectuer une réduction de la grammaire plus poussée que la seule réduction par accès au lexique. Une mise en œuvre simple de cette idée consiste à produire, en sortie de l'étiqueteur grammatical, les $n$ séquences les plus probables de catégories, et à réaliser successivement l'analyse avec chacune de ces séquences. Cette méthode n'est pas très satisfaisante d'un point de vue économique. Il est en effet possible - et c'est ce que l'on observe en pratique - que deux séquences de catégories soient très proches : elles n'associent de catégories différentes qu'à un petit nombre de mots de la phrase. Lors des analyses des $n$ séquences, toutes les parties partagées par plus d'une séquence seront analysées plus d'une fois.

L'idée présentée dans cette section consiste à représenter cette suite de $n$ séquences sous la forme d'un treillis, que l'on appellera *treillis de catégories*, et qui constitue l'entrée de l'analyseur. La particularité du treillis est qu'il limite, autant que possible, la duplication de parties communes des différentes séquences. Un tel treillis peut aussi être produit directement par l'étiquetage grammatical, comme proposé dans [Nasr & Volanschi, 2004].

La prise en compte de treillis en entrée d'un analyseur syntaxique a fait l'objet d'un grand nombre de travaux dans le domaine de la reconnaissance de la parole ([Tomita, 1986], [Nakagawa, 1987], [Chappelier et al., 1999], [Roark, 2002]). Dans ce cadre, l'entrée de l'analyseur est un treillis de mots, produit par un système de reconnaissance de la parole. Le but de l'analyse syntaxique est de déceler, parmi les différents chemins du treillis (les différentes suites de mots possibles), lesquels correspondent à des phrases reconnues par la grammaire. L'analyseur est utilisé dans un tel cadre comme un reconnaisseur (on ne s'intéresse pas particulièrement aux structures produites par l'analyseur) ou comme un *modèle de langage*, attribuant une probabilité aux séquences de mots constituant une phrase reconnue par la grammaire. Dans notre cas, il s'agit d'un treillis de catégories et non d'un treillis de mots. La méthode que nous présentons ici s'inspire de [Chappelier et al., 1999]. Elle emprunte à ces travaux l'idée de représenter la structure du treillis dans la table d'analyse. Elle s'en distingue cependant par le fait qu'à tout chemin du treillis est associée une grammaire : la grammaire formée des automates constituant les étiquettes des chemins. Ce chemin est analysé en n'utilisant que les règles de cette grammaire. Cette particularité est absente des travaux effectués dans le domaine de la reconnaissance de la parole, car la grammaire utilisée pour l'analyse est unique. Elle ne dépend pas de la structure du treillis.

### 8.4.1 Le treillis de catégories

Le treillis de catégories a été choisi comme format d'entrée unique de l'analyseur, en raison de la souplesse de ce mode de représentation. On peut en effet représenter, sous la forme d'un treillis, tous les cas mentionnés ci-dessus, que ce soit : le résultat d'une réduction de la grammaire par accès au lexique ; la meilleure solution d'un étiqueteur grammatical ; ou ses $n$ meilleures solutions. Dans le premier cas, le treillis apparaîtra comme une succession de $n$ faisceaux de transitions, où chaque faisceau représente les différents automates de $\mathcal{A}(m_i)$. Dans le second cas, le treillis sera réduit à une structure linéaire. Dans le troisième, il pourra adopter une forme quelconque. Les trois cas ont été représentés dans la figure 37.

Le treillis est représenté sous la forme d'un automate fini sur l'ensemble des catégories de la grammaire. On adoptera, par conséquent, la terminologie des automates, et nous parlerons de transitions et d'états du treillis, au lieu de sommets et d'arcs. Nous considérerons de plus que les états des automates sont numérotés par des entiers et nous désignerons chaque état par son numéro (on parlera d'état $i$ pour désigner l'état dont le numéro est $i$). Toute séquence de transitions du treillis menant d'un état $i$ à un état $j$ sera appelé un *sous-chemin* du treillis. L'ensemble des sous-chemins menant d'un état $i$ à un état $j$ d'un treillis $\mathcal{R}$ sera appelé le *sous-treillis* $\langle i, j\rangle$ de $\mathcal{R}$, noté $\mathcal{R}_{i,j}$ .

Lorsque $i$ est l'état initial du treillis, on parlera de *chemin initial* et de *sous-treillis initial*. Lorsque, de plus, $j$ est son état d'acceptation, on parlera de *chemin d'acceptation* du treillis. Les états du treillis sont regroupés en *colonnes*. La colonne $i$ est composée de tous les états constituant l'extrémité des chemins initiaux de longueur $i$, comme l'illustre la figure 38. Etant donné

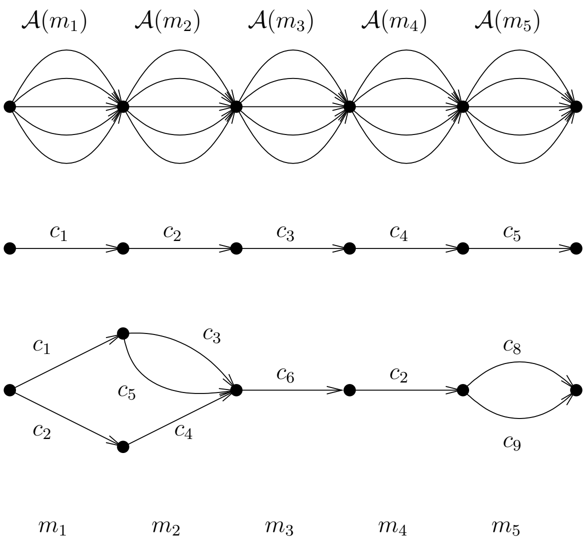



FIG. 37 – Représentation, sous la forme de treillis, du résultat d'une réduction de la grammaire par accès au lexique, de la meilleure solution d'un étiquetage grammatical et de ses $n$ meilleures solutions

un état $e$ du treillis, la colonne à laquelle il appartient est notée $col(e)$.

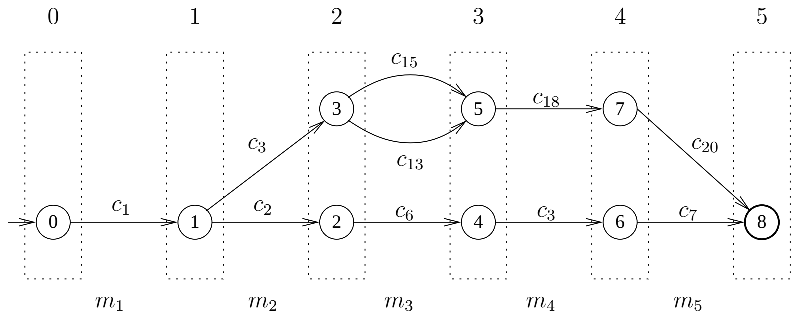



FIG. 38 – Représentation d'un treillis de catégories sous la forme d'un automate fini et d'une partition de ses états en colonnes

Le treillis de catégories est représenté formellement comme un couple $\langle \mathcal{R}, col \rangle$ où $\mathcal{R} = \langle Q_\mathcal{R}, C_\mathcal{R}, \delta_\mathcal{R}, i_\mathcal{R}, F_\mathcal{R} \rangle$ est le treillis représenté sous la forme d'un automate, et $col \in Q_\mathcal{R} \times \mathbb{N}$ est une fonction qui associe, à chaque état de l'automate $\mathcal{R}$, le numéro de sa colonne. $C_\mathcal{R}$ est le sous-ensemble de catégories de la grammaire qui apparaissent dans le treillis.

On définit, de plus, le prédicat *suivant* sur les états d'un treillis $\mathcal{R}$.

*suivant*$(i, j)$ est vrai, s'il existe un chemin de longueur supérieure ou égale à zéro dans $\mathcal{R}$, menant de l'état $i$ à l'état $j$. Nous dirons, dans ce cas, que l'état $j$ suit l'état $i$ et, symétriquement, que l'état $i$ précède l'état $j$.

### 8.4.2 Modification de l'algorithme d'analyse

La difficulté majeure qui se présente pour la prise en compte d'un treillis de catégories en entrée de l'analyseur provient du fait qu'il faut s'assurer qu'une analyse ne met en jeu que des catégories constituant des chemins d'acceptation du treillis. Pour reprendre l'exemple de la figure 38, une analyse ne peut mettre en jeu les catégories $c_1, c_3, c_6, c_{18}, c_{20}$, car il n'existe pas, dans le treillis, de chemin d'acceptation correspondant à cette suite de catégories.

Afin de garantir cette condition, on associe à toute catégorie apparaissant dans le treillis, une transition de ce treillis, appelée *transition d'ancrage* de la catégorie. La transition d'ancrage est associée, plus précisément, à une sous-analyse vierge dont l'automate correspond à cette catégorie. Si le treillis possède plus d'une transition étiquetée par une même catégorie, alors plusieurs sous-analyses vierges seront créées. Celles-ci se distingueront par leur transition d'ancrage. Le but de la transition d'ancrage est de garder, au cours du processus d'analyse, la connaissance de la transition qui a introduit cette catégorie. Pour qu'une analyse soit valide, les transitions d'ancrage des automates qui rentrent dans sa composition doivent constituer un chemin d'acceptation du treillis.

La mise en œuvre de ces nouvelles conditions nécessite la modification des définitions des sous-analyses, des opérations d'initialisation et d'attachement, de la table de sous-analyses et de l'algorithme d'analyse. Ces modifications sont décrites ci-dessous :

**Modification de la définition des sous-analyses**

Une sous-analyse correspond maintenant à un sous-treillis, et non plus à un segment de phrase. De plus, comme nous l'avons vu ci-dessus, une sous-analyse est associée à une transition du treillis : sa transition d'ancrage. Toute sous-analyse produite lors de l'analyse doit être telle que sa transition d'ancrage appartienne à son sous-treillis, ou que son sous-treillis précède sa transition d'ancrage.

Formellement, une sous-analyse est maintenant définie comme un sextuplet $X = \langle A, e, i, j, t_a, \mathcal{Y} \rangle$. La définition des éléments $A$, $e$ et $\mathcal{Y}$ est inchangée. $i$ et $j$ correspondent respectivement aux numéros de l'état initial et de l'état d'acceptation du sous-treillis correspondant à $X$, et $t_a$ est la transition d'ancrage $X$, notée $t_a(X)$.

### Modification de l'opération d'initialisation

L'opération d'initialisation ne prend maintenant plus comme paramètre un automate, mais une transition $t = \langle i, A, j \rangle$ du treillis ($i$ est l'état origine de la transition, $j$ sont état destination et $A$ son étiquette). Elle crée une sous-analyse dont l'automate est $A$, et dont la transition d'ancrage est $t$ :

$$Init(t = \langle i, A, j \rangle) = \langle A, e_0(A), 0, 0, t, \emptyset \rangle$$

Comme précédemment, on étend l'opération d'initialisation à un ensemble $\mathcal{T}$ de transitions :

$$Init(\mathcal{T}) = \bigcup_{t \in \mathcal{T}} Init(t)$$

### Modification de l'opération d'attachement

Pour que l'opération d'attachement entre deux sous-analyses $X_1$ et $X_2$ aboutisse, il faut que leurs sous-treillis soient adjacents : l'état d'acceptation du sous-treillis de $X_1$ est égal à l'état initial de $X_2$. De plus, la transition d'ancrage de $X_1$ doit suivre le sous-treillis de $X_2$. Cette condition, moins intuitive que la première, est nécessaire pour garantir que le nouveau sous-treillis, issu de la combinaison des sous-treillis de $X_1$ et $X_2$, précède la transition d'attachement de $X_1$, et finisse par l'atteindre, à un moment donné de l'analyse.

### Modification de la définition de la table de sous-analyses

Les états du treillis sont numérotés par des entiers consécutifs, à partir de 0. La numérotation respecte l'ordre des colonnes : étant donné deux états numérotés $x$ et $y$, $x$ est inférieur à $y$ si le numéro de sa colonne est inférieur au numéro de la colonne de $y$ :

$$x < y \Leftrightarrow col(x) < col(y)$$

La numérotation des états du treillis permet d'établir un lien entre la structure de ce treillis et la table d'analyse. Les indices de la table correspondent maintenant aux numéros des états du treillis. Une case $\mathcal{T}_{i,j}$ de la table correspond à un sous-treillis, si l'état $i$ précède l'état $j$. Seules ces dernières sont susceptibles de contenir des sous-analyses. Nous les appellerons les *cases valides* de la table. Les autres cases ne seront pas remplies lors d'analyse. Il s'agit des cases comportant le signe $\perp$ dans la table de la figure 39, qui correspond au treillis de la figure 38.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| $\perp$ | 1 | | | | | | | | 0 |
| | $\perp$ | 1 | 1 | | | | | | 1 |
| | | $\perp$ | $\perp$ | 1 | $\perp$ | | $\perp$ | | 2 |
| | | | $\perp$ | $\perp$ | 1 | $\perp$ | | | 3 |
| | | | | $\perp$ | $\perp$ | 1 | $\perp$ | | 4 |
| | | | | | $\perp$ | $\perp$ | 1 | | 5 |
| | | | | | | $\perp$ | $\perp$ | 1 | 6 |
| | | | | | | | $\perp$ | 1 | 7 |
| | | | | | | | | $\perp$ | 8 |

FIG. 39 – Table d'analyse - cases non valides (contenant $\perp$) et cases correspondant à des transitions du treillis des catégories (contenant 1)

**Modification de l'algorithme**

Trois modifications sont apportées à l'algorithme :

– La première modification concerne la phase d'initialisation, qui consiste à remplir les cases de la table d'analyse correspondant à des transitions du treillis, autrement dit à des couples $\langle \text{mot}, \text{catégorie} \rangle$. Dans la version originale de la table d'analyse, seules les cases de la diagonale correspondaient à de telles transitions. Cette propriété n'est plus vraie dans la nouvelle définition de la table. En effet, toute case $\mathcal{T}_{i,j}$, telle qu'il existe une transition de l'état $i$ à l'état $j$ du treillis, correspond à un mot de la phrase, précisément au mot $m_{col(j)}$. L'étape d'initialisation va, par conséquent, concerner toutes ces cases. Il s'agit des cases comportant l'entier 1 dans la table de la figure 39, étant donné le treillis de la figure 38. D'autre part, lors de l'initialisation, toutes les cases de la table non valides sont marquée comme telles. Il s'agit des cases comportant le signe $\perp$ dans la figure 39.

– La seconde modification consiste à effectuer l'opération d'attachement sur des sous-analyses correspondant à des sous-treillis adjacents. La case $\mathcal{T}_{i,j}$, par exemple, sera remplie en combinant les sous-analyses de $\mathcal{T}_{i,k}$ et de $\mathcal{T}_{k,j}$, alors que dans la version précédente - qui prenait en entrée des séquences de mots - il s'agissait des cases $\mathcal{T}_{i,k}$ et $\mathcal{T}_{k+1,j}$

– La troisième modification concerne les appels à l'opération de premier attachement entre une sous-analyse vierge $X_v$ et une sous-analyse saturée

$X_s$. Comme nous l'avons vu ci-dessus, pour que l'opération aboutisse, il est nécessaire que la transition d'ancrage de $X_v$ suive le sous-treillis de $X_s$. Pour cela, on décompose l'ensemble $I_0$ des analyses vierges en sous-ensembles $I_1$, $I_2$, $\ldots I_n$. $I_i$ est le sous-ensemble de $I_0$ dont les transitions d'ancrage suivent l'état $i$.

On trouvera une version modifiée de l'algorithme dans la figure 40.

**Entrée :**

une grammaire GDG $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I} \rangle$
une phrase à analyser $m_{1,n}$
un treillis de catégories $\langle \mathcal{R}, col \rangle$ avec $\mathcal{R} = \langle Q_{\mathcal{R}}, C_{\mathcal{R}}, \delta_{\mathcal{R}}, i_{\mathcal{R}}, F_{\mathcal{R}} \rangle$

**Sortie :** une FDP$\langle \mathcal{X}, \mathcal{F} \rangle$

**Initialisation :**

$N \leftarrow$ nombre d'états de $\mathcal{R}$
$I_0 \leftarrow Init(\delta_{\mathcal{R}})$
`Pour` $i$ `allant de` $1$ `à` $N$
    `Pour` $j$ `allant de` $i$ `à` $1$
        `si` $suivant(i, j) =$ `faux alors`
            $\mathcal{T}_{i,j} \leftarrow \perp$
        `sinon si` il existe une transition entre les états $i$ et $j$ de $\mathcal{R}$ `alors`
            $\mathcal{T}_{i,j} \leftarrow Reco(I_j, m_{col(j)})$
            `tant que` de nouvelles sous-analyses sont créées `faire`
                $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Attach1(I_0, \mathcal{T}_{i,j}))$

**Remplissage de la table :**

`Pour` $i$ `allant de` $1$ `à` $N$
    `Pour` $j$ `allant de` $i - 1$ `à` $1$
        `si` $\mathcal{T}_{i,j} \neq \perp$ `alors`
            `Pour` $k$ `allant de` $i$ `à` $j - 1$
                $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Attach(\mathcal{T}_{i,k}, \mathcal{T}_{k,j}))$
                `tant que` de nouvelles sous-analyses sont créées `faire`
                $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Reco(\mathcal{T}_{i,j}, m_{col(j)}))$
                $\mathcal{T}_{i,j} \leftarrow Factor(\mathcal{T}_{i,j}, Attach1(I_j, \mathcal{T}_{i,j}))$
$\mathcal{X} = \bigcup_{i,j} \mathcal{T}_{i,j}$
$\mathcal{F} = \bigcup_{X \in \mathcal{T}_{1,N}} X$ si $q(X) \in Q_a(A(X))$

FIG. 40 – L'algorithme d'analyse d'un treillis de catégories

# 9 Recherche des meilleures analyses

Nous avons montré, dans le chapitre 7, que l'ensemble $T_G(S)$ des arbres que la grammaire $G$ associe à la phrase $S$ pouvait être représenté de façon compacte, sous la forme d'une forêt de dépendance partagée notée $\text{FDP}_G(S)$, elle-même produite par l'analyseur syntaxique défini au chapitre 8. Nous avons montré d'autre part, dans le chapitre 5, qu'une GDGP associe une probabilité à tout arbre correspondant à une phrase de son langage. Nous nous intéresserons dans ce chapitre à la recherche de la meilleure analyse d'une phrase $S$ : l'arbre de $T_G(S)$ auquel la grammaire $G$ associe la meilleure probabilité. Nous montrerons, en particulier, que la meilleure analyse de $S$ peut être construite d'une manière efficace à partir de $\text{FDP}_G(S)$. Nous commencerons par décrire, dans la section 9.1, la recherche de la meilleure analyse d'une phrase. Nous étendrons ensuite cette recherche, en 9.3, à celle des $n$ meilleures analyses d'une phrase.

Jusque là les deux étapes d'analyse syntaxique et de recherche des meilleures analyses constituent deux étapes distinctes, la seconde s'effectuant sur le produit de la première. Nous verrons en 9.4 que les deux peuvent être réalisées simultanément dans le but de limiter le nombre d'analyses construites lors de l'analyse syntaxique.

*L'algorithme de recherche des meilleures analyses dans une* FDP *a été utilisé pour les expériences sur le Penn Treebank décrites dans [Nasr & Rambow, 2004b]. Cependant, la description de l'algorithme n'a jamais été publiée.*

## 9.1 Meilleure analyse

Nous avons vu au chapitre 5 qu'une GDGP $G = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I}, \pi \rangle$ associe à toute phrase $S$ qu'elle permet de générer, une probabilité, notée $P_G(S)$, qui n'est autre que la somme des probabilités des arbres associées à $S$ par $G$. La probabilité d'un arbre a elle-même été définie, dans la section 5.4, comme le produit des probabilités des transitions qui constituent la représentation linéaire de l'arbre et de la probabilité initiale de l'automate qui en constitue la racine. On définit de plus le *meilleur arbre* de $S$, noté $\hat{T}_G(S)$, comme l'arbre dont la probabilité est la plus élevée, parmi les arbres de $T_G(S)$. La probabilité de $\hat{T}_G(S)$ est notée $\hat{P}_G(S)$. Toutes ces définitions sont reprises ci-dessous :

– Probabilité de la phrase $S$ :

$$P_G(S) \stackrel{def}{=} \sum_{T \in T_G(S)} P_G(T) \tag{13}$$

– Probabilité d'un arbre $T$ :

$$P(T = t_{1,n}) \stackrel{def}{=} \pi(Auto(t_1)) \times \prod_{i=1\ldots n} p(t_i) \tag{14}$$

– Meilleur arbre de la phrase $S$ :

$$\hat{T}_G(S) \stackrel{def}{=} \arg \max_{T \in T_G(S)} P_G(T) \tag{15}$$

– Probabilité du meilleur arbre :

$$\hat{P}_G(S) \stackrel{def}{=} \max_{T \in T_G(S)} P_G(T) \tag{16}$$

Le nombre d'arbres appartenant à $T_G(S)$ peut être une fonction exponentielle de la longueur de $S$. De ce fait, on ne peut effectuer la somme de l'équation (13) de manière naïve, en énumérant les éléments de $T_G(S)$ à l'aide de la procédure de calcul de l'extension d'une FDP, puis en effectuant la somme de leur probabilités. Pour les mêmes raisons, la recherche de $\hat{T}_G(S)$, de l'équation (15) ne peut être réalisée par un parcours de l'ensemble $T_G(S)$. Dans les trois sous-sections suivantes, nous allons montrer que la probabilité d'une phrase, la probabilité de sa meilleure analyse et la meilleure analyse elle-même peuvent être calculées de manière efficace à partir d'une FDP.

La clef de l'efficacité des calculs provient, encore une fois, du partage de structures réalisé dans une FDP. En effet, lorsqu'une sous-analyse $X$ entre dans la composition de plusieurs autres sous-analyses, alors la probabilité de $X$, ainsi que la probabilité de son meilleur arbre, ne sont calculées qu'une fois et stockées dans la sous-analyse. Pour tirer parti de la factorisation des sous-analyses, il suffit alors de montrer que la probabilité d'une sous-analyse peut être calculée à partir de la probabilité de ses composantes, et que la probabilité d'une composante peut, à son tour, être calculée à partir des probabilités de ses sous-analyses gauche et droite, et de la probabilité de sa transition. C'est ce que nous montrerons dans la sous-section 9.1.1.

On montrera de manière similaire, dans la sous-section 9.1.2, que la meilleure probabilité d'une sous-analyse peut être calculée à partir de la meilleure probabilité de ses composantes, et que la meilleure probabilité d'une composante peut être calculée à partir des meilleures probabilités de ses sous-analyses gauche et droite et de la probabilité de sa transition.

La construction du meilleur arbre d'une sous-analyse est aussi récursive. Le meilleur arbre d'une sous-analyse est le meilleur arbre d'une de ses composantes. Ce dernier est construit en combinant le meilleur arbre des sous-analyse gauche et droite de cette composante.

On retrouve, dans le calcul de la probabilité de la meilleure analyse d'une phrase, et dans la construction de cette dernière les étapes classiques de la recherche, par la programmation dynamique, d'une solution optimale d'un problème à partir des solutions optimales de ses sous-problèmes [Cormen et al., 2001]. Dans un premier temps, on définit de manière récursive la valeur de la solution optimale du problème, en fonction de la valeur des solutions optimales des sous-problèmes. On calcule ensuite de manière ascendante, à partir des sous-problèmes minimaux, les valeurs des solutions optimales des sous-problèmes, tout en les stockant dans un tableau. La solution optimale du problème est ensuite construite de manière descendante, en combinant les solutions optimales des sous-problèmes. On retrouve là les deux étapes de l'algorithme de Viterbi ([Viterbi, 1967], [Rabiner, 1989]). Les algorithmes décrits ici sont un peu plus complexes que l'algorithme de Viterbi, du fait de la structure que nous manipulons (un graphe hétérogène) est formellement plus complexe que le treillis de Viterbi (un graphe homogène). Par conséquent, les équations récursives de calcul de probabilité ne sont pas directes : la probabilité d'une sous-analyse ne se calcule pas directement à partir des probabilités d'autres sous-analyses, mais à partir des probabilités de composantes qui, elles, sont calculées à partir de probabilités de sous-analyses.

Les équations récursives de calcul de la probabilité d'une phrase et de l'arbre de meilleure probabilité sont décrites en 9.1.1 et en 9.1.2. Nous verrons, en 9.1.3, comment construire l'arbre de meilleure probabilité après le calcul de sa probabilité. Nous finirons par décrire, en 9.1.4, comment ces principes ont été mis en œuvre.

### 9.1.1 Calcul récursif de la probabilité d'une phrase

On définit la probabilité d'une sous-analyse comme la somme des probabilités des arbres de son extension :

$$P(X) \stackrel{def}{=} \sum_{T \in \mathcal{E}(X)} P(T) \tag{17}$$

En tirant profit de la définition récursive d'une sous-analyse (une sous-analyse est associée à un ensemble de composantes, elles-mêmes formées de sous-analyses), on peut calculer la probabilité d'une sous-analyse à partir des probabilités des sous-analyses qui en sont les composantes, comme l'illustrent les équations suivantes :

$$\begin{aligned}
P(X) &= \sum_{T \in \mathcal{E}(X)} P(T) && (18)\\
&= \sum_{Y \in \mathcal{Y}(X)} \left( \sum_{T \in \mathcal{E}(Y)} P(T) \right) && (19)\\
&= \sum_{Y \in \mathcal{Y}(X)} \left( \sum_{T \in \mathcal{E}(X_G(Y)) \cdot t(Y) \cdot \mathcal{E}(X_D(Y))} P(T) \right) && (20)\\
&= \sum_{Y \in \mathcal{Y}(X)} \left( \sum_{T \in \mathcal{E}(X_G(Y))} P(T) \times p(t(Y)) \times \sum_{T \in \mathcal{E}(X_D(Y))} P(T) \right) && (21)\\
&= \sum_{Y \in \mathcal{Y}(X)} P(X_G(Y)) \times p(t(Y)) \times P(X_D(Y)) && (22)
\end{aligned}$$

L'équation (18) reprend la définition de la probabilité d'une sous-analyse comme somme des probabilités des arbres composant son extension. L'équation (19) décompose cette somme en somme de termes, chacun de ces derniers correspondant à la somme des probabilités des arbres de l'extension d'une composante de $X$. En (20) la structure de chaque arbre de l'extension d'une composante est explicitée : elle correspond à la concaténation de trois éléments ($T \in \mathcal{E}(X_G(Y)) \cdot t(Y) \cdot \mathcal{E}(X_D(Y))$). Le cœur de l'établissement des équations récursives de la probabilité d'une sous-analyse est effectué lors du passage de (20) à (21), en décomposant la probabilité d'un arbre en un produit de trois probabilités :

$$P(T) = P(T_1 \cdot t \cdot T_2) = P(T_1) \times P(t) \times P(T_2)$$

conformément à la définition de la probabilité d'un arbre, et en factorisant les termes de la somme sous la forme de trois facteurs. Le passage de (21) à (22) est réalisé en remarquant que le premier et le troisième facteurs correspondent à des probabilités de sous-analyses, telles que définies dans l'équation (18).

Les équations récursives de la probabilité d'une composante et de la probabilité d'une sous-analyse sont reprises respectivement dans les équations (23) et (24) :

$$
\begin{aligned}
P(Y) &= P(X_G(Y)) \times P(t(Y)) \times P(X_D(Y)) && (23)\\
P(X) &= \begin{cases} 1 & si\ X = \perp \\ \sum_{Y \in \mathcal{Y}(X)} P(Y) & sinon \end{cases} && (24)
\end{aligned}
$$

Nous avons vu ci-dessus que la probabilité d'une sous-analyse $X$ est la somme des probabilités des arbres de son extension. La probabilité de chaque arbre est elle-même le produit des transitions traversées lors de la génération de l'arbre. Or, la probabilité d'un arbre a été définie dans la section 5.4 comme le produit des probabilités d'une séquence de transitions et de la probabilité initiale de l'automate qui constitue la racine de l'arbre. C'est cette probabilité qui n'est pas prise en compte, pour l'instant, dans le calcul de la probabilité d'une sous-analyse. Elle est prise en compte en multipliant la probabilité de chaque sous-analyse $X$ de l'ensemble $\mathcal{F}$ des sous-analyses finales par la probabilité $\pi(Auto(X))$, qui est la probabilité que l'automate de $X$ constitue une racine d'arbre. On obtient par conséquent :

$$
P_G(S) = P(\text{FDP}_G(S) = \langle \mathcal{X}, \mathcal{F} \rangle) = \sum_{X \in \mathcal{F}} \pi(A(X)) \times P(X) \qquad (25)
$$

### 9.1.2 Calcul récursif de la probabilité de la meilleure analyse d'une phrase

Le calcul de la probabilité du meilleur arbre de l'extension d'une sous-analyse reprend les étapes du calcul de la probabilité d'une sous-analyse vu ci-dessus :

$$
\begin{aligned}
\hat{P}_G(X) &\stackrel{def}{=} \max_{T \in \mathcal{E}(X)} P(T) && (26)\\
&= \max_{Y \in \mathcal{Y}(X)} \left( \max_{T \in \mathcal{E}(Y)} P(T) \right) && (27)\\
&= \max_{Y \in \mathcal{Y}(X)} \left( \max_{T \in \mathcal{E}(X_G(Y)) \cdot t(Y) \cdot \mathcal{E}(X_D(Y))} P(T) \right) && (28)\\
&= \max_{Y \in \mathcal{Y}(X)} \left( \max_{T \in \mathcal{E}(X_G(Y))} P(T) \times p(t(Y)) \times \max_{T \in \mathcal{E}(X_D(Y))} P(T) \right) && (29)\\
&= \max_{Y \in \mathcal{Y}(X)} \left( \hat{P}_G(X_G(Y)) \times p(t(Y)) \times \hat{P}_G(X_D(Y)) \right) && (30)
\end{aligned}
$$

Ici aussi, le cœur de la représentation récursive s'effectue lors du passage de (28) à (29), en décomposant le calcul du maximum en un produit du résultat du calcul de deux maximums :

$$\max_{T \,\in\, \mathcal{E}(X_G)\cdot t\cdot\mathcal{E}(X_D)} P(T) = \max_{T\in\mathcal{E}(X_G)} P(T) \times p(t) \times \max_{T\in\mathcal{E}(X_D)} P(T)$$

Les équations récursives de la meilleure probabilité d'une composante et de la meilleure probabilité d'une sous-analyse sont décrites dans les équations (31) et (32) :

$$\hat{P}(Y) = \hat{P}(X_G(Y)) \times \hat{P}(t(Y)) \times \hat{P}(X_D(Y)) \quad (31)$$

$$\hat{P}(X) = \begin{cases} 1 & si\ X = \perp \\ \max_{Y\in\mathcal{Y}(X)} \hat{P}(Y) & sinon \end{cases} \quad (32)$$

La probabilité du meilleur arbre $\hat{T}_G(S)$ que la grammaire $G$ associe à la phrase $S$, est calculée en recherchant la sous-analyse finale $X$ qui maximise le produit de $\hat{P}(X)$ et de la probabilité que sa catégorie constitue la racine d'un arbre de dépendances :

$$\hat{P}_G(S) = \hat{P}(\text{FDP}_G(S) = \langle \mathcal{X}, \mathcal{F} \rangle) = \max_{X\in\mathcal{F}}(\pi(Auto(X)) \times \hat{P}(X)) \quad (33)$$

### 9.1.3 Construction de la meilleure analyse d'une phrase

La construction du meilleur arbre d'une phrase $S$, noté $\hat{T}_G(S)$, est réalisée après le calcul de $\hat{P}_G(S)$. A l'issue de ce calcul, on détermine la sous-analyse $\hat{X}_{\mathcal{F}}$ de $\mathcal{F}$, dans l'extension de laquelle se trouve l'arbre de meilleure probabilité :

$$\hat{X}_{\mathcal{F}} \stackrel{def}{=} \arg\max_{X\in\mathcal{F}}(\pi(Auto(X)) \times \hat{P}(X)) \quad (34)$$

On a, par conséquent :

$$\hat{T}_G(S) = \hat{T}(\hat{X}_{\mathcal{F}})$$

De façon générale, on appelle *meilleure composante* d'une sous-analyse $X$, la composante de cette dernière dont l'extension contient l'arbre permettant de réaliser le maximum $\hat{P}(X)$. La meilleure composante de $X$ est notée $\hat{Y}(X)$. Elle est déterminée lors du calcul de $\hat{P}(X)$ :

$$\hat{Y}(X) = \arg \max_{Y \in \mathcal{Y}(X)} \hat{P}(Y) \tag{35}$$

L'arbre de meilleure probabilité de l'extension de la sous-analyse $X$, noté $\hat{T}(X)$, est construit à partir de $\hat{Y}(X)$ : la meilleure composante de $X$. La construction consiste à concaténer le meilleur arbre de la composante gauche de $\hat{Y}(X)$ : $\hat{T}(X_G(\hat{Y}(X)))$, avec la transition de $\hat{Y}(X)$ : $t(\hat{Y}(X))$, et avec le meilleur arbre de sa composante droite : $\hat{T}(X_D(\hat{Y}(X)))$, comme l'illustre l'équation suivante :

$$\hat{T}(X) = \hat{T}(X_G(\hat{Y}(X))) \cdot t(\hat{Y}(X)) \cdot \hat{T}(X_D(\hat{Y}(X)))$$

$\hat{T}(X_G(\hat{Y}(X)))$ et $\hat{T}(X_D(\hat{Y}(X)))$ sont eux-mêmes construits en suivant le même principe. D'où les équations de construction du meilleure arbre d'une composante et d'une sous-analyse :

$$\hat{T}(Y) = \hat{T}(X_G(Y)) \cdot t(Y) \cdot \hat{T}(X_D(Y)) \tag{36}$$

$$\hat{T}(X) = \begin{cases} \varepsilon & si\ X = \perp \\ \hat{T}(\hat{Y}(X)) & sinon \end{cases} \tag{37}$$

### 9.1.4 Mise en œuvre

Le calcul de la probabilité d'une phrase $S$, ainsi que celui de la probabilité de sa meilleure analyse, peuvent être réalisés, soit pendant l'analyse syntaxique, simultanément à la construction de la FDP de $S$, soit après l'analyse, alors que la FDP a déjà été construite. Nous décrirons ici la deuxième solution, qui est la plus efficace. En effet, lors de l'analyse de $S$, pourront être construites des sous-analyses qui n'entreront dans la composition d'aucune analyse de $S$. Il est par conséquent inutile de calculer les probabilités de ces sous-analyses. Or, ce n'est qu'à l'issue de l'analyse, comme nous l'avons vu dans le chapitre 8, que l'on sait qu'une sous-analyse entrera ou non dans une analyse complète. Ce n'est donc qu'à l'issue de l'analyse que l'on peut décider si sa probabilité doit être calculée ou non.

Les calculs nécessitent de maintenir, au niveau de chaque sous-analyse de la FDP, la probabilité de cette dernière ainsi que la probabilité de son meilleur arbre. Pour cela, on modifie légèrement la définition des sous-analyses. Une sous-analyse est maintenant définie comme un septuplet $\langle A, q, i, j, \mathcal{S}, p, \hat{p}, \hat{c} \rangle$. La définition des cinq premiers éléments est inchangée ; $p$ est la probabilité de la sous-analyse, $\hat{p}$ est la probabilité de son meilleur arbre et $\hat{c}$ est sa meilleure composante, qui sera utilisée lors de la construction du meilleur arbre.

Les calculs de la probabilité des sous-analyses et de la probabilité de leur meilleur arbre sont réalisés de manière récursive, conformément aux équations (24) et (32). Lorsqu'une sous-analyse $X$ entre dans la composition de plusieurs autres sous-analyses, les probabilités associées à $X$ ne sont calculées qu'une fois, et stockées dans les champs correspondant de cette dernière. Ainsi, lorsque ces probabilités devront être prises en compte dans le calcul d'autres sous-analyses, elles ne seront pas recalculées. De plus, simultanément au calcul de la meilleure probabilité d'une sous-analyse $X$, le champ $\hat{c}$ de $X$ est mis à jour. Il indique dans l'extension de quelle composante de $X$ se trouve $\hat{T}(X)$. La construction de la meilleure analyse est effectuée après le calcul de la probabilité de la meilleure analyse. A l'issue de ce calcul, la sous-analyse $\hat{X}$ de la FDP est déterminé, et le meilleur arbre de $\hat{X}$ est construit conformément aux équations (37) et (36).

## 9.2 Prise en compte de la probabilité de la grammaire

Les calculs de probabilité d'un arbre, d'une phrase et du meilleur arbre de cette phrase ont été définis, dans la section 9.1, pour une grammaire donnée. Rappelons que cette grammaire a été déterminée lors de la phase d'étiquetage grammatical. Or nous avons vu, dans la section 8.4, que l'analyseur pouvait prendre en entrée plusieurs solutions produites par un étiqueteur grammatical ou, en d'autres termes, plusieurs grammaires, représentées sous la forme d'un treillis de catégories[21]. Ces différentes grammaires sont « exclusives » : à l'issue de l'analyse, une seule grammaire a été utilisée pour chacune des analyses produites. Le processus de sélection de la grammaire (l'étiquetage grammatical), est lui-même un processus probabiliste, qui associe une probabilité à chaque grammaire qu'il produit. Cette probabilité n'est autre que la probabilité attribuée par la chaîne de Markov cachée, qui constitue l'étiqueteur, à chacune des solutions de l'étiquetage.

Il semble naturel d'intégrer la probabilité de la grammaire dans la recherche

[21]Il s'agit d'un abus de langage : les solutions produites par l'étiqueteur sont en réalité des séquences de catégories. La grammaire correspondante est l'ensemble constitué par les éléments de cette séquence. On continuera néanmoins à employer le terme grammaire pour désigner ces séquences de catégories.

de la meilleure analyse d'une phrase. En effet, à l'issue de l'analyse d'une phrase $S$, deux arbres $T_1$ et $T_2$ peuvent avoir été créés à l'aide de deux grammaires $G_1$ et $G_2$. On ne peut comparer les deux arbres en ne comparant que les probabilités $P_{G_1}(T_1)$ et $P_{G_2}(T_2)$, car ces dernières ne prennent pas en compte les probabilités des deux grammaires $G_1$ et $G_2$. Il est en effet possible que $P_{G_1}(T_1)$ soit supérieure à $P_{G_2}(T_2)$ mais que $P(G_1)$ soit inférieure à $P(G_1)$[22] et que la différence des probabilités des grammaires compense la différence des probabilités des arbres. C'est la raison pour laquelle les quantités que nous allons comparer sont les produits des probabilités $P_{G_1}(T_1) \times P(G_1|S)$ et $P_{G_2}(T_2) \times P(G_2|S)$. La définition de l'arbre de meilleure probabilité d'une phrase s'en trouve modifiée : il s'agit alors de l'arbre qui maximise le produit des probabilités mentionnées, comme nous allons le voir ci-dessous.

Les différentes grammaires pouvant être utilisées pour l'analyse d'une phrase sont toutes contenues dans le treillis $\mathcal{R}$ produit par l'étiqueteur grammatical. Nous allons mettre à profit la représentation que nous avons proposée du treillis sous la forme d'un automate fini, pour identifier un treillis à un langage (le langage reconnu par l'automate), et une grammaire (la séquence de catégories correspondante) à un mot. Ainsi, on dira que la grammaire $G$ appartient au treillis $\mathcal{R}$ si le mot $G$ appartient au langage reconnu par $\mathcal{R}$ ($G \in \mathcal{L}(\mathcal{R})$). De plus, à chaque grammaire $G$ du treillis $\mathcal{R}$ est associée une probabilité, notée $P_{\mathcal{R}}(G)$ (la probabilité de l'étiquetage correspondant à $G$, attribué par la chaîne de Markov cachée). Le meilleur arbre d'une phrase $S$ étant donné un treillis $\mathcal{R}$ est défini de la manière suivante :

$$\hat{T}_{\mathcal{R}}(S) \stackrel{def}{=} \arg \max_{\substack{T \in T_{G_i}(S) \\ G_i \in \mathcal{L}(\mathcal{R})}} P_{G_i}(T) \times P_{\mathcal{R}}(G_i) \tag{38}$$

L'arbre $\hat{T}_{\mathcal{R}}(S)$ est donc, parmi tous les arbres que l'on peut construire en utilisant les différentes grammaires contenues dans $\mathcal{R}$, celui qui maximise le produit de deux quantités : la probabilité de l'arbre étant donné la grammaire ($P_{G_i}(T)$) et la probabilité de la grammaire ($P_{\mathcal{R}}(G_i)$). La probabilité d'un arbre est maintenant le produit des probabilités de deux objets, générés par deux processus stochastiques différents : l'arbre produit par la grammaire probabiliste, comme précédemment, et la séquence de catégories constituant la grammaire, produite par l'étiqueteur grammatical. La recherche de $\hat{T}_{\mathcal{R}}(S)$ dans la FDP nécessite de modifier légèrement la procédure de recherche vue

[22]Les deux probabilités $P(G_1)$ et $P(G_2)$ sont en fait les probabilités conditionnelles $P(G_2|S)$ et $P(G_1|S)$, où $S$ est la phrase analysée, car les grammaires sont déterminées à partir de cette dernière.

dans la section 9.1, afin d'y intégrer la probabilité de la grammaire, comme nous allons le voir ci-dessous.

Rappelons que la procédure de construction du meilleur arbre d'une FDP $F$ s'effectue en deux étapes. Dans une première étape, la meilleure probabilité et la meilleure composante de chaque sous-analyse de $F$ sont déterminées de manière ascendante. A l'issue de cette étape, la probabilité du meilleur arbre de $F$ est connue, et le meilleur arbre est construit de manière descendante.

Nous allons modifier l'étape ascendante en déterminant, pour chaque sous-analyse $X$- en plus de sa meilleure composante et de sa meilleure probabilité - sa *meilleure grammaire*, notée $\hat{G}(X)$, qui n'est autre que le chemin du treillis correspondant au meilleur arbre de $X$ ($\hat{T}(X)$). Ce chemin peut être calculé récursivement, d'une manière très proche de celle de la construction du meilleur arbre d'une sous-analyse. La seule difficulté réside dans la nécessité de reconstruire la séquence de catégories dans le bon ordre : l'ordre du chemin correspondant dans le treillis. Il suffit pour cela de construire, pour chaque sous-analyse $X$, sa meilleure grammaire $\hat{G}(X)$ à partir de la meilleure grammaire de sa meilleure composante. Cette dernière est construite, à son tour, grâce aux meilleures grammaires de sa sous-analyse gauche et de sa sous-analyse droite. Le processus récursif s'arrête lorsque les sous-analyses vierges sont atteintes. Ce sont en effet elles qui constituent les « extrémités » de la FDP. Chaque sous-analyse vierge est liée à une transition du treillis par l'intermédiaire de sa transition d'ancrage. C'est la catégorie qui constitue l'étiquette de cette dernière qui initie le processus de construction. Ces étapes sont représentées dans les équations récursives suivantes :

$$\hat{G}(Y) = \hat{G}(X_G(Y)) \cdot \hat{G}(X_D(Y)) \tag{39}$$

$$\tag{40}$$

$$\hat{G}(X) = \begin{cases} \varepsilon & si\ X = \perp \\ cat(t_a(X)) & si\ X\ est\ une\ sous\text{-}analyse\ vierge \\ \hat{G}(\hat{Y}(X)) & sinon \end{cases} \tag{41}$$

A l'issue de l'étape ascendante, la grammaire $\hat{G}(X)$ de chaque sous-analyse $X$ de $\mathcal{F}$ a été construite et représentée sous la forme d'une séquence de catégories. La probabilité de cette séquence est la probabilité $P_{\mathcal{R}}(\hat{G}(X))$ que l'étiqueteur grammatical associe au couple $\langle \hat{G}(X), S \rangle$. Pour calculer la meilleure probabilité de $S$, il suffit de déterminer la sous-analyse de $\mathcal{F}$ qui maximise le produit des deux probabilités :

$$
\begin{aligned}
\hat{P}_G(S) &= \hat{P}(\textsc{fdp}_G(S) = \langle \mathcal{X}, \mathcal{F} \rangle) && (42)\\
&= \max_{X \in \mathcal{F}}((\pi(Auto(X)) \times \hat{P}(X)) \times P_{\mathcal{R}}(\hat{G}(X)) && (43)
\end{aligned}
$$

L'étape descendante, qui consiste à construire le meilleur arbre de la phrase, est quasiment inchangée. Seule la détermination de la meilleure sous-analyse de $\mathcal{F}$ est modifiée, afin de prendre en compte la probabilité $P_{\mathcal{R}}(\hat{G}(X))$ :

$$
\hat{X} \stackrel{def}{=} \arg\max_{X \in \mathcal{F}}(\pi(Auto(X)) \times \hat{P}(X) \times P_{\mathcal{R}}(\hat{G}(X))) \qquad (44)
$$

## 9.3 $n$ meilleures analyses

Le calcul de la probabilité de la meilleure analyse d'une phrase à partir d'une FDP et la construction de cette meilleure analyse peuvent être étendus au calcul des probabilités des $n$ meilleures analyse d'une phrase et à leur construction. Pour cela, on associe à toute sous-analyse $X$ une liste de probabilités de longueur $n$ triée par ordre décroissant, notée $\vec{\mathcal{P}}(X)$, dans laquelle sont stockées les probabilités de ses $n$ meilleures arbres. Ces derniers sont stockés dans une liste d'arbres, notée $\vec{\mathcal{T}}(X)$. Les probabilités des arbres de $\vec{\mathcal{T}}(X)$ correspondent aux probabilités de $\vec{\mathcal{P}}(X)$ : $P(\vec{\mathcal{T}}(X)[i]) = \vec{\mathcal{P}}(X)[i]$.

Dans la suite de cette section, nous allons manipuler des listes de réels ainsi que des listes de séquences de transitions. Afin d'alléger un peu les notations, nous allons étendre certaines opérations définies sur des éléments à des opérations sur des listes de tels éléments. Ainsi, si $\vec{\mathcal{P}}$ est une liste de réels et que $n$ est un réel, $n \times \vec{\mathcal{P}}$ dénotera la liste de réel construite à partir de $\vec{\mathcal{P}}$, en multipliant chacun de ses membres par $n$. De la même manière, si $\vec{\mathcal{T}}$ est une liste de séquence de transitions, et que $t_{1,n}$ est une séquence de transitions, alors $t_{1,n} \cdot \vec{\mathcal{T}}$ (respectivement $\vec{\mathcal{T}} \cdot t_{1,n}$) dénotera la liste de mots construite à partir de $\vec{\mathcal{T}}$, en préfixant (respectivement postfixant) chacun de ses membres par $t_{1,n}$. $\vec{\mathcal{T}_1} \cdot \vec{\mathcal{T}_2}$ est obtenu en préfixant chaque élément de $\vec{\mathcal{T}_2}$ par l'élément de $\vec{\mathcal{T}_1}$ se trouvant en même position ($(\vec{\mathcal{T}_1} \cdot \vec{\mathcal{T}_2})[i] = \vec{\mathcal{T}_1}[i] \cdot \vec{\mathcal{T}_2}[i]$).

### 9.3.1 Calcul des probabilités de $n$ meilleures analyses

Le calcul des probabilités des $n$ meilleurs arbres d'une phrase nécessite la définition de deux fonctions intermédiaires.

- La fonction *apparie*$(n, \vec{\mathcal{P}_1}, \vec{\mathcal{P}_2})$ prend comme paramètres un entier $n$ et deux listes de réels $\vec{\mathcal{P}_1}$ et $\vec{\mathcal{P}_2}$ triée par ordre décroissant et retourne la liste triée de réels de longueur inférieure ou égale à $n$ composée des produits des $n$ couples $\langle p_1, p_2 \rangle \in \vec{\mathcal{P}_1} \times \vec{\mathcal{P}_2}$ qui maximisent le produit $p_1 p_2$.

  Exemple : *apparie*$(3, [5, 3, 1], [4, 3, 2]) = [20, 15, 12]$

- La fonction *fusionne*$(n, \mathcal{V})$ prend comme premier argument un entier $n$ et comme deuxième argument un ensemble $\mathcal{V}$ de listes triées de réels. La fonction retourne une liste de réels de longueur inférieure ou égale à $n$ composée des $n$ plus grands réels contenus dans les listes de $\mathcal{V}$.

Exemple : *fusionne*$(3, \{[5,4,1], [12,9,2]\}) = [12,9,5]$

La liste des $n$ meilleures probabilités de la composante $Y$, notée $\vec{\mathcal{P}}(Y)$, est construite à partir des listes des $n$ meilleures probabilités de ses sous-analyses gauches $\vec{\mathcal{P}}(X_G(Y))$ et droites $\vec{\mathcal{P}}(X_D(Y))$ et de la probabilité de sa transition $p(t(Y))$. Le principe consiste à rechercher, les $n$ couples de $\vec{\mathcal{P}}(X_G(Y)) \times \vec{\mathcal{P}}(X_D(Y))$ dont le produit est maximal, recherche réalisée par la fonction *apparie* :

$$\vec{\mathcal{P}}(Y) = p(t(Y)) \times \mathit{apparie}(n, \vec{\mathcal{P}}(X_G(Y)), \vec{\mathcal{P}}(X_D(Y))) \tag{45}$$

La liste des $n$ meilleures probabilités de la sous-analyse $X$ est construite à partir des listes de ses composantes, elle consiste à rechercher parmi les listes des $n$ meilleures probabilités des composantes de $X$, les $n$ valeurs les plus élevées, recherche réalisée par la fonction *fusionne* :

$$\vec{\mathcal{P}}(X) = \mathit{fusionne}(n, \{\vec{\mathcal{P}}(Y) \mid Y \in \mathcal{Y}(X)\}) \tag{46}$$

La liste des $n$ meilleures probabilités de la FDP $\langle \mathcal{X}, \mathcal{F} \rangle$est construite à partir des listes des $n$ meilleures probabilités des sous-analyses de $\mathcal{F}$ et de la probabilité que l'automate d'une sous-analyse de $\mathcal{F}$ soit sélectionné pour démarrer une dérivation :

$$\vec{\mathcal{P}}(\langle \mathcal{X}, \mathcal{F} \rangle) = \mathit{fusionne}(n, \{\pi(A(X)) \times \vec{\mathcal{P}}(X) \mid X \in \mathcal{F}\}) \tag{47}$$

### 9.3.2 Construction des $n$ meilleures arbres

La construction des $n$ meilleurs arbres d'une sous-analyse nécessite aussi la définition de deux fonctions intermédiaires.

– La fonction *arg-apparie*$(n, \vec{\mathcal{T}}_1, \vec{\mathcal{T}}_2)$ prend en paramètres un entier $n$ et deux listes d'arbres $\vec{\mathcal{T}}_1$ et $\vec{\mathcal{T}}_2$ et retourne un couple de listes d'arbres $\langle \vec{\mathcal{T}}', \vec{\mathcal{T}}'' \rangle$ de longueur identique, inférieure ou égale à $n$. Les arbres de $\vec{\mathcal{T}}'$ proviennent de la liste $\vec{\mathcal{T}}_1$ et le arbres de $\vec{\mathcal{T}}''$ de la liste $\vec{\mathcal{T}}_2$. Les couples $\langle \vec{\mathcal{T}}'[i], \vec{\mathcal{T}}''[i] \rangle$ sont les couples $\langle T', T'' \rangle \in \vec{\mathcal{T}}_1 \times \vec{\mathcal{T}}_2$ qui maximisent le produit $P(T')P(T'')$.

– La fonction *arg-fusionne*$(n, \mathcal{V})$ prend comme premier argument un entier $n$ et comme deuxième argument un ensemble $\mathcal{V}$ de listes d'arbres. La fonction retourne une liste d'arbres de longueur inférieure ou égale à $n$ composée des $n$ meilleurs arbres contenus dans les listes de $\mathcal{V}$.

La liste des $n$ meilleurs arbres d'une composante $Y$, notée $\vec{\mathcal{T}}(Y)$, est construite à partir des listes des meilleurs arbres de ses deux sous-analyses gauche et droite et de sa transition, à l'aide de la fonction *arg-apparie* :

$$\vec{\mathcal{T}}(Y)[i] = \begin{cases} \vec{\mathcal{T}}'[i] \cdot t(Y) \cdot \vec{\mathcal{T}}''[i] \\ \text{avec } \langle \vec{\mathcal{T}}', \vec{\mathcal{T}}'' \rangle = \textit{arg-apparie}(n, \vec{\mathcal{T}}(X_G(Y)), \vec{\mathcal{T}}(X_D(Y))) \end{cases} \quad (48)$$

La liste des $n$ meilleurs arbres d'une sous-analyse $X$ est construite à partir des listes des $n$ meilleurs arbres de ses composantes :

$$\vec{\mathcal{T}}(X) = \textit{arg-fusionne}(n, \{\vec{\mathcal{T}}(Y) \mid Y \in \mathcal{Y}(X)\}) \quad (49)$$

La liste des $n$ meilleurs arbres de la FDP $\langle \mathcal{A}, \mathcal{I} \rangle$ est construite à partir des listes des $n$ meilleurs arbres des sous-analyses de $\mathcal{I}$ :

$$\vec{\mathcal{T}}(\langle \mathcal{X}, \mathcal{F} \rangle) = \textit{fusionne}(n, \{\pi(\textit{Auto}(X)) \times \vec{\mathcal{P}}(X)) \mid X \in \mathcal{F}\}) \quad (50)$$

## 9.4 Elagage de la FDP pendant l'analyse

Nous avons présenté dans la section 9.1 une méthode de construction de la meilleure analyse d'une phrase à partir d'une FDP. Dans cette approche, la recherche de la meilleure analyse est effectuée après l'analyse syntaxique. Il est possible de prendre en compte les probabilités des sous-analyses pendant l'analyse syntaxique afin d'éliminer celles dont la probabilité est assez faible pour supposer qu'elles n'apparaîtront pas dans une analyse correcte de la phrase. L'avantage de cet élagage est que le nombre de sous-analyses dans chaque case de la matrice de sous-analyses peut être maintenu arbitrairement bas. Bien entendu, plus l'élagage sera sévère, plus le risque d'éliminer, lors de l'élagage, une sous-analyse rentrant dans la composition de l'analyse correcte d'une phrase sera élevée. Cette méthode d'élagage est plus connue sous le terme anglais de *beam search*, introduit à l'origine dans les systèmes de reconnaissance de parole par [Lowerre, 1968].

L'élagage est mis en œuvre lors de l'analyse syntaxique. Après le remplissage d'une case de la matrice seul un sous-ensemble des sous-analyses de cette dernière est conservé, les autres sont éliminées. Plusieurs techniques peuvent être utilisés pour déterminer les sous-analyses à conserver, nous avons utilisé la technique du *seuil dynamique* ([Jelinek, 1998]). Etant donné l'ensemble des sous-analyses de la case $\mathcal{T}_{i,j}$, on détermine dans un premier temps la probabilité $\hat{P}$ de la meilleure sous-analyse :

$$\hat{P} = \max_{X \in \mathcal{T}_{i,j}} P(X)$$

qui permet de calculer un seuil dynamique[23] $\tau$ :

$$\tau = \frac{\hat{P}}{K}$$

où $K$ est une constante, appelée *rayon de coupure* dont la valeur est déterminée de manière empirique.

L'élagage est effectué en ne conservant dans la case $\mathcal{T}_{i,j}$ que les sous-analyses dont la probabilité est supérieure ou égale au seuil $\tau$. La mise en œuvre de l'élagage suppose que les probabilités des sous-analyses soient calculées pendant l'analyse syntaxique et non après comme nous l'avons décrit en 9.1.4.

[23] Le seuil est dit dynamique car il est différent pour les différentes cases de la matrice des sous-analyses.

## Troisième partie

# Expériences

Les quatre chapitres qui constituent cette partie décrivent le volet expérimental de notre travail. Ce dernier repose sur les implémentations que nous avons effectuées de l'algorithme d'analyse syntaxique, exposé dans le chapitre 8, et de l'algorithme de recherche des arbres les plus probables d'une FDP, exposé dans le chapitre 9.

Nous commencerons par décrire, dans le chapitre 10, le cadre expérimental dans lequel s'inscrivent les expériences. Nous y évoquerons, en particulier, l'extraction automatique des grammaires à partir d'un corpus arboré et l'étiquetage grammatical. Ces deux processus ne sont pas au cœur de notre travail, mais ils constituent des éléments cruciaux du dispositif expérimental.

Les deux chapitres 11 et 12 décrivent des expériences menées sur deux corpus différents : le corpus LE MONDE pour le premier et le Penn Treebank pour le second. Les expériences sur le corpus LE MONDE constituent, à notre connaissance, les premiers travaux d'extraction automatique de grammaires et d'analyse syntaxique réalisés à partir de ce corpus. Le chapitre 12 résumera les expériences que nous avons réalisées sur le Penn Treebank. Un bilan des expériences effectuées sera enfin présenté dans le chapitre 13.

# 10 Cadre expérimental

Les expériences décrites dans les deux chapitres 11 et 12 mettent en jeu quatre processus : l'extraction de grammaires, l'étiquetage grammatical, l'analyse syntaxique et la recherche des arbres les plus probables de la FDP produite par l'analyseur. Les deux derniers processus ont été décrits en détails dans les chapitres 8 et 9. Nous allons donner ici quelques détails sur les deux premiers.

Nous commencerons par décrire brièvement, dans la section 10.1, l'algorithme d'extraction de grammaires TIG et la construction de grammaires GDGP à partir de ces dernières. Nous donnerons ensuite, en 10.2, un bref aperçu du processus d'étiquetage grammatical. Nous terminerons, en 10.3, par l'introduction des différentes mesures d'évaluation utilisées dans les deux derniers chapitres.

## 10.1 Construction des grammaires

Les grammaires utilisées dans ces expériences ont été extraites automatiquement des deux corpus LE MONDE et Penn Treebank, d'après les techniques d'extraction décrites dans [Chen, 2002]. A l'issue de la phase d'extraction, une grammaire TIG est créée. Cette dernière est ensuite transformée, soit en une GDG canonique en utilisant l'algorithme décrit dans le chapitre 6 soit en une GDG au format positionnel (GDGP-POS) ou bigramme (GDGP-2G) en utilisant des variantes de l'algorithme de transformation. Les grammaires extraites sont ensuite enrichies de probabilités, pour donner naissance à des GDGP-POS et des GDGP-2G qui seront utilisées pour effectuer l'analyse syntaxique. Les deux étapes d'extraction de TIG et de transformation de ces dernières en GDGP sont brièvement décrites en 10.1.1 et en 10.1.2.

### 10.1.1 L'algorithme d'extraction des TIG

Notre objectif n'est pas de donner ici une description détaillée de la procédure d'extraction d'une TIG développée par John Chen. Il consiste simplement à offrir une idée générale de son fonctionnement et des connaissances linguistiques qui permettent de paramétrer le processus d'extraction et, par conséquent, les grammaires extraites. On trouvera dans [Chen, 2002] une description détaillée de ces travaux ainsi que les résultats de l'extraction sur le corpus Penn Treebank. L'adaptation de ce travail au corpus LE MONDE est décrit en détail dans [Dybro Johansen, 2004].

L'algorithme d'extraction de grammaires prend en entrée une partie du corpus arboré, appelée corpus d'apprentissage, et produit une TIG. Le corpus est traité arbre par arbre. Pour chaque arbre syntagmatique $T$, en entrée, l'algorithme produit un arbre de dérivation TIG : des arbres élémentaires ainsi que la description des opérations d'adjonction ou d'insertion qui permettent de les combiner pour produire l'arbre $T$. Les arbres élémentaires sont créés en « découpant » $T$, de manière à former des arbres élémentaires initiaux ou auxiliaires. A l'issue de la création d'un arbre élémentaire, deux situations peuvent se présenter, selon que le schéma de cet arbre a déjà été créé lors du découpage de $T$, ou d'un arbre précédent du corpus d'apprentissage, ou qu'il s'agit de la première occurrence d'un tel schéma d'arbre. Dans ce dernier cas, ce nouveau schéma est ajouté à la grammaire en cours de construction et un identifiant lui est associé. Cet identifiant constitue un *supertag* ou encore une catégorie GDG.

La taille de la grammaire, calculée en nombre de schémas d'arbres élémentaires, croît au fur et à mesure du traitement du corpus. L'évolution de la taille de la grammaire en fonction de la taille du corpus est un aspect important du processus d'extraction. En effet, l'idée même d'une grammaire (ensemble fini de règles permettant de générer un ensemble infini de structures) suppose que le processus d'extraction de la grammaire converge : à partir d'une certaine taille de corpus, la taille de la grammaire ne doit plus croître. Cette convergence n'a pas été atteinte pour les deux corpus traités dans les chapitres 11 et 12. Le résultat pratique de ce fait est que certains arbres élémentaires du corpus de test (corpus sur lequel sont effectuées les expériences d'analyse) n'existent pas dans la grammaire extraite. Ces arbres élémentaires seront appelés *arbres élémentaires inconnus*, ou *catégories inconnues* dans le cadre des GDG.

La décomposition d'un arbre syntagmatique en arbres élémentaires repose d'une part sur un principe de découpage, et d'autre part sur des paramètres linguistiques qui vont guider le processus. Ces paramètres se présentent sous deux formes. La première est une table d'identification des têtes lexicales, ou *table de percolation* dont le principe a été introduit par [Magerman, 1995]. Cette table permet d'associer une tête lexicale à chaque nœud syntagmatique d'un arbre. Cette association est réalisée en identifiant, parmi les fils d'un nœud syntagmatique $n$, un *nœud tête* $n_H$, indiquant que la tête lexicale de $n$ est la tête lexicale de $n_H$. Cette dernière est elle-même la tête lexicale du nœud tête de $n_H$, et ainsi de suite. Cette information permet de faire « remonter » les étiquettes des feuilles lexicales dans les nœuds internes de l'arbre syntagmatique. Un exemple du produit d'un tel processus est représenté dans la figure 41, extraite de [Dybro Johansen, 2004].

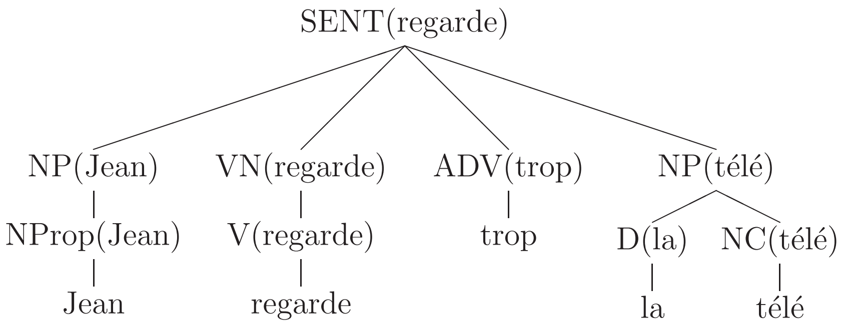



FIG. 41 – Remontée des étiquettes lexicales dans un arbre syntagmatique

L'autre information nécessaire au processus de découpage est la répartition des dépendants des mots de la phrase en actants et en circonstants. Cette distinction est cruciale pour décider si un nœud de l'arbre syntagmatique en cours de découpage donnera naissance à un arbre auxiliaire ou à un arbre initial. La situation est représentée schématiquement dans la figure 42. Quatre des cinq fils du nœud syntagmatique étiqueté $X$ de l'arbre de la partie gauche de la figure ont été étiquetés $A_i$ ou $C_i$, selon qu'ils représentent un actant ou un circonstant de la tête de $X$. Cette dernière est identifiée, grâce à la table de percolation, comme étant la tête du fils étiqueté $H$. A l'issue du processus de découpage, les fils $A_i$ donneront naissance à des nœuds d'insertion (les nœuds $A_1 \downarrow$ et $A_2 \downarrow$ de l'arbre $\alpha_1$) et à des arbres initiaux ($\alpha_2$ et $\alpha_3$) tandis que les fils $C_i$ donneront naissance à des arbres auxiliaires ($\beta_1$ et $\beta_2$).

### 10.1.2 Construction des GDGP

A l'issue du processus d'extraction d'une grammaire à partir d'un corpus $O$, un quadruplet $\langle G, O', \mathcal{T}_A, \mathcal{T}_R \rangle$ est produit.

– $G$ est la grammaire TIG, qui se présente sous la forme d'un ensemble de schémas d'arbres élémentaires, où chaque arbre élémentaire est associé à une catégorie.

– $O'$ est un appariement qui associe, à toute occurrence d'un mot du corpus $O$, l'identifiant de l'arbre élémentaire de la grammaire $G$ que cette occurrence ancre. $O'$ peut aussi être vu comme un corpus étiqueté qui associe une catégorie à chaque occurrence d'un mot.

– $\mathcal{T}_A$ est la table d'attachement, qui spécifie pour chaque arbre élémentaire $T_i$

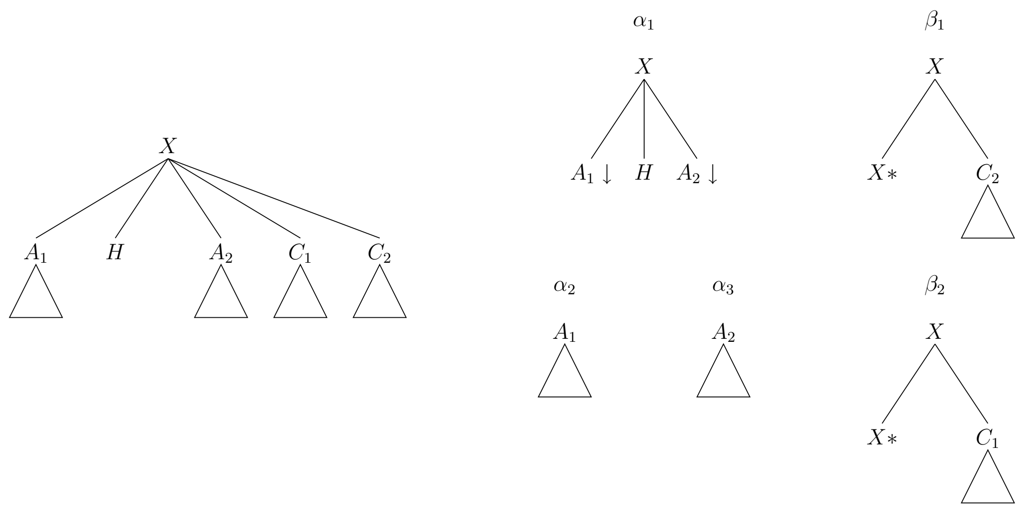



FIG. 42 – Découpage d'un arbre dérivé en arbres élémentaires

le nombre d'occurrences des différents attachements (insertion ou adjonction) observés sur les différents nœuds de $T_i$. Dans le cas d'une adjonction, sont indiquées, en plus, l'ordre de cette dernière ainsi que l'adjonction précédente sur le même nœud.

– $\mathcal{T}_R$ est la table des racines. Elle indique, pour chaque schéma d'arbre élémentaire de $G$, le nombre de fois que ce dernier constitue la racine d'un arbre de dérivation dans le corpus $O$.

La construction d'une grammaire GDGP $G' = \langle \mathcal{C}, \Sigma, \mathcal{F}, \mathcal{A}, \theta, \mathcal{I}, \pi \rangle$ à partir d'un quadruplet $\langle G, O', \mathcal{T}_A, \mathcal{T}_R \rangle$ se décompose en deux parties : une partie *algébrique* et une partie *probabiliste*.

La partie algébrique consiste à construire une GDG canonique à partir de la TIG $G$, en suivant l'algorithme du chapitre 6, ou en une GDG au format positionnel ou bigramme. Les procédures utilisées dans ces deux derniers cas sont des variantes de l'algorithme décrit dans le chapitre 6. A l'issue de cette étape, un automate de l'ensemble $\mathcal{A}$ est construit pour tout arbre élémentaire de $G$. Les noms des automates, ou encore des catégories de la GDGP, sont les mêmes que les noms des schémas d'arbres, ou *supertag*s.

La partie probabiliste consiste à estimer les paramètres d'une GDGP-POS ou d'une GDGP-2G, à partir des comptes contenus dans les tables $\mathcal{T}_A$ et $\mathcal{T}_R$. Différentes méthodes d'estimation ont été utilisées pour les différents types de paramètres. Elles sont décrites ci-dessous :

#### Probabilités initiales

Les probabilités initiales $\pi$ de $G'$ sont estimées à partir de la table des racines $\mathcal{T}_R$ au moyen d'une simple estimation par maximum de vraisemblance. Pour tout automate $A$, la probabilité $\pi(A)$ n'est rien d'autre que le nombre de fois où l'automate $A$ constitue la racine d'un arbre du corpus $O$, divisé par le nombre d'arbres composant le corpus.

#### Probabilités des transitions actancielles

Les probabilités associées aux transitions actancielles sont estimées de manière similaire pour les GDGP-POS et les GDGP-2G. Ces probabilités correspondent à la probabilité de choisir une catégorie donnée pour pourvoir un site actanciel. Elles sont calculées à partir des comptes contenus dans la table $\mathcal{T}_A$. Certaines transitions actancielles construites correspondent à des attachements qui n'ont jamais été observés dans le corpus d'apprentissage. Les comptes leur correspondant dans la table $\mathcal{T}_A$ sont par conséquent nuls. Afin d'attribuer une probabilités à ces événements, nous avons appliqué une méthode simple de lissage de probabilité, connue sous le nom de *add one smoothing*. Elle consiste à augmenter de 1 les comptes de tous les attachements. Ainsi, les événements jamais observés auront un compte de 1 et les événements observés $n$ fois auront un compte de $n+1$. Les probabilités sont ensuite calculées par maximum de vraisemblance sur ces nouveaux comptes.

Les limites de cette méthode d'estimation sont bien connues ([Gale & Church, 1994]). La principale provient du fait que la méthode peut avoir tendance à réserver une masse de probabilité trop importante aux événements non observés et à diminuer ainsi, de façon brutale, les probabilités des événements observés par rapport à leur estimation par maximum de vraisemblance. Malgré cela, nous avons conservé cette méthode d'estimation car, dans notre cas, le nombre d'événements sur lesquels répartir la masse de probabilité est modeste (par comparaison aux événements pris en compte dans l'estimation de $n$-grams en reconnaissance de la parole, par exemple). Les probabilités attribuées aux événements observés sont par conséquent peu affectées par le lissage des probabilités. Néanmoins, nous avons légèrement modifié la méthode initiale en ajoutant aux comptes une quantité $x \leq 1$. La valeur optimale de $x$ a été déterminée de manière expérimentale sur le corpus de développement.

#### Probabilités des transitions modificatrices

Les probabilités associées aux dépendances modificatrices ont été calculées différemment pour les GDGP-2G et les GDGP-POS.

Dans le cas des GDGP-2G, les probabilités des transitions modificatrices sont des probabilités conditionnelles d'une catégorie, étant donné la catégorie précédente (ces probabilités sont appelées probabilités bigrammes). La structure d'un site modificateur de GDGP-2G, repris du chapitre 5, apparaît dans la figure 43.

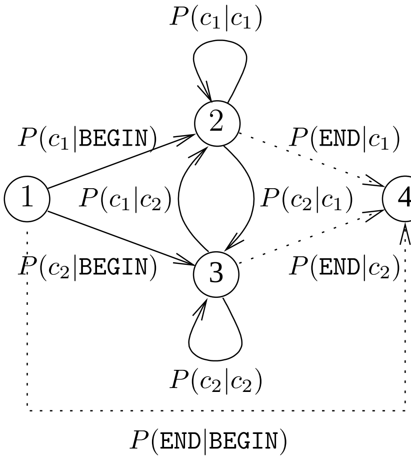



FIG. 43 – Site modifieur bigramme

Les probabilités bigrammes sont estimées à partir des comptes contenus dans la table $\mathcal{T}_A$. Pour les mêmes raisons que celles évoquées ci-dessus, l'estimation n'a pas été réalisée par maximum de vraisemblance, mais par une variante de la méthode de combinaison linéaire des probabilités bigrammes et unigrammes (la probabilité d'occurrence d'une catégorie indépendamment de la catégorie précédente), proposée par [Jelinek & Mercer, 1980]. Cette méthode permet d'estimer la probabilité conditionnelle $P(c_2|c_1)$ comme la combinaison linéaire des deux probabilités $P_{MV}(c_2|c_1)$ et $P_{MV}(c_2)$ (les probabilités estimées par maximum de vraisemblance) :

$$P(c_2|c_1) = \lambda_1 P_{MV}(c_2|c_1) + \lambda_2 P_{MV}(c_2) \text{ avec } \lambda_1 + \lambda_2 = 1 \tag{51}$$

La différence entre la méthode initiale et la notre est que, dans la première, les valeurs des coefficients de pondération sont calculés pour chaque historique. Ainsi, l'équation (51) se réécrit :

$$P(c_2|c_1) = \lambda_1(c_1)P(c_2|c_1) + \lambda_2(c_1)P(c_2) \text{ avec } \lambda_1(c_1) + \lambda_2(c_1) = 1 \qquad (52)$$

où $\lambda_1(c_1)$ et $\lambda_2(c_1)$ sont les coefficients spécifiques à l'historique $c_1$. Dans notre cas, une seule valeur a été calculée pour les coefficients $\lambda_1$ et $\lambda_2$. Ces valeurs ont été déterminées de manière expérimentale sur un corpus de développement.

Dans le cas des GDGP-2G, les probabilités des transitions modificatrices sont des probabilités conditionnelles qu'une catégorie occupe un site modificateur à une position donnée. Le nombre maximal de positions d'un site est un paramètre du modèle, dont la valeur est déterminée *a priori*. La structure d'un site modificateur à deux positions, repris du chapitre 5 apparaît dans la figure 44.

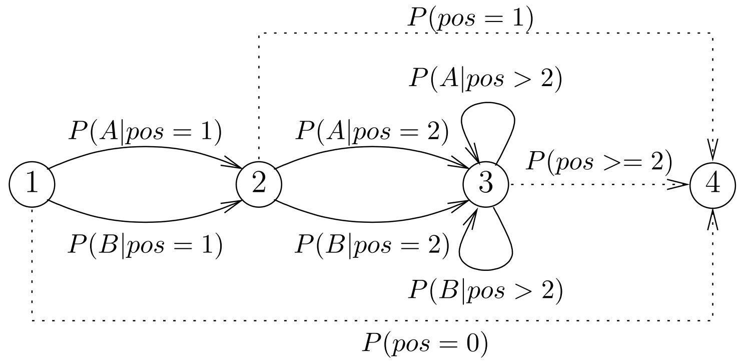



FIG. 44 – Site modifieur à deux positions

Le nombre de positions, pour un site particulier, est déterminé en fonction du nombre de positions observées dans le corpus d'apprentissage. Les probabilités conditionnelles des transitions du site sont estimées à partir des comptes de la table $\mathcal{T}_A$. L'estimation de chaque site est effectuée par la même méthode que celle des sites actanciels : la méthode du *add one smoothing*.

## 10.2 Etiquetage grammatical

L'étiquetage grammatical est réalisé à l'aide d'un étiqueteur probabiliste fondé sur un modèle de Markov caché. Nous ne décrirons pas ici en détail la problématique de l'étiquetage grammatical. Pour cela le lecteur pourra se reporter aux travaux de Srinivas Bangalore [Bangalore, 1997, Bangalore & Joshi, 1999b] ou de François Toussenel [Toussenel, 2004].

Le modèle probabiliste de l'étiqueteur repose sur deux types de paramètres. Le premier correspond aux probabilités d'émission du modèle de Markov caché qui, dans notre cas, correspondent à la probabilité d'un mot $m$ étant donné une catégorie $c$ : $P(m|c)$. Le second type de paramètres correspond aux probabilités de transition, dans notre cas, la probabilité d'une catégorie $c$ étant donné les $n$ catégories précédentes : $P(c_i|c_{i-n} \dots c_{i-1})$. Les paramètres du modèle sont appris sur un corpus d'apprentissage, qui n'est autre que le corpus $O'$, obtenu à l'issue du processus d'extraction d'une grammaire. Le processus d'étiquetage est réalisé grâce à l'algorithme de Viterbi.

L'étiquetage grammatical se distingue d'un étiquetage morpho-syntaxique classique, avant tout, par la taille de son jeu d'étiquettes, qui est, dans notre cas, de l'ordre de plusieurs milliers. Ce nombre élevé d'étiquettes provoque des problèmes sévères de manque de données, lors de l'apprentissage des paramètres du modèle. En effet, du fait de la taille du jeu d'étiquettes, certains événements présents dans le corpus sur lequel sont effectuées les expériences, appelé corpus de test, n'auront jamais été observés dans le corpus d'apprentissage. Ces événements peuvent être de quatre types différents, décrits ci-dessous. Les deux premiers sont propres à l'étiquetage grammatical, tandis que les deux derniers se retrouvent dans l'étiquetage morpho-syntaxique classique.

– Certaines catégories du corpus de test n'ont jamais été observées dans le corpus d'apprentissage. Nous verrons en effet, dans les chapitres 11 et 12, que le nombre de catégories créées à l'issue du processus d'extraction de la grammaire ne s'est pas stabilisé. Par conséquent, les catégories de certains mots du corpus de test n'ont tout simplement pas été créés lors de la création de la grammaire. Il s'agit d'une situation inédite par rapport à l'étiquetage morpho-syntaxique classique, où le jeu d'étiquettes est connu à l'avance. Ce problème n'est pas résolu à l'heure actuelle. Par conséquent, dans ce cas, toute occurrence d'un mot qui devrait être étiqueté par une telle catégorie sera systématiquement mal étiqueté.

– Certains couples $\langle m, c \rangle$ du corpus de test n'ont pas été observés dans le corpus d'apprentissage. La probabilité d'émission $P(m|c)$, estimée par maximum de vraisemblance, est par conséquent nulle. Là aussi, aucune solution n'a été mise en œuvre pour résoudre le problème associé à ce type d'événement.

– Certains mots du corpus de test n'ont jamais été observés dans le corpus d'apprentissage, menant là aussi à des probabilités d'émission nulles. Il s'agit du problème classique des mots inconnus en étiquetage morpho-syntaxique ([Weischedel et al., 1993]). Il a été résolu, dans notre cas, en modifiant l'estimation des probabilités d'émission. Cette dernière suppose une distribution standard de la probabilité qu'un mot inconnu appartienne à une des catégories.

– Certains $n$-grams du corpus de test n'ont jamais été observés dans le corpus d'apprentissage. Il s'agit là aussi d'un problème classique d'estimation de paramètres dans le cadre des modèles de Markov cachés, pour lequel plusieurs méthodes d'estimations ont été développées. La technique utilisée dans les expériences décrites ici est celle du repli sur des $n$-grams d'ordre inférieur, proposées par [Katz, 1987].

## 10.3 Mesures d'évaluation des résultats

Les expériences décrites dans le reste de cette partie mettent en jeu trois processus : l'extraction de la grammaire, l'étiquetage grammatical et l'analyse. Des mesures différentes sont utilisées pour évaluer chacun d'entre eux.

### 10.3.1 Grammaire

Une grammaire extraite, $G$, est évaluée par sa *taille*, sa *couverture* et son *ambiguïté moyenne*.

La taille de $G$ n'est autre que le nombre de catégories, ou encore d'automates, qu'elle comprend.

La couverture de $G$, étant donné un corpus $O$, est le rapport du nombre d'occurrences de catégories inconnues dans $O$ par la taille de $O$ (le nombre total d'occurrences de mots dans $O$).

L'ambiguïté moyenne de $G$ est, comme son nom l'indique, la moyenne du nombre d'analyses différentes que $G$ associe aux phrases de $O$.

### 10.3.2 Etiquetage

Le résultat de la tâche d'étiquetage d'un corpus $O$ est évaluée, de manière classique, par la *précision par mot*. Il s'agit du rapport du nombre d'occurrences de $O$ correctement étiquetées par le nombre total d'occurrences de $O$.

Lorsque l'étiqueteur produit plusieurs solutions, on appelle *précision maximale* de cet ensemble, la précision de la solution la plus proche de la référence.

### 10.3.3 Analyse

Le résultat de l'analyse syntaxique est évalué par deux mesures : la *précision par arbre* (ou A-précision) et la *précision par dépendance* (ou D-précision). Etant donné un corpus $O$ et une grammaire $G$, la précision par arbre est le nombre de phrases de $O$ pour lesquelles l'arbre correct a été construit par l'analyseur, et extrait de la FDP, divisé par le nombre de phrases de $O$. Lorsque plusieurs arbres sont extraits de la FDP, la A-précision mesure la présence de l'arbre correct parmi les arbres produits.

La A-précision est une mesure assez grossière, car elle ne permet pas de savoir si un résultat est partiellement correct. Des mesures plus précises des résultats d'une analyse syntaxique ont été définies pour les arbres syntagmatiques. C'est le cas en particulier des mesures des campagnes d'évaluation PARSEVAL ([Black et al., 1991]). Ces dernières distinguent *précision*, *rappel* et *croisement*. Etant donné un arbre syntagmatique d'une phrase produit par l'analyseur, que l'on appellera arbre *hypothèse*, et l'arbre correct de cette phrase, appelé arbre *référence*, les trois mesures vont comparer les syntagmes de l'hypothèse et de la référence. Un syntagme est défini par son étiquette et par son extension : les mots qu'il regroupe. La précision d'une hypothèse est définie comme le nombre de syntagmes de l'hypothèse présentes dans la référence. Le rappel est le nombre de syntagmes de la référence présents dans l'hypothèse. Le croisement est le nombre de syntagmes de l'hypothèse qui croisent des syntagmes de la référence ($[_H\ [_R\ ]_H\ ]_R$).

La comparaison d'arbres de dépendances est plus simple que la comparaison d'arbres syntagmatiques, du fait que tous les arbres de dépendances d'une même phrase comportent le même nombre de dépendances ($n-1$ dépendances pour une phrase composée de $n$ mots). Etant donné un arbre de dépendances hypothèse et un arbre de dépendances référence, on définit la D-précision de l'hypothèse comme le rapport du nombre de dépendances de l'hypothèse présentes dans la référence sur le nombre total de dépendances de l'hypothèse (ou de la référence, les deux étant égaux).

Lorsque plusieurs hypothèses sont extraites de la FDP, on définit la D-précision de l'ensemble des hypothèses comme la D-précision de la meilleure hypothèse.

Les analyses partielles sont aussi évaluées grâce à la D-précision. Rappelons qu'une analyse partielle d'une phrase $S$ de longueur $n$ est une collection de $k$ arbres de dépendances, où chaque arbre correspond à un segment de $S$. L'ensemble des arbres de la collection définit un ensemble de dépendances de cardinalité $n - k - 1$. On calcule la D-précision de cet ensemble de la même manière que l'on a calculé la D-précision d'une analyse complète : le nombre de dépendances de l'analyse partielle présentes dans la référence, divisé par le nombre total de dépendances de la référence. Ainsi, une absence de dépendance dans une analyse partielle compte comme une dépendance erronée.

# 11 Expériences sur le corpus LE MONDE

Les expériences décrites dans ce chapitre ont été réalisées sur le corpus LE MONDE. Les grammaires ont été extraites par Ane Dybro Johansen, à partir de la méthode proposée par [Chen, 2002]. L'étiquetage grammatical a été réalisé par François Toussenel, à l'aide d'un étiqueteur fondé sur les chaînes de Markov cachées.

Nous donnerons, dans la section 11.1, quelques détails sur le corpus utilisé. Les grammaires extraites du corpus seront décrites dans la section 11.2. Deux séries d'expériences d'analyse seront décrites en 11.3 et en 11.4. La première consiste à fournir, en entrée de l'analyseur, des phrases dont les mots ont été étiquetés par la catégorie correcte. Dans la seconde, l'entrée de l'analyseur est constitué par un treillis de catégories produit par l'étiqueteur.

## 11.1 Le corpus

Les expériences ont été effectuées sur la partie du corpus LE MONDE étiquetée syntaxiquement [Abeillé et al., 2003]. Le corpus a été divisé en trois parties : un corpus d'apprentissage (90% du corpus complet), un corpus de test (5%) et un corpus de développement. Le corpus d'apprentissage a servi à l'extraction des grammaires, des tables d'attachement, des tables de racines ainsi qu'à la constitution d'un corpus étiqueté par les catégories définies par les différentes grammaires. C'est sur ce dernier qu'ont été estimés les paramètres d'un étiqueteur grammatical. Le corpus de développement a servi à la mise au point des grammaires, ainsi qu'à l'estimation des coefficients de pondération des probabilités des GDGP-2G. Le corpus de test a été utilisé pour évaluer les performances de l'analyseur et de l'étiqueteur grammatical. Les tailles des trois corpus sont reproduites dans le tableau 1.

| | Apprentissage | Test | Développement |
|---|---|---|---|
| Nombre de phrases | 13 716 | 773 | 865 |
| Nombre de mots | 358 545 | 19 308 | 21 401 |

TAB. 1 – Tailles des corpus d'entraînement, de test et de développement

La longueur moyenne des phrases du corpus d'apprentissage est de $26,1$ mots. La répartition des phrases selon leur longueur est représentée dans la figure 45.

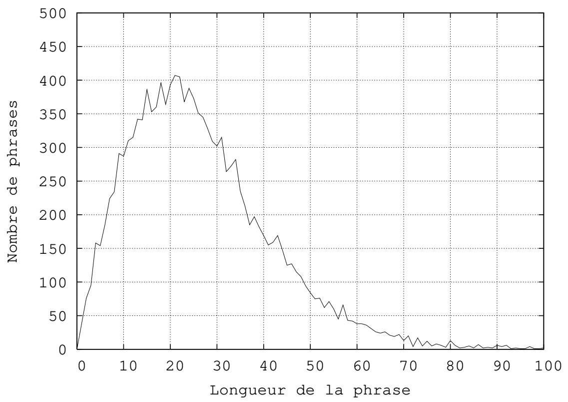



FIG. 45 – Répartition du nombre de phrases du corpus d'apprentissage selon leur longueur

## 11.2 Les grammaires

Comme nous l'avons évoqué dans le chapitre 10, le processus d'extraction de grammaires, proposé par [Chen, 2002], nécessite de distinguer, parmi les dépendants d'un mot, ses actants de ses circonstants. Cette distinction est réalisée dans le Penn Treebank (le corpus dont s'est servi [Chen, 2002]) en utilisant diverses informations, en particulier les étiquettes sémantiques présentes dans le corpus. L'absence d'étiquettes sémantiques, et de rôles fonctionnels, dans le corpus LE MONDE, rend plus difficile la distinction actant/circonstant, en particulier pour les groupes prépositionnels. [Dybro Johansen, 2004] propose trois heuristiques pour contourner le problème. La première, notée $H_1$, revient à considérer que tous les groupes prépositionnels constituent des circonstants. $H_3$ considère, quant à elle, que tous les groupes prépositionnels sont des actants. L'heuristique $H_2$ constitue une position intermédiaire entre $H_1$ et $H_3$. Elle repose sur une liste de 14 prépositions établie manuellement dont on considère qu'elles introduisent des actants. Les 103 autres prépositions (principalement des locutions prépositionnelles) introduisent des circonstants.

Ces trois heuristiques permettent de générer trois grammaires TIG que nous appellerons respectivement $G_1$, $G_2$ et $G_3$. Ces dernières se distinguent principalement par le nombre de catégories qu'elles définissent. $G_3$ définit, naturellement, le plus de catégories. En effet, en TAG (et en TIG), les actants sont intégrés dans les arbres élémentaires de leur gouverneur. Par conséquent, plus une grammaire définit d'actants, plus elle produira de schémas d'arbres élémentaires différents.

La distinction actant/circonstant exerce aussi une influence sur l'ambiguïté de la grammaire : en définissant plus d'actants parmi les groupes prépositionnels, une grammaire réduit les cas d'ambiguïté de rattachements prépositionnels. Dans le cas de $G_3$, par exemple, qui traite tous les groupes prépositionnels comme actants, une catégorie prévoit exactement le nombre de prépositions qu'elle doit régir. A l'inverse, pour $G_1$, les parties du discours *nom*, *verbe*, *adjectif* et *adverbe* peuvent gouverner un nombre quelconque de prépositions, augmentant ainsi la combinatoire des rattachements prépositionnels. Le nombre de catégories défini par chaque grammaire, ainsi que leur couverture, leur ambiguïté moyenne et le temps moyen d'analyse par phrase sur le corpus de test, ont été représentés dans le tableau 2.

| Grammaire | Taille | Couverture | Ambiguïté moyenne | Temps d'analyse moyen |
|---|---|---|---|---|
| $G_1$ | 3 808 | $0,993$ | 11 003 | $2,22$ ms |
| $G_2$ | 5 910 | $0,988$ | 750 | $1,39$ ms |
| $G_3$ | 7 123 | $0,985$ | 346 | $0,13$ ms |

TAB. 2 – Taille des trois grammaires $G_1$, $G_2$ et $G_3$ ainsi que leur couverture, leur ambiguïté moyenne et le temps moyen d'analyse par phrase sur le corpus de test

Les résultats du tableau 2 confirment et quantifient les prédictions théoriques. $G_3$ définit approximativement deux fois plus de catégories que $G_1$, et $G_2$ se situe entre les deux. De plus, la taille de la grammaire influence directement l'ambiguïté moyenne et le temps d'analyse. La comparaison de l'ambiguïté moyenne pour $G_1$ et pour $G_3$ permet en outre de quantifier la part d'ambiguïté provenant des rattachements prépositionnels. En effet, ces deux grammaires ne se distinguent que par leur traitement des rattachements prépositionnels[24], qui ne sont jamais ambigus pour $G_3$. La différence de l'ambiguïté moyenne pour $G_1$ et $G_3$ est très largement dûe aux rattachements prépositionnels. Parmi les 11 003 analyses construites en moyenne par phrase pour $G_1$, à peu près 10 600 proviennent de rattachements prépositionnels, soit une proportion de l'ordre de 95 %.

[24]Ceci n'est pas tout à fait exact. La procédure d'extraction décrite dans [Dybro Johansen, 2004] ne permet pas d'adjoindre un circonstant entre deux actants d'un même gouverneur. Par conséquent, dans la grammaire $G_3$, les circonstants situés entre deux groupes prépositionnels - traités comme actants du fait de l'heuristique $H_3$ - sont aussi traités comme des actants. Ce phénomène est néanmoins marginal et nous le négligerons ici.

Le tableau 2 permet aussi d'observer que les trois heuristiques exercent une faible influence sur les couvertures des grammaires qu'elles définissent : le rapport des tailles des grammaires $G_1$ et $G_3$ vaut $0,53$ alors que le rapport de leur couverture n'est que de $1,008$. Un accroissement considérable du nombre de catégories ne diminue donc que très faiblement la couverture des grammaires. Cette différence de comportement s'explique aisément à la lumière de la figure 46, empruntée à [Dybro Johansen, 2004]. Les trois graphiques de la figure retracent l'évolution du nombre de catégories définies lors de l'extraction des trois grammaires. Dans chaque graphique, les trois courbes indiquent les catégories observées au moins une fois, au moins deux fois et au moins trois fois, à différentes étapes de l'extraction. On observe qu'à la fin du processus, 63% des catégories de $G_3$ n'ont été observées qu'une fois (phénomène habituellement désigné par *hapax legomena* (« lu une seule fois », en grec)), 54% dans le cas de $G_1$. Ces chiffres montrent qu'un nombre important de catégories a une probabilité d'occurrence médiocre. Par conséquent, leur influence sur la couverture de la grammaire est limitée, expliquant la faible influence du nombre de catégories sur la couverture de la grammaire.

Les courbes de la figure 46 montrent aussi que la croissance du nombre de catégories n'est pas stabilisée à l'issue de l'extraction, ce qui semble indiquer que la taille du corpus d'apprentissage n'est pas suffisant pour atteindre la convergence des grammaires. Cette conclusion doit néanmoins être modérée, en observant que le nombre de catégories apparaissant au moins deux fois dans le corpus d'apprentissage est quasiment stabilisé à l'issue de l'extraction. Seules les courbes retraçant l'évolution des *hapax* continuent de croître à l'issue du processus d'extraction. Malgré son influence limitée sur la couverture de la grammaire, l'évolution de la courbe des *hapax* est un phénomène troublant. Il semble s'expliquer, en partie, par des erreurs dans le corpus d'apprentissage (une erreur dans un arbre syntaxique du corpus d'apprentissage peut provoquer la création d'un arbre élémentaire aberrant dont la probabilité d'être synthétisé une deuxième fois est quasi-nulle). Ce phénomène n'a pas été quantifié pour l'instant. Les autres *hapax* modélisent des phénomènes rares, susceptibles de se répéter dans un corpus plus important.

L'ambiguïté et le temps d'analyse étant fortement dépendants de la longueur des phrases, nous avons représenté, dans la figure 47, les ambiguïtés et temps d'analyse moyens et maximaux pour les phrases de longueurs identiques. Ces résultats ont été calculés sur le corpus d'apprentissage, afin d'obtenir des chiffres plus significatifs. Les graphiques de la figure 47 permettent d'observer que le nombre d'analyses évolue de manière exponentielle par rapport à la longueur des phrases.

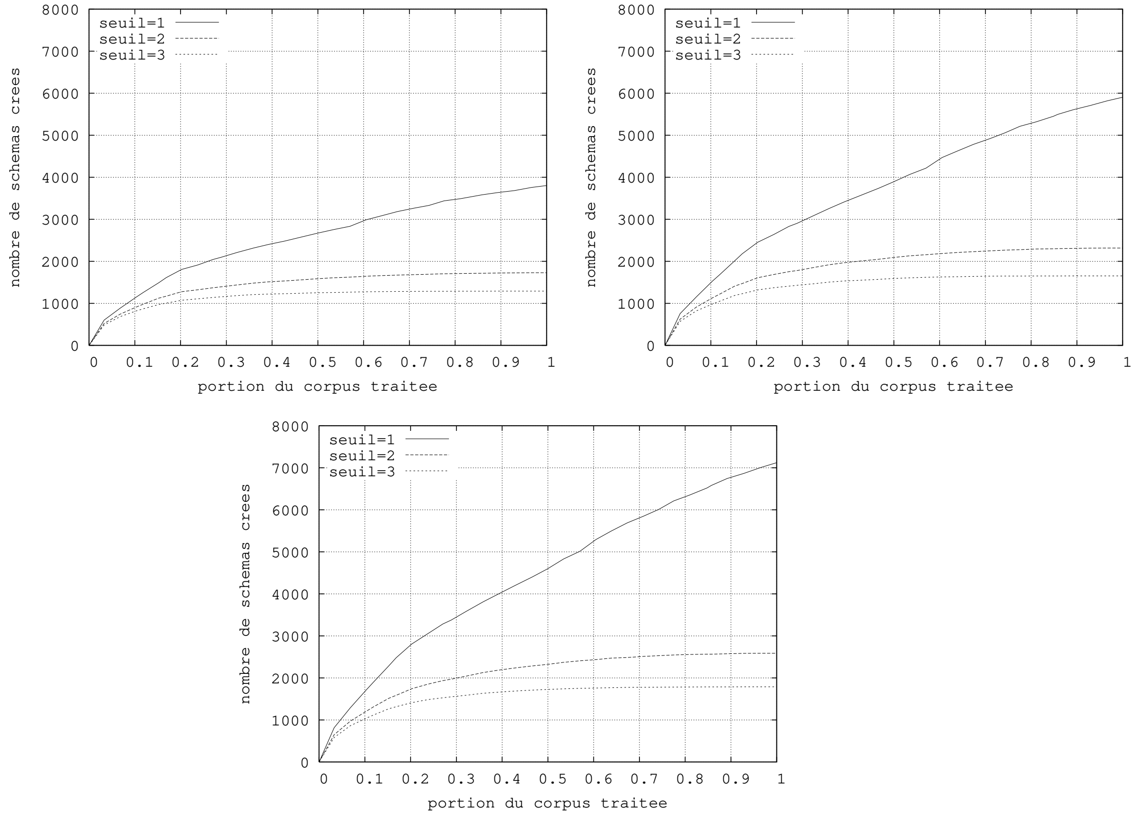



FIG. 46 – Croissance de la taille des trois grammaires $G_1$ (en haut à gauche), $G_2$ (en haut à droite) et $G_3$ (en bas) en fonction de la fraction traitée du corpus d'apprentissage. Les trois courbes de chaque graphique représentent les schémas d'arbres observés au moins une fois (seuil = 1), au moins deux fois (seuil = 2) et au moins trois fois (seuil = 3).

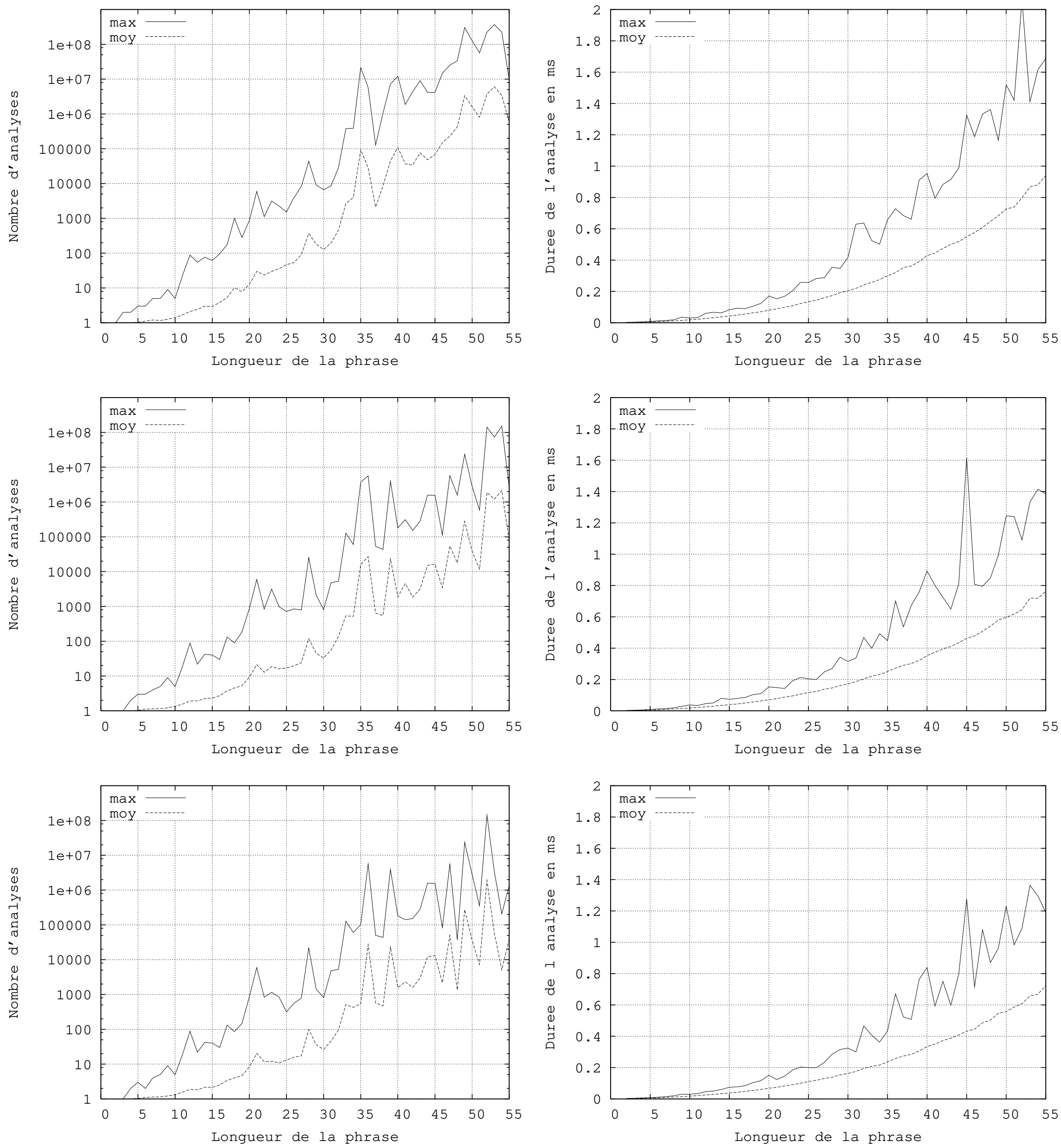



FIG. 47 – Ambiguïté et temps d'analyse, moyens et maximaux, selon la longueur des phrases, pour les trois grammaires $G_1$, $G_2$ et $G_3$ (de haut en bas)

## 11.3 Performances de l'analyseur étant donné un étiquetage correct

Les expériences décrites dans cette section consistent à analyser les phrases du corpus de test dans lesquelles chaque occurrence de mot a été étiquetée par sa catégorie correcte. Ce corpus constitue, en quelque sorte, le résultat d'un étiqueteur grammatical parfait, ne commettant aucune erreur d'étiquetage. Le corpus sur lequel ont été menées les expériences ne correspond en fait pas exactement au corpus de test, mais à une partie de ce dernier : la partie constituée des phrases ne contenant pas de catégories inconnues. Rappelons que certaines phrases du corpus de test contiennent des mots dont la catégorie correcte n'a pas été créée lors de l'extraction de la grammaire. Ces phrases ont été éliminées du corpus de test pour les expériences de cette section. La taille du corpus de test amputé de ces phrases est par conséquent inférieure à celle du corpus de test original. Nous l'appellerons taille réelle du corpus. Cette taille est différente pour les trois grammaires, car chacune possède une couverture différente.

Deux séries d'expériences ont été réalisées. La première repose sur une GDGP-2G et la seconde sur une GDGP-POS. Le but de ces expériences est d'évaluer les meilleurs résultats que l'on peut obtenir étant donné les trois grammaires $G_1$, $G_2$ et $G_3$.

### 11.3.1 Le modèle bigramme (GDGP-2G)

Dans cette section, $G_1$, $G_2$ et $G_3$ désignent les GDGP-2G extraites à l'aide des heuristiques $H_1$, $H_2$ et $H_3$. La A-précision et la D-précision calculées sur le corpus de test ont été représentées pour les trois grammaires dans le tableau 3. Ces chiffres ont été calculés sur l'arbre de probabilité maximale de chaque phrase. Les expériences, conformément à l'intuition, montrent que les performances de la grammaire $G_3$ sont supérieures à celles de $G_2$, elles-mêmes supérieures à celle de $G_1$. Ce résultat peut s'expliquer par le fait que la tâche d'analyse est plus facile pour $G_3$ qu'elle ne l'est pour $G_2$ et pour $G_1$. En effet, comme nous l'avons observé ci-dessus, le nombre d'analyses différentes par phrase varie en fonction des grammaires. La grammaire $G_3$ produit en moyenne moins d'analyses que $G_2$ et que $G_1$. La tâche consistant à choisir l'analyse correcte, ou une analyse qui s'en rapproche, est par conséquent plus facile pour $G_3$ qu'elle ne l'est pour $G_2$ et pour $G_1$.

Les résultats du tableau 3 permettent de comparer entre elles les trois grammaires. Mais ils sont difficiles à évaluer dans l'absolu, sans avoir de point

| Grammaire | Taille réelle du corpus de test | A-précision | D-précision |
|---|---|---|---|
| $G_1$ | 15 161 mots | $0,606$ | $0,964$ |
| $G_2$ | 12 437 mots | $0,682$ | $0,971$ |
| $G_3$ | 11 498 mots | $0,722$ | $0,976$ |

TAB. 3 – A-précision et D-précision des trois grammaires sur le corpus de test, dont la taille réelle dépend de la grammaire - Modèle bigramme

de repère permettant d'estimer la difficulté de la tâche consistant à réaliser l'analyse syntaxique de phrases correctement étiquetées. Nous avons créé un tel point de repère en construisant, pour chaque grammaire $G_i$, un modèle aléatoire $B_i$. Comme son nom l'indique, ce dernier a été construit en attribuant aux différents paramètres du modèle (probabilités des transitions des automates et probabilités des racines) des valeurs aléatoires. Les D-précisions obtenues par les trois grammaires $B_1$, $B_2$ et $B_3$ sur les corpus de test sont respectivement égales à $0,934$, $0,951$ et $0,957$.

Ces résultats appellent plusieurs remarques. La première est leur niveau élevé. Il montre que, dans un système où l'analyse syntaxique est réalisée à la suite d'un étiquetage grammatical, l'essentiel des choix a été réalisé lors de l'étiquetage, du moins pour les trois grammaires $G_1$, $G_2$ et $G_3$. On observe, en effet, qu'en choisissant la catégorie des mots d'une phrase et, par conséquent, la grammaire utilisée lors de l'analyse syntaxique, l'ambiguïté résiduelle est suffisamment limitée pour qu'un modèle aléatoire effectue les bons rattachements, dans 95% des cas. Cette conclusion permet de quantifier la prédiction de S.Bangalore et A.Joshi : « supertagging is almost parsing ». D'autre part, on peut observer que les gains apportés par les modèles probabilistes sont médiocres : les résultats de $G_1$ sont supérieurs à ceux de $B_1$ de $3,2\%$. La différence est de 2% pour le couple $G_2$, $B_2$ et de $1,9\%$ pour le couple $G_3$, $B_3$.

Nous n'avons pas, pour l'instant, d'analyse précise de la cause des erreurs produites par l'analyseur ni de la possibilité d'influer sur les résultats en améliorant le modèle probabiliste. Nous avons néanmoins identifié les erreurs les plus courantes commises par les trois grammaires sur le corpus de développement. Pour cela, nous avons extrait les 10 catégories morphosyntaxiques qui ont été le plus fréquemment « mal rattachées » : celles dont les gouverneurs n'ont pas été correctement identifiés dans la phrase. Le résultat apparaît dans la figure 4. Nous ne sommes pas en mesure, pour le moment, d'expliquer les causes des différents types d'erreurs. Ces causes sont d'ordres divers. Certaines proviennent des choix syntaxiques effectués pour l'étiquetage du corpus. D'autres proviennent de la procédure d'extraction des grammaires et des grammaires extraites. D'autres encore proviennent des erreurs de rattachements effectués par l'analyseur. Nous nous bornerons, par conséquent, à quelques commentaires.

| $G_1$ | | $G_2$ | | $G_3$ | |
|---|---|---|---|---|---|
| Dep | % | Dep | % | Dep | % |
| `PONCTW` | $47,12$ | `PONCTW` | $47,77$ | `PONCTW` | $44,97$ |
| `PREP` | $18,12$ | `V-FIN` | $12,14$ | `V-FIN` | $13,06$ |
| `V-FIN` | $8,45$ | `PREP` | $8,91$ | `CONJ-C` | $10,06$ |
| `ADV` | $4,83$ | `CONJ-C` | $6,88$ | `V-INF` | $7,1$ |
| `V-INF` | $4,83$ | `ADV` | $5,26$ | `ADV` | $7,1$ |
| `PART-PAS` | $4,22$ | `V-INF` | $4,85$ | `PART-PAS` | $4,73$ |
| `PAR-PRE` | $3,62$ | `PAR-PRE` | $3,64$ | `PAR-PRE` | $2,95$ |
| `CONJ-C` | $3,62$ | `PART-PAS` | $2,43$ | `CONJ-S` | $2,36$ |
| `CONJ-S` | $1,51$ | `CONJ-S` | $2,02$ | `ADJ` | $2,36$ |
| `NOM-COM` | $1,21$ | `ADJ` | $1,62$ | `PREP` | $1,77$ |

TAB. 4 – Catégories morpho-syntaxiques les plus fréquemment « mal rattachées »

On observe qu'à peu près la moitié des erreurs provient des signes de ponctuation dite *faible* (`PONCTW`). Cette catégorie regroupe principalement les *virgules*, *guillemets*, *parenthèses*, *tirets* et *deux-points*. Parmi ces signes, seules les virgules présentent un réel intérêt. C'est pourquoi, elles seules ont été prises en compte pour l'analyse d'erreur. Le second type d'erreur le plus fréquent, pour $G_1$, est celui des rattachements prépositionnels. Comme on pouvait s'y attendre, la part des rattachements prépositionnels dans les erreurs est plus importante pour $G_1$ qu'elle ne l'est pour $G_2$ et pour $G_3$.

### Evolution de la précision en augmentant le nombre d'analyses produites

La figure 48 représente l'évolution des A-précision et D-précision des trois grammaires $G_1$, $G_2$ et $G_3$ calculées sur les $n$ solutions les plus probables produites par l'analyseur. On observe, sans surprise, que la précision augmente avec le nombre de solutions. Les courbes $B_i$ restent en dessous des courbes $G_i$, mais la distance entre les deux diminue en augmentant le nombre de solutions. La croissance des courbes $B_i$ s'explique par le fait que, pour une valeur donnée, $x$, du nombre de solutions prises en compte, toutes les phrases possédant un nombre d'analyses inférieur ou égal à $x$ auront obligatoirement leur bonne analyse dans l'ensemble des $x$ meilleures analyses. Par conséquent, pour ces phrases, le modèle utilisé n'a aucune importance. Au fur et à mesure que l'on augmente le nombre de solutions, la proportion de ces phrases augmente, expliquant ainsi le rapprochement des deux courbes, qui finiront par se rejoindre lorsque tous les arbres auront été extraits de la FDP.

Les courbes de la figure 48 appellent une autre question : celle de savoir si l'analyse correcte a été produite pour toutes les phrases du corpus. On observe, en effet, qu'en prenant en compte les 200 meilleures analyses, la proportion de phrases pour lesquelles l'analyse correcte a été trouvée est inférieure à 1. Elle est égale à $0,856$, $0,869$ et $0,894$, pour les trois grammaires $G_1$, $G_2$ et $G_3$. En augmentant le nombre d'arbres extraits, ces chiffres augmentent, mais restent inférieur à 1. Nous ne savons pas si, pour les autres phrases, la bonne analyse a effectivement été construite mais ne figure pas dans l'ensemble des arbres extraits. Nous avons néanmoins vérifié manuellement que toutes les phrases pour lesquelles l'analyse correcte n'appartient pas à l'ensemble des 10 000 premières analyses possèdent plus de 10 000 analyses différentes, laissant ouverte la possibilité que l'analyse correcte se trouve parmi les analyses non prises en compte.

### 11.3.2 Le modèle positionnel (GDGP-POS)

Dans cette section, $G_1$, $G_2$ et $G_3$ désignent les GDGP-POS extraites à l'aide des heuristiques $H_1$, $H_2$ et $H_3$. La A-précision et la D-précision calculées sur le corpus de test ont été représentées pour les trois grammaires dans le tableau 3. Comme précédemment, ces chiffres ont été calculés sur l'arbre de probabilité maximale de chaque phrase. Les performances sont inférieures à celles obtenues par les GDGP-2G. Ces résultats semblent montrer que la connaissance de la nature des attachements précédents, dans le cas de rattachements multiples à un même site, est une information plus pertinente que la position de l'attachement. On observe aussi que le passage du modèle à zéro position (équivalent au modèle GDGP-INIT) à un modèle à une position provoque une modeste amélioration des résultats. Cette amélioration ne se produit plus lors du passage d'un modèle à une position à un modèle distinguant deux positions, sauf pour la grammaire $G_3$, pour laquelle une légère amélioration est observée.

Les modèles bigrammes semblent en définitive plus performants que les modèles positionnels. C'est sur les premiers que le reste des expériences va être mené.

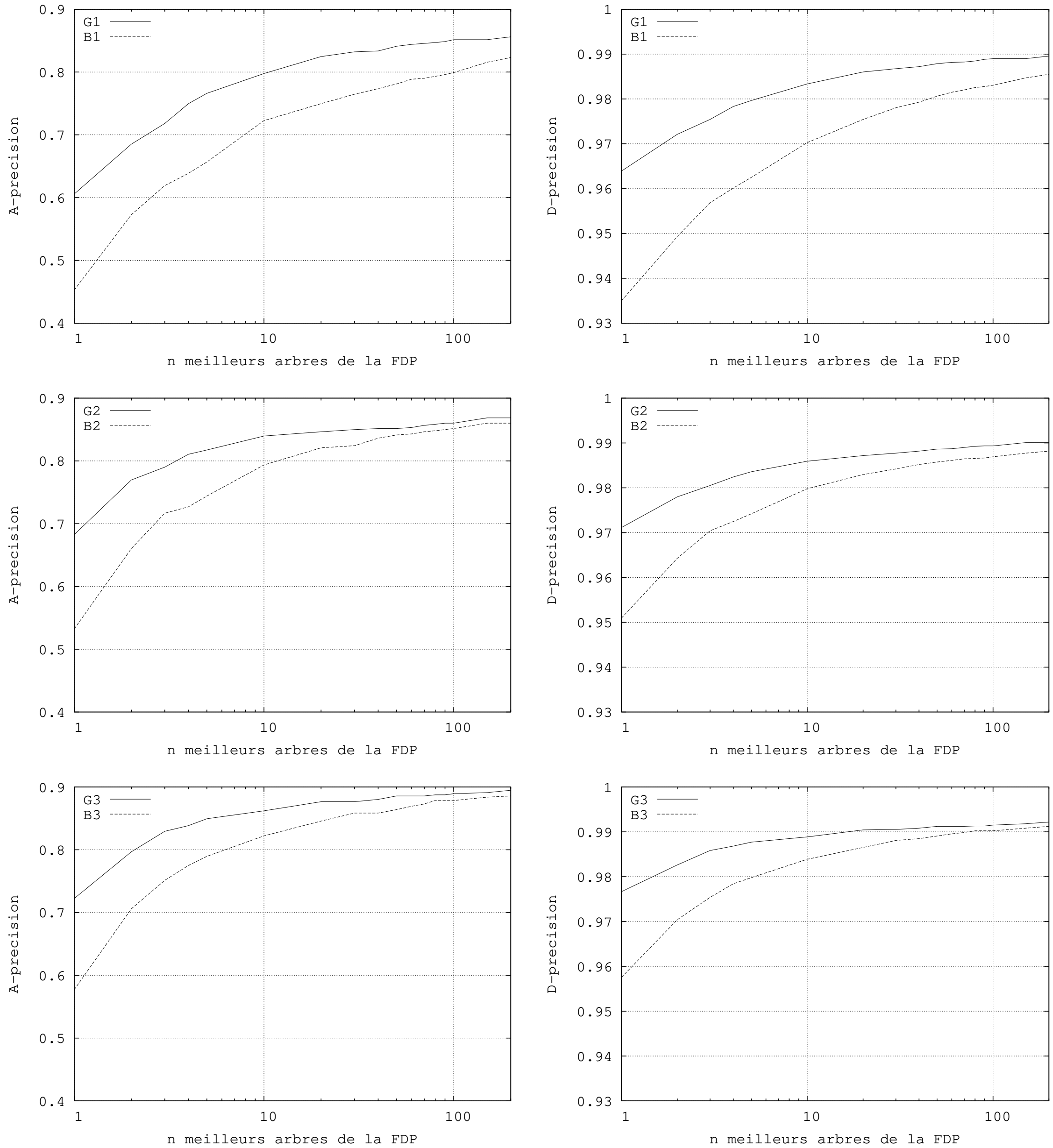



FIG. 48 – A-précision et D-précision du modèle bigramme et du modèle aléatoire pour les trois grammaires $G_1$, $G_2$ et $G_3$ (de haut en bas), en fonction du nombre d'arbres extraits de la FDP

| Grammaire | Nb positions | A-précision | D-précision |
|---|---|---|---|
| $G_1$ | 0 | $0,411$ | $0,924$ |
| | 1 | $0,451$ | $0,939$ |
| | 2 | $0,437$ | $0,935$ |
| $G_2$ | 0 | $0,515$ | $0,946$ |
| | 1 | $0,645$ | $0,966$ |
| | 2 | $0,646$ | $0,966$ |
| $G_3$ | 0 | $0,564$ | $0,956$ |
| | 1 | $0,688$ | $0,971$ |
| | 2 | $0,695$ | $0,972$ |

TAB. 5 – A-précision et D-précision des trois grammaires sur le corpus de test - Modèle positionnel, une position

## 11.4 Performances de l'analyseur couplé à un étiqueteur

Les résultats de la section précédente ont permis d'évaluer la couverture des grammaires produites par le processus d'extraction, ainsi que l'apport d'un modèle probabiliste pour choisir un arbre dans la forêt produite à l'issue de l'analyse syntaxique. Les conditions dans lesquelles les expériences ont été effectuées sont artificielles, car elles supposent un étiqueteur grammatical parfait. Les expériences décrites dans cette section sont plus proches des conditions réelles. En effet, l'entrée du système est constituée de l'intégralité du corpus de test. Les phrases sont traitées, dans un premier temps, par un étiqueteur grammatical produisant un nombre donné de solutions, représentées sous la forme d'un treillis de catégories. Le treillis est ensuite fourni en entrée à l'analyseur syntaxique qui produit, lui-même, un nombre donné de solutions. Nous commencerons par donner, ci-dessous, les résultats de l'étiqueteur, puis les résultats de l'analyseur sur les sorties de l'étiqueteur.

### 11.4.1 Performances de l'étiqueteur

Les précisions des $n$ solutions les plus probables de l'étiqueteur, pour les trois grammaires $G_1$, $G_2$ et $G_3$, sur le corpus de test sont représentées dans la figure 49. Rappelons que la précision de l'étiqueteur, lorsque ce dernier produit plus d'une solution, a été définie comme la précision de sa meilleure solution, appelée précision maximale.

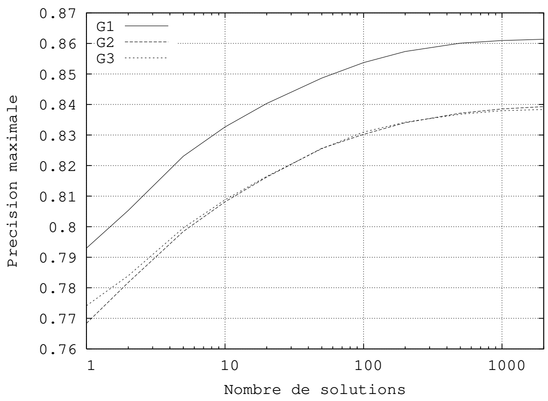



FIG. 49 – Précision de l'étiquetage en fonction du nombre d'hypothèses produites

On observe sans surprise que la précision dépend du nombre de catégories de la grammaire mais aussi de la nature de ces catégories. En effet, les résultats des deux grammaires $G_2$ et $G_3$ sont très proches, bien que $G_3$ possède approximativement 20% de catégories de plus que $G_2$. On observe aussi que la précision maximale n'augmente quasiment plus à partir de 200 solutions. Les résultats obtenus sont nettement inférieurs à ceux obtenus par S.Bangalore sur le Penn Treebank ($92,2\%$ pour la meilleure solution de l'étiqueteur). La différence majeure entre les expériences de S.Bangalore et les expériences décrites ici concerne le nombre de catégories : 300 dans le premier cas et, respectivement, 3 808, 5 910 et 7 123 dans le second. Cette différence explique probablement l'écart entre les résultats obtenus par S.Bangalore et les résultats décrits ici.

### 11.4.2 Performances de l'analyseur

La D-précision de la solution la plus probable de l'analyseur, en fonction du nombre de solutions produites par l'étiqueteur, pour les trois grammaires, est représentée dans la figure 50. Les meilleurs résultats sont obtenus pour la grammaire $G_1$, alors que $G_3$ obtenait les meilleurs résultats pour une entrée correctement étiquetée. Les résultats de l'analyse couplée à l'étiquetage suivent donc la tendance observée sur l'étiquetage. Le meilleur résultat obtenu pour $G_1$ est de $0,746$ (D-précision) en prenant en entrée de l'analyseur le treillis des 200 meilleures solutions de l'étiqueteur, dont la précision maximale est égale à $0,853$.

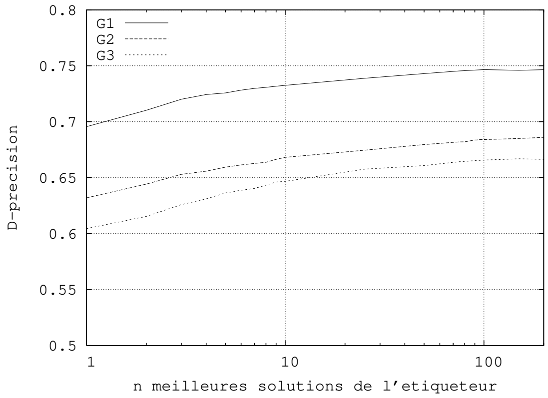



FIG. 50 – A-précision et D-précision de la solution la plus probable de l'analyseur en fonction du nombre de solutions de l'étiqueteur grammatical - Modèle bigramme

Les résultats obtenus sont très nettement inférieurs à ceux obtenus sur des phrases correctement étiquetées pour la même grammaire. Ces derniers étaient de $0,964$ (D-précision). Cette différence permet de quantifier l'influence des erreurs d'étiquetage sur les performances de l'analyse : une diminution de 15% de précision de l'étiqueteur provoque une diminution de $22,4\%$ de la D-précision de l'analyse. Cette comparaison est un peu biaisée du fait que, dans le premier cas, seule la bonne séquence de catégories était fournie à l'analyseur, alors qu'ici, un treillis de 200 séquences de catégories lui est fourni.

Afin de mieux apprécier l'influence des erreurs d'étiquetage sur les erreurs d'analyse, nous avons tracé, dans la figure 51, la D-précision du résultat de l'analyse par rapport à la précision de l'étiquetage. Cette figure montre que la relation entre les deux variables est assez proche d'une relation linéaire, sur l'intervalle que nous avons à notre disposition (une précision de l'étiqueteur variant de $0,79$ à $0,85$). Cet intervalle est malheureusement trop limité pour effectuer une extrapolation et déterminer la précision d'étiquetage nécessaire pour obtenir une D-précision donnée.

Afin d'évaluer la difficulté de la tâche d'analyse prenant en entrée un treillis de catégories, nous avons eu recours aux modèles aléatoires introduits dans la section précédente. Les D-précisions obtenues par la grammaire aléatoire $B_1$ (équivalente à la grammaire $G_1$ mais dont les paramètres ont été déterminés aléatoirement) sont représentées dans la figure 52. La D-précision de $G_1$ a été reprise dans la figure pour faciliter la comparaison des deux courbes.

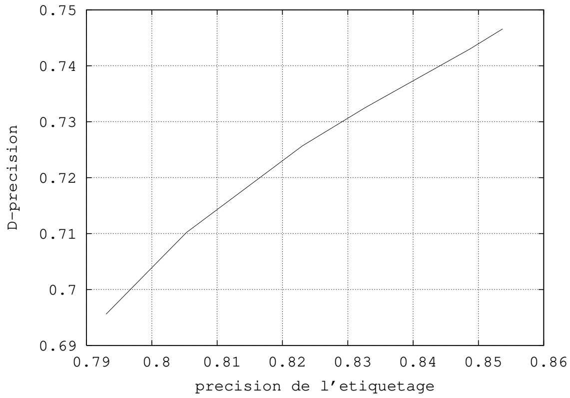



FIG. 51 – D-précision de l'analyseur en fonction de la précision de l'étiqueteur

On observe que les performances du modèle aléatoire diminuent lorsque l'on augmente le nombre de solutions produites par l'étiqueteur. L'apport du modèle probabiliste est ici nettement plus important qu'il ne l'était pour une entrée correcte. En effet, le gain varie entre $14,8\%$ et $30,4\%$ alors qu'il était de $3,2\%$ pour une entrée correcte. En définitive, le modèle probabiliste ne semble pas assez précis pour discriminer les bonnes analyses pour une phrase correctement étiquetée. En revanche, son apport est significatif lorsque l'entrée de l'analyseur est constituée d'un treillis de catégories.

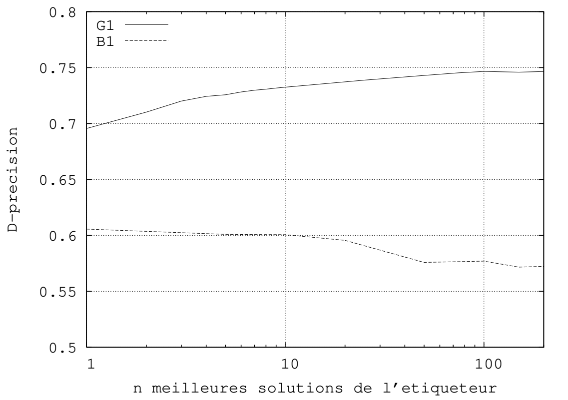



FIG. 52 – A-précision et D-précision de la solution la plus probable de l'analyseur avec le modèle bigramme et avec le modèle aléatoire de la grammaire $G_1$. L'entrée de l'analyseur est constitué des $n$ solutions les plus probables de l'étiqueteur.

# 12 Résumé des expériences effectuées sur le Penn Treebank

Les expériences résumées dans ce chapitre ont été réalisées sur le Penn Treebank([Marcus et al., 1993]). Les grammaires ont été produites par John Chen et Owen Rambow, à l'université Columbia, d'après les principes exposés dans [Chen, 2002]. L'étiquetage grammatical a été réalisé par François Toussenel, en utilisant les mêmes outils que pour les expériences sur le corpus LE MONDE. Les expériences réalisées sont identiques à celles du chapitre précédent. C'est la raison pour laquelle nous les détaillerons moins que nous ne l'avons fait pour le corpus LE MONDE.

## 12.1 Le corpus

Le corpus a été divisé en trois parties, servant de corpus d'apprentissage, de développement et de test, dont les tailles sont représentées dans le tableau 6. La taille du corpus complet est de 1 036 905 mots, soit plus du double du corpus LE MONDE.

| | Apprentissage | Test | Développement |
|---|---|---|---|
| Nombre de phrases | 39 797 | 3 840 | 1 744 |
| Nombre de mots | 949 993 | 46 451 | 40 461 |

TAB. 6 – Tailles des corpus d'entraînement, de test et de développement

## 12.2 La grammaire

Contrairement aux expériences du chapitre précédent, une seule grammaire a été utilisée ici. Elle est appelée $D_6$ et ses caractéristiques sont représentées dans le tableau 7. L'extraction de $D_6$ tire profit d'un certain nombre de particularités du Penn Treebank, en particulier de la présence d'étiquettes sémantiques, pour distinguer les actants des circonstants. $D_6$ est constituée de 4 726 automates, ce qui la place entre les grammaires $G_1$ et $G_2$ du chapitre précédent. L'ambiguïté moyenne se situe aussi entre celles de $G_1$ et de $G_2$. La couverture de la grammaire est légèrement supérieure à celle de la grammaire $G_1$, dont la taille est pourtant sensiblement inférieure. Cette différence s'explique probablement par la différence des tailles des corpus d'apprentissage.

| Grammaire | Taille | Couverture | Ambiguïté moyenne | Temps d'analyse moyen |
|---|---|---|---|---|
| $D_6$ | 4 726 | $0,995$ | 4 069 | $0,23$ ms |

TAB. 7 – Grammaire $D_6$ - Taille, couverture, ambiguïté moyenne et le temps moyen d'analyse par phrase sur le corpus de test

## 12.3 Performances de l'analyseur étant donné un étiquetage correct

Les résultats obtenus par les modèles bigrammes et positionnels sont représentés dans le tableau 8. Les résultats du modèle bigramme sont, ici encore, supérieurs à ceux du modèle positionnel. D'autre part, les précisions sont supérieures, pour les deux types de modèles, aux précisions observées sur le corpus LE MONDE. En effet, la D-précision sur le corpus LE MONDE était égale à $0,976$ pour la grammaire $G_3$ (la grammaire qui obtenait les meilleurs résultats) et elle est de $0,979$ ici, pour une grammaire plus ambiguë.

| | | GDGP-2G | | GDGP-POS | | |
|---|---|---|---|---|---|---|
| Grammaire | Taille réelle du corpus de test | A-prec. | D-prec. | Nb pos. | A-prec. | D-prec. |
| $D_6$ | 40 461 mots | $0,71$ | $0,979$ | 0 | $0,575$ | $0,961$ |
| | | | | 1 | $0,581$ | $0,962$ |
| | | | | 2 | $0,664$ | $0,972$ |

TAB. 8 – A-précision et D-précision de la grammaire $D_6$ sur le corpus de test - Modèle bigramme et modèle positionnel

Il est difficile d'expliquer les différences des résultats observés sur les deux corpus. Plusieurs paramètres influent en effet sur le résultat de l'analyse, parmi lesquels on peut citer : la taille du corpus d'apprentissage, les choix d'annotation, les processus d'extraction des grammaires et la variété des structures syntaxiques dans les deux corpus. Il est évident que la taille du corpus d'apprentissage constitue un avantage, mais il est difficile d'en mesurer l'importance. Nous avons néanmoins comparé les gains des modèles bigrammes entre le français et l'anglais en recourant à un modèle aléatoire, de la même façon que nous l'avions fait dans le chapitre précédent. Le gain apporté par le modèle bigramme, par rapport à un modèle aléatoire est de $2,7\%$. Il est comparable aux gains obtenus sur le corpus LE MONDE. Ce résultat semble montrer que la différence de taille des corpus d'apprentissage ne se traduit pas par une différence sensible de la qualité de la grammaire probabiliste.

## 12.4 Performances de l'analyseur couplé à un étiqueteur

La figure 53 retrace l'évolution de la précision maximale de l'étiqueteur grammatical en fonction du nombre de solutions produites par ce dernier. On observe que la précision est sensiblement supérieure à celle observée sur le corpus LE MONDE : la précision obtenue sur la grammaire $G_1$, ne contenant que 3 803 automates était égale à $0,793$, alors la précision de $D_6$ (4 726 automates) est égale à $0,811$. De plus, le gain apporté sur la précision maximale, en augmentant le nombre de solutions de l'étiqueteur, est légèrement supérieur pour $D_6$ qu'il ne l'était pour $G_1$. Dans le premier cas, il est égal à 9% et dans le second à $8,5\%$. Ces différences s'expliquent probablement par la différence de taille des corpus d'apprentissage. On peut aussi observer, sur la courbe de la figure 53, le même tassement de la précision pour un nombre de solutions supérieur à 200 que celui que nous avions observé dans le chapitre précédent. Là aussi, l'augmentation du nombre de solutions de l'étiqueteur ne permet pas d'augmenter la précision maximale au-delà d'une certaine limite.

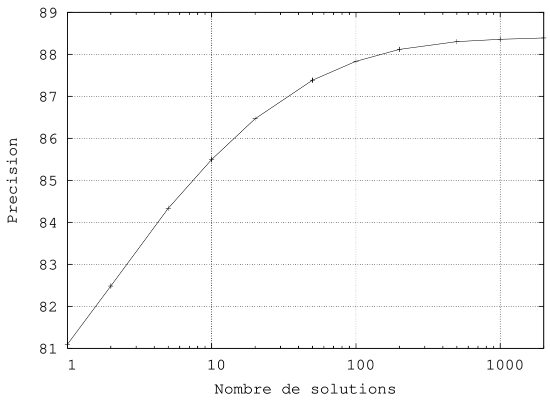



FIG. 53 – Précision de l'étiquetage en fonction du nombre d'hypothèses produites par l'étiqueteur

La figure 54 retrace l'évolution de la D-précision de l'arbre le plus probable en fonction du nombre de solutions produites par l'étiqueteur. Comme on pouvait s'y attendre, les résultats sont sensiblement supérieurs à ceux obtenus sur le corpus LE MONDE : pour un treillis d'entrée composé des 200 meilleures solutions de l'étiqueteur, la D-précision de l'arbre le plus probable est égale à $0,746$ pour le corpus LE MONDE et à $0,799$ pour le Penn Treebank, soit une augmentation de $7,1\%$. Cette différence était prévisible car l'étiqueteur, aussi bien que l'analyseur, ont des performances supérieures dans le cas du Penn Treebank.

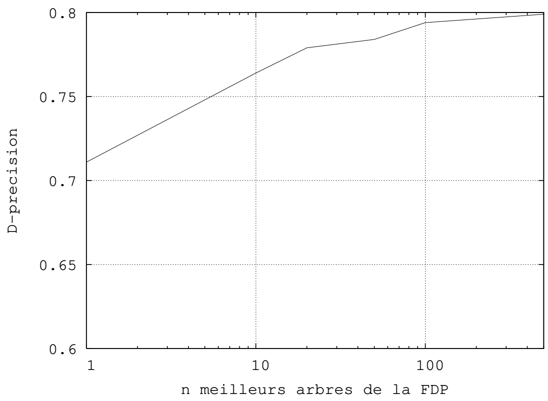



FIG. 54 – D-précision de la solution la plus probable de l'analyseur en fonction du nombre de solutions de l'étiqueteur grammatical - Modèle bigramme

# 13 Bilan

Les expériences effectuées dans les chapitres 11 et 12 ont permis d'évaluer le système décrit dans de document. Les meilleurs résultats obtenus sont de $0,746$ (D-précision) pour le corpus LE MONDE et de $0,799$ pour le Penn Treebank. Ces résultats sont, pour le moment, assez décevants et sont significativement inférieurs à l'état de l'art, pour ce qui concerne les expériences sur le Penn Treebank. L'analyseur de Michael Collins ([Collins, 1999]), par exemple, atteint une précision de $0,9$ sur le même corpus. Il est difficile de déterminer, à l'heure actuelle, si l'approche que nous avons choisie pourrait amener à des résultats comparables, ou supérieurs, à l'état de l'art. Les expériences ont en effet révélé un certain nombre de sources d'erreurs, certaines desquelles pourraient être traitées, améliorant ainsi les résultats. Trois sources majeures d'erreurs sont présentées ci-dessous.

La principale source d'erreurs que nous avons identifiée, à l'issue des expériences, est l'étiquetage grammatical. Nous avons vu qu'en prenant en compte l'ensemble des 200 meilleures solutions d'un étiqueteur, la précision de la meilleure solution de cet ensemble est en moyenne de $0,85$ (85% des mots se voient attribuer une étiquette correcte) pour le français et $0,88$ pour l'anglais. L'amélioration de ces résultats est une condition nécessaire pour augmenter les performances du système global (analyseur couplé à l'étiqueteur) car une erreur d'étiquetage se traduit généralement par une erreur (ou plus) d'analyse. Plusieurs solutions peuvent être envisagées pour améliorer les performances de l'étiquetage. La plus simple consiste à diminuer le jeu d'étiquettes en définissant des catégories moins fines et, par conséquent, des grammaires plus ambiguës. Dans une telle perspective, la tâche de l'étiqueteur sera simplifiée et la tâche de l'analyseur deviendra plus ardue, ce qui aura probablement pour effet de dégrader ses performances. Les performances générales du système s'apprécieront si l'amélioration des performances de l'étiqueteur compense la dégradation des performances de l'analyseur. D'autres solutions plus radicales peuvent être envisagées, telle que le recours à des méthodes d'étiquetage différentes. La méthode choisie ici est celle des chaînes de Markov cachées, qui est la méthode initialement choisie par S.Bangalore dans ses travaux sur l'étiquetage grammatical. D'autres méthodes peuvent être envisagées pour réaliser cette tâche, en particulier des méthodes symboliques (non probabilistes) qui permettraient de vérifier si une séquence de catégories est « valide » : si elle peut mener à une analyse complète de la phrase. Ce type de contrôle global (portant sur toute la séquence) ne peut être effectué par un modèle de Markov caché, du fait de l'hypothèse de Markov.

Si l'étiquetage constitue la principale source d'erreur, elle n'en est cependant pas la seule. Nous avons vu qu'en présentant, en entrée de l'analyseur, une phrase correctement étiquetée, certaines erreurs étaient néanmoins commises par l'analyseur. Dans le cas de l'anglais, la D-précision atteignait $0,979$ et, pour le français, $0,976$ (pour la grammaire $G_3$, la moins ambiguë des trois grammaires testées). Le niveau élevé de ces résultats et, par conséquent, la faiblesse de la marge de progression possible, ne nous ont pas poussé à porter beaucoup d'effort sur les modèles probabilistes, en particulier sur les méthodes d'estimation des paramètres : les méthodes mises en œuvre sont assez simples et elles pourraient être améliorées. Cependant, cette amélioration risque de devenir indispensable pour l'analyse de grammaires plus ambiguës (solution possible au problème des mauvaises performances de l'étiquetage). En effet, nous avons pu observer, dans les expériences portant sur le français, que l'ambiguïté de la grammaire avait une influence sur les performances de l'analyseur : la D-précision pour la grammaire $G_3$ (7 123 catégories) est égale à $0,976$, et elle est égale à $0,964$ pour la grammaire $G_1$ (3 808 catégories). Dans la perspective de grammaires beaucoup plus ambiguës, dont le nombre de catégories seraient de l'ordre de quelques centaines, l'amélioration des modèles probabilistes et l'estimation de leurs paramètres risquent de devenir une priorité.

La couverture des grammaires utilisées constitue une autre source d'erreurs. Nous avons vu que de manière générale, la couverture des grammaires (proportion de mots du corpus de test pour lesquels la catégorie n'a pas été créée) était élevée. Dans le cas du français, la couverture des grammaires $G_1$, $G_2$ et $G_3$ était égale, respectivement, à $0,976$, $0,971$ et $0,964$. Cependant, la proportion de phrases comportant au moins un mot pour lequel la catégorie n'a pas été créée est nettement plus élevée. Elle est égale, pour les trois grammaires, à $13,7\%$, $24,2\%$ et $28,7\%$. L'analyse correcte de ces phrases ne pourra par conséquent pas être construite par l'analyseur. Une solution évidente à ce problème est l'accroissement de la taille du corpus d'apprentissage. Mais de tels corpus sont rares et chers à élaborer, et il est peu probable que l'on puisse disposer, dans un avenir proche, de corpus significativement plus importants. De plus, l'accroissement de la taille des corpus permettra d'atténuer le problème, mais pas de le résoudre complètement.

# 14 Conclusions et perspectives

Les travaux décrits dans ce document avaient pour objectif de réaliser un analyseur syntaxique probabiliste pour grammaires de dépendances, dont la grammaire est extraite automatiquement d'un corpus arboré. Le document est organisé en trois parties, recouvrant trois aspects différents de nos travaux. Dans la première partie, un formalisme syntaxique probabiliste est défini. Il sert de base aux deux algorithmes d'analyse syntaxique et de recherche de la meilleure analyse d'une phrase, décrits dans la seconde partie. Des expériences sur deux corpus différents sont présentées dans la troisième partie. Nous allons reprendre ci-dessous les points principaux des trois parties puis nous présenterons brièvement les directions dans lesquelles ce travail pourrait être poursuivi.

Le formalisme des grammaires génératives de dépendances probabilistes, décrit dans la première partie du document repose sur la notion d'automate lexicalisé. Trois idées clefs sont à l'origine de la définition de tels automates.

– La première est liée aux structures de dépendances, plus spécifiquement à la conception d'un système de réécriture pour grammaires de dépendances. L'objectif d'un tel système est de générer des arbres de dépendances comme produit d'un processus de réécriture de chaînes. Nous avons vu dans le chapitre 2 que la définition d'un tel système est confrontée à deux problèmes : la spécification de la position du gouverneur vis-à-vis de ses dépendants et la génération d'un nombre quelconque de dépendants d'un même gouverneur. Cette dernière doit en effet être réalisée sans recourir aux règles récursives des grammaires syntagmatiques, qui introduisent des niveaux intermédiaires dans l'arbre de dérivation. Celui-ci ne correspond alors plus à un arbre de dépendances. Nous avons vu qu'une solution à ce problème avait été proposée par [Abney, 1996b], elle consiste à définir des règles de réécriture dont la partie droite est constituée par une expression régulière. Nous avons adopté cette idée mais avons remplacé les expressions régulières par des automates finis.

– La seconde idée clef est la possibilité de décomposer le processus génératif en étapes élémentaires, correspondant au franchissement d'une transition. Chaque transition franchie se traduit par l'établissement d'une seule dépendance dans l'arbre de dépendances en cours de génération. Cette « finesse » du processus génératif nous a permis de concevoir un mode de représentation des arbres de dépendances, appelé arbres de dépendances binaires, qui est aux structures de dépendances ce que la forme normale de

Chomsky est aux structures syntagmatiques. Les arbres de dépendances binaires permettent à leur tour la représentation économique d'une forêt d'arbres de dépendances sous la forme d'un objet complexe appelé forêt de dépendances partagée. La possibilité de représenter une forêt d'arbres de dépendances sous une forme compacte constitue, à notre connaissance, un aspect novateur de notre travail.

– La troisième idée clef réside dans la possibilité qu'offrent les automates lexicalisés d'établir une correspondance entre un modèle algébrique et un modèle probabiliste. Nous avons vu, en effet, que les grammaires probabilistes contemporaines ont adopté comme modèle algébrique les grammaires hors-contexte, alors que leurs modèles probabilistes reposent sur d'autres hypothèses : ils sont lexicalisés et dépendent du contexte. Cette absence de correspondance entre les deux modèles a pour effet d'obscurcir le lien entre le modèle algébrique et le modèle probabiliste. L'avantage des automates lexicalisés probabilistes, de ce point de vue, est la possibilité qu'ils offrent d'adapter la structure des automates (le modèle algébrique) au modèle probabiliste. Nous avons pu en effet définir des structures différentes d'automates pour implémenter des modèles probabilistes différents : le modèle bigramme et le modèle positionnel. Par ailleurs, cette meilleure correspondance entre les deux modèles revêt un aspect pratique : elle permet l'indépendance de l'algorithme d'analyse vis-à-vis du modèle probabiliste implémenté. Un seul algorithme est défini, qui permet l'analyse de toute grammaire GDGP, quel que soit le modèle probabiliste qu'elle implémente. La possibilité de modifier la structure des automates de la grammaire en fonction des modèles probabilistes implémentés constitue, à notre connaissance, un autre aspect novateur de notre travail.

La seconde partie du document a permis d'introduire deux algorithmes, un algorithme d'analyse syntaxique et un algorithme de recherche des arbres les plus probables d'une phrase. L'interface entre les deux algorithmes est constitué par les forêts de dépendances partagées évoquées ci-dessus.

L'algorithme d'analyse est d'une facture assez classique. Il mêle les principes de deux algorithmes standards pour grammaires hors-contexte : l'algorithme CYK et l'algorithme de Earley. Ce mélange n'est pas une nouveauté, on le retrouve à plusieurs reprises dans la littérature. Il permet d'effectuer une analyse ascendante, à la CYK, sans qu'il soit nécessaire de transformer la grammaire sous forme normale de Chomsky préalablement à l'analyse. Cette possibilité découle de la définition de la notion de sous-analyse (couple formé d'un automate et d'un état de ce dernier), qui peut être vue comme la généralisation des règles pointées de l'algorithme de Earley. L'originalité de notre algorithme provient de sa capacité à prendre en compte un treillis de catégories, produit par un étiqueteur grammatical. La prise en compte d'un treillis en entrée d'un analyseur n'est, en elle-même, pas nouvelle, elle a déjà été mise en œuvre pour la reconnaissance de la parole. L'originalité, dans notre cas, vient du fait que chaque chemin du treillis correspond à une grammaire et qu'un chemin ne peut être analysé qu'à l'aide de la grammaire qu'il définit.

L'algorithme d'analyse produit une forêt de dépendances partagées comprenant toutes les structures syntaxiques que la grammaire associe à la phrase analysée. Cette forêt constitue l'entrée de l'algorithme de recherche, qui en extrait les $n$ arbres les plus probables. L'algorithme repose sur les principes de la programmation dynamique et présente de nombreuses similarités avec l'algorithme de Viterbi.

Le mode d'interaction instauré entre les deux algorithmes évoqués ci-dessus constitue un autre point original de notre travail : les deux étapes d'analyse et de recherche sont en effet autonomes. L'analyse est réalisée indépendamment du modèle probabiliste : elle produit toutes les analyses de la phrase d'entrée. Cette indépendance est voulue. Elle laisse la porte ouverte à l'utilisation de l'analyseur dans un cadre non probabiliste, où l'intégralité de la forêt de dépendances partagées, produite par l'analyseur, constituerait l'entrée d'un autre module. Ce dernier pourrait être un module d'interprétation sémantique, ou, dans le cadre d'une application de traduction automatique, un module de transfert[25]. Nous avons néanmoins prévu la possibilité de prendre en compte, lors de l'analyse syntaxique, le modèle probabiliste implémenté par la grammaire. Ce dernier est utilisé pour élaguer la forêt de dépendances partagée au fur et à mesure de sa construction, en ne gardant, à chaque étape de l'analyse, que les sous-analyses les plus prometteuses.

La troisième partie du document concerne l'évaluation du système d'analyse. Les principales conclusions de cet aspect de notre travail ont été résumées à la fin de la partie concernée. L'objectif de ce volet de notre travail était - en plus de mettre à l'épreuve les différents objets et traitements définis - d'évaluer un système d'analyse syntaxique en deux étapes : une étape d'étiquetage grammatical suivie d'une étape d'analyse. Comme nous l'avons souligné dans la conclusion de la partie III, nous n'avons pu à l'heure actuelle, valider ni invalider cette idée.

Il est peut-être important de revenir ici sur les raisons qui nous ont poussé

[25]De tels modules doivent, bien entendu, être conçus de manière à prendre en entrée une forêt de dépendances partagées.

à nous engager dans la voie d'une analyse en deux étapes. Elles trouvent leur origine dans les expériences d'étiquetage grammatical, menées par S.Bangalore, qui avaient donné des résultats encourageants : la précision de l'étiquetage dépassait $0,9$ (plus de 90% des mots des phrases étiquetées l'étaient correctement), elle atteignait $0,97$ lorsque les trois meilleures solutions de l'étiqueteur étaient prises en compte. Ces résultats rendaient particulièrement attractive la perspective de prendre en entrée de l'analyseur la sortie d'un étiqueteur. Il suffisait, en effet, de montrer qu'un analyseur prenant en entrée une phrase (presque) correctement étiquetée pouvait aboutir à de bons résultats. C'est ce qui fut partiellement fait dans les premières séries d'expériences (décrites dans les sections 11.3 et 12.3 : « Performances de l'analyseur étant donné un étiquetage correct » ). Nous avons en effet montré que pour un étiquetage parfait, les résultats de l'analyse dépassaient de manière significative les meilleurs résultats du moment, atteignant $0,976$ ($97,6\%$ de dépendances correctes dans l'arbre le plus probable) pour le français et $0,979$ pour l'anglais. Restait alors à coupler l'étiqueteur et l'analyseur. Hélas, les résultats de l'étiqueteur dans notre contexte se sont révélés considérablement inférieurs aux résultats obtenus par Bangalore, très probablement du fait de la différence de taille des jeux d'étiquettes. En effet, les résultats obtenus se situaient aux alentours de 85% de mots correctement étiquetés (88% pour l'anglais), en prenant en compte les 200 meilleures solutions de l'étiqueteur. Dans ces conditions, les résultats de l'analyseur se sont avérés bien inférieurs à l'état de l'art : $0,746$ pour le français et $0,799$ pour l'anglais.

Plusieurs solutions ont été évoquées dans le bilan de la partie III pour tenter d'améliorer les résultats de l'ensemble. La première consiste à rééquilibrer la répartition du travail entre les deux modules d'étiquetage et d'analyse. D'une part, cette évolution permettra d'améliorer, éventuellement, les résultats. D'autre part, elle permettra de mettre en œuvre une idée séduisante à nos yeux, qui consiste à étudier le comportement du système en faisant varier l'ambiguïté des grammaires. La seconde solution consiste à modifier les principes de l'étiquetage en tirant profit de la structure des automates. En effet, actuellement, lors de l'étiquetage grammatical, une catégorie est vue comme un objet atomique, dépourvu de structure. Or une catégorie est associée à un automate et ce dernier peut imposer des contraintes à son contexte : un automate correspondant à un verbe fini imposera, par exemple, la présence, dans la phrase, d'un automate réalisant son sujet. La prise en compte de ces contraintes, éventuellement couplées au modèle probabiliste de l'étiqueteur pourrait en améliorer les performances. Ces deux évolutions constituent les voies dans lesquelles nous allons poursuivre notre travail.

# Références


[Abeillé et al., 2003] Abeillé, A., Clément, L., & Toussenel, F. (2003). *Building a treebank for French.* Edité dans Abeillé, A., éditeur, *Treebanks.* Kluwer, Dordrecht.

[Abney, 1996a] Abney, S. (1996a). A grammar of projections. Universität Tübingen.

[Abney, 1996b] Abney, S. (1996b). *Statistical Methods and Linguistics.* Edité dans Klavans, J. & Resnik, P., éditeurs, *The Balancing Act : Combining Symbolic and Statistic Approaches to Language*, Language, Speech, and Communication, pages 1–48. MIT Press.

[Albert et al., 2001] Albert, J., Giammarresi, D., & Wood, D. (2001). *Normal Form Algorithms for Extended Context-Free Grammars. Theoretical Computer Science*, 267 :35–47.

[Allauzen et al., 2003] Allauzen, C., Mohri, M., & Roark, B. (2003). *Generalized Algorithms for Constructing Statistical Language Models.* Edité dans *41st Meeting of the Association for Computational Linguistics*, pages 40–47, Sapporo, Japon.

[Alshawi, 1996] Alshawi, H. (1996). *Head Automata and Bilingual Tiling : Translation with Minimal Representation.* Edité dans *34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 167–176.

[Bangalore, 1997] Bangalore, S. (1997). *Complexity of Lexical Descriptions and its Relevance to Partial Parsing.* Thèse de doctorat, University of Pennsylvania, Philadelphia, USA.

[Bangalore & Joshi, 1999a] Bangalore, S. & Joshi, A. K. (1999a). *Supertagging : An Approach to Almost Parsing. Computational Linguistics*, 25(2) :237–265.

[Bangalore & Joshi, 1999b] Bangalore, S. & Joshi, A. K. (1999b). *Supertagging : An Approach to Almost Parsing. Computational Linguistics*, 25(2) :237–265.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ.

[Bikel, 2004] Bikel, D. (2004). *Intricacies of Collins' Parsing Model. Computational Linguistics.* sous presse.

[Billot & Lang, 1989] Billot, S. & Lang, B. (1989). *The structure of shared forests in ambiguous parsing.* Edité dans *27th Meeting of the Association for Computational Linguistics (ACL'89)*, pages 143–151, Vancouver, Canada.

[Black et al., 1991] Black, E., Abney, S. P., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M. P., Roukos, S., Santorini, B., & Strzalkowski, T. (1991). *A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars*. Edité dans *Proceedings DARPA Speech and Natural Language Workshop*, pages 306–311, Pacific Grove, CA. Morgan Kaufmann.

[Black et al., 1992] Black, E., Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., & Roukos, S. (1992). *Towards History-based Grammars : Using Richer Models for Probabilistic Parsing*. Edité dans *Proceedings DARPA Speech and Natural Language Workshop*, pages 134–139, Harriman, New York. Morgan Kaufmann.

[Booth, 1969] Booth, T. L. (1969). *Probabilistic representation of formal languages*. Edité dans *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pages 74–81.

[Boyer & Lapalme, 1985] Boyer, M. & Lapalme, G. (1985). *Generating paraphrases from meaning-text semantic networks. Computational Intelligence*.

[Bröker, 2000] Bröker, N. (2000). *Unordered and non-projective dependency grammars. Traitement Automatique des Langues*, 41(1) :246–272.

[Bröker & Neuhaus, 1997] Bröker, N. & Neuhaus, P. (1997). *The complexity of recognition of linguistically adequate dependency grammars*. Edité dans *35th Meeting of the Association for Computational Linguistics (ACL'97)*, Madrid, Spain.

[Buchsbaum et al., 2000] Buchsbaum, A., Giancarlo, R., & Westbrook, J. (2000). *On the Determinization of Weighted Finite Automata*. *SIAM Journal on Computing*, 30(5) :1502–1531.

[Candito & Kahane, 1998] Candito, M. & Kahane, S. (1998). *Can the derivation tree represent a semantic graph ? An answer in the light of Meaning-Text Theory*. Edité dans *International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 21–24, Philadelphie.

[Carroll & Weir, 1997] Carroll, J. & Weir, D. (1997). *Encoding frequency information in lexicalized grammars*. Edité dans *Fifth International Workshop on Parsing Technologies (IWPT 97)*, pages 8–17.

[C.Emele & Dorna, 1998] C.Emele, M. & Dorna, M. (1998). *Ambiguity Preserving Machine Translation using Packed Representation*. Edité dans *Proceedings of the 19th International Conference on Computational Linguistics (COLING-ACL'98)*, Montreal.

[Chappelier & Rajman, 1998] Chappelier, J.-C. & Rajman, M. (1998). A practical bottom-up algorithm for on-line parsing with stochastic context-free grammars. Rapport technique, Ecole Polytechnique Fédérale de Lausanne, Département Informatique.

[Chappelier et al., 1999] Chappelier, J.-C., Rajman, M., & Rozenknop, A. (1999). *Lattice Parsing for Speech Recognition*. Edité dans *5ème conférence sur le Traitement Automatique des Langues Naturelles (TALN'1999)*, Cargèse, France.

[Charniak, 1997] Charniak, E. (1997). *Statistical parsing with a context-free grammar and word statistics*. Edité dans *AAAI-97*, Menlo Park. AAAI Press.

[Chen, 2002] Chen, J. (2002). *Towards Efficient Statistical Parsing using Lexicalized Grammatical Information*. Thèse de doctorat, University of Delaware.

[Chiang, 2000] Chiang, D. (2000). *Statistical parsing with an automatically extracted tree adjoining grammar*. Edité dans *Data Oriented Parsing*, pages 299–316. CSLI Publications.

[Chomsky, 1957] Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

[Chomsky, 1962] Chomsky, N. (1962). Context-free grammars and pushdown storage. Quarterly progress report, No 65, Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Etats-Unis.

[Chomsky, 1963] Chomsky, N. (1963). *Formal Properties of Grammars*. Edité dans Luce, R. D., Bush, R., & Galanter, E., éditeurs, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. Wiley, New York.

[Chomsky & Miller, 1958] Chomsky, N. & Miller, G. A. (1958). *Finite-state Languages*. *Information and Control*, 1 :91–112.

[Church, 1988] Church, K. W. (1988). *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text*. Edité dans *Second Conference on Applied Natural Language Processing*, pages 136–143. ACL.

[Church & Patil, 1982] Church, K. W. & Patil, R. (1982). *Coping with Syntactic Ambiguity*. *American Journal of Computational Linguistics*, 8(3-4) :139–149.

[Collins, 1999] Collins, M. (1999). *Head-driven Statistical Models for Natural Language Parsing*. Thèse de doctorat, University of Pennsylvania, Philadelphia.

[Collins, 2003] Collins, M. (2003). *Head-Driven Statistical Models for Natural Language Parsing*. *Computational Linguistics*, 29(4) :589–637.

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms, Second Edition.* MIT Press - McGraw-Hill.

[Doran et al., 1994] Doran, C., Egedi, D., Hockey, B.-A., Srinivas, B., & Zaidel, M. (1994). *XTAG System - A wide coverage grammar for English.* Edité dans *Proceedings of the 16th International Conference on Computational Linguistics (COLING'94)*, Kyoto, Japon.

[Dybro Johansen, 2004] Dybro Johansen, A. (2004). Extraction automatique de grammaires à partir d'un corpus français. Mémoire de DEA, Université Paris 7.

[Earley, 1968] Earley, J. (1968). *An Efficient Context-Free Parsing Algorithm.* Thèse de doctorat, Carnegie Mellon University, Pittsburgh, PA.

[Earley, 1970] Earley, J. (1970). *An efficient context-free parsing algorithm. Communcations of the ACM*, 8(6) :451–455.

[Eilenberg, 1974] Eilenberg, S. (1974). *Automata, languages, and machines, Volume A.* Academic Press, New York.

[Eilenberg, 1976] Eilenberg, S. (1976). *Automata, languages, and machines, Volume B.* Academic Press, New York.

[Eisner, 1996] Eisner, J. (1996). *Three New Probabilistic Models for Dependency Parsing : An Exploration.* Edité dans *Proceedings of the 17th International Conference on Computational Linguistics (COLING'96)*, pages 340–345.

[Eisner, 2000] Eisner, J. (2000). *Bilexical Grammars and Their Cubic-Time Parsing Algorithms.* Edité dans Bunt, H. & Nijholt, A., éditeurs, *Advances in Probabilistic and Other Parsing Technologies*, volume 16 of *Text, Speech and Language Technology*, pages 29–61. Kluwer Academic Publishers, Dordrecht/Boston/London.

[Erbach, 1994] Erbach, G. (1994). *Bottom-Up Earley Deduction.* Edité dans *Proceedings of the 16th International Conference on Computational Linguistics (COLING'94)*, volume 2, pages 796–802, Kyoto.

[Gaifman, 1965] Gaifman, H. (1965). *Dependency Systems and Phrase-Structure Systems. Information and Control*, 8 :304–337.

[Gale & Church, 1994] Gale, W. A. & Church, K. W. (1994). *What is wrong with adding one ?* Edité dans Oostdijk, N. & de Haan, P., éditeurs, *Corpus-based Research into Language*, pages 189–198. Rodopi, Amsterdam.

[Gildea, 2001] Gildea, D. (2001). *Corpus variation and parser performances.* Edité dans *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, Pittsburgh, USA.

[Gladkij & Mel'čuk, 1975] Gladkij, A. & Mel'čuk, I. A. (1975). *Tree grammars I. A formalism for syntactic transformations in natural languages. Linguistics*, 150 :47–82.

[Graham et al., 1980] Graham, S. L., Harrison, M. A., & Ruzzo, W. L. (1980). *An Improved Context-Free Recognizer. ACM Transactions on Programming Languages and Systems*, 2(3) :415–462.

[Hays, 1964] Hays, D. G. (1964). *Dependency Theory : A Formalism and some Observations. Language*, 40 :511–525.

[Jelinek, 1998] Jelinek, F. (1998). *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MASS.

[Jelinek & Mercer, 1980] Jelinek, F. & Mercer, R. L. (1980). *Interpolated estimation of Markov source parameters from sparse data.* Edité dans Gelsema, E. S. & Kanal, L. N., éditeurs, *Proceedings, Workshop on Pattern Recognition in Practice*, pages 381–397. North Holland, Amsterdam.

[Joshi et al., 1975] Joshi, A., Levy, L., & Takahashi, M. (1975). *Tree Adjunct Grammars. J. Comput. Syst. Sci.*, 10 :136–163.

[Joshi & Rambow, 2003] Joshi, A. & Rambow, O. (2003). *A Formalism for Dependency Grammars Based on Tree Adjoining Grammars.* Edité dans *First International Conference on Meaning-Text Theory*, pages 219–228, Paris, France.

[Joshi, 1985] Joshi, A. K. (1985). *Tree adjoining grammars : how much context-sensitivity is required to provide reasonable structural descriptions ?* Edité dans Dowty, D. R., Karttunen, L., & Zwicky, A., éditeurs, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge.

[Jurafsky & Martin, 2000] Jurafsky, D. & Martin, J. (2000). *Speech and Language Processing.* Prentice Hall, Upper Saddle River, New Jersey, Etats Unis.

[Kahane, 2000] Kahane, S. (2000). *Des grammaires formelles pour définir une correspondance.* Edité dans *6ème conférence sur le Traitement Automatique des Langues Naturelles (TALN'2000)*, Lausanne, Suisse.

[Kahane, 2002] Kahane, S. (2002). Grammaire d'unification sens-texte : Vers un modèle mathématique articulé de la langue. Habilitation à diriger des recherches, Université Paris 7.

[Kahane, 2004] Kahane, S. (2004). *Grammaires d'unification polarisées.* Fès, Maroc.

[Kahane & Mel'čuk, 1999] Kahane, S. & Mel'čuk, I. (1999). *La synthèse sémantique ou la correspondance entre graphes sémantiques et arbres syn-*

*taxiques ? Le cas des phrases à extraction en français contemporain. Traitement Automatique des Langues*, 40(2) :25–85.

[Kahane et al., 1998] Kahane, S., Nasr, A., & Rambow, O. (1998). *Pseudo Projectivity : A Polynomially Parsable Non-Projective Dependency Grammar.* Edité dans *33rd Meeting of the Association for Computational Linguistics (ACL'98)*, pages 646–652, Montréal Canada.

[Kasami, 1965] Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Rapport Technique AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA†.

[Katz, 1987] Katz, S. M. (1987). *Estimation of Probabilities from sparse data for the Language Model component of a speech recogniser. IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3) :400–401.

[Lecerf, 1960] Lecerf, Y. (1960). *Programme des conflits, modèle des conflits. Bulletin bimestriel de l'*ATALA, 4,5.

[Lombardo, 1996] Lombardo, V. (1996). *An Earley-Style Parser for Dependency Grammars.* Edité dans *COLING'96*, Copenhagen.

[Lombardo & Lesmo, 2000] Lombardo, V. & Lesmo, L. (2000). *A formal theory of dependency syntax with empty units. Traitement Automatique des Langues*, 41(1) :180–209.

[Lowerre, 1968] Lowerre, B. T. (1968). *The Harpy Speech Recognition System.* Thèse de doctorat, Carnegie Mellon University, Pittsburgh, PA.

[Magerman, 1995] Magerman, D. M. (1995). *Statistical Decision-Tree Models for Parsing.* Edité dans *ACL-95*, pages 276–283, Cambridge, MA. ACL.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). *Building a Large Annotated Corpus of English : The Penn Treebank. Computational Linguistics*, 19(2) :313–330.

[Maxwell & Kaplan, 1993] Maxwell, J. & Kaplan, R. (1993). *The Interface between Phrasal and Functional Constraints. Computational Linguistics*, 19(4) :571–590.

[McEnery & Wilson, 2001] McEnery, T. & Wilson, A. (2001). *Corpus Linguistics.* Edinburgh University Press.

[Mel'čuk, 1988] Mel'čuk, I. A. (1988). *Dependency Syntax : Theory and Practice.* State University of New York Press, New York.

[Mel'čuk & Pertsov, 1987] Mel'čuk, I. A. & Pertsov, N. V. (1987). *Surface Syntax of English.* John Benjamins, Amsterdam/Philadelphia.

[Mel'čuk & Zholkovsky, 1970] Mel'čuk, I. A. & Zholkovsky, A. (1970). *Towards a functioning meaning-text model of language. Linguistics*, 57 :10–47.

[Mertens, 2002] Mertens, P. (2002). *Parsing Dependency Grammar using ALE*. Edité dans *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, volume 2, pages 653–659, Taipei, Taiwan.

[Nakagawa, 1987] Nakagawa, S. (1987). *Spoken Language Recognition by Time-Synchronous Parsing Algorithm of Context Free Grammar*. Edité dans *International Conference on Acoustics, Speech and Signal Processing (ICASSP87)*, volume 2, pages 829–832, Dallas (Texas).

[Nasr, 1995] Nasr, A. (1995). *A Formalism and a Parser for Lexicalised Dependency Grammars*. Edité dans *4th International Workshop on Parsing Technologies*, pages 186–195, Prague.

[Nasr, 1996] Nasr, A. (1996). *Un modèle de reformulation automatique fondé sur la Théorie Sens Texte : Application aux langues contrôlées*. Thèse de doctorat, Université Paris 7.

[Nasr, 2003] Nasr, A. (2003). *Factoring Sufrace Syntactic Structures*. Edité dans *First International Conference on Meaning-Text Theory*, pages 249–258, Paris, France.

[Nasr et al., 1999] Nasr, A., Estève, Y., Béchet, F., Spriet, T., & de Mori., R. (1999). *A Language Model Combining N-grams and Stochastic Finite State Automata*. Edité dans *Eurospeech*, volume 5, pages 2175–2178, Budapest, Hungary.

[Nasr & Rambow, 2004a] Nasr, A. & Rambow, O. (2004a). *A Simple String-Rewriting Formalism for Dependency Grammar*. Edité dans *Workshop on Recent Advances in DependencyGrammar, International Conference on ComputationalLinguistics (Coling)*, Genève.

[Nasr & Rambow, 2004b] Nasr, A. & Rambow, O. (2004b). *Supertagging and Full Parsing*. Edité dans *7th International Workshop on Tree Adjoining Grammars and Related Frameworks*, Vancouver, Canada.

[Nasr et al., 2002] Nasr, A., Rambow, O., Chen, J., & Bangalore, S. (2002). *Context-Free Parsing of a Tree Adjoining Grammar Using Finite-State Machines*. Edité dans *Proceedings the Sixth Workshop on Tree Adjoining Grammars (TAG+ 6)*, Venise, Italie.

[Nasr & Volanschi, 2004] Nasr, A. & Volanschi, A. (2004). *Couplage d'un étiqueteur morpho-syntaxique et d'un analyseur partiel représentés sous la forme d'automates finis pondérés*. Edité dans *TALN'2004*, pages 329–338, Fez, Morocco.

[Polguère, 1990] Polguère, A. (1990). *Structuration et mise en jeu procédurale d'un modèle linguistique déclaratif dans un cadre de génération de texte*. Thèse de doctorat, Université de Montréal.

[Pollock, 1997] Pollock, J.-Y. (1997). *Langage et cognition, introduction au programme minimaliste de la grammaire générative*. Presses Universitaires de France.

[Rabiner, 1989] Rabiner, L. R. (1989). *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE*, 37(2) :257–286.

[Rambow et al., 2002] Rambow, O., Bangalore, S., Butt, T., Nasr, A., & Sproat, R. (2002). *Finite State Parser with Application Semantics*. Edité dans *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, Taipei, Taiwan.

[Rambow & Joshi, 1997] Rambow, O. & Joshi, A. (1997). *A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena*. Edité dans Wanner, L., éditeur, *Recent Trends in Meaning-Text Theory*, pages 167–190. Benjamins Academinc Publishers.

[Resnik, 1992] Resnik, P. (1992). *Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing*. Edité dans *COLING-92*, pages 418–424, Nantes, France.

[Roark, 2002] Roark, B. (2002). *Markov parsing : lattice rescoring with a statistical parser*. Edité dans *40th Meeting of the Association for Computational Linguistics (ACL'02)*, pages 287–294, Philadelphie, USA.

[Robinson, 1970] Robinson, J. J. (1970). *Dependency structures and transformational rules. Language*, 46(2) :259–285.

[Schabes, 1990] Schabes, Y. (1990). *Computational and Mathematical Properties of Lexicalized Grammars*. Thèse de doctorat, Université de Pennsylvanie, Philadelphie, Etats-Unis.

[Schabes, 1992] Schabes, Y. (1992). *Stochastic Lexicalized Tree-Adjoining Grammars*. Edité dans *COLING-92*, pages 426–433, Nantes, France.

[Schabes & Waters, 1995] Schabes, Y. & Waters, R. C. (1995). *Tree Insertion Grammars : A Cubic-Time, Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced. Computational Linguistics*, 21(4479-513).

[Tesnière, 1959] Tesnière, L. (1959). *Eléments de syntaxe structurale*. Klincksieck, Paris.

[Tomita, 1986] Tomita, M. (1986). *An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition*. Edité dans *International Conference on Acoustics, Speech and Signal Processing (ICASSP86)*, volume 3, pages 1569–1572, Tokyo Japon.

[Toussenel, 2004] Toussenel, F. (2004). *Why Supertagging is Hard.* Edité dans *Proceedings the Seventh Workshop on Tree Adjoining Grammars (TAG+ 7)*, Vancouver, Canada.

[Viterbi, 1967] Viterbi, A. J. (1967). *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory*, IT-13(2) :260–269.

[Weischedel et al., 1993] Weischedel, R., Schwartz, R., Palmucci, J., Meteer, M., & Ramshaw, L. (1993). *Coping with Ambiguity and Unknown Words through Probabilistic Models. Computational Linguistics*, 19(2) :359–382.

[Woods, 1970] Woods, W. A. (1970). *Transition Network Grammars for Natural Language Analysis. Transactions of the ACM*, 13(10) :591–606.

[Younger, 1967] Younger, D. (1967). *Recognition and parsing of context-free grammar in time* $n^3$. *Information and Control*, 10 :189–208.