UNIVERSITE D'AIX-MARSEILLE

UNIVERSITE DE SFAX

ECOLE DOCTORALE MATHEMATIQUES ET INFORMATIQUE DE MARSEILLE

ECOLE DOCTORALE DES SCIENCES ECONOMIQUES, GESTION ET INFORMATIQUE

LABORATOIRE PAROLE ET LANGAGE

MULTIMEDIA, INFORMATION SYSTEMS AND ADVANCED COMPUTING LABORATORY

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : Mathématique et Informatique

Spécialité : Informatique

Raja BENSALEM Ep. BAHLOUL

Titre de la thèse : Construction de ressources linguistiques arabes à l'aide du formalisme de grammaires de propriétés en intégrant des mécanismes de contrôle

Thesis title: Building Arabic linguistic resources using the property grammar formalism by integrating control mechanisms

Soutenue le JJ/MM/AAAA devant le jury :

Lamia BELGHUITH	Université de Sfax	Présidente
Emmanuel MORIN	Université de Nantes	Rapporteur
Mounir ZRIGUI	Université de Monastir	Rapporteur
Marie CANDITO	Université Paris Diderot	Examinatrice
Kais HADDAR	Université de Sfax	Directeur de thèse
Philippe BLACHE	Université Aix-Marseille	Directeur de thèse

Numéro national de thèse/suffixe local :

Résumé : La construction de ressources linguistiques arabes riches en informations syntaxiques constitue un enjeu important pour le développement de nouveaux outils de traitement automatique. Cette thèse propose une approche pour la création d'un treebank de l'arabe intégrant des informations d'un type nouveau reposant sur le formalisme des Grammaires de Propriétés. Une propriété syntaxique caractérise une relation pouvant exister entre deux unités d'une certaine structure syntaxique. Cette grammaire est induite automatiquement à partir du treebank arabe ATB, ce qui constitue un enrichissement de cette ressource tout en conservant ses qualités. Cet enrichissement a été également appliqué aux résultats d'analyse d'un analyseur état de l'art du domaine, le Stanford Parser, offrant la possibilité d'une évaluation s'appuyant sur un ensemble de mesures obtenues à partir de cette ressource. Les étiquettes des unités de cette grammaire sont structurées selon une hiérarchie de types permettant la variation de leur degré de granularité, et par conséquent du degré de précision des informations. Nous avons pu ainsi construire, à l'aide de cette grammaire, d'autres ressources linguistiques arabes. Nous avons dans un second temps, sur la base de cette nouvelle ressource, développé un analyseur syntaxique probabiliste basé sur les propriétés syntaxiques. Il s'agit du premier analyseur de ce type, que nous avons appliqué à l'arabe. Une grammaire de propriétés lexicalisée probabiliste fait partie de son modèle d'apprentissage pour pouvoir affecter positivement le résultat d'analyse et caractériser ses structures syntaxiques avec les propriétés de ce modèle. Nous avons enfin évalué les résultats de cette approche en les comparant à celles du Stanford Parser.

Mots-clés : grammaires de propriétés, langue arabe, treebanks, mécanismes de contrôle, analyseur syntaxique probabiliste.

Abstract : The building of syntactically informative Arabic linguistic resources is a major issue for the development of new machine processing tools. We propose in this thesis to create an Arabic treebank that integrates a new type of information, which is based on the Property Grammar formalism. A syntactic property is a relation between two units of a given syntactic structure. This grammar is automatically induced from the Arabic treebank ATB. We enriched this resource with the property representations of this grammar, while retaining its qualities. We also applied this enrichment to the parsing results of a state-of-the-art analyzer, the Stanford Parser. This provides the possibility of an evaluation using a measure set, which is calculated on this resource. We structured the tags of the units in this grammar according to a type hierarchy. This permit to vary the granularity level of these units, and consequently the accuracy level of the information. We have thus been able to construct, using this grammar, other Arabic linguistic resources. Secondly, based on this new resource, we developed a probabilistic syntactic parser based on syntactic properties. This is the first analyzer of this type that we have applied to Arabic. In the learning model, we integrated a probabilistic lexicalized property grammar that may positively affect the parsing result and describe its syntactic structures with its properties. Finally, we evaluated the parsing results of this approach by comparing them to those of the Stanford Parser.

Key-words: property grammars, Arabic language, treebanks, control mechanisms, probabilistic parser.

Dédicace

C'est avec une grande émotion et le coeur débordant d'une reconnaissance infinie que je dédie ce mémoire de thèse, fruit de la faveur imméritée de Dieu, à ceux et celles qui n'ont épargné aucun effort pour me permettre d'assurer la satisfaction d'un travail accompli mais également l'occasion de pouvoir partager ma joie et mon bonheur avec :

Mes chers parents Mohamed et Fekria, dont l'amour et l'infinie tendresse ainsi que leur sens de sacrifice m'a permis de mener à terme ce travail : Que Dieu me permette de rendre un tant soit peu le bonheur qu'ils n'ont jamais cessé de me prodiguer pour contribuer à ma prospérité. Qu'ils en soient remerciés. Tous les mots du monde ne sauraient exprimer l'immense amour que je vous porte, ni la profonde gratitude que je vous témoigne pour tous les efforts et les sacrifices que vous n'avez jamais cessé de consentir pour mon instruction et mon bien-être.

Mon cher mari Mohamed BAHLOUL dont la tendresse de laquelle il m'a entouré, le soutien moral et physique continu, les encouragements et les conseils pertinents m'ont été d'une aide précieuse. Qu'il trouve à travers ce travail l'expression de mon estime et mon sincère attachement et reconnaissance. Que Dieu nous réunisse pour le restant de notre vie en toute affection et amour.

Mes chers petits enfants Mariem et Ismail, qui ont pu supporter mon éloignement et ont pu continuer d'être sages en dépit des longues heures d'absence, quand j'étais plongée dans la préparation de cette thèse. Qu'ils

grandissent et apprennent le meilleur de leur vie. Je leur souhaite un avenir plein d'amour, de bonheur et de réussite.

Ma chère soeur Dorra, et mes chers frères Ahmed yessine et Mohamed Amine, pour toute la complicité et l'entente qui nous unissent, pour toute l'ambiance dont ils m'ont entouré, pour toute leur spontanéité et élan chaleureux, ce travail est un témoignage de mon attachement et de mon amour. Puisse Dieu le tout puissant exhausser tous leurs vœux.

Ma chère belle-mère Souad et mon cher beau-père Abdelaziz, qui m'ont accueilli à bras ouverts dans leur famille. Je les remercie d'être toujours à mes côtés, par leur aide, leur amour et leur tendresse. Je leur prie Dieu le tout puissant pour qu'il leur donne bonheur et santé et prospérité.

Mes chères belles-sœurs Abir et Dorra et mes chers beaux-frères Ali et Omar pour leur gentillesse, leur bonté, leur compréhension, et leurs encouragements. Qu'ils trouvent dans ce modeste travail l'expression de mon affection la plus sincère.

Mes chers amis et proches, dont l'affection et le soutien continus m'ont permis de préserver pour ne pas perdre de vue mon objectif.

Et enfin toutes les personnes qui se sont intéressées à mon travail et ne se sont pas fait faute de m'apporter leur concours par leurs nombreux conseils et l'intérêt que leur a suscité mon travail.

Raja

Remerciements

Je voudrais, en premier lieu, témoigner mes vifs et sincères remerciements au bon DIEU, notre Créateur incontestable, pour sa miséricorde, son secours, ses bénédictions et grâces.

Je tiens aussi à exprimer ma gratitude à mon directeur de thèse associé à l'université de Sfax, monsieur le maître de conférences Kais HADDAR et mon directeur de thèse associé à l'Université Aix-Marseille, monsieur le professeur Philippe BLACHE. Je les remercie pour leurs conseils judicieux et leurs critiques qui m'ont permis de mener à bien ce travail. Au-delà de l'apport scientifique et de recherche, je remercie messieurs HADDAR et BLACHE pour leur confiance, leurs encouragements, leur gentillesse et pour tout ce que j'ai appris grâce à eux.

Je remercie madame le professeur Lamia BELGUITH de l'université de Sfax d'avoir accepté de présider le jury de ma soutenance et pour son intérêt à ce travail.

Je remercie monsieur le professeur Emmanuel MORIN de l'université de Nantes et monsieur Mounir ZRIGUI de l'université de Monastir pour l'intérêt qu'ils ont porté de à mon sujet de thèse en acceptant d'être rapporteurs et pour l'analyse minutieuse qu'ils ont menée sur ce mémoire de thèse.

Je tiens aussi à adresser mes remerciements les plus sincères à madame le maître de conférences Marie CANDITO de l'université Paris DIDEROT pour avoir consacré du temps à ce mémoire de thèse en acceptant de l'examiner.

Merci à toute ma famille, à mes parents et à mon mari qui m'ont soutenu en toutes circonstances et sans eux, ce travail n'aurait pu arriver à terme. Merci pour la confiance qu'ils ont placé en moi.

Enfin, que tous ceux qui ont participé de près ou de loin à la réalisation de ce travail, trouvent ici le témoignage de ma profonde reconnaissance.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Motivations

La disponibilité de ressources linguistiques richement annotées est un enjeu déterminant pour le traitement automatique des langues et leurs applications, dans de nombreux domaines. Les informations associées aux données linguistiques brutes sont stratégiques sur plusieurs plans (e.g. sémantique, syntaxique, lexical). La possibilité de création de ce type de ressource de haut niveau s'appuie sur le développement de différentes méthodes facilitant leur extraction et leur exploitation. L'analyse syntaxique constitue l'un de ces axes de recherche participant à réaliser cet objectif. La tâche de cette analyse consiste à annoter les textes pour leur associer des informations de différents types concernant leurs unités et structures. Elle s'appuie généralement sur des grammaires développées à priori. Celles-ci peuvent être construites manuellement à l'aide d'experts en construisant par exemple les règles de production qui caractérisent les constructions de la langue traitée. Le développement de la grammaire peut être également réalisé automatiquement en induisant les règles à partir des textes déjà annotés comme les treebanks. Les textes de ces treebanks sont choisis par des linguistes et représentés sous forme de structures morphosyntaxiques à plusieurs niveaux d'analyse (lexical, syntaxique, sémantique).

Les grammaires qui peuvent être construites dans le cadre de l'analyse syntaxique sont très variées et se basent sur des formalismes offrant chacune une représentation déterminée répondant à certains besoins de recherche. Parmi ces formalismes, il existe ceux basés sur la satisfaction de contraintes. On trouve dans cette famille les Grammaires de Dépendances (GD) (Hays, 1964) ou les grammaires syntagmatiques dirigées par les têtes (Head-driven Phrase Structure Grammar, HPSG) (Pollard et Sag, 1994). Pour celles-ci, les contraintes portent sur des informations non élémentaires. Ce sont des arbres locaux dans le cas des HPSG, et des relations de dépendances pour les GD. Ces formalismes requièrent la construction d'une structure locale avant d'utiliser les contraintes qui s'y appliquent. La satisfaisabilité des contraintes ne peut alors pas être vérifiée sur un système de contraintes global. Une approche s'appuyant effectivement sur la satisfaction de contraintes doit pouvoir accéder directement aux valeurs des variables, sans passer par la construction d'une structure globale. Le formalisme de

grammaires de propriétés (GP) répond à ce besoin. Il offre une représentation décentralisée et locale de l'information linguistique. En effet, il est possible de représenter d'une manière indépendante, tout type d'information, mais également des informations incomplètes, partielles et non canoniques, difficilement prises en compte dans les approches classiques. De plus, la décentralisation des informations qui caractérise les GP permet de prendre en compte les différentes interprétations possibles pour une même entrée en fonction du contexte. Les GP décrivent pour chaque structure syntaxique ou catégorie un ensemble de contraintes, dites propriétés (syntaxiques, sémantiques, etc.). Les propriétés syntaxiques spécifient des relations pouvant exister entre les unités d'une même structure. Ces relations peuvent porter par exemple sur la précédence linéaire, l'unicité, l'obligation, l'adjacence, l'exigence, l'exclusion ou la dépendance. La flexibilité et la facilité de représentation des informations qui caractérisent le formalisme de GP nous ont conduits à proposer de nouvelles ressources linguistiques pour la description et le traitement de l'arabe.

Une des difficultés rencontrées pour le traitement de la langue arabe tient au manque de ressources (corpus, dictionnaires, lexiques, etc.) robustes et riches. En particulier, pour ce qui concerne la dimension syntaxique, chacun des treebanks arabes possède des qualités spécifiques qui ne retrouvent pas nécessairement dans les autres. Le treebank arabe ATB est considéré comme le treebank le plus riche en informations morphosyntaxiques et sémantiques. Il a été validé par des linguistes et il est utilisé comme entrée de plusieurs autres treebanks arabes. Cependant, ce treebank représente une ressource insuffisante dans sa forme actuelle puisqu'il ne couvre pas tous les phénomènes linguistiques particuliers et il reçoit des améliorations continuelles. Il s'agit cependant à ce jour de la ressource de référence dans le domaine.

Le départ de cette ressource (l'ATB) pour générer à l'aide du formalisme de GP de nouvelles ressources à large couverture, héritant les qualités du treebank d'origine tout en gagnant en temps de construction, semble être une directive avantageuse. Ainsi, l'enrichissement de l'ATB avec les propriétés d'une GP permet d'obtenir un nouveau treebank intégrant des informations syntaxiques. Ces informations enrichissantes ont été implicites dans le treebank d'origine. Ce dernier montre seulement des relations entre le nœud parent et le nœud fils dans un arbre syntaxique. En effet, ni l'ordre, ni la désignation des unités obligatoires, ni les statistiques sur les catégories ne sont mentionnés. La GP utilisée pour l'enrichissement est induite également à partir de l'ATB. Cet enrichissement peut être appliqué aux résultats d'analyse d'un analyseur état de l'art du domaine, le Stanford Parser, offrant la possibilité d'une évaluation s'appuyant sur un ensemble de mesures obtenues à partir de cette ressource. Les étiquettes des unités de

cette grammaire sont structurées selon une hiérarchie de types permettant la variation de leur degré de granularité, et par conséquent du degré de précision des informations. Sur la base de la GP obtenue, il est en plus possible de développer un analyseur syntaxique probabiliste basé sur les propriétés syntaxiques. Il s'agit du premier analyseur de ce type pouvant être appliqué à l'arabe. Une grammaire de propriétés lexicalisée probabiliste peut faire partie de son modèle d'apprentissage pour pouvoir affecter positivement le résultat d'analyse et caractériser ses structures syntaxiques avec les propriétés de ce modèle. La ressource résultante peut être bénéfique dans différents axes de recherches du domaine de traitement automatique des langues comme la traduction automatique, les systèmes de question-réponse ou le résumé automatique.

Problématique

Plusieurs problèmes, et qui constituent des verrous spécifiques pour le développement de ressources syntaxiques sont cités ci-dessous.

- **Spécificités de la langue arabe :** On parle surtout de l'absence des voyelles et de la nature agglutinative qui caractérisent les mots arabes. Les informations associées à ces mots sont manquantes. L'ATB, constituant l'entrée de nos contributions, fournit une variété d'informations (ou annotations) selon plusieurs formats et à différents niveaux d'analyse répondant à différents besoins.
- **Complexité de la GP :** Plusieurs facteurs peuvent rendre la complexité de la GP exponentielle. Ainsi, l'autorisation de description de tout type d'information et selon plusieurs interprétations dans les propriétés est coûteuse. La prise en compte de toutes les unités indépendamment de leur position, les relations distantes et les phénomènes de ce type multiplient le nombre de configurations à représenter. De plus, les catégories du treebank ATB à partir duquel est induite notre GP se caractérisent par une forte granularité. Cela peut affecter la taille des informations à représenter dans la GP. Il faut alors intégrer des mécanismes de contrôle réduisant ces catégories et optimisant l'interprétation des propriétés de la GP.
- **Interprétation des propriétés :** Certaines propriétés faciles à déduire, d'autres sont plus complexes, nécessitant des heuristiques voire un accès à des ressources externes. Ces ressources peuvent ne pas générer la même représentation structurelle des données que le treebank ATB.
- **Intégration des propriétés :** Les propriétés sont des informations syntaxiques décrivant une certaine catégorie syntaxique. L'intégration d'une propriété dans la description d'une structure syntaxique de l'ATB ou d'un résultat d'analyse de Stanford peut ne pas être directe

à cause de la différence de granularité des catégories dans le cas de l'enrichissement de treebanks ou de différence d'étiquetage dans le cas de l'enrichissement des résultats d'analyse.

- **Evaluation des propriétés :** Les propriétés sont des relations entre deux unités d'une structure syntaxique. La difficulté est de développer des modules suffisamment robustes permettant l'évaluation de la satisfaction de ces propriétés sur d'autres structures syntaxiques ayant la même étiquette.
- **Choix des mesures d'évaluation des résultats d'analyse de Stanford parser :** Les propriétés ne sont pas des informations numériques pour les utiliser directement pour évaluer un résultat d'analyse.

Contributions

En admettant dans notre thèse, la mise en œuvre de la problématique décrite ci-dessus, il est nécessaire pour nous de suivre un ensemble de démarches intégrant différents mécanismes de contrôles. L'application de ces démarches revient à réaliser les objectifs suivants, et qui constituent autant de contributions proposées dans le cadre de cette thèse. Ainsi, nous visons à construire une GP pour l'arabe à large couverture d'une façon contrôlée et rapide reposant sur une entrée validée par des linguistes : le treebank ATB. Nous allons générer cette GP selon une représentation syntagmatique dans laquelle différents types de propriétés sont encapsulés dans la description des catégories syntaxiques.

Nous allons étudier l'entrée (le treebank ATB) à partir de laquelle la grammaire sera induite. Cela permet d'en filtrer les informations fournies afin de choisir le format qui répond aux besoins d'induction de la GP et aux exigences de traitement du corpus de test par notre analyseur syntaxique probabiliste de propriétés. Notre étude concernera également les ressources externes à exploiter pour interpréter certains types de propriétés tout en adaptant leur représentation à celle d l'ATB. Nous allons procéder à l'application des formalismes nécessaires pour définir des critères de contrôle sur les catégories grammaticales de l'entrée permettant la possibilité de varier leur granularité et par conséquent de renforcer leur réutilisation.

A l'aide de la grammaire obtenue, nous cherchons à créer en plus un nouveau treebank de l'arabe enrichi par des informations syntaxiques reposant sur la GP construite. Cet enrichissement ne représente pas un simple ajout d'annotations plus riches mais plutôt une hybridation du treebank d'origine par un nouveau formalisme. Pour cela nous allons représenter les catégories du nouveau treebank selon un format adéquat capable d'héberger les nouvelles informations syntaxiques. Nous allons également mettre en correspondance les catégories de la

structure syntaxique du treebank ou du résultat d'analyse avec celles de la GP pour pouvoir accéder aux propriétés qui la décrivent. Nous implantons dans ce cadre la variation de granularité des catégories du treebank à enrichir selon celles de la GP et nous développons une table de correspondance entre l'étiquetage des catégories du résultat d'analyse et celui de la GP. Nous développons en plus des solveurs de contraintes permettant l'évaluation des propriétés qui décrivent une structure syntaxique du treebank ou un résultat d'analyse. Leur évaluation se base sur la vérification de leur satisfaction dans cette structure, ou encore la vérification de la présence ou de l'absence des unités membres d'une telle propriété dans cette structure. Nous allons exploiter les statistiques sur les propriétés obtenues de l'enrichissement d'un certain résultat d'analyse donné dans des mesures d'évaluation de l'analyseur Stanford Parser.

A l'aide de la GP et en s'inspirant de l'enrichissement des treebanks, nous allons développer un analyseur syntaxique probabiliste de propriétés en se basant sur la GP construite. Nous estimons améliorer l'efficacité de l'analyseur en intégrant dans le modèle d'apprentissage une GP composée de données lexicalisées analysées d'une manière approfondie. Nous réalisons des expérimentations de l'analyseur proposé sur un corpus de test, tout en les comparants avec celles appliquées pour les résultats d'analyse enrichis de Stanford Parser. Pour avoir une analyse cohérente, nous devons adapter le modèle d'apprentissage construit selon les formes exigées par l'algorithme d'analyse choisi.

Les ressources linguistiques résultant de la réalisation des objectifs indiqués ci-dessus héritent des qualités qui caractérisent le treebank source ATB. Parmi celles-ci, nous citons la conformité de son étiquetage à des consensus connus, la validation par des linguistes de ses annotations et la richesse linguistique de son corpus source. Les ressources à obtenir se distinguent également par la granularité variable de leurs catégories permettant ainsi leur génération en différentes versions, ce qui favorise leur robustesse et leur large couverture.

Organisation du mémoire de thèse

Le présent mémoire de thèse s'articule principalement autour de trois grandes parties.

1. La première partie sera consacrée à la présentation d'un état de l'art sur les trois principaux thèmes de notre thèse, à savoir l'analyse syntaxique, les grammaires de propriétés et les treebanks. Trois chapitres successifs traitent ces thèmes.

Le premier chapitre présente l'architecture générale d'un analyseur syntaxique, et les étapes de prétraitement réalisées pour générer l'entrée de l'analyseur syntaxique. Il montre ensuite les représentations structurelles possibles de sa sortie et revient sur les deux composants principaux, qui sont la grammaire et l'algorithme d'analyse.

Le deuxième chapitre porte sur le formalisme de GP. Avant de présenter son principe, il explique tout d'abord les notions fondamentales liées aux contraintes. Il passe ensuite à la définition des problèmes de satisfaction de contraintes et des solutions adoptées pour ce type de problème. Par la suite, il montre l'utilité de l'accès aux contraintes selon plusieurs formalismes. Finalement, il propose un aperçu sur le formalisme de GP, ses composants et ses domaines d'utilisation.

Le troisième chapitre commence par définir la notion de treebank tout en la distinguant d'autres notions proches. Il présente ensuite leurs formats de conception en spécifiant leurs différentes caractéristiques ainsi que les schémas d'annotations qu'ils peuvent adopter. Puis, il décrit les méthodes et les outils de développement de ces treebanks. Finalement, il décrit les treebanks les plus connus, et qui sont développés pour la langue arabe que nous allons traiter. Cet aperçu est représenté sous forme d'une comparaison entre ces différents treebanks en fonction de plusieurs facteurs dans le but de choisir le treebank le plus convenable avec les besoins de nos contributions.

2. La deuxième partie de notre thèse porte sur les démarches de développement que nous avons proposées pour mettre en œuvre nos contributions.

Le quatrième chapitre se focalise sur la description de la démarche d'induction d'une GP arabe à granularité variable à partir du treebank arabe ATB. Il justifie le choix du treebank ATB en tant que source d'induction et fournit une étude détaillée de ce treebank pour nous faciliter son exploitation.

Le cinquième chapitre concerne les démarches d'enrichissement et d'évaluation appliquées sur des textes annotés syntaxiquement pour les convertir en des ressources riches de propriétés syntaxiques. Ce chapitre fournit une étude empirique d'un ensemble de syntagmes arabes pour définir les propriétés syntaxiques qui peuvent les enrichir. Il aborde les étapes à réaliser pour enrichir le treebank ATB en décrivant les solveurs de contraintes appliqués pour ceci. Il montre enfin les étapes d'enrichissement et d'évaluation des résultats d'analyse de Stanford Parser avec des propriétés syntaxiques.

Le sixième chapitre propose une description du moteur de notre analyseur syntaxique probabiliste de propriétés. Il illustre alors son architecture générale et présente respectivement le déroulement de ses deux composants principaux : le modèle d'apprentissage et l'algorithme d'analyse.

3. La dernière partie de ce mémoire présente les expérimentations suivies des interprétations en découlant, qui sont respectives aux démarches des trois chapitres de la partie précédente.

Ainsi, le septième chapitre présente les expérimentations ainsi que les discussions nécessaires obtenues de la mise en œuvre de la démarche d'induction d'une GP arabe à granularité variable. Les expérimentations décrivent les entrées et les sorties. Un jeu de la granularité des catégories est également appliqué dans les expérimentations pour déterminer son effet sur la validité des propriétés induites.

Le huitième chapitre décrit les différentes expérimentations concernant l'enrichissement du treebank ATB et l'enrichissement et l'évaluation du résultat d'analyse de Stanford Parser. Les expérimentations portent sur les entrées à savoir la GP utilisée, l'ATB et le résultat d'analyse, et sur les sorties qui sont des textes annotées en plus avec des propriétés syntaxiques. Les résultats des mesures d'évaluation du résultat d'analyse sont en plus interprétés.

Le neuvième chapitre présente finalement ses expérimentations et discussions dans trois sections respectivement sur les caractéristiques des entrées de notre méthode, sur celles du modèle d'apprentissage construit et sur celles du résultat d'analyse avec l'algorithme CYK. Les résultats détaillés obtenus de ces expérimentations portent sur les catégories lexicalisées, les propriétés et les règles et leurs suites de catégories lexicales.

Première partie

Etat de l’art

Chapitre 1

Analyse syntaxique

1 Introduction

Nous considérons que la tâche d'analyse syntaxique occupe un rôle important dans le processus de traitement automatique du langage naturel. En effet, plusieurs domaines de recherche exécutant ce processus comme la traduction automatique, la recherche d'information et la résolution des ambiguïtés linguistiques ont besoin de l'analyse syntaxique. Plus cette tâche est correctement accomplie, plus les performances des systèmes qui l'utilisent sont élevées. Pour cela, les efforts de recherche s'accumulent sans cesse pour offrir de nouveaux outils d'analyse syntaxique de plus en plus performants.

Le présent chapitre sera consacré à la présentation de quelques outils de prétraitement et d'analyse syntaxique et plus particulièrement à ceux adoptant une approche statistique. Il est alors organisé comme suit : la section 2 donnera un aperçu général sur le processus d'analyse syntaxique. La section 3 sera consacrée à l'explication des étapes de prétraitement pour une analyse syntaxique. Les représentations structurelles possibles de la sortie de l'analyse syntaxique seront exposées dans la section 4. Tandis que les sections 5 et 6 feront respectivement l'objet des deux composants principaux utilisés par l'analyseur syntaxique à savoir : la grammaire et l'algorithme d'analyse.

2 Analyse syntaxique

L'analyse syntaxique permet de traiter une phrase et de lui affecter une certaine structure syntaxique. Cette structure peut avoir différentes représentations possibles. La phrase à analyser syntaxiquement fait partie d'un énoncé pouvant avoir la forme écrite ou parlée. Cet énoncé peut éventuellement subir un prétraitement avant l'analyse de ses phrases à proprement parler. Dans plusieurs cas, l'analyse syntaxique n'est pas considérée comme objectif final mais plutôt une étape intermédiaire pour un traitement postérieur comme la détermination du sens de la phrase ou encore de son contexte.

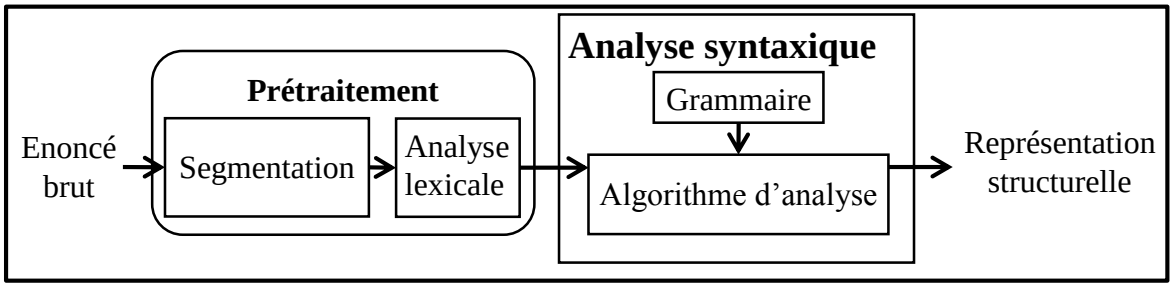


Figure 1.1 Processus général de prétraitement et d'analyse syntaxique

Comme montré dans la figure 1.1, la phase de prétraitement passe par des étapes de segmentation et d'analyse lexicale d'un énoncé brut. Tandis que la phase d'analyse se base sur la création d'une grammaire pour faire tourner son algorithme d'analyse.

3 Prétraitement

Avant d'établir l'analyse syntaxique, les phrases entrées doivent être prétraitées. Deux étapes principales doivent être appliquées dans le cadre du prétraitement de ces phrases, à savoir la segmentation et l'analyse lexicale. La sous-section suivante montre le processus général de prétraitement à travers un exemple de texte arabe. Les deux autres sous-sections donnent plus de détails sur chaque étape, et plus particulièrement dans le cas du traitement de l'arabe.

3.1 Processus général de prétraitement

La phase de prétraitement part d'un texte arabe brut pour le générer bien segmenté et analysé lexicalement. Ceci nécessite tout d'abord l'application d'une étape de segmentation des phrases et mots. Il faut ensuite établir l'analyse lexicale de chaque mot segmenté. Celle-ci se base sur l'étiquetage de la partie de discours (Part-Of-Speech, POS) de chaque mot, son analyse morphologique, et sa désambiguïsation morphologique. Voici un exemple de texte arabe :

Texte arabe	أبصرنا حسنا . فعدنا.
Translitération[1]	>abSarnA HasanAF. faEudonA
Traduction	Nous avons vu Hassan. Nous sommes alors revenus

La figure 1.2 montre l'application du processus de prétraitement sur le texte arabe ci-dessus :

Etapes \ **Texte brut**		أبصرنا حسنا . فعدنا .					
Segmentation	**en Phrases**	أبصرنا حسنا .			فعدنا.		
	en Mots	أبصرنا	حسنا	.	ف	عدنا	.
Analyse lexicale	**Etiquetage**	-Verbe au passé 1ère personne du pluriel -Verbe passif	-Verbe au passé 3ème personne du pluriel (dual) -Nom propre accusatif -Nom singulier indéfini accusatif -Adjectif accusatif	Ponctuation	…	…	
	Désambiguïsation	-Verbe au passé 1ère personne du pluriel	-Nom propre accusatif	Ponctuation			

Figure 1.2 : Processus de prétraitement de l'analyse syntaxique d'un texte arabe brut

[1] Système de translitération de Buckwalter : http:qamus.org/transliteration.htm

Dans la figure 1.2 le texte arabe prétraité est premièrement segmenté en deux phrases. Pour chaque phrase, chaque mot représentant une agglutination est segmenté à son tour. Par exemple, dans la deuxième phrase, le mot avant la segmentation «فعدنا. » est décomposé après la segmentation en trois mots. Ensuite, une affectation des différentes natures grammaticales possibles est réalisée à chacun des mots. Finalement, l'une de ces natures est choisie. Par exemple, le mot «أبصرنا» peut être un verbe à la forme active à la première personne du pluriel ou à la forme passive. Le choix s'est fait sur la première possibilité.

3.2 Segmentation

La segmentation concerne le processus de découpage d'un texte écrit en une suite de phrases décomposée à leur tour en une suite de tokens. Pour segmenter le texte en phrases, les signes de ponctuation, comme le point, le point d'exclamation, les deux points et les parenthèses, sont généralement considérés comme caractères de séparation. Dans certains cas, il est possible qu'on trouve des ambigüités de segmentation. Ceci est remarqué s'il existe par exemple des points dans les abréviations ou avec le chiffre décimal ou encore dans les dates.

Pour ce qui concerne la segmentation des mots, l'espace représente un caractère de séparation dans la phrase pour la plupart des cas (et pour la plupart des langues), même si ce paramètre à lui seul ne suffit pas à la segmentation. Cependant, il existe des mots écrits sous formes agglutinées. Ces formes doivent être décomposées dans le cadre d'une étape appelée « tokenization ». Ainsi, le terme « mot » ne peut pas identifier précisément la sortie de notre segmentation. On parle plutôt du terme « token » pour qualifier une unité après la séparation des agglutinations. Les agglutinations dans la langue arabe sont très fréquentes et peuvent ne pas être toutes détectées à cause de l'absence de vocalisation et de signes diacritiques. Ceci peut engendrer plusieurs ambiguïtés à différents niveaux d'analyse. Ainsi, le caractère "ف" (fa) dans le mot "فِرَقٌ" (firaqN, « groupes ») est un caractère original alors que dans le mot "فَرَقَّ" (faraq~a, « alors il s'attendrit »), il est un préfixe. Sans la vocalisation, ces deux mots sont factorisés en un seul mot.

Pour le problème de segmentation, plusieurs travaux ont proposé différentes solutions en fonction du domaine de recherche traité. La plupart de ces solutions réalisent la « tokenisation » en se basant sur une étape d'analyse morphologique des mots avant la séparation des agglutinations. Cette analyse permet de déterminer les frontières des composants internes (préfixe, base et suffixe) de chaque mot.

Dans (Boella et al., 2011), où la segmentation porte sur les hadiths[2], une première segmentation se réalise entre la chaîne des transmetteurs (sanad) et le sujet du hadith (matn). La première partie (la chaine des transmetteurs) contient des expressions fonctionnelles récurrentes basées sur des verbes et des prépositions. De ces expressions fonctionnelles, les auteurs se sont inspirés un ensemble d'expressions régulières à appliquer pour détecter chaque transmetteur et obtenir finalement un graphe de relations entre les transmetteurs de tous les hadiths ensembles. La deuxième partie, quant à elle a été segmentée à l'aide d'une version améliorée de l'analyseur morphologique BAMA v1.0 (Buckwalter, 2002). La segmentation de 7305 hadiths du livre SaHiyH AlbuxaAri[3] a permis d'atteindre, en termes de performance, 96.1% pour les « sanad» et 87.5% pour les « matn ».

Pour la segmentation des textes en unités discursives minimales (segments qui ne se chevauchent pas), Keskes et al. (2013) ont utilisé une approche d'apprentissage supervisée multi-classes. Ils ont plus précisément utilisé le classifieur de Stanford[4] basé sur le modèle d'entropie maximale. Le but est de prédire ces unités même si elles sont imbriquées. La classification a été réalisée sur la base d'une combinaison de traits typographiques, morphologiques, lexicaux, syntaxiques et sémantiques qui peuvent couvrir jusqu'à 174 connecteurs d'unités possibles. En l'absence des traits syntaxiques et sémantiques, ce segmenteur atteint ses meilleures performances à 88.5% pour les textes éducatifs (composés de 924 unités discursives minimales) et 84.5% pour des textes du corpus ATB (composés de 706 unités discursives minimales). La segmentation de ces unités pour des textes arabes oraux proposée par Lhioui et al. (2015) apporte également des performances encouragentes atteignant 79% pour 140 dialogues de demandes de réservation dans un hôtel. Cette segmentation se base sur une approche hybride combinant un processus linguistique et un autre probabiliste. Le premier intègre des expressions régulièrement pour séparer les agglutinations dans les mots arabes. Tandis que le processus probabiliste, il utilise une grammaire à contexte libre probabiliste pour regrouper les unités lexicales en unités discursives. Cette grammaire sont est représentée par un modèle de markov caché et ses probabilités sont estimées grâce à l'algorithme d'espérance-maximisation.

La segmentation de l'analyseur MADAMIRA (Pasha et al., 2014) applique une suite de composants représentant la combinaison des analyseurs MADA (Habash et Rambow, 2005 ;

[2] Hadiths : textes arabes qui transmettent les paroles et les actes du prophète Mohamed « Que la paix est la bénédiction de Dieu soient sur lui »
[3] http://hadith.al-islam.com/Loader.aspx?pageid=194&BookID=24
[4] https://nlp.stanford.edu/software/classifier.html

Roth et al., 2008 ; Habash et al., 2009 ; Habash et al., 2013) et AMIRA (Diab et al., 2007). En effet, chaque mot est tout d'abord analysé morphologiquement pour générer toutes les analyses possibles indépendantes du contexte du mot. Un composant de modélisation de traits morphologiques vient ensuite pour appliquer des modèles SVM et des modèles de langage sur le texte source et les analyses déjà générées. Le but est de prédire les traits relatifs à chaque mot. A ces prédictions, un composant de classement calcule des scores pour générer celles meilleures. Ces meilleures prédictions sont finalement employées par le composant « tokenizer » pour produire la segmentation du texte en tokens. Cette segmentation peut être adoptée par la suite pour produire la segmentation du texte en structures syntaxiques.

Stanford Segmenter prédit la segmentation correcte en utilisant un modèle CRF (Conditional Random Field) de Markov (Green et DeNero, 2012). Il a été appliqué dans le cadre de la traduction automatique de l'anglais vers l'arabe. Il propose de segmenter chaque hypothèse de traduction arabe en « tokens » durant l'étape de décodage (la recherche de la traduction optimale). Les tokens prennent la forme d'une séquence de caractère en se basant sur le modèle CRF. La segmentation est évaluée en fonction de l'accord entre les classes morpho-syntactiques prédites pour les tokens formés. La performance de cette segmentation atteint 98.6% en l'appliquant sur le corpus de développement des parties[5] 1-3 de l'ATB. Une extension de ce segmenteur (Monroe et al., 2014) a été conçue pour traiter en plus le dialecte égyptien. Les performances ont détérioré à 98.23% pour l'ATB mais ont dépassé 92.09% contre 91.60% pour le segmenteur de l'analyseur MADA-ARZ v4.0 (Habash et al., 2013) désigné spécialement au dialecte égyptien.

Le segmenteur Farasa (Abdelali et al, 2016) est basé sur un modèle SVM qui utilise une variété de lexiques et traits pour faire le classement des segmentations possibles d'un mot. Les traits calculent des scores (probabilités, probabilités conditionnelles, moyennes) sur les combinaisons de traits morphologiques (préfixe, suffixe, racine). La base de ces scores est une collection de modèles de racines, de lexiques généraux et de lexiques d'entités nommées. En bénéficiant de la richesse de ses ressources, Farasa génère ses segmentations dans un temps relativement très faible par rapport aux segmenteurs de MADAMIRA et de Stanford. Ses performances ont été comparables et même élevées par rapport aux deux autres segmenteurs en le testant sur les tâches recherche d'information et de traduction automatique de et vers l'arabe.

Pour un texte arabe bien écrit électroniquement, la segmentation, comme le montre la figure 1.2, représente la première étape à réaliser dans le processus de TALN. Toute erreur de

[5] LDC2010T13, LDC2011T09, LDC2010T08

traitement liée à cette étape peut conduire à d'autres erreurs aux niveaux d'analyse suivants : lexical, syntaxique, sémantique, etc., et inversement. Cette influence s'impose à tous les domaines de recherche en TALN. Ainsi, pour la langue arabe qui a une morphologie riche, une segmentation en tokens plus correcte a amélioré la performance aussi bien de la traduction automatique de l'arabe vers l'anglais (Habash et Sadat, 2006) que de l'analyse syntaxique arabe (Green et Manning, 2010).

3.3 Analyse lexicale

L'analyse lexicale est adoptée pour affecter à chaque mot segmenté une certaine étiquette identifiant la classe lexicale[6] de ce mot ainsi que ses propriétés spécifiques. Cette affectation est résumée dans les étapes nommées « étiquetage POS » et « analyse morphologique ». Dans certains cas, cette tâche ne peut pas être directement appliquée. Un mot peut avoir différentes étiquettes possibles. L'emploi d'une résolution de ce genre d'ambiguïtés lexicales représente elle-même une sous-étape indépendante qu'on appelle « la désambiguïsation morphologique ». Dans les deux sous-sous-sections suivantes, nous donnons un bref aperçu sur les termes déjà cités tout en montrant quelques travaux relatifs et plus particulièrement pour le traitement de la langue arabe.

3.3.1 Etiquetage POS et analyse morphologique

L'étiquetage de parties de discours (POS, Part-Of-Speech) nécessite généralement l'accès à un lexique construit à priori pour trouver toutes les étiquettes POS attribuées à chaque mot segmenté. Le lexique contient l'ensemble des mots qui sont reconnus par une langue donnée. Dans cette ressource linguistique, le mot n'est pas considéré comme une unité atomique. Il est représenté par une structure de composants internes appelés morphèmes qui qualifient ses propriétés morphosyntaxiques comme la nature, le genre, le nombre. Le lexique, qui offre ces propriétés, est construit en se basant généralement sur des règles lexicales. Ces règles montrent comment les combinaisons de propriétés ont été formées pour générer chaque mot reconnu par la langue. Parfois, grâce à ces règles, seulement les lemmes simples sont disponibles dans le lexique, et les formes fléchies sont générées automatiquement. L'utilisation d'un lexique pour

[6] La liste la plus simple et la moins détaillée de classes (ou catégories) lexicales regroupe le verbe, le nom, l'adverbe, l'adjectif, la préposition, la conjonction et le déterminant. Pour l'arabe, cette liste se limite à trois classes qui sont le nom, le verbe et la particule. Vu qu'elle ne donne pas une description détaillée de la structure interne du mot en relation, cette liste est juste utilisée par les linguistes comme un point de départ pour la sélection d'étiquettes POS beaucoup plus informatives.
.

l'analyse lexicale est beaucoup plus efficace en termes d'espace de stockage que d'avoir une grande liste de mots atomiques. C'est parce que dans le cas réel, il n'existe pas un lexique qui peut couvrir tous les mots du langage naturel. Pour un mot non reconnu par le lexique, il suffit de lui appliquer une analyse morphologique.

Il est à noter également que les lexiques peuvent avoir d'autres formes outre les propriétés morphosyntaxiques. Dans (Linard et al. 2016) par exemple, et dans le cadre de la traduction automatique statistique, les auteurs ont construit des lexiques bilingues à partir de corpus comparables tout en impliquant des langues pivots. Les mots de ces lexiques sont définis en se basant sur la détermination de la liste leurs contextes dans le corpus.

D'une manière générale, l'analyse morphologique utilise des règles lexicales pour délimiter les morphèmes d'un mot et les étiqueter par les informations nécessaires qui les qualifient. Pour la langue arabe, qui est caractérisée par une forte utilisation des formes agglutinatives, l'analyseur morphologique doit séparer du mot entré certains morphèmes dans plusieurs cas. C'est le cas notamment des préfixes comme les préfixes verbaux (ta–, ya–, na–) et des proclitiques comme les conjonctions wa– et fa–, les prépositions bi– et li–, l'article défini et des suffixes comme les suffixes verbaux (e.g –to, –taA, –Ani et –wna) et des enclitiques comme les pronoms possessifs (e.g. –hu, –haA et –humA). La solution que propose un analyseur morphologique arabe définit les affixes (préfixes, infixes et suffixes), les clitiques (proclitiques et enclitiques) et le radical du mot entré en se basant sur la détermination de la racine et du schème du mot. La racine arabe est une séquence de trois (rarement quatre ou cinq) consonnes appelées radicaux. Le schème est une combinaison de consonnes (C) et de voyelles (V) telles que les consonnes font référence aux radicaux de la racine. Par exemple :
Pour le mot : [7] "فارسه" (fArsh, « son chevalier »), nous obtenons la solution d'analyse suivante :

Racine	ف ر س (f r s)
Schème	CVCCV
Infixe	ا (A)
Enclitique	ه (h)

Les résultats d'analyse du mot donné dans l'exemple ci-dessus concernent 4 informations morphologiques. La racine montre que ce mot se base sur 3 lettres. Le schéma montre que la deuxième et la cinquième lettre du mot ne sont pas des lettres de base. La première est un infixe, l'autre est un enclitique.

Dans la littérature, différents travaux de recherche ont essayé de présenter les meilleures solutions d'analyse morphologique des mots arabes. Ainsi, dans (Rafea et Shalaan, 93),

[7] En arabe les textes sont lus de droite à gauche, les lettres du mot sont lues dans l'ordre suivant : ف, ا, ر, س

l'analyse morphologique porte sur les mots arabes fléchis. Pour un mot entré, il applique une recherche exhaustive dans un réseau de transitions augmenté pour déterminer les différentes analyses morphologiques possibles de ce mot tout en se référant à un lexique. Pour chaque analyse, on définit les composants internes suivants : les préfixes, le stem et les suffixes du mot. Le parcours dans ce réseau se base sur des heuristiques et des ensembles de règles associés aux arcs. Chaque ensemble de règles contient des conditions et des actions contextuelles et non contextuelles qu'il faut réaliser pour délimiter un composant interne du mot.

L'analyseur morphologique de Xerox[8] (Beesley, 2001) utilise la technologie de traducteurs à états finis sur des dictionnaires de racines et de schèmes des mots. Acceptant en entrée des mots voyellés ou pas, l'analyse se fait à deux niveaux : un niveau pour la détermination des racines des mots et de leurs schèmes grâce à un algorithme automatique qui les combinent. L'autre niveau permet d'analyser les préfixes, les clitiques et d'autres formes comme les prépositions et les conjonctions. Cet analyseur offre une large couverture des stems qui peut atteindre 90 milles.

BAMA (Buckwalter, 2002) est l'analyseur morphologique arabe proposé par Tim Buckwalter. Il est utilisé par le LDC (Linguistic Data Consortium). BAMA se base sur un dictionnaire de lexiques des stems arabes et des listes de préfixes et de suffixes. Il peut analyser jusqu'à 77800. Cet analyseur consiste à lister toutes les segmentations possibles d'un mot entré, puis à les chercher dans ces lexiques et listes d'affixes. Il vérifie ensuite si les composants internes sont compatibles entre eux en examinant les correspondances préfixe-stem, préfixe-suffixe et stem-suffixe. La version améliorée de BAMA est nommée SAMA (Standard Arabic Morphological Analyzer). Les mots que cette version peut analyser ainsi que le nombre de solutions d'analyse qu'elle peut proposer ont largement augmentés (Maamouri et al., 2010).

Dans MADAMIRA (Pasha et al., 2014), l'analyse morphologique est appliquée pour choisir la meilleure segmentation en tokens comme indiqué dans la section précédente.

Alkhalil Morpho Sys 2 (Boudchiche et al., 2016) est un analyseur morphosyntaxique qui peut traiter les mots non voyellés de l'arabe standard moderne. Il utilise une approche symbolique qui modélise une variété de règles morphologiques arabes. Ces règles se basent sur différentes ressources linguistiques comme la base de données des racines et des clitiques des mots et leurs schèmes voyéllés possibles. Ces schèmes peuvent couvrir plus que 4 millions stemms voyellés caractérisés par un grand nombre de traits morphosyntaxiques. Ceci a apporté

[8] https://open.xerox.com/Services/arabic-morphology

une couverture qui atteint 99.31% de mots analysés d'un ensemble de corpus[9] composés de plus de 72 millions mots incluant des signes diacritiques de l'arabe classique et de l'arabe moderne. La version ancienne de cet analyseur « Alkhalil Morpho Sys 1 » (Boudlal, 2010) et les analyseurs BAMA (Buckwalter, 2002), SAMA (Graff et al., 2009) ont apporté par contre des couvertures plus faibles (respectivement 88.51%, 80.13% et 90.18%). Il est à noter que la nouvelle version apporte des corrections et des enrichissements de la base de données des racines et clitiques des mots par rapport à celle ancienne.

3.3.2 Désambiguïsation morphologique

La désambiguïsation morphologique vise à sélectionner parmi plusieurs solutions possibles l'analyse morphologique correcte qui correspond à un mot entré. Cette sélection se base plusieurs facteurs comme la compatibilité contextuelle de cette solution avec les mots adjacents et son occurrence dans le corpus d'étude.

Dans la langue arabe, les mots sont généralement ambigus dans leur analyse morphologique. Dans le treebank ATB, un mot a en moyenne deux analyses morphologiques (Habash et Rambow, 2005). Il est souvent possible de trouver des mots qui sont composés des mêmes lettres mais qui sont prononcés différemment. Le manque ou l'absence de vocalisation par exemple peut générer des ambiguïtés à différents niveaux d'analyse débutant par les ambigüités morphologiques. Différents types de ces ambigüités ont été spécifié dans (Attia, 2006). Ci-dessous quelques types sont présentés :

Le premier type d'ambiguïtés concerne le cas de deux lemmes composés des mêmes lettres mais l'une possède un son doublé par rapport à l'autre qui n'est pas montré au niveau écriture.

Verbe "درس" (drs, ?)	
دَرَسَ darasa étudier	دَرَّسَ dar~asa enseigner

Le deuxième type d'ambiguïtés porte sur les mots dérivés d'un même lemme comme l'ambiguïté entre les formes active, passive et impérative de certains verbes.

Verbe "أقفل" (>qfl, « fermer »)		
أَقْفَلَ >aqofala ferme (active)	أُقْفِلَ >uqofila est fermé (passive)	أَقْفِلْ >aqofil fermes (impératif)

[9] Corpus Tashkeela : http://sourceforge.net/projects/tashkeela/. et le corpus RDI http://www.rdi-eg.com/RDI/TrainingData/.

Le troisième type d'ambigüités est lié aux mots dérivés de lemmes différents et ayant subi des opérations d'alternation orthographiques (comme la suppression et l'assimilation) :

Verbe "يصل" (ySl, ?)			
يَصِلُّ (أصلّ) yaSil~u (>aSal~a) Pourrir	**يَصِلُ** (وصل) yaSilu (waSala) Arriver	**يُصَلِّ** (صلّى) yuSal~i (Sal~aY) Prier	**يَصْلِ** (صلى) yaSoli (SalaY) Griller

Le quatrième type d'ambigüités concerne les préfixes ou les suffixes qui ont la même forme graphique comme les cas suivants :

- Le préfixe t- peut indiquer la 3ème personne du féminin ou la 2ème personne du masculin.

Verbe "تعمل" (taEomulu , travailler)	
تَعْمَلُ taEomulu tu travailles	**تَعْمَلُ** taEomulu elle travaille

- Le suffixe –t génère une confusion entre quatre traits possibles.

Verbe "عملت" (Eml-t , travailler)			
عملتُ Eamiltu j'ai travaillé	**عملتَ** Eamilta tu as travaillé (masc)	**عملتِ** Eamilti tu as travaillé (fém)	**عملتْ** Eamilato elle a travaillé

- Le suffixe du dual et le pluriel dans le cas accusatif ont les mêmes lettres :

Nom "العاملين" (AlEmlyn, « les travailleurs »)	
العَامِلَيْنِ AlEamilayni travailleurs (dual)	**العَامِلِينَ** AlEamiliyna Travailleurs (pluriel)

Parmi les solutions qui ont été proposées pour résoudre ces ambiguïtés morphologiques nous pouvons citer les deux exemples suivants : MADA (Morphological Analysis and Disambiguation for Arabic) (Habash et Rambow, 2005; Roth et al., 2008; Habash et al., 2009; Habash et al., 2013) utilise plus que 19 attributs pour choisir la meilleure analyse parmi les analyses possibles produites par l'analyseur BAMA (Buckwalter Arabic Morphological Analyzer) (Buckwalter, 2002) qui fonctionne avec le modèle SVM. MADA ajoute aux 14 attributs morphologiques générés par BAMA, cinq attributs qui se basent sur les statistiques des chaines n-grammes et leurs variations orthographiques. Les choix d'analyse effectués par MADA dépendent des scores qu'il affecte à la meilleure analyse dans chaque contexte traité.

Ayed et al. (2014) proposent une nouvelle approche de classification possibiliste pour désambiguïser 14 attributs morphologiques des textes arabes non voyellés. Cette approche

permet d'apprendre et de tester des données imprécises. Pour chaque mot donné, elle détermine pour chaque attribut les valeurs correspondant aux deux mots qui le précèdent et qui le suivent. Des modèles de classification ont été établis pour calculer les mesures de possibilité et de nécessité de chaque attribut morphologique décrivant le mot testé. Pour la plupart des attributs, cette approche apporte le meilleur taux de désambiguïsation par rapport aux classifieurs SVM, Bayésien Naïf et les arbres de décisions. La moyenne des taux des attributs dépasse les 87% pour ce classifieur contre moins de 77% pour les autres. La non intégralité de cette désambiguïsation revient à l'ordre relativement aléatoire des mots d'une phrase qui caractérise la langue arabe.

4 Représentations structurelles

La représentation structurelle est le résultat d'analyse syntaxique d'une phrase entrée. Celle-là prend la forme d'un arbre construit à base de constituants ou à base de dépendances. Les deux sous-sections suivantes donnent respectivement plus de détails concernant ces deux types de représentations.

4.1 Représentations à base de constituants

La représentation structurelle à base de constituants est un arbre dans lequel les mots d'une phrase apparaissent sous forme de feuilles et les nœuds non terminaux représentent des catégories syntaxiques qui regroupent des structures syntaxiques dans la phrase comme le syntagme nominal (Noun Phrase, NP) ou le syntagme verbal (Verbal Phrase, VP). C'est pour cette raison qu'on appelle les grammaires qui adoptent ce type de représentation les grammaires syntagmatiques. Plusieurs analyseurs syntaxiques (parsers) utilisent les grammaires syntagmatiques comme Stanford Parser (Green et Manning, 2010), Berkeley parser (Petrov, 2009) et Bikel parser (Bikel, 2004) qui a été employé dans l'analyse du treebank arabe Penn Arabic Treebank (ATB) (Maamouri et al., 2004). La figure 1.3 illustre la représentation en structure de constituants de la phrase suivante selon l'ATB :

" خمسون ألف سائح زاروا لبنان و سوريا في ايلول الماضي" (xmswn Alf sAˆyH zArwA lbnAn wswryA fy Aylwl AlmADy « 50 milles touristes ont visité le Liban et la Syrie en Septembre dernier. »).

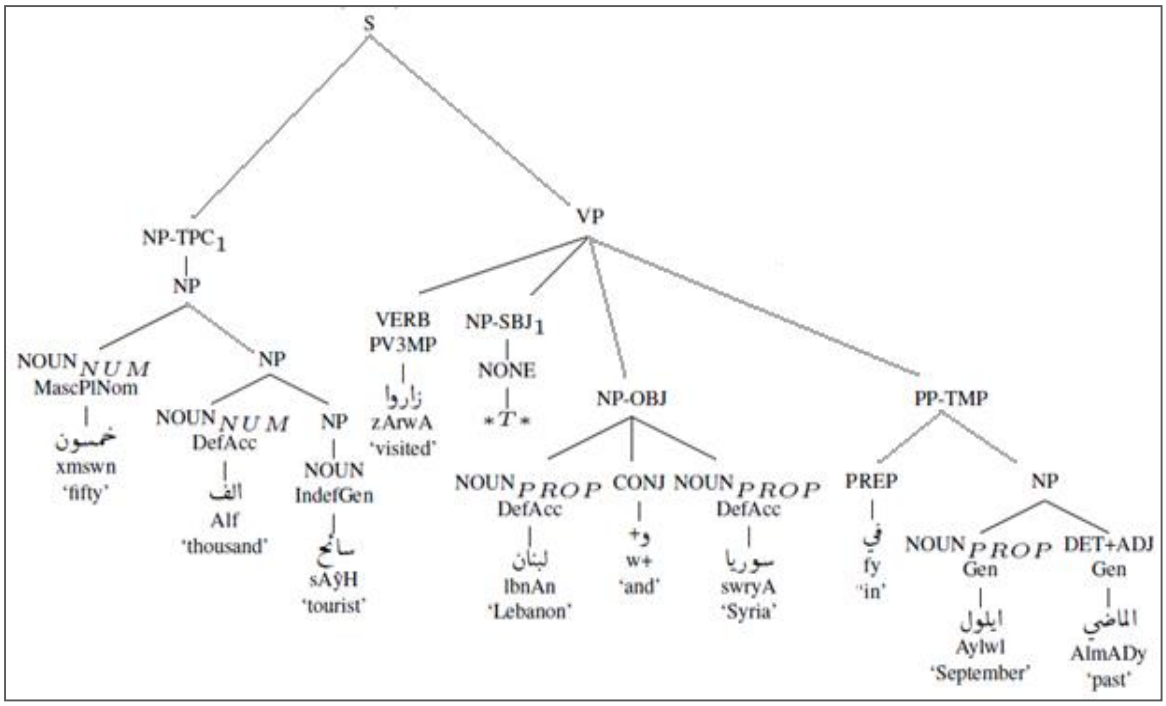


Figure 1.3 Exemple de représentation structurelle à base de constituants à partir de l'ATB (Habash et al. 2009)

La figure 1.3 suivante ci-dessus montre par exemple que les mots "ألف " (milles) et "سائح" (touristes) sont regroupés dans un syntagme étiqueté par la catégorie NP (syntagme nominal). Ce syntagme est encapsulé dans la structure "خمسون ألف سائح" (cinquante milles touristes) étiquetée elle-aussi par la catégorie NP.

4.2 Représentations à base de dépendances

Les représentations structurelles à base de dépendances prennent aussi les formes d'arbres à ceci près que les mots d'une phrase sont les nœuds de l'arbre. Tandis que les arcs qui relient un nœud à un autre représentent leurs relations de dépendances. Cette structure est très adaptée pour les treebanks arabes car elle n'est pas contraignante avec l'ordre relativement libre des mots dans la phrase qui caractérise la langue arabe. A l'exception de l'ATB, les treebank arabes les plus connus utilisent cette représentation structurelle dans leurs annotations à savoir : QADT (Quranic Arabic Dependency Treebank) (Dukes et al., 2010), PADT (Prague Arabic Dependency Treebank) (Hajic et al., 2004) et CATiB (Columbia Arabic Treebank) (Habash et al. 2009). Le treebank coranique QADT, plus particulièrement, offre une représentation hybride qui combine les dépendances avec la décomposition en syntagmes. La figure 1.4 illustre un exemple de représentation à base de dépendances selon le treebank CATiB. Tandis que la figure 1.5 montre une phrase annotée par le treebank QADT.

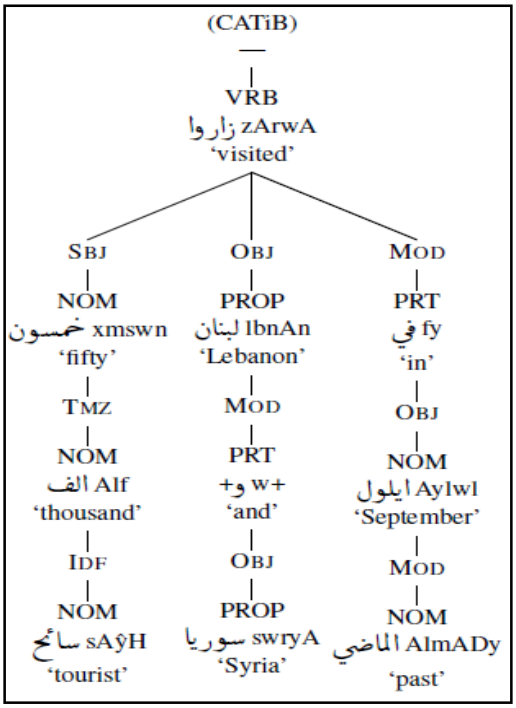


Figure 1.4 : Exemple de représentation structurelle à base de dépendances à partir de CATiB (Habash et al. 2009)

La figure ci-dessus montre un exemple de représentation structurelle à base de dépendances qui montre qu'il y a une relation de dépendance « sujet » (SBJ) entre le nœud "زاروا" étiqueté comme verbe (VRB) et le nœud "خمسون" étiqueté comme nom (NOM).

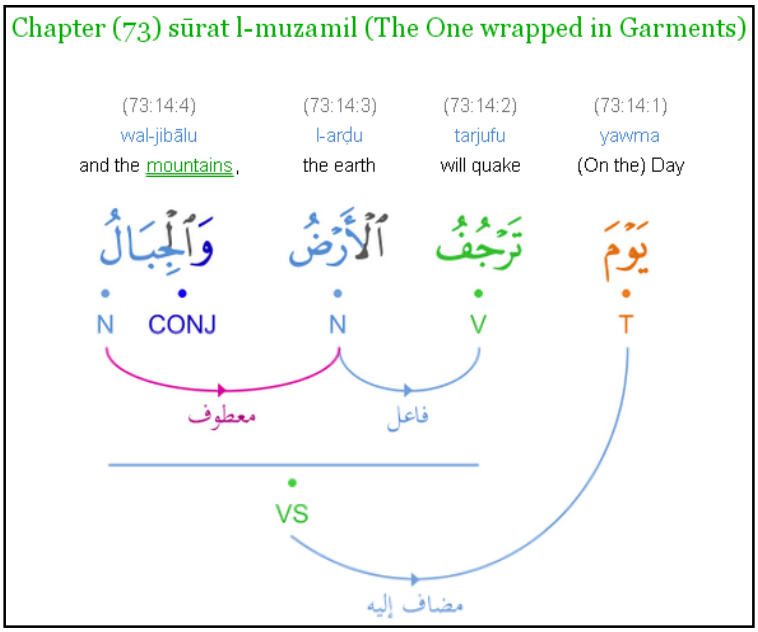


Figure 1.5: Exemple de représentation structurelle hybride à partir de QADT[10]

[10] http://corpus.quran.com/treebank.jsp?chapter=73&verse=14

L’exemple illustré de la figure 1.5 montre que la structure syntaxique "يوم ترجف الأرض و الجبال" contient la relation de dépendance"مضاف إليه" (est possédé par) entre son premier mot " يوم " et le reste qui représente lui-même la sous-structure VS (phrase verbale).

Les analyses représentées ci-dessus ont été construites en se basant sur deux grands composants à savoir : la grammaire et l’algorithme d’analyse. Les deux sections suivantes s’occupent de les présenter et de fournir quelques travaux qui y sont reliés.

5 Grammaire

La grammaire regroupe l’ensemble des règles linguistiques qui s'appliquent pour construire la structure syntaxique de la phrase. Dans leur ensemble, ces règles permettent de reconnaitre les phrases correctes d’une certaine langue. Autour de ces définitions, circulent certains concepts de base utilisés par les grammaires. Nous allons les identifier d’une manière formelle dans la sous-section suivante avant de citer quelques formalismes de grammaires.

5.1 Concepts de base formels

Tout langage, noté L, regroupe l’ensemble des chaines de caractères représentées par des suites de mots. L’alphabet de ce langage représente l’ensemble fini Σ qui regroupe tous les mots aperçus dans le langage L. Une phrase de ce langage est une séquence $S = (\alpha_1 \alpha_2 \ldots \alpha_n)$, où $\alpha_i \in L$ ($i \in [1, n]$ et $n \in \mathbb{N}$). Tout langage L respecte un ensemble de règles fournies par une grammaire donnée G. Cette grammaire est représentée selon un certain formalisme. La grammaire G est en fait une description du langage L. Elle porte sur un lexique et des règles. Le lexique fournit d’une manière structurée l’alphabet de L et les règles décrivent comment les mots du lexique sont concaténés pour produire des structures syntaxiques plus étendues.

5.2 Survol de quelques formalismes

Plusieurs formalismes ont été utilisés pour produire des grammaires dans l’analyse syntaxique. Les sous-sections suivantes sont consacrées à la description du fonctionnement de certains formalismes appliqués pour la langue arabe et à la présentation de quelques travaux justificatifs.

5.2.1 Grammaire à contexte libre

La grammaire à contexte libre, ou grammaire hors-contexte, (Bloomfield, 1933 ; Chomsky, 1959), appelée en anglais Context-Free Grammar (CFG), est une grammaire syntagmatique. Elle est définie formellement comme un 4-uplet G = (N, Σ, R, S) où :

1. N est un ensemble fini de symboles non terminaux,
2. Σ est un ensemble fini de symboles terminaux,
3. R est un ensemble fini de règles de production tel que R = {α → β | α ∈ N, β ∈ (N U Σ)*},
4. S est le symbole de départ qui est généralement le symbole des phrases.

Par exemple, la règle PP → Prep NP est une dérivation indique qu'un syntagme prépositionnel (PP) représentant un symbole non terminal peut être composé d'une préposition (Prep) et d'un syntagme nominal (NP). Dans la CFG, le symbole non terminal présent dans la partie droite de la règle (PP) peut être remplacé par sa partie gauche sans prise en compte du contexte dans lequel il se situe. Toute règle de la CFG spécifie deux relations : « la dominance immédiate » du nœud parent par rapport à ses fils (α domine β) et « la précédence linéaire » entre les nœuds sous-jacents d'un parent (l'ordre des symboles qui forment β). Des relations syntaxiques comme l'accord, la sous-catégorisation et l'unicité ne sont représentées par ce type de grammaires. Il est à noter que la CFG a été utilisée comme grammaire intermédiaire par plusieurs autres formalismes qui traitent les structures à base de constituants (Gazdar et al. 1985 ; Green et Manning, 2010).

5.2.2 Grammaires d'unification

Les formalismes appliqués par les grammaires d'unification utilisent à des degrés variés les contraintes. Ils offrent une représentation riche des informations linguistiques en spécifiant les différents attributs qui caractérisent les unités linguistiques à plusieurs niveaux d'analyse (e.g. morphologique, syntaxique, sémantique) selon une représentation homogène reposant sur des structures de traits. Une structure de traits est représentée par une matrice d'attributs-valeurs. Par exemple, la forme« écrivent » peut être décrite par la structure de traits suivante :

$$\begin{bmatrix} \text{Catégorie} & \text{verbe} \\ \text{Accord} & \begin{bmatrix} \text{Nombre} & \text{pluriel} \\ \text{Personne} & 3 \end{bmatrix} \\ \text{Temps} & \text{présent} \end{bmatrix}$$

Les grammaires d'unification peuvent également traiter des relations syntaxiques comme l'accord et la sous-catégorisation. Ceci permet de mieux modéliser les phénomènes linguistiques complexes comme les relatives et les coordinations. Les grammaires les plus connues qui adoptent ce formalisme sont les grammaires syntagmatiques généralisées (Generalized Phrase Structure Grammar, GPSG, Gazdar et al. 1985)**,** les grammaires syntagmatiques guidées par les têtes (Head-driven Phrase Structure Grammar, HPSG), (Pollard et Sag, 1994) ou encore les grammaires fonctionnelles lexicales (Lexical Functional Grammar, LFG, Kaplan et Bresnan 1982).

GPSG représente à la fois le niveau syntaxique et le niveau sémantique. Cette grammaire utilise les règles de la CFG non pas pour appliquer des transformations des structures grammaticales selon les dérivations mais plutôt pour générer des métarègles.

Son formalisme provenant de celui de GPSG, on considère que HPSG en est son successeur. Au lieu d'appliquer des transformations des structures grammaticales, elle fournit des représentations parallèles mutuellement contrôlées. Cette grammaire représente sous forme de structures de traits des entrées lexicales (appelées signes lexicaux), des règles grammaticales et des principes linguistiques universels et spécifiques. Les travaux de (Haddar et al., 2010) et (Bouekdi et Haddar, 2014) construisent des HPSGs pour traiter respectivement les phénomènes linguistiques arabes suivants : les relatives et les coordinations. Les deux grammaires ont été intégrées dans la plateforme LKB (Copestake, 2002) pour générer automatiquement des analyseurs syntaxiques. Au niveau performances, le premier analyseur génère 84% analyses non ambiguës d'un corpus de test à 500 phrases arabes contenant essentiellement des relatives. Tandis que le deuxième analyseur apporte un succès d'analyse de plus que 72% d'un extrait du corpus ATB composé de 600 phrases contenant 370 coordinations.

Les grammaires LFG bénéficient quant à elles à la fois des caractéristiques de la CFG et des réseaux de transitions augmentés[11]. LFG représente la phrase donnée par deux structures corrélées : la structure de constituants, C-Structure, qui est dérivée des règles de la CFG, et la structure fonctionnelle F-structure, qui est produite par les relations grammaticales (comme les fonctions de sujet et d'objet) des réseaux de transitions augmentés à travers des structures de traits unifiées. Le travail de (Tounsi et al., 2009) construit les annotations F-structures de la LFG par extraction automatique à partir de l'ATB des informations comme les catégories POS

[11] Les réseaux de transition augmentés (Bolc, 1983) se basent non pas sur des règles mais plutot sur des automates à états finis. Chaque automate correspond à un symbole non terminal. Il contient des transitions d'un état à un autre marquées par les catégories grammaticales. Une phrase n'est acceptée que si elle peut traverser des automates à partir d'un état initial jusqu'à un état final.

et les relations grammaticales comme le sujet et l'objet. L'évaluation des annotations obtenues produit une performance qui atteint 95 % parmi 250 phrases choisies aléatoirement de l'ATB. Cette annotation a été utilisée par (Attia et al., 2012) pour extraire automatiquement les cas de sous-catégorisation et les dépendances distantes pour plusieurs fins comme l'amélioration de l'analyse syntaxique en temps et en qualité.

5.2.3 Grammaires de dépendances

Les grammaires de dépendances (Hays, 1964) spécifient les relations de dépendances qui peuvent exister entre deux mots. Une relation est définie par le couple <dépendant, tête>. La tête (en anglais Head) est le mot duquel dépendent tous les autres mots de la phrase. Le dépendant peut être un complément ou un modifieur de la tête (e.g. l'adjectif est un modifieur qui dépend du nom). Dans sa forme de base, la grammaire de dépendances est un 4-uplets DG = (C, ∑, R, F) où :

- C : ensemble fini des catégories lexicales,
- ∑ : ensemble fini des symboles terminaux,
- R : ensemble fini des règles de dépendances entre les catégories de C,
- F : fonction d'affectation F : ∑ → C.

Il existe des extensions de cette grammaire où le dépendant peut avoir plus d'une tête, les règles sont remplacées par des contraintes ou encore l'application d'analyse multidimensionnelle des dépendances dans la phrase (dimension syntaxique, dimension sémantique) (Karlsson, 1990 ; Afonso et al. 2002 ; Debusmann et al. 2006).

5.2.4 Grammaires d'arbres adjoints

Dans les grammaires d'arbres adjoints (Joshi et al. 1975), les arbres sont utilisés comme des objets élémentaires. Une grammaire est un 5-uplets définie par TAG = (N, ∑, I, A, S) où :

- N : ensemble fini des symboles non terminaux,
- ∑ : ensemble fini des symboles terminaux,
- I : ensemble fini des arbres initiaux,
- A : ensemble fini des arbres auxiliaires,
- S ∈ N : symbole de départ de la phrase.

Pour générer la dérivation d'une phrase, on construit son arbre dérivé d'arbres initiaux (I) et d'arbres auxiliaires (A) en appliquant des opérations d'adjonction et/ou de substitution. Les arbres initiaux sont les structures syntagmatiques minimales non récursives. Tandis que les

arbres auxiliaires représentent les structures récursives ou les constituants s'attachant à des structures de base (comme les adverbes).

Le formalisme de grammaires d'arbres adjoints a été étendu (Schabes, 1992 ; Resnik, 1992) et a été appliqué sur plusieurs langues comme le français (Abeillé, 1988), le chinois (Bikel et Chiang, 2000), et l'arabe (Habash et Rambow, 2004). Pour la grammaire d'arbres adjoints arabe, elle a été extraite à partir de l'ATB pour le générer sous forme d'une représentation structurelle à base de dépendances. Habash et Rambow (2004) ont proposé des procédures spécifiques aux différentes formes de structures syntaxiques.

5.2.5 Grammaires catégorielles combinatoires

La grammaire catégorielle combinatoire (Steedman, 1985) est une extension du formalisme de grammaires catégorielles (Bar-Hillel, 1953). Ce dernier construit la grammaire grâce à la combinaison de catégories primitives (comme les noms et les verbes) pour créer des structures plus complexes. Cette combinaison ne se fait pas en se basant sur des règles mais plutôt sur la définition syntaxique des mots dans le lexique. Pour l'appliquer à l'analyse syntaxique et sémantique, l'extension de (Steedman, 1985) ajoute au lexique des règles combinatoires. Ces règles spécifient les catégories pouvant être combinées avec chaque entrée du lexique et les structures résultantes. La grammaire de cette extension peut être définie ainsi : CCG = (N, Σ, f, R, S) où :

- N : ensemble fini de symboles non terminaux,
- Σ : ensemble fini de symboles terminaux,
- f : $\Sigma \rightarrow C(N)$: fonction associant chaque élément de Σ à un sous-ensemble de C(N) ; C(N) est l'ensemble le plus petit qui réalise ces deux conditions :
    - $N \subseteq C(N)$
    - Si c1, c2 ∈ C(N) alors (c1/c2), (c1\c2) ∈ C(N),
- R : ensemble fini de règles combinatoires (avant ou après),
- S ∈ N : symbole de la phrase.

L'inconvénient de ce formalisme est qu'il est possible d'obtenir plusieurs dérivations possibles menées d'une même structure dérivée, ce qui représente une ambiguïté.

5.2.6 Grammaires de propriétés

Le formalisme de grammaires de propriétés (Blache, 2001) est une approche à base de contraintes. Il fournit une représentation locale et décentralisée des informations linguistiques directement sur les catégories syntaxiques. Cette représentation est indépendante du type et de

la position de l'information. Elle peut couvrir même les informations incomplètes, partielles et non canoniques. Une grammaire de propriétés GP est formée par un ensemble de propriétés indépendantes l'une de l'autre. Ces propriétés expriment différentes relations (syntaxiques, sémantiques, etc.) entre les catégories qui forment la structure syntaxique. Elles peuvent être très spécifiques (concernant un ensemble limité de catégories) ou au contraire très générales.

Vu que l'objectif de notre thèse consiste à exploiter le formalisme de grammaires de propriétés pour la construction de ressources linguistiques arabes, nous allons consacrer un chapitre entier pour une description détaillée de ce formalisme.

6 Algorithmes d'analyse

L'analyse consiste à utiliser les informations de la grammaire (typiquement des règles) pour la description d'une entrée donnée selon une certaine démarche dans le but de distinguer l'analyse correcte et la générer sous forme d'une certaine représentation structurelle. Cette démarche peut être observée de plusieurs points de vue. Elle peut signifier par exemple le sens de traitement de l'entrée (de gauche à droite ou de droite à gauche) ou bien la façon de parcours d'une représentation structurelle (parcours en profondeur ou parcours en largeur) ou encore le sens de construction de la représentation structurelle de l'entrée (analyse ascendante ou analyse descendante). Ainsi, l'analyse ascendante (en anglais buttom-up) (Glennie, 1960), vise à retrouver la dérivation de la phrase en entrée en partant des mots fournis par cette entrée et tente de construire des structures partielles de plus en plus larges jusqu'à ce qu'elle forme la phrase. Tandis que l'analyse descendante (en anglais top-down) (Yngve, 1959) réalise l'opération inverse en partant du niveau phrase et en essayant d'appliquer les règles pour retrouver le niveau mots. Ceci se fait en morcelant la phrase en éléments de plus en plus réduits jusqu'à atteindre les unités lexicales. Pour donner plus de détails concernant ces algorithmes, nous présentons, dans les deux sections suivantes, deux exemples très connus d'algorithmes d'analyse avant de citer les approches d'analyse suivies pour générer une grammaire utilisable par ces algorithmes.

6.1 Exemples d'algorithmes d'analyse

Parmi les algorithmes d'analyse syntaxique les plus connus, nous décrivons dans ce qui suit l'algorithme CYK (Kasami, 1965 ; Younger, 1967) et l'algorithme Earley (Earley, 1970). Ces algorithmes peuvent fonctionner sur une CFG probabiliste, la grammaire qui nous intéresse le plus puisque nous traitons dans notre thèse l'analyse syntaxique probabiliste. Nous nous occupons alors de les décrire dans les deux sous-sous-sections suivantes :

6.1.1 Algorithme CYK

L'algorithme de Cocke, Younger et Kasami (CYK), suit une approche non déterministe qui travaille directement sur la grammaire. Cette grammaire doit être adaptée à la forme normale de Chomsky (en anglais Chomsky Normal Form, CNF) (Chomsky, 1959) qui n'accepte que les deux formes de règles suivantes : $X \rightarrow Y\ Z$ et $X \rightarrow a$ ($\forall X, Y, Z \in N$ et $\forall a \in \Sigma$). La phrase en entrée de cet algorithme est analysée de manière ascendante (Bottom-up) dans un temps de calcul en ordre de n^3 où n représente la longueur de la phrase entrée. Le principe général de l'algorithme est simple et se résume comme suit : L'analyse d'une phrase S non vide est réalisée dynamiquement en commençant par les structures élémentaires. Ainsi, on définit pour la sous-chaine de mots S[i..j] (pour tout i, j tels que $1 \leq i \leq j \leq n$), l'ensemble des non-terminaux P[i, j] qui l'engendrent. Ces ensembles sont construits par récurrence sur plusieurs niveaux en fonction de la longueur de la sous-chaine traitée ayant la valeur |j-i| avec $i \leq j$. Si la longueur est nulle, l'analyse porte sur un seul mot de S. Dans ce cas, P[i, i] est l'ensemble des non-terminaux N tel que $N \rightarrow S[i]$ est une règle de la grammaire. Sinon, l'analyse porte sur une sous-chaine de mots de S de longueur supérieure à 1. Dans ce cas, P[i, j] est l'ensemble des non-terminaux N tels qu'il existe une règle $N \rightarrow B\ C$ de la grammaire où B et C sont des non-terminaux tels que $B \in P[i, k]$ et $C \in P[k+1, j]$ (avec $k \in [i, j-1]$). L'analyse de la phrase est réussie si cette phrase est engendrée par la grammaire. Cela veut dire que l'ensemble P[1, n] contient l'axiome de la grammaire noté S.

6.1.2 Algorithme d'Earley

L'algorithme d'Earley adopte également une approche non déterministe et effectue l'analyse dans un temps de calcul en l'ordre de n^3. Cette analyse est faite d'une manière dynamique mais descendante (top-down), à l'opposé de l'algorithme CYK. Le principe de cet algorithme consiste à stocker des ensembles d'états (définis par des items) dans des tables T[j] avec $j \in [0, n]$ dans n+1 grandes étapes. Un item est caractérisé par la règle de production à appliquer, la position du mot de S qui a été reconnu le dernier dans cette règle et la position du mot de S qui a été reconnu le premier dans cette règle. A l'étape 0, les items à stocker dans T[0], sont ceux relatifs aux règles de production qui dérivent de l'axiome de la grammaire. A l'étape j (avec $j > 0$), on stocke dans T[j] les items qui peuvent reconnaitre les sous-chaines S[i..j] quelque soit $i \in [1, j[$. Cette reconnaissance se fait d'une manière progressive en affectant et en utilisant les ensembles T[0]..T[j-1]. Pour cela, plusieurs items portant sur une même règle de production peuvent être stockés dans différentes tables T[j] en fonction du niveau de

progression de la reconnaissance des S[i..j]. Une reconnaissance d'une sous-chaine passe par une succession de plusieurs opérations de lecture, de prédiction et de complétion. La lecture consiste à avancer dans l'analyse d'un mot de S à son successeur en se basant sur une prédiction. La prédiction consiste à trouver une dérivation d'un un non-terminal à travers une règle de production. La complétion vient pour valider la reconnaissance de toute une sous-chaine S[i..j]. L'analyse de S sera réussie si et seulement si on peut stocker dans la table T[n] un item caractérisé par une règle de production S → A validée par une opération de complétion (avec A est un non-terminal).

Pour les phrases longues, l'algorithme d'Earley parait plus efficace que l'algorithme CYK au niveau espace de stockage car il ne charge dans ses tables que les items sur les constituants qu'il a besoin. Cependant, cela produit un travail redondant dans certain cas ce qui n'est pas le cas avec l'algorithme CYK qui charge tous les constituants.

Comme résultat d'analyse, le produit des deux algorithmes est un ensemble de solutions possibles. Ils n'arrivent pas à résoudre l'ambiguïté syntaxique par la génération de l'analyse correcte.

Généralement, l'analyse ascendante (celle suivie par l'algorithme CYK) est plus bénéfique que celle descendante surtout avec les grammaires trop récursives comme la langue arabe. L'avantage que présente l'analyse descendante est qu'elle n'explore pas les dérivations guidant à une fausse racine qui est l'axiome.

6.2 Approches d'analyse

Les approches d'analyse montrent les méthodes suivies pour générer les grammaires utilisées dans l'analyse syntaxique d'un corpus de test. Il existe deux familles d'approches avec lesquelles un analyseur syntaxique peut fonctionner à savoir : l'approche symbolique et l'approche statistique. Les sous-sous-sections sont consacrées à la présentation de quelques exemples d'analyseurs faisant partie respectivement de ces deux approches.

6.2.1 Analyseurs syntaxiques symboliques

Les analyseurs syntaxiques symboliques se basent sur des règles écrites manuellement par des linguistes. Ceci permet de représenter des relations syntaxiques très détaillées et les phénomènes linguistiques spécifiques au langage traité. Dans la littérature, il existe plusieurs analyseurs à base de règles manuelles. Nous présentons quelques-uns destinés pour le traitement de la langue arabe.

Ainsi, l'analyseur d'Othman et al. (2003) suit une approche ascendante à base de règles manuelles. L'analyseur morphologique qu'il utilise, adopte la technique de Réseaux de Transitions Augmentés (RTA) (Rafea et al., 1993) pour générer ses annotations. Sa grammaire est formée de l'ensemble des règles syntaxiques extraites manuellement de textes de l'arabe standard moderne portant sur le domaine de l'agriculture. Chaque règle est en outre caractérisée par des contraintes syntaxiques et sémantiques.

L'analyseur d'Al-Daoud et Basata (2009) se focalise sur le traitement des phrases arabes verbales simples. Il accomplit son analyse lexicale en se basant sur les racines de mots arabes fournies par un lexique. Son analyse syntaxique se réfère à une CFG créée manuellement. Elle suit un algorithme d'analyse avec une démarche ascendante et exécute des procédures mutuellement récursives appliquant chacune une règle de la CFG. En cas d'ambiguïté lexicale ou syntaxique, cet analyseur offre la possibilité d'interaction avec l'utilisateur pour choisir la solution la plus cohérente.

Boukedi et al. (2014) ont développé une grammaire HPSG (Head-driven Phrase Structure grammar) qui peut représenter les différentes formes de coordination en se basant sur une topologie étudiée à priori. La figure 1.6 suivante montre la représentation de la matrice d'attributs valeurs (AVM) de la conjonction arabe modélisée dans cette grammaire.

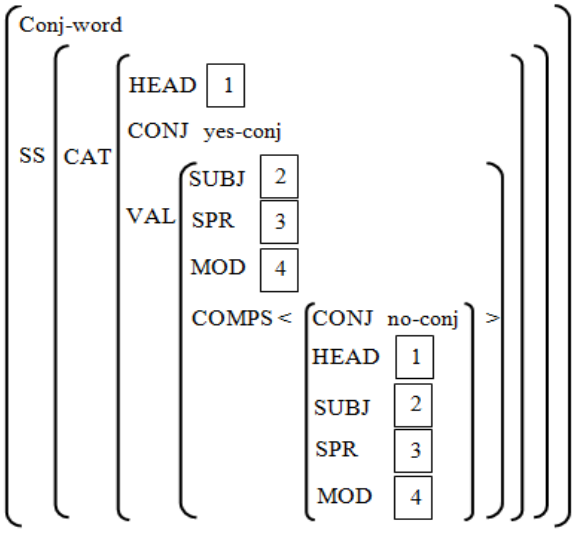


Figure 1.6 : Représentation AVM de la conjonction arabe

La représentation ci-dessus indique que la conjonction, étant le « Head » de la structure de coordination, doit avoir la même valence (notée VAL) que son complément (noté COMPS). Cette grammaire a été expérimentée dans le générateur automatique d'analyseurs LKB (Linguistic Knowledge Building). Ceci est fait en spécifiant les règles syntaxiques de la grammaire et en langage de description de types (en anglais TDL). Dans le but d'augmenter

l'extensibilité du lexique, Boukedi et al, ont exploité la spécification en TDL pour introduire également des règles lexicales permettant de générer automatiquement les formes dérivées de chaque entrée. Comme exemple, la figure 1.7 suivante montre la spécification en TDL d'une règle lexicale.

```
verbe-accompli-fem-sing-3p-lr :=
%suffix (* ْت)
  l2m-flex &
  [SS #synsem &
      [LOC[CAT [TETE[ASPECT ماضي, DEC مبني],
                VAL.SUJ < [LOC [
                                CONT.IND [PER 3e,
                                          NOMB مفرد,
                                          GEN مؤنث]]] >],
           CONT. IND[PER 3e,
                     NOMB مفرد,
                     GEN مؤنث]]],
   ARGS < [SS #synsem] >].
```

Figure 1.7: Exemple de règle lexicale spécifiée en TDL

La règle illustrée par la figure 1.7 ci-dessus permet de conjuguer un verbe à la troisième (3e) personne du féminin (مؤنث) singulier (مفرد) au passé composé (ماضي). Le terme %suffix ajoute une terminaison à la forme canonique indiquée dans le lexique. L'évaluation de cette analyse de 600 phrases contenant 370 coordinations a apporté une F-mesure de 64,75%. Le tableau suivant récapitule les caractéristiques de chacun des analyseurs proposés ci-dessus.

Analyseurs	Grammaire	Niveaux de Règles	Domaine des textes	Technique	Approche	Couverture
(Othman et al., 2003)	CDG	Syn/Sem	agriculture	RTA	ascendante	Tout
(Al-Daoud et Basata, 2009)	CFG	Lex/Syn	-	Procédures mutuelles récursives	ascendante	Phrases verbales simples
(Boukedi et al., 2014)	HPSG	Lex/Syn/Sem	Presse (ATB)	LKB	-	+ Phénomène de coordination

Tableau 1.1 : Tableau récapitulatif d'analyseurs symboliques arabes

A partir du tableau ci-dessus, nous remarquons que l'analyseur de Boukedi et al. (2014) destiné au traitement du phénomène de coordination utilise différents niveaux de règles (lexicales (Lex), syntaxiques (Syn) et sémantiques (Sem)) en le comparant avec les deux autres analyseurs. Pour déterminer ses règles, cet analyseur se base sur une ressource bien validée qui est l'ATB.

Malgré l'avantage acquis des analyseurs symboliques, ils souffrent de plusieurs limites (Geman et Johnson, 2002 ; Nivre, 2006) à savoir : La construction de la grammaire est très coûteuse car elle nécessite le traitement d'une quantité importante de textes. Plus cette quantité est petite, plus la performance de l'analyseur généré est faible. En plus, la grammaire construite

reste incomplète car l'effort manuel qu'elle coûte est énorme et la compréhension des structures syntaxiques ne couvre pas toujours tous les cas possibles. Cette grammaire peut générer également des analyses correctes syntaxiquement mais ambiguës sémantiquement. Finalement, les analyseurs symboliques ne sont pas très robustes vu qu'il est possible qu'aucune analyse ne peut être générée pour une phrase entrée.

6.2.2 Analyseurs syntaxiques statistiques

L'approche statistique, qui a connu historiquement un grand succès en reconnaissance de parole, a été proposée pour surmonter les limites des analyseurs syntaxiques symboliques. En effet, au lieu d'écrire des règles manuellement, on applique une phase d'apprentissage sur un grand corpus de textes pour produire un modèle intégrant par exemple les règles de la grammaire et estimant la distribution de ces règles et de plusieurs autres données dans ce corpus. La phase de test vient après pour analyser un ensemble X de nouvelles phrases. Ceci passe par deux grandes étapes : la génération de l'ensemble GEN(x) regroupant les analyses possibles de chaque phrase $x \in X$, et l'évaluation de ces analyses $y \in GEN(x)$ pour sélectionner celle meilleure en se basant sur une fonction de rangement des scores d'analyse EVAL(y). Ce rangement facilite la tâche de désambiguïsation et réduit les contraintes imposées sur le langage généré par la grammaire, ce qui favorise la robustesse de l'analyseur. Un tel modèle peut être également généralisé pour d'autres langues ou domaines. La seule contrainte de ces modèles réside dans le fait que les modèles statistiques marquant les meilleures performances sont appris à partir de corpus annotés (e.g. les treebanks) rares, coûteux et spécifiques à quelques domaines particuliers comme la presse et la médecine. Ceci n'assure pas l'obtention d'une grammaire à large couverture et la performance de l'analyseur diminue lorsqu'on analyse des phrases d'un domaine différent de celui caractérisant le corpus d'apprentissage.

Plusieurs méthodes ont été appliquées pour construire les modèles d'apprentissage. Nous présentons dans ce qui suit les méthodes adoptées par les analyseurs les plus connus surtout pour la langue arabe tout en débutons par le modèle statistique le plus simple : la CFG probabiliste.

Le modèle que Charniak a proposé dans (Charniak, 1996) représente une CFG probabiliste (PCFG) définie par le 5-uplet $G' = (N, \Sigma, R, P, S)$ où :

1. N est un ensemble fini de symboles non terminaux,
2. Σ est un ensemble fini de symboles terminaux,
3. R est un ensemble fini de règles de production tel que $P = \{\alpha \rightarrow \beta \mid \alpha \in N, \beta \in (N \cup \Sigma)^*\}$,

4. P est l'ensemble fini de probabilités p(r) associées aux règles r ∈ R telle que :

$$\sum_{\beta} p(\alpha \rightarrow \beta) = 1 , \forall \alpha \epsilon N,$$

5. S est le symbole de départ qui est généralement le symbole des phrases.

S → VP PUNC	0.6	WHADVP → ADV	1.0
S → VP	0.4	ADV → كيف	1.0
NP → NOUN NP	0.7	IV → نعرف	0.6
NP → PRON	0.3	IV → يفكر	0.4
VP → PRT IV SBAR	0.2	NOUN → غير	1.0
VP → IV NP	0.8	PART → لم	1.0
PRT → PART	1.0	PUNC → .	1.0
SBAR → WHADVP S	1.0	PRON → نا	1.0

Figure 1.8 : Grammaire hors-contexte probabiliste portant sur quelques phrases arabes

La figure 1.8 ci-dessus montre un exemple virtuel d'une grammaire PCFG avec laquelle la phrase arabe x = "لم نعرف كيف يفكر غيرنا" (lm nErf kyf yfkr gyrnA/ « Nous n'avons pas su comment pensent les autres ») peut être analysée selon la représentation structurelle y illustrée par la figure 1.9.

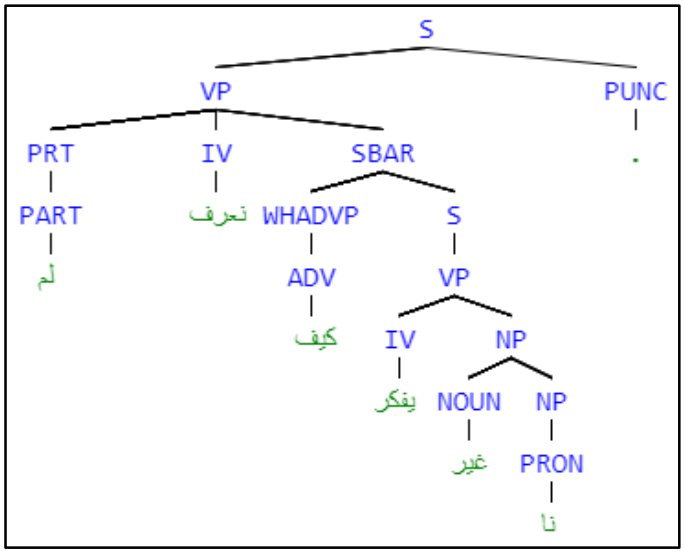


Figure 1.9 : Représentation structurelle d'une phrase analysée avec la PCFG de la figure 1.8

La probabilité P(y) de l'arbre d'analyse y ∈ GEN(x) est définie par le produit des probabilités de toutes les règles appliquées pour générer y : P(y)= 0.6 x 0.4 x 0.7 x 0.3 x0.2 x 0.8 x 1.0 x 1.0 1.0 x 1.0 x 0.6 x 0.4 x1.0 x 1.0 x 1.0 x 1.0 = 0.0019.

Dans les simples PCFG, les dépendances entre les règles ne sont pas traitées. Par exemple, la probabilité de la règle NP → PRON est calculée indépendamment du contexte du pronom

dans l'arbre syntaxique. Dans plusieurs langues, le pronom prend la fonction d'un sujet plutôt qu'un objet. Pour traiter ces indépendances, des recherches ont été effectuées pour générer des modèles génératifs plus complexes. Dans ce qui suit, nous présentons les techniques les plus utilisées dans ces recherches.

6.2.2.1 Modèles génératifs

Un modèle génératif définit une distribution de probabilité jointe P(x, y) qui réunit des données relatives à la phrase x et des données relatives à l'analyse produite y. L'exemple le plus connus de modèles génératifs est celui de Collins (1997) qui améliore une PCFG apprise sur le corpus Penn treebank par l'application de la lexicalisation. Ceci veut dire que chacun de ses symboles non terminaux dans un arbre de dérivation est caractérisé par le token et l'étiquette POS du constituant Head de la dérivation de ce symbole. Il s'écrit sous la forme A(a) où A est le symbole non-terminal est a =⟨ w, t⟩ est le couple token et étiquette POS du Head.

Dans son modèle 1, Collins (1997) écrit chaque règle de la grammaire sous la forme P(h) → $L_n(l_n)$ …$L2(l_1)H(h)R_1(r_1)$ …$R_m(r_m)$ où H est l'étiquette POS du constituant Head dans la structure étiquetée par le symbole non terminal P, L_1 …L_n et R_1 …R_m sont les modifieurs qui dépendent de H, alors que h, l_1…l_n et r_1…r_m sont les token respectifs du Head, modifieurs , L_1 …L_n et R_1 …R_m.

Le modèle 2 intègre une distinction entre les compléments et les auxiliaires en suffixant les symboles non terminaux des premiers par « -C ». Ceci permet de faciliter l'analyse syntaxique à plusieurs niveaux comme la définition des cadres de sous-catégorisation pour les subordonnées et l'évitement d'erreurs liées aux compléments dépendants. La figure 1.10 montre deux analyses d'une phrase dont les compléments doivent être dépendants.

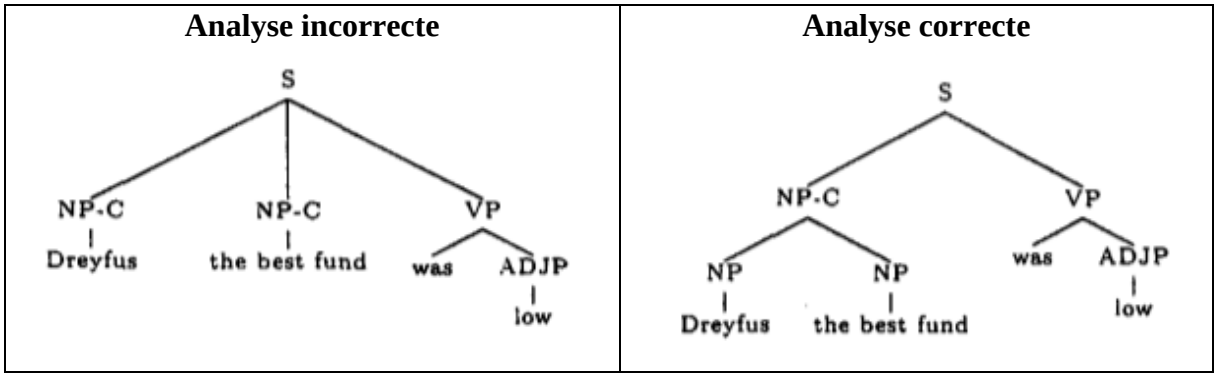


Figure 1.10 : Analyses avec le modèle 2 de Collins respectant la dépendance de compléments dans la phrase « Dreyfus the best fund was low » / « Dreyfus, le meilleur fonds était faible »

Le modèle 3 traite en plus les déplacements QU- à partir des subordonnées relatives. En effet, à partir d'une déclaration simple comme « le père voyage à Tunis », on produit une question « qui ». La structure syntaxique « le père » a sa trace « t » dans la question « **Qui** t voyage à Tunis ? » et dans la proposition relative de « C'est le père **qui** t voyage à Tunis. Le corpus d'apprentissage Penn Treebank mentionne déjà ces traces et le modèle 3 les exploitent pour l'analyse syntaxique. Pour chercher l'analyse maximale, Collins utilise l'algorithme CYK.

L'analyseur de Bikel (2004) offre une amélioration des modèles de Collins (Collins, 1997). Il vérifie l'efficacité des étapes de génération appliquées (comme l'élagage de certains nœuds d'arbres d'analyse) et les techniques utilisées (comme la lexicalisation de la grammaire). Il propose l'emploi de nouvelles informations sémantiques pour caractériser les mots outre l'étiquette POS. L'analyseur de Bikel est employé pour la génération automatique des analyses préliminaires des phrases du treebank arabe ATB. L'analyse des parties 1-3 de ce treebank apporte une performance de 77,99%. Par conséquent, ces analyses subissent une révision totale par des linguistes pour validation.

L'analyseur statistique de Stanford (Klein et Manning, 2003) appelé Stanford Parser, suit une approche statistique. Son modèle d'apprentissage a connu plusieurs améliorations comme l'intégration d'une représentation des dépendances dont les préférences sont combinées par une inférence exacte en utilisant l'algorithme A*. Cette représentation est simple, uniforme et facilement accessible par les utilisateurs non-linguistes. Elle décrit les relations grammaticales entre les mots d'une phrase donnée (e.g. le complément adjectival, le modifieur de l'adverbe, l'objet direct). Stanford Parser exécute pour l'analyse, l'algorithme CYK et grâce aux probabilités de son modèle, il produit la meilleure solution en se basant sur l'algorithme de Viterbi. Stanford Parser offre également des outils d'analyse à appliquer en étape de prétraitement comme Stanford Segmenter pour la segmentation des phrases en tokens et Stanford POS-Tagger pour l'étiquetage POS comme le Penn treebank (PTB) (Taylor et al., 2003). L'analyseur de Stanford et a été adapté à plusieurs langues comme l'anglais (Marneffe et al., 2006 ; Schuster et Manning, 2016) le chinois (Levy et Manning, 2003 ; Chang et al., 2009), l'allemand (Rafferty et Manning, 2008), l'arabe (Green et Manning ,2010), et le français (Green et al., 2011). Autour de cette variété de langues, Stanford Parser a été également appliqué dans une variété de domaines. En effet, pour la reconnaissance de parole, les analyses des phrases du corpus TCT (Tsinghua Chinese Treebank) ont générées par le Stanford parser chinois (Cheng et al., 2010). Le but est d'apprendre ces analyses pour construire trois modèles : un modèle d'analyse syntaxique, un modèle d'étiquetage POS et un modèle de reconnaissance

du Head des structures syntaxiques en adoptant un étiquetage CRF. Pour le problème d'attribution d'auteurs qui est un problème de classification, Abuhaiba et Eltibi (2016) ont généré un modèle de langage étendu d'une PCFG (XPCFG). Cette PCFG est induite des annotations syntaxiques produites par le Stanford parser arabe pour un ensemble de textes arabes. Même, pour la traduction automatique, Shquier et Al-Howiti (2014) ont proposé dans leur traducteur d'arabe-anglais d'utiliser le Stanford parser arabe pour détecter les structures syntaxiques qui forment la phrase source (en arabe). Ils ont également exploité les relations de dépendances fournies par Stanford parser pour résoudre certaines ambigüités syntaxiques ce qui a permis de réduire les ambiguïtés liées au choix du sens des mots. Le Stanford parser arabe a été également employé dans un système question/réponse pour analyser chaque question morphologiquement et syntaxiquement Waheeb and Babu, 2016). Ceci permet d'extraire la portion de la question qui fait référence à la réponse (le focus) y compris son constituant Head et les modifieurs de ce Head. Dans la modélisation, Stanford parser arabe a participé dans le processus de génération des modèles de cas d'utilisation à partir de l'ensemble des besoins des utilisateurs. Cet ensemble est analysé pour obtenir des structures syntagmatiques facilitant la détermination des acteurs potentiels et des cas d'utilisation (Arman and Jabbarin, 2014). Les performances de Stanford varient d'une langue à une autre. Pour l'arabe, il n'a pas généré les meilleurs résultats du fait que Berkeley parser le dépasse de plus que 3,5%. Cette différence est encore plus faible avec d'autres langues (Green et Manning, 2010).

L'analyseur statistique arabe de Khoufi et al. (2014b) utilise un modèle d'apprentissage supervisé qui fait sa classification selon l'algorithme SVM. Les vecteurs entrés à apprendre représentent les règles de production induite d'une analyse approfondie du Penn Arabic Treebank (ATB). Les attributs de classification portent sur les étiquettes POS des cinq premiers mots d'une structure syntaxique. L'étiquette POS de cette structure représente la classe du vecteur. Ainsi un syntagme prépositionnel (PP) composé uniquement d'une préposition (PREP) suivie d'un nom (NOUN) aura la forme de vecteur suivante : (PREP, NOUN,? ,? ,? , PP). Les performances de cet analyseur ont atteint généralement une f-mesure de 75,37% sur le treebank ATB3. Ceci a été justifié par son incapacité à traiter des structures syntaxiques très complexes. Khoufi et al. (2016) ont également implémenté un analyseur hybride de l'arabe qui combine l'analyse numérique avec l'analyse linguistique. Le but est de réduire le temps d'analyse qui est exponentiel avec la taille de la phrase à analyser. Cet analyseur hybride procède alors à une étape d'analyse numérique qui décompose la phrase en fragments principaux (en anglais chunks) en se référant à un modèle. Ce dernier a été obtenu de l'application de la classification supervisée avec les CRF (Conditional Random Fields) (Khoufi

et al., 2014a). Chaque chunk sera, ensuite, analysé indépendamment des autres en utilisant la PCFG induite de l'ATB 3. Contre 71,12% obtenus avec le modèle à base de CRF, les résultats d'évaluation de l'analyse de cette hybridation ont augmenté pour atteindre 83,24%.

Plusieurs autres techniques ont été appliquées avec les modèles génératifs, les plus connues sont présentées dans la sous-section 7.1.3.

6.2.2.2 Modèles discriminatifs

Les modèles discriminatifs portent seulement la probabilité conditionnelle P(y|x) du résultat d'analyse y étant donné la phrase entrée x. L'avantage majeur de ce genre de modèles est qu'il existe vraiment une dépendance entre les données de x et de y, ce qui sera beaucoup plus utile pour la résolution des ambigüités syntaxiques. L'outil le plus connu qui adopte ce genre de modèle est celui de Charniak et Johnson (2005) qui reçoit pour chaque phrase entrée une liste composée des 50 meilleures analyses déjà produites par le modèle génératif de Charniak (2000). Cet outil sélectionne parmi cette liste la meilleure analyse possédant le score le plus élevé. Il utilise pour cela un modèle d'entropie maximale (Johnson et al., 1999; Riezler et al., 2002) qui permet de classifier chaque analyse en fonction d'un grand nombre d'attributs qui atteint plus que 1,1 millions. Ces attributs concernent des informations aperçues dans au moins 5 phrases du corpus d'apprentissage qui regroupe les sections 1-21 du Penn Treebank. Ce modèle a été évalué sur la section 24 du Penn Treebank contenant des phrases d'au plus 100 mots et a généré un score F1 de 91.0%.

6.2.3 Exemples de modèles génératifs

Le calcul de la probabilité de toute l'analyse d'une phrase donnée effectué sur un modèle génératif, se base sur la distribution des règles de ce modèle qui participent à cette analyse. Ce calcul peut se faire selon plusieurs méthodes. La sous-section suivante montre trois exemples de méthodes construisant des modèles qui s'inscrivent dans le cadre des modèles génératifs, à savoir : les modèles à base d'historique, les modèles à base de transformations et les modèles orientés données. Les sous-sous-sections suivantes donnent un bref aperçu d'eux.

6.2.3.1 Modèles à base d'historique

Ce genre de modèle a été proposé par Black et al. (1992). Son principe consiste à relier chaque paire (x, y) ($x \in X$ étant la phrase entrée et $y \in GEN(x)$), à une séquence de décisions de dérivation $D = (d_1,...,d_m)$. La probabilité jointe P(x, y) est calculée ainsi :

$$P(x,y) = P(d_1,\ldots,d_m) = \prod_{i=1}^{m} P(d_i \mid \phi\,(d_1,\ldots,d_{i-1}))$$

De ce fait, chaque décision d_1 dans une dérivation D est conditionnée par son historique (d_1,...,d_{i-1}) déjà construit dans l'arbre syntaxique. La fonction Φ est utilisée dans le cadre de l'apprentissage automatique pour grouper les historiques dans des classes d'équivalences.

6.2.3.2 Modèles à base de transformations

Ces modèles permettent d'effectuer des transformations sur la PCFG simple selon certaines techniques. En effet, la technique de division d'états est employée par exemple pour annoter les nœuds non terminaux (sauf les préterminaux) par leurs parents (Johnson, 1998 ; Klein et Manning, 2003). Ainsi, le symbole non terminal NP dont le parent est S se distingue d'un autre NP dont le parent est PP. Cette technique a augmenté largement la performance d'analyse (Gildea, 2001; Bikel, 2004). Cette annotation a été ensuite améliorée par l'emploi d'une classification non supervisée pour la division d'états (Matsuzaki et al., 2005 ; Petrov et al., 2006 ; Liang et al, 2007).

Dans la technique de Markovisation, ce sont les règles de production n-aires de la PCFG qui sont transformées en des règles unaires et binaires. Cela nécessite la création de nouveaux symboles non terminaux pour coder les éléments de l'historique de la dérivation. Par exemple la règle ternaire « VP → V NP PP » est transformée en la suite de règles suivantes :

VP → ⟨ VP:V … PP⟩

⟨ VP:V … PP⟩ → ⟨ VP:V … NP⟩ PP

⟨ VP:V … NP⟩ → ⟨ VP:V⟩ NP

⟨ VP:V⟩ → V

Dans la première règle, on représente le VP par un nouveau symbole ⟨ VP:V … PP⟩ qui porte sur un VP dont le fils Head est V et sur un PP qui est le fils le plus à droite. Dans la deuxième et la troisième règle, on génère le fils le plus à droite (respectivement PP et NP) après un nouveau symbole regroupant les autres fils qui est respectivement ⟨ VP:V … NP⟩ et ⟨ VP:V⟩ .

Toutefois, l'efficacité de ce genre de modèles a diminué à cause de l'augmentation incrémentale des transformations en fonction du nombre de symboles non terminaux et de règles. Des analyseurs utilisant ce genre de modèles, comme l'analyseur de Berkeley, ont appliqué un élagage approfondi des nouveaux nœuds obtenus des transformations a pour régler leur efficacité. En effet, l'analyseur de Berkeley (Petrov et al., 2006) a choisi de faire son

annotation syntaxique en apprenant une nouvelle grammaire à partir d'une simple CFG. Ses symboles non terminaux sont ceux de la CFG mais qui ont subi des opérations alternatives de divisions et de fusions à plusieurs niveaux de granularité. La division est effectuée pour avoir une grammaire à large couverture. Tandis que la fusion est employée pour généraliser quelques symboles et contrôler la taille de la grammaire. L'application alternative de ces deux opérations fait le compromis entre ces deux besoins. Les performances de cet analyseur ont atteint un score F1 de 90.2% en le testant sur le Penn Treebank. Pour le treebank arabe ATB (parties 1-3), il apporte 84,29% contre 80,67% pour Sanford et 77,99% pour Bikel (Green et Manning, 2010).

6.2.3.3 Modèles orientés données (MOD)

Le principe de ce genre de modèles génératifs consiste à appliquer le treebank directement en tant que grammaire probabiliste. Une nouvelle phrase est analysée en fait en se basant sur la combinaison de fragments d'analyses perçus dans le corpus d'apprentissage. Bod (1998) et Bod et al. (2003) proposent une implémentation de ce principe qui considère que les fragments d'analyses sont des arbres élémentaires (des sous-arbes d'analyses) obtenus du corpus d'apprentissage. Un arbre élémentaire est caractérisé par une racine et des nœuds internes étiquetés par des symboles de N et des feuilles étiquetés par des symboles de (Σ U N).

L'ensemble de tous les arbres élémentaires forme une grammaire appelée grammaire d'arbres de substitution (en anglais Tree Substitution Grammar, TSG). Deux arbres élémentaires α et β peuvent être combinés par une opération de substitution $\alpha \circ \beta$ pour obtenir un arbre unifié. Cette substitution ne peut être réalisée que si la racine de β a la même étiquette que le fils le plus à gauche de α.

Maltparser (Nivre, 2007a) est aussi un système d'analyse orientée données, mais qui fonctionne sur des treebanks à base de dépendances. Ce système est générique du fait qu'il fonctionne sur plusieurs langues sans intégrer des ajustements spécifiques aux langues traitées. Maltparser fait son apprentissage sur les treebanks à base de dépendances en le décomposant en arbres élémentaires. Il construit un modèle de classification de ses données selon des algorithmes comme les SVM. Ce modèle inclut pour chaque phrase annotée, une variété d'attributs comme les étiquettes POS des derniers tokens traités et non encore traités, le dernier token traité, ses types de dépendances, son premier et son dernier dépendant. En utilisant le modèle construit, Maltparser analyse les nouvelles phrases selon un algorithme basé sur des opérations de transitions (empilement, dépilement) effectuées sur les nœuds. Ces transitions sont liées à la satisfaction d'un ensemble de contraintes de dépendances satisfaites entre les nœuds. En l'appliquant sur la version à base de dépendance du Penn Treebank (Yamada et

Matsumoto, 2003), Maltparser a apporté 86.3%. Avec la même structure à base de dépendances, l'apprentissage d'un modèle pour l'arabe avec Maltparser a apporté 74.75% dans le cadre d'une évaluation de plusieurs analyseurs à base de dépendances (Nivre et al., 2007b). Le corpus d'apprentissage était le treebank PADT (Prague Arabic Dependency Treebank) développé par Hajič et al. (2004). Dans la même évaluation, Maltparser a atteint la meilleure performance avec un score de 76.52% en faisant la combinaison de six modèles d'apprentissage. L'apprentissage d'un modèle avec MaltParser sur un autre treebank arabe à savoir CATiB (Habash et Roth, 2009), a apporté une performance de 81% et a même atteint 84% en intégrant des attributs morphologiques non spécifiés par CATiB comme le genre, le nombre, la personne, le lemme et la définition (Marton et al., 2013). Le tableau suivant montre une comparaison entre les différents analyseurs statistiques appliqués pour l'arabe les performances principalement en fonction de leurs performances.

Analyseurs statistiques	**Algorithme d'apprentissage**	**Technique utilisée**	**F-mesure**
(Bikel, 2004)	-	Lexicalisation + infos Syn et Sem	77,99
Stanford (Green et Manning, 2010)	A*	Dépendences grammaticales	80,67
(Khoufi et al., 2014a)	SVM	Classification	75, 37
(Khoufi et al., 2016)	CRF	Hybride (Décomposition puis analyse)	**83,24**
Berkeley (Petrov, 2009)	-	Division et Fusion	**84,29**
Maltparser sur PADT (Nivre et al., 2007b)	SVM	MOD	74, 75
Maltparser sur CATiB (Marton et al., 2013)	SVM	MOD	**84,00**

Tableau 1.2 : Tableau comparatif entre les analyseurs statistiques

Le tableau ci-dessus montre que les analyseurs de Berkeley (Petrov, 2009), Maltparser (Marton et al., 2013) et (Khoufi et al., 2016) apportent respectivement les meilleures performances. Bien que les techniques d'analyse que ces analyseurs utilisent soient différentes, elles partagent l'idée de décomposition de l'arbre d'analyse à construire en arbres élémentaires.

7 Conclusion

Le présent chapitre nous a permis dans un premier temps de choisir les outils de prétraitement des textes entrés à notre analyseur syntaxique ainsi que la représentation structurelle qui en résulte. Dans un second temps, la présentation des différents formalismes de

grammaires et approches d'analyse nous a guidé à les évaluer pour fixer celles qui satisfont le plus nos besoins d'analyse. Le bref aperçu fourni s'est focalisé sur les outils destinés au traitement de l'arabe, la langue qui nous intéresse le plus. Le formalisme de grammaire de propriétés a été défini dans ce chapitre rapidement comme moyen pour créer une grammaire dans un analyseur syntaxique. Ce terme représente aussi un thème important dans notre thèse. Pour cela nous avons choisi de lui consacrer un chapitre entier, étant le chapitre suivant, pour montrer son utilité et son apport par rapport à d'autres formalismes dans différents domaines d'application.

Chapitre 2

Grammaires de propriétés

1 Introduction

Le deuxième thème de notre problématique concerne le formalisme de grammaires de propriétés. Ce formalisme permet de décrire une langue en représentant d'une manière décentralisée ses informations linguistiques par des contraintes locales appelées « propriétés ».

Le présent chapitre s'occupe de donner un aperçu sur le cadre dans lequel s'inscrit ce formalisme. Il commence dans la section 2 par la définition de la notion de contraintes qui représente le cœur de ce formalisme. Il passe ensuite à présenter dans la section 3 comment les contraintes sont utilisées dans le cadre des problèmes de satisfaction de contraintes. Il aborde par la suite dans la section 4 la question d'utilité des contraintes et de leurs niveaux d'accès dans plusieurs autres théories linguistiques. Il consacre finalement la section 5 à la présentation du principe du formalisme de GP, ses différents composants et ses domaines d'utilisation.

2 Notion de contrainte

Une contrainte est définie comme étant une relation logique (une propriété qui doit être vérifiée) entre différentes inconnues, appelées variables. Chaque variable prend ses valeurs dans un ensemble donné, appelé domaine. Ainsi, une contrainte restreint les valeurs que peuvent prendre simultanément les variables. Par exemple, la contrainte « x - 3*y = 12 » restreint les valeurs qu'on peut affecter simultanément aux variables x et y (Laurière, 1978). Il faut noter aussi que, généralement, l'ordre de présentation des contraintes d'un certain problème n'est pas significatif. Il suffit en fait que toutes les contraintes soient satisfaites.

L'arité d'une contrainte est le nombre de variables sur lesquelles elle porte. Ainsi, une contrainte peut être : unaire (portant sur une variable), binaire (mettant en relation deux variables) ou n-aire (mettant en relation un ensemble de n variables).

Le problème qui est conçu pour satisfaire ces contraintes est appelé Problème de Satisfaction de Contraintes (en anglais Constraint Satisfaction Problem, CSP). La section suivante fera l'objet de la présentation de ses différents aspects.

3 Problèmes de satisfaction de contraintes

Le traitement d'un CSP exige la modélisation des contraintes de ce problème puis l'utilisation de solveurs de contraintes pour le résoudre. Les deux sous-sections suivantes s'occupent de présenter ces deux étapes.

3.1 Modélisation d'un CSP

Un CSP se modélise sous la forme d'un ensemble de contraintes impliquant un certain nombre de variables, chacune prenant ses valeurs dans un domaine. Plus formellement, un CSP se définit par un triplet (X, D, C) tel que :

$X = \{X_1, X_2, ..., X_n\}$ est l'ensemble des variables du problème,

$D = \{D_1, D_2, ..., D_n\}$ correspond aux domaines de valeurs respectifs des variables de X.

$C = \{C_1, C_2, ..., C_m\}$ est l'ensemble des contraintes reliant les différentes variables de X.

Dans un CSP, le but est tout simplement de trouver une solution affectant des valeurs aux variables, de sorte que toutes les contraintes soient satisfaites. L'affectation consiste en effet à instancier certaines variables par des valeurs de leurs domaines. L'affectation est dite totale si elle instancie toutes les variables du problème ; elle est par contre dite partielle si elle n'en instancie qu'une partie. Si toutes les variables sont instanciées tout en respectant toutes les contraintes, on a une solution exacte. Si une contrainte n'est pas valide bien que l'affectation des variables est totale, on parle d'une contrainte violée. L'affectation (totale ou partielle) est dite consistante si elle ne viole aucune contrainte.

3.2 Solveurs de contraintes

Les recherches dans le domaine des CSP ont pour objectif de concevoir des méthodes efficaces pour les résoudre par la satisfaction de toutes leurs contraintes. Cependant, la plupart des CSP font partie d'une classe de problèmes appelés NP-complets (Garey et Johnson, 1979) pour lesquels tous les algorithmes connus pour les résoudre nécessitent, dans le pire des cas, un temps exponentiel. Pour diminuer ce coût, plusieurs solveurs ont été proposés, permettant de prendre un CSP en entrée et de produire en sortie une solution à ce CSP si ce problème est consistant. Ces solveurs peuvent être classés en deux catégories : solveurs complets ou systématiques et solveurs incomplets ou stochastiques. Ceux de la première classe font un parcours exhaustif de toutes les affectations possibles de variables pour trouver une solution. Si la taille de l'espace de recherche est assez grande, ils peuvent prendre beaucoup de temps pour arriver à la trouver, ce qui n'est pas utile dans ce cas (Garey et Johnson, 1979). Les solveurs incomplets, par contre, font un parcours aléatoire (non systématique) de l'espace de recherche pour trouver une solution acceptable en un minimum de temps possible. Dans ce cas, trouver une solution, même acceptable, n'est pas garanti.

3.2.1 Solveurs complets

Selon Dechter (1990), les techniques d'exploration de l'espace de recherche adoptées par les solveurs complets peuvent être classées selon les trois catégories suivantes :

- **Exploration systématique :** L'algorithme de retour-en-arrière (backtrack en anglais) (Kumar, 1992) est le plus courant dans cette catégorie. Il parcourt l'espace de recherche en profondeur et affecte au fur et à mesure des valeurs aux variables, et vérifie à chaque fois la consistance de l'instanciation partielle courante. Cette dernière peut être écartée en cas de violation de contrainte, et il y aura un retour en-arrière à la dernière instanciation partielle consistante.
- **Exploration prospective :** Son principe général (Haralick et Elliott, 1980) consiste à faire une anticipation, appelée en anglais look-ahead, des valeurs des variables à garder en ajoutant une étape de filtrage au principe de retour-en-arrière. En effet, après chaque affectation de valeur à une variable, ils éliminent des domaines des variables non encore instanciées les valeurs qui sont incompatibles avec cette affectation. Ces modifications ne seront pas prises en compte s'il y aura ensuite un retour-en-arrière de cette affectation. Le nombre de tests des contraintes et le nombre des valeurs éliminées sont les critères qui distinguent un algorithme de ce type d'un autre. Parmi ses algorithmes, on trouve Forward-checking, Partial-look-ahead, Full-look-ahead et Real-full-look-ahead.
- **Exploration rétrospective :** Les algorithmes de cette catégorie affectent le processus de recherche pour faire un retour-en-arrière « intelligent ». En effet, ils bénéficient des informations implicites concernant les cas d'échec pour économiser des essais postérieurs de valeurs ou pour sélectionner un meilleur point de retour, d'où leur noms en anglais « look-back » qui indique l'exploitation des expériences passées. Parmi ces algorithmes, on peut citer Back-jumping (Dechter, 1990), Back-checking (Haralick et Elliott, 1980), Back-marking (Caschnig, 1977).

3.2.2 Solveurs incomplets

A la différence des solveurs vus dans la sous-section précédente, les solveurs incomplets ne construisent pas les affectations au cours du processus de recherche mais ils partent d'une affectation complète des variables et essaient de l'améliorer par la suite à travers des modifications successives de leurs valeurs. Ces modifications peuvent être subies de manière probabiliste d'où le critère stochastique qui caractérise ce type de recherche. L'objectif ici est que cette suite de modifications puisse générer finalement une solution sans pour autant pouvoir

prouver l'inconsistance du problème puisque ce type de solveur ne peut pas couvrir toutes les affectations possibles.

Après avoir expliqué les différents aspects des problèmes qui s'occupent de satisfaire les contraintes ainsi que les solutions proposées pour résoudre ce type de problème, nous envisageons de découvrir dans la section suivante les façons d'utilisation des contraintes dans les théories linguistiques (présentées d'un autre point de vue dans le chapitre précédant) ainsi que le niveau d'accès de ces contraintes aux valeurs des variables. Le formalisme de grammaires de propriétés en tant théorie linguistique aura une section suivante entière.

4 Utilité et niveau d'accès des contraintes dans les théories linguistiques

En considérant le problème d'analyse comme un CSP, plusieurs théories linguistiques se rejoignent à l'utilisation des contraintes en tant que élément fondamental mais se diffèrent aux niveaux d'accès de leurs contraintes aux valeurs des variables du problème. Les deux sections suivantes montrent respectivement l'utilité des contraintes dans ces différentes théories et les niveaux d'accès de ces contraintes aux variables.

4.1 Utilité des contraintes

Les contraintes sont utilisées de manière systématique dans plusieurs théories linguistiques. Nous citons dans ce qui suit quelques exemples d'elles appliqués respectivement sur les grammaires de contraintes, les grammaires de dépendences par contraintes et les HPSG.

4.1.1 Grammaire de contraintes (Karlsson, 1990)

La grammaire de contraintes utilise les contraintes pour le filtrage des caractéristiques des catégories en se basant sur leurs informations contextuelles structurées en traits à des niveaux de spécificité bien déterminés, comme la nature grammaticale. Voici un ensemble de contraintes pour l'anglais comme exemple (Blache, 2001) :

1. (@w = 0 "PREP" (-1 DET))
2. (@w = 0 VFIN (-1 TO))
3. ("that" =! "Rel " (-1 NOMHEAD) (1 VFIN))

La première contrainte interdit qu'une préposition soit précédée d'un déterminant, la seconde empêche la précédence de la particule « to » par un verbe tensé, alors que la dernière indique que « that » représente un relatif s'il est précédé d'une tête nominale et suivi d'un verbe tensé.

4.1.2 Grammaire de Dépendance par Contraintes (Maruyama, 1990)

Comme son nom l'indique, les contraintes seront évidemment appliquées au sein de cette grammaire aux relations de dépendances. Ces contraintes permettent en fait de contrôler la construction de ces relations en liant le type de relation de dépendance et les caractéristiques des mots reliés. Les deux contraintes suivantes peuvent former un exemple explicatif :

1. word(pos(x)) = PP ⇒(word(mod(x)) ∈ {PP, NP, V}, mod(x) ≺ pos(x))
2. mod(x) ≺ pos(y) ≺ pos(x) ⇒mod(x) ⪯ mod(y) ⪯ pos(x)

La première contrainte indique qu'un syntagme prépositionnel (PP), à une position x, peut modifier l'un des éléments de l'ensemble {PP, NP, V} et que ces éléments le précèdent. La deuxième contrainte quant à elle indique que le croisement des relations de modifications entre deux catégories n'est pas autorisé.

4.1.3 HPSG (Pollard, 1996)

La théorie HPSG (Head-driven Phrase Structure Grammar) considère chaque information linguistique qu'elle modélise comme une contrainte. Ses informations linguistiques prennent la forme d'une matrice de structures de traits où chaque trait représente un couple attribut/valeur. Les contraintes d'une entrée syntagmatique dans cette approche sont modélisées comme suit :

$$\begin{bmatrix} SYNSEM & \left[SYN \left[COMPS\langle\rangle \right] \right] \\ HD-DTR & \left[word \; SS \left[SYN \left[COMPS\langle \boxed{1}, \ldots, \boxed{n} \rangle \right] \right] \right] \\ NHD\langle [SS\boxed{1}], \ldots, [SS\boxed{n}] \rangle & \end{bmatrix}$$

Figure 2.1: Matrice d'attributs-valeurs des syntagmes composés de compléments

La contrainte de l'exemple illustré par la figure 2.1 ci-dessus porte sur les syntagmes composés de compléments en plus du fils tête. Elle exige que ces compléments doivent être des fils non-tête dans ce syntagme.

De ce fait, l'utilisation systématique des contraintes est très bénéfique vu que ces contraintes permettent non seulement de traiter des informations de différents niveaux de spécificité mais aussi elles donnent une vision générale et incrémentale du problème d'analyse linguistique.

Du point de vue opérationnel, lorsque l'approche se base sur les contraintes, elle peut monter du niveau de la théorie de grammaire pour être interprétée au niveau informatique et ce, à travers sa modélisation en un CSP (Johnson et al., 1999) entre les deux niveaux. Cette

nouvelle modélisation offre également la possibilité de manipuler les données incomplètes et générer des solutions approchées ou partielles, ce qui favorise encore plus la robustesse de l'approche (Blache, 2000).

Les apports bénéfiques des implémentations informatiques des théories linguistiques à base de contraintes en tant que CSP ont été bien observés. Ceci a incité les approches modernes à les exploiter par des approches modernes dans la résolution du problème d'analyse syntaxique. Il suffit en effet de spécifier un ensemble de variables, leurs domaines ainsi qu'un système de contraintes portant sur ces variables. Le contrôle permanent du processus d'analyse se fait à travers la vérification continue de la satisfaction de contraintes et donc de la consistance de ce système tout en permettant de réduire au fur et à mesure le coût de traitement des domaines de définition des variables. Le problème majeur qui réside ici concerne la spécification des variables des contraintes par la théorie linguistique. La plupart des théories linguistiques représentent ces variables par des structures de haut niveau, ce qui empêche l'accès immédiat à leurs valeurs. La section suivante s'occupera de l'explication de ce problème.

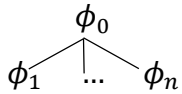
4.2 Niveau d'accès des contraintes

Comme nous l'avons déjà mentionné dans la section précédente, les exemples de théories linguistiques cités ci-dessus font partie des approches basées sur les contraintes. Ces approches font référence à la notion de contrainte de manière explicite. Elles représentent les informations linguistiques d'une manière parallèle à tous les niveaux (Carpenter, 1992 ; Pollard, 1996) et non pas séparée à la façon des méthodes séquentielles (Prince et Smolensky, 1993). Cette vision parallèle repose sur un système de contraintes interactives (Pollard, 1996) au lieu des systèmes de règles des méthodes séquentielles. Cela permet d'assurer la grammaticalité des structures. Les deux sous-sous-sections suivantes montrent avec plus de détail comment accèdent les contraintes de chaque formalisme aux variables du problème.

4.2.1 Cas des HPSG

Les HPSG sont des approches à base de contraintes modélisées en CSP pour faire l'analyse (Sag, 1999). Elles représentent, en effet, toute information linguistique, quelque soit son niveau, sous forme de contrainte. Leurs caractéristiques, mentionnées ci-dessous, supportent ceci (Blache, 2000) :

- Bonne formation : Φ est une structure arborescente bien formée si et seulement si tous les sous-arbres de Φ satisfont une entrée lexicale ou une règle de grammaire ρ.

- Satisfaction lexicale : la structure lexicale $\begin{matrix} F \\ | \\ \omega \end{matrix}$ satisfait l'entrée lexicale ⟨ ω, δ⟩ seulement si F satisfait δ.
- Satisfaction syntagmatique : le sous-arbre local Φ =  satisfait une règle de grammaire ρ = $\delta_0 \rightarrow \delta_1 \ldots \delta_n$ si et seulement si :
  1. la séquence ⟨ $\phi_0, \phi_1, \ldots, \phi_n$⟩ satisfait la description ⟨ $\delta_0, \delta_1, \ldots, \delta_n$⟩ ,
  2. Φ satisfait le Principe de Compositionnalité Sémantique,
  3. si ρ est une règle portant sur les têtes, alors Φ satisfait le principe des traits de tête, le principe de valence et le principe d'héritage Sémantique.

Selon la caractéristique de bonne formation, toute information linguistique décrivant des structures lexicales ou des principes universels, joue le rôle de contraintes. La satisfaction de ces contraintes est exprimée sur des arbres. La vérification d'une contrainte exige alors la construction préalable d'un arbre local. Ainsi, il faut sélectionner une règle de la grammaire, correspondant à un schéma d'arbre, puis instancier cet arbre en fonction des informations à vérifier. Par conséquent, le domaine de définition des variables ne peut pas se limiter simplement à un système de contraintes.

Ce problème est dû à la dépendance de l'information linguistique de la connaissance des têtes. Ainsi, pour chercher une entrée lexicale dans la grammaire au cours de l'analyse, il faut tout d'abord monter à la racine de l'arbre local puis descendre vers l'entrée recherchée. La détermination de la racine est liée à la détermination de sa tête. Ceci nécessite la préparation au préalable du schéma de l'arbre local à parcourir. Ce schéma regroupe la tête et les constituants de la structure sur laquelle une contrainte peut être vérifiée. On applique un ensemble de règles pour construire ce schéma. Cette préparation au préalable rend indirecte l'accès aux variables contraintes, étant les structures syntaxiques. C'est une limitation dans l'utilisation des HPSG.

4.2.2 Cas des Grammaires de dépendance et contraintes

Les théories génératives y comprises les grammaires de dépendances et contraintes (CDG) représentent les informations linguistiques en ensembles d'équations (Mel'ˇcuk, 1988). Ces équations peuvent être interprétées comme des contraintes dans le cadre d'un CSP (Maruyama, 1990 ; Duchier, 1999). Cependant, comme les HPSG, l'accès aux valeurs des variables contraintes n'est pas direct avec les CDG. Ainsi, lors de la définition d'une relation entre deux mots d'une phrase, il faut spécifier le type de dépendance qu'elle représente, ce qui ne peut pas

être généré directement par les CDG. Il faudrait alors générer au préalable l'ensemble toutes les relations de dépendances possibles entre ces deux mots. Cet ensemble représente le domaine des variables contraintes et le système de contraintes va servir pour son filtrage.

En conclusion, pour résumer les problèmes liés à l'interprétation du processus d'analyse comme des problèmes de satisfaction de contraintes selon les théories linguistiques décrites ci-dessus, nous constatons qu'ils sont de deux types : le premier vient du caractère génératif de ces théories, le second de leur manque de déclarativité. La récapitulation de ces deux problèmes est présentée dans ce qui suit :

- Problème de **générativité** qui génère les deux conséquences suivantes sur la conception de l'analyse :
  - La grammaticalité d'un énoncé est obligatoire dans l'analyse pour construire une structure syntaxique (on ignore ainsi le cas où l'information est incomplète ou malformée). L'analyse dépend de la dérivation à trouver qui conduit à cet énoncé et lui associe une structure syntaxique. Ceci est incompatible avec le langage en tant qu'ensemble hétérogène de données.
  - La dépendance de l'information linguistique d'une structure construite préalablement selon un ordre d'analyse.
- Problème de **déclarativité** qui manque dans ces théories. En effet, avec ces théories, il existe une dépendance entre les contraintes (représentant les principes). C'est parce qu'il faut construire au préalable un arbre local pour vérifier les principes. Les principes ne sont pas alors vérifiés isolément mais dépendamment de la connaissance des règles hiérarchiques. Ceci est un problème cumulé au problème lié à la représentation des contraintes. Ces dernières sont représentées de manière détaillée et hiérarchisée sous forme de structures de traits respectant le système d'héritage. Ces structures posent un problème de déclarativité dans la mesure où les traits qui les utilisent et leur propagation sont contrôlés par des principes. Ainsi le type de trait caractérisant une certaine information linguistique est représenté en fonction du type de propagation ou de contrôle à lui affecter. De manière réciproque, les principes universels sont établis en fonction de la forme de la structure de traits. L'information linguistique est donc représentée dépendamment de certaines structures procédurales. Lors d'une interprétation de ces théories sous forme de satisfaction de contraintes, il faut définir un ensemble de variables et un système de contraintes portant sur ces variables. Les principes sont vus comme des contraintes. Mais les contraintes vont porter sur des structures complexes devant être construites en cours

d'analyse. L'ensemble de contraintes ne peut être considéré comme un système : chaque contrainte est vérifiée sur une structure donnée, ce qui lui donne un effet passif.

5 Formalisme de Grammaires de Propriétés (GP)

Le problème rencontré dans les approches à base de contraintes réside dans le fait que les contraintes portent sur des informations non élémentaires (en HPSG sur un arbre local, et en CDG sur une relation de dépendance). Ces approches requièrent, en effet, la construction d'une structure locale de ces informations avant d'utiliser les contraintes qu'elles ont décrites. La satisfaisabilité des contraintes ne peut alors pas être vérifiée sur un système de contraintes global. Une approche s'appuyant effectivement sur la satisfaction de contraintes doit pouvoir accéder directement aux valeurs des variables, sans passer par la construction d'une structure élémentaire. L'approche de Grammaires de Propriétés (GP) (Bès, 1999 ; Blache, 1999 ; Blache, 2000) répond vraiment à ce besoin. Cette section sera consacrée à présenter ses objectifs, ses principes, mais surtout les éléments essentiels sur lesquels elle repose et finalement quelques exemples de domaines d'application de ce formalisme.

5.1 Objectifs des GP

Sur le plan théorique et comme décrit ci-dessus, l'apparition d'une nouvelle approche se justifie par les problèmes posés par les approches existantes. Le plan méthodologique scientifique est encore plus important dans l'élaboration des approches (Blache, 2001). Dans (Bès, 1998), l'auteur favorise encore la nécessité de fournir des systèmes de représentation des conditions d'observation et des prédictions pour pouvoir proposer des hypothèses générales. Dans le formalisme de GP, on reste à un niveau de représentation très général sans émettre des hypothèses sur la représentation des objets linguistiques (les traits non locaux en HPSG par exemple gardent leur structure). On a adopté pour réaliser cette représentation une démarche regroupant la description générale des informations linguistiques avec une description empirique de ces informations à travers des observations. Ceci conduit à l'élaboration d'hypothèses exprimées sous forme de propriétés. A travers l'application de cette démarche basée sur le formalisme de GP, différents objectifs sont apparus se résumant en les points suivants :

- Dégager des principes de haut niveau ainsi que des propriétés plus spécifiques
- Organiser ces propriétés en familles

- Offrir un cadre d'analyse linguistique prenant en compte partiellement ou entièrement l'ensemble de ces propriétés et principes en fonction des besoins sans mettre en cause la cohérence de cet ensemble. Selon le contenu de cet ensemble, il est possible d'avoir une analyse superficielle ou au contraire approfondie.
- Proposer une méthode d'analyse à granularité variable pour l'analyse syntaxique automatique.

Par ailleurs, en focalisant la vision sur l'application du formalisme de GP au domaine de TALN, nous constatons qu'il apporte une approche différente des celles existantes. Généralement, les travaux dans ce domaine se distinguent en fonction de leur utilisation d'une base de description linguistique ou pas. Deux différents types d'approches peuvent être définis : les approches théoriques et les approches industrielles. Cette utilisation est très complexe et pose deux types de questions en TALN : (i) le maintien de sa cohérence et (ii) son efficacité d'implantation.

- **La cohérence** : cette question se pose lors de l'enrichissement de la capacité expressive du langage de description sur lequel s'appuie une certaine approche. Cet enrichissement rend la description linguistique plus complexe ce qui nécessite la vérification de sa cohérence.
- **L'efficacité** : Cette question se pose à cause du double problème lié à l'utilisation ou pas d'une théorie linguistique. En effet, la seule façon d'obtenir généralement une analyse automatique fiable, détaillée et robuste consiste à utiliser explicitement une théorie linguistique, mais ceci cette utilisation est très complexe et difficile. La non utilisation est également trop restrictive. La prise en compte des paramètres fondamentaux (la cohérence, la généralité, l'efficacité, la réutilisabilité, la gestion du développement des descriptions) ne rend aucune des deux démarches satisfaisante.

L'approche de GP proposée par (Blache, 2001) est radicalement différente. Elle rend possible l'expression des relations entre les objets indépendamment de la structure des objets eux-mêmes. Aucune structure interne d'objets ne doit être définie au préalable. Les relations peuvent être de différents types (syntaxiques, sémantiques, etc.) et de différents niveaux de description indépendants les uns des autres.

Nous récapitulons les constations citées ci-dessus favorisant le formalisme de GP au profit des autres théories linguistiques modernes lors de l'interprétation du problème d'analyse comme problème de satisfaction des contraintes, en différentes caractéristiques du formalisme de GP présentées dans ce qui suit :

- **Capacité expressive de description enrichie :** Toute information syntaxique est représentée est décrite par un ensemble de propriétés (relations) exprimées sous forme de contraintes. Par exemple, pour un syntagme verbal en français, plusieurs propriétés peuvent être dégagées comme : la liste de ses éléments possibles (verbe, auxiliaire, clitique), la contrainte d'avoir un auxiliaire fléchi si le participe passé d'un verbe se construit avec être, etc.
- **Extensibilité :** Toute comme les autres théories génératives modernes, la hiérarchisation de la structure syntaxique est préservée, mais en rendant systématique l'indépendance des propriétés entre elles. Cette option permet une extensibilité dans la représentation des informations linguistiques. Ainsi il est possible d'intégrer à tout moment de nouvelles propriétés.
- **Flexibilité :** A la différence de plusieurs autres approches modernes, tout type d'information linguistique quelque soit sa position dans l'énoncé à analyser peut être représenté dans ce formalisme. Les informations incomplètes, partielles et non canoniques peuvent être décrites. C'est parce que ce formalisme traite la notion de grammaticalité en acceptant d'avoir des propriétés vérifiées satisfaites, mais également violées, lors de la description et l'analyse de l'énoncé entré. La notion de grammaticalité est calculée par un ratio des propriétés violées par rapport aux propriétés vérifiées. , ce qui caractérise la flexibilité de ce formalisme.
- **Appui effectif sur la notion de contrainte :** Les informations statistiques sont caractérisées directement par un ensemble de contraintes (propriétés). Lorsqu'on est un intéressé dans l'analyse par certains phénomènes linguistiques bien déterminés, il est possible de se limiter à traiter un sous-ensemble de ces contraintes.

En plus des qualités indiquées ci-dessus, le formalisme de GP prend en compte les paramètres fondamentaux fixés pour évaluer les théories linguistiques qui sont principalement : la déclarativité, la généralité et la réutilisabilité.

5.2 Principes des GP

L'approche de GP permet une représentation décentralisée et locale des informations linguistiques. En effet, il est possible de représenter d'une manière indépendante, tout type d'information, mais également des informations incomplètes, partielles et non canoniques. De même, cette forme de représentation permet de prendre en compte les différentes interprétations possibles pour une même entrée en fonction du contexte. Cette représentation favorise la

généralité de l'approche. Dans ce cas, l'analyse ne se base pas sur la forme des informations linguistiques traitées, mais sur leurs propriétés. Pour cela, ces informations doivent être encapsulées par types indépendants les uns des autres. La granularité d'analyse peut également être variée selon le degré de finesse (précision) des informations linguistiques manipulées et représentées en propriétés sans pour autant ennuyer la cohérence de cet ensemble d'informations. On pourra avoir une analyse approfondie en utilisant la totalité des propriétés. Cette analyse est utile pour différentes tâches comme la traduction automatique et l'apprentissage. Une analyse superficielle peut également être réalisée se contentant d'une partie des propriétés. Le processus d'analyse sera vraiment allégé, mais le niveau de grammaticalité de la GP obtenue devient général. Ce type d'analyse est utile par exemple pour l'annotation de corpus, le filtrage d'information ou le guidage de la synthèse de la parole.

5.3 L'analyse syntaxique avec les GP

Pour faire l'analyse syntaxique avec les GP, nous nous n'intéressons pas à associer simplement une structure syntaxique à un énoncé, mais plutôt de le caractériser, quelle que soit sa forme, à l'aide d'un ensemble de propriétés à vérifier leur satisfaction. La caractérisation d'un énoncé sera alors formée par l'ensemble des propriétés satisfaites et celles qui ne le sont pas (violées). Avec cette stratégie d'analyse, il est également possible de varier la granularité de l'analyse sans changer le fonctionnement de l'analyseur lui-même. C'est par ce que les contraintes sont représentées de manière indépendante. Elles sont regroupées dans des sous-systèmes caractérisant chacun une catégorie syntaxique (formés par les projections des propriétés pour prendre en considération la granularité d'analyse choisie). L'analyse avec ce formalisme revient en fait à vérifier pour chaque catégorie syntaxique la satisfaisabilité de son sous-système de contraintes. Nous avons alors une véritable stratégie d'analyse par contraintes. Pour analyser un énoncé donné, un processus de trois étapes est à appliquer : L'énumération de l'ensemble des catégories possibles de cet énoncé y compris celles syntaxiques susceptibles d'être des catégories mères pour les catégories dégagées, la construction des suites possibles des catégories énumérées, et finalement le calcul de la caractérisation des suites par la vérification de la consistance des sous-systèmes de contraintes correspondant aux catégories syntaxiques de ces suites. L'apport du formalisme de GP est très bien décrit grâce à cette notion de caractérisation. En effet, aucune règle syntagmatique ni schéma de règle n'est nécessaire pour décrire syntaxiquement un énoncé. Les notions de dérivation et de bonne formation ne se posent pas. Il suffit de fournir un ensemble de systèmes de contraintes décrivant cet énoncé de façon simple et directe et peu importe sa forme, et de vérifier leur satisfaction. En plus, si nous

avons fixé certains phénomènes à viser, il est possible de sélectionner un sous-ensemble de contraintes relatifs à ces phénomènes pour les vérifier.

5.4 Eléments essentiels des GP

Après avoir présenté les objectifs et les principes des GP, il est nécessaire maintenant de connaitre la définition d'une GP pour se familiariser encore plus sur ce formalisme. Ainsi, une GP est formée par un ensemble de propriétés exprimant différentes relations (propriétés) entre les catégories qui forment la structure syntaxique. Cette grammaire s'inscrit dans le cadre des grammaires syntagmatiques tout en adoptant une structure syntaxique hiérarchisée décrite par des propriétés. Dans ce qui suit, nous présentons ses éléments essentiels, à savoir les catégories et les propriétés, tout en expliquant les différents aspects qui les caractérisent.

5.4.1 Catégories

Une catégorie en GP est formée par une structure de traits typés définissant les informations linguistiques pouvant intervenir dans la spécification de contraintes. Chaque trait est un couple <étiquette, valeur>. La valeur d'un trait peut être atomique ou complexe. La structure de traits formant la catégorie peut alors être plus ou moins spécifiée selon les besoins d'analyse. De nouveaux traits peuvent également être introduits dans la même structure. Ceci permet de représenter les différents niveaux d'analyses (syntaxique, sémantique, . . .) sans affecter le processus général d'analyse (Blache, 2001).

5.4.1.1 Traits et typage de traits

Comme nous avons déjà mentionné, les traits d'une catégorie sont typés. Ceci veut dire que les traits partageant des caractéristiques communes sont regroupés dans un type spécifique. A ce type, plusieurs sous-types peuvent être associés héritant automatiquement ses traits, ainsi que des traits spécifiques (dits appropriés aux sous-types). Les types et leurs sous-types peuvent être organisés sous la forme d'hiérarchies de types, tels qu'un descendant est plus spécifique que son ancêtre et deux frères (descendants d'un même ancêtre) se distinguent par un ensemble non vide de traits. Nous citons comme exemple celui de la figure 2.2 (Blache, 2001) :

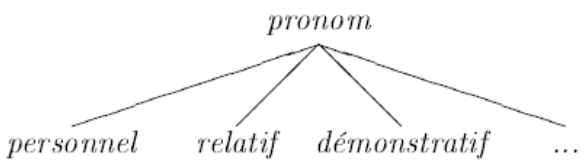


Figure 2.2: Exemple de type et ses sous-types

Dans l'exemple illustré par la figure 2.2 ci-dessus, les nœuds pronom personnel, pronom relatif et pronom démonstratif sont des spécifications du type général pronom. Le type spécifique pronom relatif par exemple ajoute d'autres informations (e.g. la caractéristique d'introduction d'une proposition) en plus des informations propres à pronom les récupérant automatiquement.

Grâce à l'utilisation des types, les informations manipulées deviennent non seulement structurées et factorisées, mais aussi cohérentes dans la mesure où chaque type n'aura jamais une information qui ne lui correspond pas.

5.4.1.2 Hiérarchies de types

Comme nous avons déjà mentionné, les types et leurs sous-types sont organisés sous la forme d'hiérarchies de types. Ces hiérarchies distinguent différents niveaux de spécification de l'information. Chaque catégorie définie par un ensemble de traits est alors associée à un certain niveau de spécification (granularité) de son type. Une hiérarchie est représentée sous la forme d'un arbre où la racine représente un type et les nœuds descendants sont des sous-types plus spécifiques de leurs nœud parent. La figure 2.3 ci-dessous illustre la hiérarchie du type « *cat* » caractérisant les catégories lexicales en français.

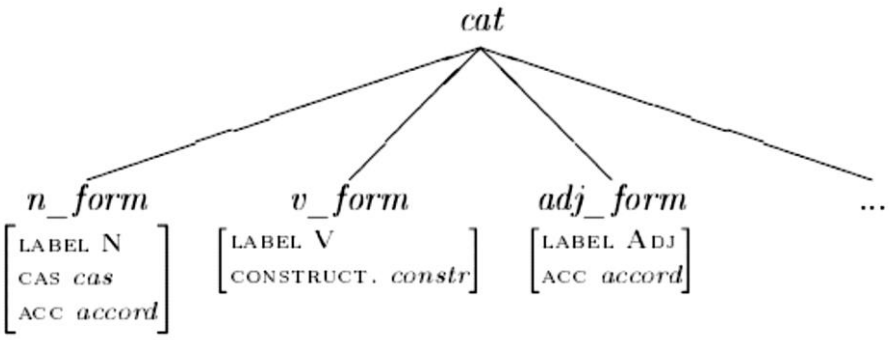


Figure 2.3: Hiérarchie du type « cat» caractérisant les catégories lexicales en français

Dans la hiérarchie du type « cat » donnée par la figure 2.3 ci-dessus, un seul trait, appelé CAT, est spécifié pour ce type. Ce trait a une valeur cat complexe (donc formée de plusieurs traits). Le premier trait (noté LABEL) composant sa valeur indique simplement l'étiquette de la catégorie lexicale traitée (e.g. N, V, Adj, etc.) pour désigner les différentes natures grammaticales (Nom, Verbe, Adjectif, etc). Tous les traits sont représentés dans une matrice sous le type concerné, leurs étiquettes sont en majuscules et leur type est en italique.

Grâce à cette hiérarchie de types, les catégories lexicales seront distinguées par sous-types en fonction de la différence des valeurs de traits appropriés à ces sous-types. Cette hiérarchie

de types introduit des sous-hiérarchies pour caractériser les traits à valeur complexe. C'est le cas par exemple du trait CONSTRUCT pour décrire une catégorie donnée contenant une construction verbale et du trait ACC pour désigner l'accord d'une catégorie donnée. Nous remarquons ici l'importance de la notion de « trait approprié » dans le contrôle du niveau d'intégration d'un trait à la structure de traits formant une certaine catégorie. Cette notion permet aussi d'exprimer les relations qui peuvent exister entre deux sous-types. Pour mieux expliquer cette idée, nous présentons dans la figure 2.4 suivante les deux hiérarchies de types « *construction_verbale* » et « *flexion* » qui décrivent le syntagme verbal noyau.

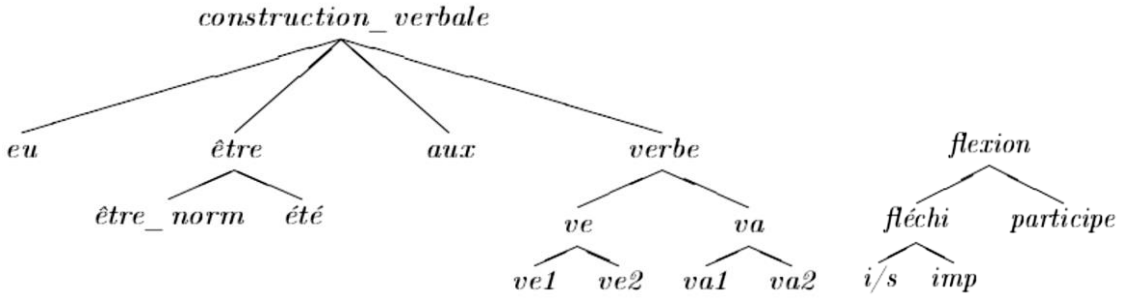


Figure 2.4: Hiérarchies de types « construction_verbale » et « flexion »

Dans la figure 2.4 ci-dessus, le trait CONSTRUCT, qui est déjà mentionné dans la figure 2.3, est caractérisé par le type « *construction_verbale* ». Ce type par exemple est spécifié par 4 sous-types possibles. Le type « flexion » porte par contre sur un autre trait, celui des mots fléchis ou les participes. Selon la langue française, il existe un certain nombre de contraintes entre ces deux types (« *construction_verbale* » et « *flexion* »). Nous les représentons intuitivement par les implications illustrées dans la figure suivante :

$$verbe \Rightarrow flexion$$
$$aux \Rightarrow i/s$$
$$être\_norm \Rightarrow fléchi$$

Figure 2.5: Relations possibles entre les deux types « construction_verbale » et « flexion »

Les relations, données par la figure 2.5 ci-dessus, indiquent que tout verbe doit avoir une caractéristique de flexion. Plus particulièrement, les verbes être normaux ne peuvent être que fléchi. De même, les flexions des auxiliaires ne peuvent être que du sous-type *i/s*. Ceci exige l'introduction d'un trait (soit FLEXION) caractérisant les sous-types *flexion* appropriés aux différents sous-types du trait CONSTRUCT.

Grâce à la formalisation des catégories en structures de traits organisés en hiérarchies de types, il devient possible de spécifier la granularité de l'analyse (générale ou détaillée) par la fixation d'un niveau de spécification bien déterminé des catégories utilisées. Ceci permet de

réaliser un jeu d'étiquettes à travers les projections pour générer des propriétés compatibles avec le niveau de granularité choisi.

Les exemples illustrés par la figure 2.6 suivante décrivent deux entrées représentant respectivement deux formes du verbe « aimer » pour montrer la variabilité des valeurs d'un trait en fonction de son type :

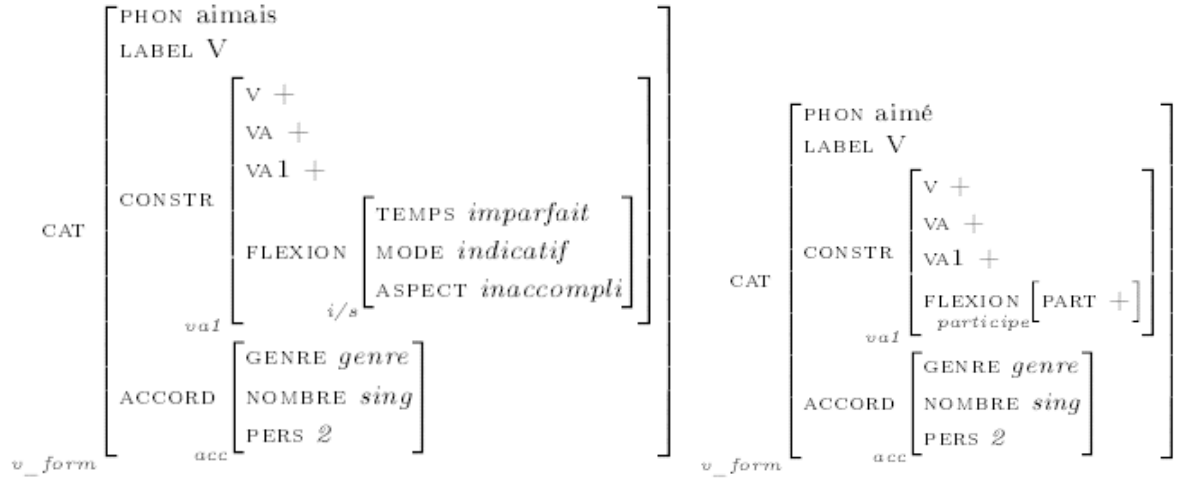


Figure 2.6: Exemples de représentation des catégories des deux entrées « aimais » et « aimé»

Dans les exemples donnés par la figure 2.6 ci-dessus, l'existence du trait FLEXION qui change de forme selon qu'il est de type i/s ou participe montre la différence entre ces deux entrées.

5.4.2 Propriétés

Les propriétés sont utilisées pour fournir un encodage encapsulé de l'information linguistique. Cet encodage est effectué à travers des contraintes locales portant sur un ensemble de catégories dégagées d'un énoncé donné. De même, chaque catégorie est décrite par un sous-ensemble de propriétés. Comme nous l'avons déjà souligné à plusieurs reprises, ce qui caractérise les propriétés est le fait qu'elles sont toutes définies au même niveau, c'est à dire qu'elles ne sont ni dépendantes les unes des autres ni ordonnées entre elles. En plus, elles représentent des relations traitant toutes les informations de manière explicite. Cette représentation est différente des celle de constituants qui se limite à la définition explicite d'une relation unique : la hiérarchie. Les relations hiérarchiques représentent l'information syntaxique de manière holistique. Cette représentation ignore les cas où cette information est incomplète ou mal formée (Blache et Rauzy, 2012). Ce type de relations ne traite pas les phénomènes linguistiques complexes comme les relatives, les anaphores et les coordinations. Les propriétés par contre peuvent décrire ces phénomènes grâce à leur représentation décentralisée et locale des informations linguistiques.

5.4.2.1 Types de propriétés

Les propriétés peuvent être du niveau lexical (comme les propriétés morphologiques ou phonologiques) ou bien du niveau syntaxique. Selon (Blache, 2001), les propriétés syntaxiques portent sur sept différents types de contraintes. Les catégories que ces propriétés décrivent sont donc d'un niveau syntagmatique. La liste de propriétés est non exhaustive est peut être enrichie. On cite les sept types suivants à partir de (Blache, 2001) : la linéarité, la constituance, l'unicité, l'obligation, l'exigence, l'exclusion et la dépendance. Les propriétés d'unicité, d'obligation et de constituance sont des relations unaires portant sur une seule catégorie. Les autres sont par contre des relations binaires faisant la connexion entre deux catégories. Dans (Bès, 1999a), on regroupe également les propriétés de constituance, d'unicité, d'obligation, d'exigence et d'exclusion sous le terme de propriétés d'existence. Nous présentons en plus de détails ces différents types en se basant sur (Blache, 2001) puis nous joignons, dans une sous-section suivante, quelques exemples explicatifs de propriétés.

- **Linéarité** ($\prec$) : La notation $A \prec_{SX} B$ indique une relation d'ordre linéaire décrivant le syntagme SX entre les deux catégories A et B. Ceci veut dire que si A et B apparaissent à la fois dans le syntagme SX, on trouve que A précède absolument B.
- **Constituance** (const) : La notation const(SX) = C regroupe l'ensemble des catégories susceptibles d'être utilisées dans la formation du syntagme SX. Toute catégorie apparaissant dans le syntagme SX devrait faire partie de l'ensemble C.
- **Unicité** (unic) : La notation unic(SX) = C regroupe l'ensemble des catégories ne pouvant apparaître qu'une seule fois dans la réalisation du syntagme SX.
- **Obligation** (oblig) : La notation oblig(SX) = C regroupe l'ensemble des noyaux possibles du syntagme SX. Ceci veut dire que le syntagme SX doit avoir obligatoirement dans chacune de ses réalisations une catégorie de l'ensemble C. Cette catégorie appelée noyau est également unique.
- **Exigence** ($\Rightarrow$) : La notation $A \underset{\Rightarrow}{\square\square} B$ exprime l'exigence de cooccurrence entre les deux catégories A et B dans le syntagme SX. Ceci indique que, si la catégorie A apparait dans une réalisation du syntagme SX, il faut absolument trouver la catégorie B dans la même réalisation. La relation réciproque n'est pas absolument vraie. C'est-à-dire que la présence de la catégorie B n'exige pas la présence de la catégorie A.
- **Exclusion** ($\otimes$) : La notation $A \underset{\otimes}{\square\square} B$ exprime l'impossibilité de cooccurrence entre les deux catégories A et B dans le syntagme SX. Ceci indique que, si la catégorie A apparait dans une réalisation du syntagme SX, il faut ne faut jamais trouver la catégorie B dans la même

réalisation. Cette relation est symétrique, ce qui fait que la relation réciproque devrait être également vraie. C'est-à-dire que la présence de la catégorie B impose la présence de la catégorie A.

- **Dépendance** ($\rightsquigarrow$) : La notation A $\rightsquigarrow_{SX}$ B désigne une relation de dépendance entre les deux catégories A et B dans le syntagme SX. Ceci indique que dans le syntagme SX, A dépend de B. Cette relation de dépendance est essentiellement sémantique mais peut être syntactico-sémantique prenant en compte la connexion structurale avec la connexion sémantique comme le fait les grammaires de dépendances (Tesnière, 1959).

5.4.2.2 Exemples de propriétés

Nous pouvons présenter comme exemples de propriétés, ceux proposés par (Blache, 2001). Ces exemples, portant sur des structures syntagmatiques tirées de la langue française, décrivent des propriétés du syntagme nominal (NP). Nous citons alors :

- La propriété *const(NP) = {Det, Noun, AdjP, Sup}* indique la constituance. Elle énumère la liste des catégories possibles du NP. Cette liste comprend un déterminant, un nom, un syntagme adjectival et un superlatif.
- La propriété *unic(NP) = {Det, Card}* constate l'interdiction d'avoir plus d'un déterminant ou d'un cardinal dans un même NP.
- La propriété *oblig(NP) = {Noun, AdjP}* indique que le NP doit avoir comme l'un des composants un nom ou un syntagme adjectival.
- La propriété $Det \prec_{NP} N$ indique qu'un déterminant précède toujours un nom dans un NP.
- La propriété $Nc \overset{NP}{\Rightarrow} Det$ montre que si un nom commun se présente dans un NP, un déterminant doit être trouvé aussi dans le même NP.
- La propriété $Card \overset{NP}{\otimes} Det[indef]$ montre qu'un cardinal et un déterminant indéfini ne peuvent pas coexister dans un NP.
- La propriété $Det \rightsquigarrow_{NP} Noun$ souligne qu'un déterminant dépend d'un nom dans un NP.

A partir des exemples donnés ci-dessus, nous pouvons constater que les GP fournissent des propriétés permettant de représenter plusieurs relations d'une manière simple et directe. Cela favorise leur utilisation dans plusieurs axes de recherche du domaine de TALN. La sous-section suivante montre quelques domaines d'utilisation des GP.

5.5 Domaines d'utilisation des GP

L'apport du formalisme de GP a été exploité par plusieurs travaux desquels nous pouvons citer quelques-uns par domaines d'application comme l'analyse syntaxique (symbolique et stochastique) et l'enrichissement des treebanks.

5.5.1 Analyse syntaxique symbolique

Parmi les travaux qui ont exploité les GP pour l'analyse syntaxique symbolique, nous pouvons commencer par citer celui de Duchier et al. (Duchier et al. 2010) qui a utilisé une sémantique formelle pour modéliser l'analyse syntaxique en GP sous la forme d'un problème de satisfaction de contraintes. Cette sémantique est définie en théorie de modèles en se basant sur le formalisme de GP. En 2011, ces auteurs ont poursuivi ce travail par l'exploitation du nouveau modèle d'analyse pour développer une extension traitant de nouveaux types de propriétés qui permettent l'expression de relations entre constituants syntaxiques sous la forme de contraintes sur les structures de traits (Duchier et al.2011).

Plus anciennement, Vanrullen et al. (2005) ont formulé mathématiquement les GP pour les utiliser comme formalisme d'analyse par satisfaction de contraintes mais pour assurer une granularité d'analyse syntaxique contrôlable.

Prost (Prost, 2009), quant à lui, il a proposé un analyseur à base de GP permettant de générer la structure de constituant de mérite maximum d'une phrase donnée pouvant être agrammaticale.

5.5.2 Analyse syntaxique stochastique

A la différence des analyseurs syntaxiques symboliques qui se basent purement sur des règles de productions, les analyseurs syntaxiques stochastiques utilisent des heuristiques définis en fonction de la fréquence d'apparition des structures. Ceci permet d'alléger largement le processus d'analyse par rapport à l'approche symbolique en réduisant le temps d'exécution ainsi que la consommation mémoire. La contribution de Blache et Rauzy (Blache et Rauzy, 2013) s'inscrit dans cet axe de recherche. Elle propose, en effet, une démarche d'hybridation d'un contrôle symbolique avec une analyse syntaxique probabiliste. Cette démarche repose sur des heuristiques représentant les scores d'apparition des propriétés. Ils se sont basés sur la définition des propriétés les plus probables pour une affectation donnée ainsi que sur la quantification de la cooccurrence de contraintes satisfaites simultanément.

5.5.3 Enrichissement de treebanks

A cause du manque de ressources linguistiques robustes et riches décrivant véritablement la langue naturelle, plusieurs efforts se sont dirigés à construire de plusieurs façons de nouvelles ressources pour diverses langues. L'enrichissement des treebanks était l'un des efforts fournis pour cette perspective. Leur motivation repose sur la richesse et la fiabilité de ces ressources linguistiques déjà disponibles pour construire d'autres nouvelles plus robustes et surmontant leurs insuffisances en terme de représentation actuelle et de couverture de phénomènes linguistiques particuliers. La première contribution utilisant les GP dans ce cadre a été proposée par Blache et Rauzy (Blache et Rauzy, 2012 ; Rauzy et Blache, 2012) fournissant une représentation syntaxique enrichie du treebank français FTB. Cette représentation ajoute aux constituants (catégories grammaticales) les propriétés syntaxiques qui peuvent les relier. Ceci a permis de rendre explicite plusieurs relations implicites. Un tel corpus serait largement bénéfique pour le genre d'analyseurs qu'ils ont proposé ensuite en 2013. Blache a aussi utilisé généré une grammaire de propriétés pour la langue chinoise pour un éventuel enrichissement de corpus pour cette langue (Blache, 2014). Dans le même cadre, Bensalem et elkaroui ont développé la même technique d'induction de GP à partir du treebank arabe ATB. La grammaire obtenue a servi ensuite pour l'enrichissement de l'ATB par ces propriétés (Bensalem et al., 2015).

5.5.4 Autres domaines

L'enrichissement de corpus avec des représentations à base de GP ne s'est pas limité aux treebanks, mais s'est étendu à des corpus sonores. C'est le cas notamment de la base de données Aix-MARSEC (Auran et al., 2004). En effet, Aix-MARSEC est formée de deux principaux composants : les enregistrements sonores français numérisés du corpus MARSEC et leurs annotations. Aux neufs différents niveaux de spécification que les annotations de cette base de données ont atteints, s'ajoute deux niveaux supplémentaires : l'annotation syntaxique ainsi qu'un système de GP relatif.

Pour la même langue, Guénot et al. (Guénot et al., 2003) ont proposé un outil graphique appelé Accolade permettant le développement des GP avec une représentation complète et ergonomique.

6 Conclusion

Pour récapituler, tout au long de ce chapitre nous avons fait un bref aperçu sur les théories linguistiques basées sur les contraintes et plus particulièrement le formalisme de grammaire de propriétés. La modélisation d'un tel formalisme comme un CSP pour réaliser l'analyse l'a favorisé par rapports aux autres approches. Ainsi, il s'appuie effectivement sur les contraintes qui sont les propriétés puisqu'il accède directement aux variables contraintes qui sont les catégories grammaticales. Nous avons alors défini les notions fondamentales qui entourent ce formalisme pour finir par citer quelques domaines d'application. Le troisième mot-clé de notre thèse occupe entièrement le troisième chapitre à savoir : les treebanks. L'importance de ces ressources est remarquée à travers leur utilisation dans toutes les contributions de notre thèse.

Chapitre 3

Treebanks

1 Introduction

Les treebanks sont des ressources linguistiques annotées de façon contrôlée à l'aide de linguistes. Ils sont également soumis à des consensus, ce qui favorise leur fiabilité. Ces points forts incitent à les utiliser dans différents domaines comme l'enseignement, et le TALN. De nombreux travaux proposés dans le domaine TALN exploitent les treebanks pour réaliser plusieurs phases comme l'évaluation d'analyseurs syntaxiques et l'induction de grammaires et l'optimisation des analyses. Ainsi, il est possible d'induire pour l'analyse, à partir des treebanks, plusieurs outils linguistiques comme les grammaires probabilistes, les grammaires de propriétés et les grammaires profondes. Dans ce contexte, nous avons mené des recherches sur les treebanks pour comprendre comment sont-ils construits et quels rôles jouent-ils pour pouvoir exploiter ceux qui satisfont nos besoins.

Les réponses aux questions déjà posées sont organisées dans le présent chapitre comme suit : Nous commençons tout d'abord par spécifier la notion de « treebank » Nous présentons ensuite leurs formats de conception en décrivant leurs différentes caractéristiques ainsi que les schémas d'annotations qu'ils peuvent adopter. Puis, nous exposons les méthodes et les outils de développement de ces treebanks. Finalement, nous jetons l'œil principalement sur les treebanks les plus connus, et qui sont développés pour la langue arabe qui nous intéresse le plus. Dans ce cadre, une comparaison entre ces différents treebanks est menée en fonction de plusieurs facteurs pour faciliter le choix du treebank à exploiter.

2 Définition des treebanks

Le terme « treebank » a été spécifié par Geoffrey Leech (Sampson, 2003). Il peut être défini comme étant un corpus annoté linguistiquement de telle sorte qu'il inclut des analyses grammaticales au-delà du niveau POS. Ces analyses sont représentées sous forme d'arbres, d'où l'origine du terme « tree » dans « treebank » qui désigne le mot arbre en anglais. Néanmoins, dans l'usage courant, le terme « treebank » ne concerne pas seulement cette forme de représentation, mais plutôt tout type de corpus analysé grammaticalement.

Il faut noter aussi que la définition du terme « treebank » se limite, habituellement, à l'utilisation d'analyses grammaticales annotées manuellement ou avec une post-correction, ce qui le différencie du terme « corpus parsé » (« parsed corpus » en anglais) qui est plus employé avec les corpus analysés automatiquement (Abeillé et al., 2003).

Depuis les années 70, les treebanks ont été représentés sous quelques formes. Le corpus annoté développé dans (Teleman, 1974) représente un exemple significatif et riche. Ce treebank

génère une annotation syntaxique manuelle d'environ 300,000 mots du suédois parlé et écrit. Cette annotation traite aussi bien les structures de syntagmes que les fonctions grammaticales. Toutefois, c'est uniquement dans les quinze dernières années que les treebanks apparaissent à des langues diverses, développés surtout en combinant un traitement automatique avec une annotation manuelle ou avec une post-correction. Pour plus de détails sur les treebanks, nous vous invitons à consulter les références suivantes (Abeillé, 2003 ; Hinrichs et Simov, 2002 ; Nivre et Hinrichs, 2003 ; Kübler et al., 2004).

3 Conception des treebanks

Le développement de treebank nécessitant beaucoup d'effort, ils sont conçus de telle sorte qu'ils permettent de servir à la fois plusieurs objectifs. Leur efficacité à satisfaire un objectif spécifique forme déjà un point critique et discutable, et plusieurs choix de conception peuvent être considérés pour représenter les différents points de vue (Abeillé, 2003). Pour expliquer les détails de conception d'un treebank, nous présentons dans ce qui suit les caractéristiques du corpus source à annoter ainsi que les schémas (formats) d'annotation employés à ce corpus.

3.1 Caractéristiques du corpus source

La conception d'un treebank nécessite tout d'abord un bon choix des données sources à annoter. Ce choix dépend de plusieurs considérations à prendre en compte comme l'utilisation prévue du treebank, la disponibilité des données et des outils d'analyse, la nature de leur langue (écrite ou parlée), leur genre (texte équilibré ou concernant un domaine spécifique), leur taille.

Concernant le choix de la nature du langage à utiliser pour concevoir un treebank, les théories de représentation syntaxique se sont concentrées beaucoup plus au langage écrit au profit de celui parlé pour la conception des corpus linguistiques et surtout des treebanks. Il existe, cependant, quelques treebanks, pouvant être en nombre croissant, développés à base de données de langages parlés, tels que le corpus Christine pour l'anglais (Sampson, 2003), les Treebanks Tübingen de l'allemand, de l'anglais et du japonais parlés (Hinrichs et al., 2000), et le corpus danois parlé (CGN) (Wouden et al., 2002).

Il faut également définir le genre du corpus à traiter dans le treebank, il s'agit de choisir un échantillon équilibré abordant différents domaines dans différentes natures de langues ou bien de se concentrer à un texte dans un domaine bien spécifique. Historiquement, plusieurs treebanks se sont basés sur des corpus préétablis, héritant donc leurs choix de conception. Ainsi, le corpus Susanne Corpus (Sampson, 1995) est fondé sur une partie du corpus Brown de

l'anglais américain (Kučera et Francis, 1967), qui représente un corpus équilibré. Actuellement, la majorité des treebanks sont, toutefois, basés sur des ressources contemporaines uni-langues ayant la particularité d'être facilement accessibles. Ce sont les articles de presse. C'est notamment le cas de la section « Wall Street Journal » du « Penn Treebank" (Marcus et al., 1993), représentant un modèle de treebanks dans une vaste gamme de langues. Il existe également d'autres formes de treebanks. Les treebanks historiques, par exemple, sont basés sur des données de périodes antérieures comme le Penn-Helsinki Parsed Corpus du moyen anglais (Kroch et Taylor, 2000) et le Partially Parsed Corpus du portugais médiéval (Rocio et al., 2003). Tandis que les treebanks parallèles, ils sont basés sur des textes dans une langue ainsi que leurs traductions dans d'autres langues. Le « Czech-English Penn Treebank », par exemple, a été développé pour la tâche de traduction automatique (Čmejrek et al., 2004).

La taille du corpus est un point important à considérer. Malgré les recherches avancées en automatisation du processus d'annotation, la tâche d'annotation linguistique et surtout celle des structures grammaticales exige encore beaucoup d'efforts. Pour cela, il fallait faire un compromis entre la quantité des données à inclure et la quantité d'annotations à appliquer à ces données. Pour cela, selon l'usage prévu, il est préférable de construire de petits treebanks si l'on veut intégrer des annotations détaillées, comme le cas du corpus Susanne (Sampson, 1995), ou bien de plus grands mais avec moins de détails en annotation comme la version voyéllée originale du « Penn Treebank » (Marcus et al., 1993).

3.2 Schémas d'annotation

Pour déterminer le format d'annotation pour un treebank, il faut suivre deux étapes : La première porte sur l'analyse linguistique. Elle permet aux auteurs de définir la nature de la structure syntaxique, de spécifier le choix des catégories linguistiques et d'établir les directives d'annotations des phénomènes linguistiques particuliers (comme les anaphores et les coordinations). La seconde étape, quant à elle, concerne la représentation formelle (appelée aussi l'encodage). Elle permet de spécifier le choix du format à utiliser pour l'annotation (un langage de balisage ou un texte normal), de la manière de stockage des données (dans un seul fichier ou dans plusieurs), etc (Nivre, 2008).

La plupart des schémas d'annotation sont organisés dans un certain nombre de couches, la plus basse contient les annotations avec des étiquettes POS ainsi que des traits morpho-syntactiques, de lemmes ou des analyses morphologiques. La Figure 3.1 montre un exemple significatif extrait du corpus Susanne (Sampson, 1995).

AT	The	The
VVDv	took	take
NN1c	swipe	swipe
VVGt	handling	handle
IO	of	of
JJ	federal	federal
NN2	funds	fund
YG	-	-
VVNt	granted	grant

Figure 3.1: Annotation de niveau mot dans le corpus Susanne

L'annotation de niveau mot caractérisant l'extrait donné par la figure 3.1 ci-dessus, concerne certains segments, chacun dans une ligne de trois colonnes. Chaque ligne contient successivement un l'étiquette POS (y compris ses traits morphosyntaxiques) du segment, le segment lui-même et son lemme.

Le niveau d'annotation structurelle (syntaxique ou même sémantique) nous concerne plus que celui de mots, vu que c'est le niveau qui distingue les treebanks des autres corpus annotés. Le choix du schéma d'annotation pour les treebanks à large échelle dépend de plusieurs facteurs, comme la théorie linguistique à adopter (spécifique ou neutre), les caractéristiques grammaticales de la langue à analyser, les schémas d'annotations utilisés pour d'autres langues (dans le cas d'une étude comparative ou de développement de treebanks parallèles), les préférences des différents groupes d'utilisateurs linguistiques (chercheurs, développeurs, enseignants, étudiants, etc.).

Le nombre de différents schémas d'annotations augmente avec le nombre de treebanks de différentes langues qui est en croissance continue. Généralement, nous pouvons distinguer trois genres d'annotation principaux à partir desquels quelques treebanks existants combinent deux ou tous les trois : l'annotation de constituants, l'annotation fonctionnelle et l'annotation sémantique.

L'annotation de constituants permet de spécifier des étiquettes POS uniquement pour les constituants qui sont des mots ou des structures syntagmatiques. Le corpus Lancaster (Garside et al., 1992) et la version originale du « Penn Treebank » (Marcus et al., 1993) représentent des exemples de treebanks qui suivent ce type d'annotation. La Figure 3.2 montre un exemple significatif extrait du treebank « IBM Paris » qui utilise le schéma d'annotation du corpus Lancaster.

```
[N Vous_PPSA5MS N]
[V accedez_VINIP5
        [P a_PREPA
                [N cette_DDEMFS session_NCOFS
                N]
        P]
        [Pv a_PREP31 partir_PREP32 de_PREP33
                [N la_DARDFS fenetre_NCOFS
                        [A Gestionnaire_AJQFS
                                [P de_PREPD
                                        [N
                                        taches_NCOFP
                                        N]
                                P]
                        A]
                N]
        Pv]
V]
```

Figure 3.2 : Annotation de constituants dans le treebank « IBM Paris »

L'inconvénient de ce genre d'annotation réside dans le fait que les structures qu'il produit sont plates, vu qu'il évite d'utiliser des catégories syntagmatiques intermédiaires, ce qui peut augmenter largement le nombre de constructions pour une même catégorie syntagmatique. Dans (Charniak, 1996), par exemple, on a pu extraire à partir d'un échantillon de 300,000 mots du Penn Treebank, jusqu'à 10,605 règles hors-contexte, dont seulement 3943 apparaissent plus qu'une fois dans l'échantillon. Des variations de ce genre d'annotation ont été adoptées. D'une part, dans les treebanks français, on a annoté des chunks syntaxiques au lieu d'un arbre de structures syntagmatiques complet (Abeillé et al., 2003 ; Vilnat et al., 2003). D'autre part, dans les treebanks Tübingen de l'allemand, on a introduit une couche de champs topologiques en haut de la structure des constituants de base (Hinrichs et al., 2000).

L'annotation fonctionnelle décrit les liens de dépendances entre les différents éléments dans le corpus. Ce genre d'annotation est considérée par les défendeurs de la syntaxe de dépendances, comme Mel'čuk (1988), plus fondamentale que celui de constituants. En effet, ils soutiennent une idée différente de celle adoptée depuis (Chomsky 1965), en considérant que les fonctions grammaticales sont dérivables de la structure fonctionnelle mais pas de celle de constituants. Ce genre d'annotation est de plus en plus adopté dans les dernières années. Plusieurs treebanks basés sur ce genre d'annotation peuvent être cités : le « Prague Dependency Treebank » de la langue tchèque (Hajič 1998 ; Böhmová et al., 2003), « le METU Treebank » de la langue turque (Oflazer et al., 2003), le « Danish Dependency Treebank » de la langue danoise (Kromann, 2003), etc. chimney. La figure 3.3 suivante montre un exemple d'annotation

fonctionnelle à travers la phrase («Kominík vymetá komíny. » Les ramoneurs de cheminées ramonent les cheminées.) dans le « Prague Dependency Treebank ».

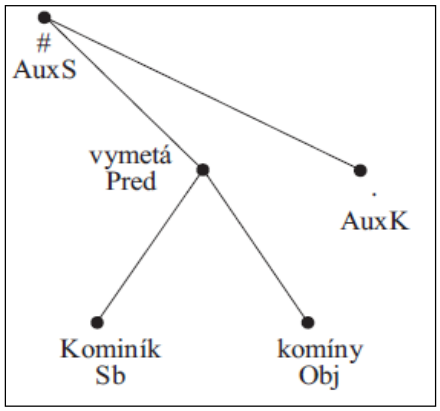


Figure 3.3: Annotation fonctionnelle dans le « Prague Dependency Treebank »

L'annotation fournie par la figure 3.3 ci-dessus apparait directement au-dessus de l'annotation morphologique sans aucune couche de structure de constituants.

La combinaison entre les structures de constituants et les fonctions grammaticales est le genre d'annotation le plus dominant et existe à différentes variations :

- Le « Penn Treebank II » (Marcus et al. 1994), basé sur l'annotation de constituants, par exemple, ajoute des étiquettes grammaticales à son schéma d'annotation d'origine. Le nouveau schéma d'annotation a été adopté par plusieurs autres treebanks comme le « Penn Arabic Treebank » (Maamouri et Bies, 2004), le « Penn Chinese Treebank » (Xue et al., 2004) et le « Penn Korean Treebank » (Han et al., 2002).
- L'annotation de Tiger pour l'allemand (Brants et al., 2002), est une autre manière de combinaison, elle est représentée dans un graphe tel que les nœuds forment les catégories structurelles et les arêtes représentent des fonctions syntaxiques. Cette représentation accepte les branches croisées pour modéliser les constituants discontinus.
- L'annotation de VISL (Visual Interactive Syntax Learning) permet également une combinaison acceptant les constituants discontinus. Elle a été développée pour des fins pédagogiques et appliquée à, 22 langues à petite échelle, puis utilisée pour des treebanks plus grand pour le portuguais (Afonso et al., 2002) et le danois (Bick, 2003).
- L' « Italian Syntactic-Semantic Treebank » (Montemagni et al., 2003) introduit dans sa combinaison deux couches d'annotation indépendantes, l'une pour les constituants, l'autre pour les dépendances.

L'annotation sémantique ajoute à l'annotation fonctionnelle des rôles sémantiques. Dans le « Proposition Bank » (Kingsbury et Palmer, 2003), qui est basé sur le « Penn Treebank », des structures de prédicats-arguments sont ajoutées dans une nouvelle couche. Dans le « Prague Dependency Treebank », en plus des liens de dépendances qu'il représente (comme dans la figure 3.3), une couche d'analyses tectogrammaticales a été introduite décrivant des rôles sémantiques (Hajičová 1998). Le « Sinica treebank » du chinois quant à lui, utilise une combinaison de structures de constituants et d'annotation fonctionnelle impliquant des rôles sémantiques (Chen et al., 2003). Une autre manière d'annotation sémantique consiste à définir les sens des mots. Elle est adoptée par l' « Italian Syntactic-Semantic Treebank » (Montemagni et al., 2003) et le « Hellenic National Treebank » du greque (Stamou et al., 2003) adoptent ce genre d'annotation sémantique. Le treebank Tiger (Kunz et Hansen-Schirra, 2003), et le « Penn Discourse Treebank » (Miltsakaki et al., 2004), par contre, permettent d'annoter les phénomènes sémantiques du discours. Malgré ces différents exemples de treebanks, le rôle de l'annotation sémantique est considéré plutôt marginal dans le développement des treebanks, ce rôle peut être renforcé dans le futur.

Indépendamment du genre d'annotation employé, il y a un intérêt croissant aux schémas d'annotation se basant dans l'annotation des phrases du corpus sur une théorie linguistique spécifique. En effet, la théorie de HPSG (Head-Driven Phrase Structure Grammar) a été utilisée par des treebanks pour l'anglais (Oepen et al., 2002) et le bulgare (Simov et al., 2002). Par contre la théorie FGD (Functional Generative Description) a été employée par le « Prague Dependency Treebank » (Sgall et al. 1986). Le « Penn Treebank » a été aussi appliqué mais pour une annotation selon la théorie de CCG (Combinatory Categorial Grammar) sous une version appelée « CCG-bank » (Hockenmaier et Steedman, 2002), et aussi la théorie de Lexical-Functional Grammar (Cahill et al., 2002).

Bien qu'une théorie linguistique neutre soit destinée à un grand groupe d'utilisateurs, elle peut tomber dans le risque d'être non suffisamment informative ou exigeant trop de compromis pour être utile à des applications spécifiques. La théorie spécifique, quant à elle, est clairement utile pour un groupe d'utilisateurs limité. Récemment, la combinaison des deux théories est l'objet de plusieurs efforts dans le but de maximiser l'utilité globale de la communauté de la recherche par l'utilisation de schémas d'annotation riches avec des conversions bien définies à des schémas plus spécifiques (Nièvre, 2003; Sasaki et al., 2003). En plus pour réduire l'effort requis pour produire un ensemble de treebanks à théories spécifiques, basés sur les mêmes données de la langue, un tel schéma peut permettre des comparaisons systématiques entre les différentes théories.

La raison pour laquelle la discussion tout au long de cette section a été axée sur l'annotation des treebanks pour le langage écrit est le fait l'annotation des données du langage parlé pose des difficultés particulières nécessitant des extensions des schémas d'annotation existants. La section Standard du « Penn Treebank » (Taylor et al., 2003) introduit un exemple d'annotation de disfluences avec le même schéma d'annotation que le treebank emploie. Mais plus généralement, on ne peut savoir dans quelle mesure les schémas d'annotation développés pour le langage écrit sont adéquats pour l'annotation du langage parlé, où ils nécessitent la prise en compte des notions interactives comme les actes de dialogue.

4 Méthodes et outils de développement des treebanks

Les méthodes et les outils de développement des treebanks ont largement évolué des premiers treebanks, où les annotations étaient manuelles jusqu'aux treebanks actuels, où les annotations utilisent des outils en émergence qui combinent plus ou moins le travail manuel avec le traitement automatique. Dans ce qui suit, nous présentons un aperçu des différentes méthodes et outils adoptés pour développer les treebanks.

4.1 Méthodes de développement des treebanks

Pour les premiers treebanks, l'annotation manuelle était leur seule solution faisable (e.g Teleman (1974) et Järborg (1986) pour le suédois). Cette annotation présentait l'inconvénient non seulement d'être gourmande en main-d'œuvre et donc coûteuse pour les grands volumes de données, mais aussi de ne pas garantir la cohérence entre les annotateurs s'ils sont très nombreux.

Depuis les années 90, on a commencé à introduire l'aspect automatique dans l'annotation à travers le corpus Susanne (Sampson, 1995). Une annotation purement automatique est plus avantageuse que celle manuelle dans la mesure où elle est à la fois cohérente et non couteuse. Toutefois, elle introduit un taux d'erreurs considérable, qui augmente typiquement avec la complexité du schéma d'annotation. Ceci indique qu'il est préférable d'utiliser l'annotation automatique seulement lorsque la quantité des données à annoter rend l'annotation manuelle ou semi-automatique très coûteuse, comme pour le corpus de Bank of English qui contient, 200 millions mots (Järvinen, 2003).

Etant donnés les inconvénients et les avantages de l'annotation manuelle et automatique, la plupart des treebanks actuels utilisent une combinaison des deux façons d'annotations afin

de maximiser l'efficacité du processus d'annotation tout en optimisant la fiabilité des données produites.

La méthode classique de combinaison consiste à réaliser une première étape d'analyse syntaxique (complète ou partielle) suivie d'une seconde étape de post-correction des erreurs détectées de l'étape d'analyse. Cette méthode a été utilisée pour le développement du « Penn Teebank » (Taylor et al., 2003) et du « Prague Dependency Treebank » (Böhmová et al., 2003). Une variation de cette méthode consiste à réaliser manuellement la désambiguïsation au lieu de recourir à la post-correction manuelle, (par exemple permettre à l'annotateur humain de choisir l'analyse correcte parmi l'ensemble des analyses possibles produites par l'analyseur automatique). Le « TreeBanker » (Carter 1997) et les treebanks « LinGO Redwood » (Oepen et al., 2002) ont adopté cette variation. L'inconvénient de cette méthode classique réside dans le fait que l'intervention manuelle (post-correction ou sélection d'analyses) peut augmenter le risque d'avoir des annotations influencées par le choix du correcteur humain en acceptant l'analyse proposée, même dans les cas douteux.

La méthode d'annotation interactive développée dans le cadre du projet NEGRA permet de réduire ce risque (Brants et al., 2003). Elle utilise, en effet, une cascade d'analyseurs automatiques dirigés par les données donnant à l'annotateur humain la possibilité de corriger la sortie d'un analyseur avant d'être introduit comme entrée dans le suivant (Brants et Plaehn, 2000). Grâce à ces analyseurs, le processus peut être aussi amorcé, puisque leur performance s'améliore progressivement avec la taille du treebank.

Outre la définition de la façon d'annotation (manuelle, semi-automatique ou automatique), il existe un autre point à considérer pour le développement d'un treebank, à savoir : l'ordre dans lequel les données sont fournies aux annotateurs humains pour la post-correction. Dans (Wallis, 2003), on défend l'idée que procéder à la correction transversale (e.g : vérifier tous les cas d'une même construction ensembles) peut améliorer la cohérence de l'annotation, au lieu d'utiliser la correction classique (phrase par phrase). Toutefois, la correction transversale est difficile à mettre en œuvre. En plus, dans le cas où on a des systèmes d'annotation multicouches, il faut définir l'ordre dans lequel ces différentes couches doivent être traitées et si les couches peuvent être annotées en parallèle ou bien, il existe des dépendances entre certaines couches exigeant un ordre particulier (Taylor et al ., 2003).

Il reste, enfin, un point à indiquer concernant la cohérence dans l'annotation du treebank. Cette dernière peut être améliorée en laissant plusieurs personnes annoter ou corriger les mêmes phrases et comparer leurs travaux. Cette procédure est, toutefois, très coûteuse et ne peut donc être utilisée que pour une petite partie du treebank pour vérifier l'accord inter-annotateurs. Une

méthode moins coûteuse peut être utilisée. Elle permet de faire l'analyse automatisée pour détecter les erreurs ou les incohérences potentielles dans l'annotation, comme proposé par (Dickinson et Meurers, 2003) et (Ule et Simov, 2004) par exemple.

4.2 Outils et normes

La plupart des outils utilisés pour le développement de treebanks sont aussi nécessaires pour tout corpus annoté, comme les segmenteurs (« tokenizers » en anglais) et les étiqueteurs (« taggers » en anglais) de POS. Les outils spécifiques au développement de treebanks, quant à eux, sont conçus essentiellement pour le prétraitement syntaxique ou pour une annotation spécialisée. Les exemples les plus connus d'analyseurs syntaxiques utilisés dans le développement de treebanks sont l'analyseur Fidditch (Hindle, 1994) et l'analyseur statistique dans (Collins et al. 1999) utilisés respectivement pour le « Penn Treebank » et le « Prague Dependency Treebank ». Les analyseurs partiels (ou « chunkers ») sont, par contre, utilisés pour le prétraitement syntaxique vue que l'analyse partielle est plus précise que celle complète. Le processus d'analyse peut être décomposé en cascade pour réaliser une annotation interactive, comme nous l'avons déjà mentionné dans la section précédente. L'outil interactif Annotate (Brants et Plaehn, 2000) est développé dans ce cadre pour établir les étapes de segmentation et d'étiquetage. Pour l'édition graphique, on a utilisé, par exemple, l'outil TrEd développé dans le « Prague Dependency Treebank » et l'outil Emacs qui est une extension d'un éditeur existant (Taylor et al., 2003 ; Abeillé et al., 2003).

Malgré l'abondance d'outils utilisés dans le développement de treebanks, il y a un manque d'outils normalisés. Seulement les projets de treebanks les plus influents comme le « Penn Treebank », le « Prague Dependency Treebank » et le projet TIGER ont conçu leurs propres normes. Ce manque s'explique par le fait que les schémas d'annotation utilisés dans les treebanks sont très variés. Même l'encodage n'est pas uniforme dans les schémas d'annotation. La présence d'un format de corpus unifiant les différentes représentations s'avère utile pour répondre au besoin accentué de réutilisation des outils et des ressources. Ceci permet non seulement d'offrir une interopérabilité des données annotées, mais aussi d'étendre l'applicabilité des outils divergents pour plusieurs treebanks et dans différents contextes de recherche. Pustylnikov et Mehler ont été les premiers qui proposent en 2008 un format unifiant les représentations des treebanks (Pustylnikov et Mehler, 2008). Ce travail consiste à transformer les treebanks de dépendances de 11 langues à un format unique en traitant les corpus sur trois niveaux de caractéristiques : le premier niveau porte sur les caractéristiques liées au genre du corpus (comme la structure syntaxique), le deuxième niveau inclut les

caractéristiques spécifiques des données alors que le troisième niveau, il porte sur les caractéristiques liées aux formats des treebanks de destination.

5 Utilisation des treebanks

La recherche linguistique représente l'un des domaines d'utilisation les plus importants généralement pour les corpus annotés et spécifiquement pour les treebanks dès leurs premières apparitions. Le domaine de traitement du langage naturel occupe une place plus centrale, au cours des dernières années, dans les travaux de développement de nouveaux treebanks et devient en plus la principale force motrice derrière ces travaux. Un aperçu sur ces deux domaines d'utilisation sera l'objet des deux sous-sections présentées ci-dessous. En plus de ces deux domaines, l'utilisation de treebanks s'étend à des cadres pédagogiques comme l'enseignement. C'est notamment le cas du projet VISL[12] qui a développé des treebanks d'enseignement pour 22 langues avec des outils pédagogiques y compris les jeux interactifs.

5.1 Recherche linguistique

Pour faire des recherches sur des phénomènes linguistiques dépendant de propriétés syntaxiques, l'utilisation d'un treebank s'avère utile plutôt que d'un corpus ordinaire, dans la mesure où il permet d'affiner ces recherches de sorte qu'on peut trouver les constructions particulières grâce à son annotation correcte. Les données du treebank sont utilisées dans la recherche linguistique de différentes manières. En effet, il y a des données qui sont utilisées surtout pour effectuer des études quantitatives permettant d'obtenir des fréquences et des probabilités. Ces dernières constituent des informations complémentaires aux données du treebank qui peuvent influencer la description des catégories linguistiques et des règles, comme dans (Bod et al., 2003). Il y a d'autres types de données spécifiques à des études qualitatives comme la recherche d'exemples authentiques d'une certaine construction linguistique.

5.2 Traitement du langage naturel

Deux manières d'utilisation des treebanks peuvent être essentiellement distinguées dans le domaine de traitement du langage naturel. La première consiste à évaluer les systèmes de traitement du langage naturel, en particulier des analyseurs syntaxiques. La seconde vise à induire des ressources linguistiques, en particulier des analyseurs linguistiques à travers l'apprentissage automatique des treebanks.

[12] http://visl.edu.dk

Au niveau de l'évaluation qui est un domaine très actif actuellement, et plus particulièrement celle des analyseurs syntaxiques, il y a deux types de données à utiliser. En effet, on a, d'une part, les suites de tests, (des collections de phrases par exemple), pour mesurer la couverture d'un analyseur syntaxique en termes de nombre de constructions qu'il peut traiter, indépendamment de la fréquence de chacune d'elles (Lehmann et al. 1996). D'autre part, on a des échantillons représentatifs au niveau fréquences des différentes constructions, extraits du treebank pour mesurer le rendement moyen prévu de l'analyseur en cas de distribution naturelle des données du treebank comme dans le corpus d'évaluation. Le plus important dans l'évaluation consiste à savoir comment mesurer la performance d'un analyseur par rapport à un échantillon d'un treebank annoté manuellement. Il ne faut pas adopter la mesure évidente, étant la proportion des phrases en sortie de l'analyseur correspondant à l'annotation de cet échantillon, car il s'agit d'une évaluation métrique qui ne tient pas compte de la forme d'erreur produite par l'analyseur (qu'elle soit une erreur d'un seul constituant ou de plusieurs). Les mesures d'évaluation les plus utilisés traitent, par contre, la correspondance partielle entre la sortie de l'analyseur et l'échantillon. Les plus connus sont celles de Parseval (Black et al. 1991), utilisant les données du « Penn Treebank » pour l'évaluation et basées sur le nombre de constituants correspondants entre la sortie de l'analyseur et l'échantillon. Ils existent d'autres mesures fondées sur les relations de dépendances et générant une meilleure façon de comparaison des analyseurs utilisant différentes représentations (Lin 1998 ; Carroll et al. 1998).

L'induction des ressources linguistiques pour l'analyse, surtout des grammaires probabilistes, représente une utilisation très réussie des treebanks au cours de la dernière décennie. Les exemples de ressources linguistiques ne manquent pas : les grammaires hors-contexte simples (Marcus et al. 1993 ; Hajič, 1998 ; Abeillé et al., 2003), les grammaires hors contextes probabilistes (Charniak, 1996 ; Mohri et Roark, 2006 ; Tounsi et VanGenabith, 2010), les grammaires de propriétés (Blache et Rauzy, 2012 ; Denis et al., 2009 ; Blache, 2014).

L'analyse à large couverture a également connu de grands progrès. Cette analyse utilise ce qu'on appelle « les grammaires profondes », qui sont obtenues par l'induction de modèles statistiques pour la sélection d'analyses induits principalement à partir des treebanks comme dans (Riezler et al., 2002) et (Toutanova et al., 2002). L'analyse orientée données (Bod, 1998) est une approche encore plus radicale. Elle élimine complètement la notion traditionnelle de grammaire et utilise un modèle probabiliste défini directement sur le treebank. Outre l'induction des grammaires et l'optimisation des analyseurs syntaxiques, les treebanks peuvent être utilisés pour induire d'autres ressources linguistiques étant pertinentes pour le traitement du langage

naturel comme par exemple l'extraction de cadres de sous-catégorisation (Briscoe et Carroll, 1997).

Ce que nous pouvons dégager de ce premier aperçu est la conclusion que les treebanks sont des ressources très précieuses aussi bien pour les applications de recherche linguistique que de traitement du langage naturel. Ces ressources riches en annotations peuvent être même indispensables lorsqu'il s'agit d'opter pour une analyse robuste à large couverture, quelle que soit la méthode d'analyse de base qui a été prévue. L'exploitation des treebanks arabes pour construire de nouvelles ressources linguistiques peut également enrichir l'ensemble des outils de traitement automatique de cette langue. Ceci nécessite l'étude des différents aspects qui caractérisent ces treebanks pour choisir le treebank qui répond aux besoins posés dans la problématique.

6 Treebanks pour la langue arabe

Nous pouvons citer quatre principaux treebanks pour la langue arabe : l'ATB (Maamouri et Bies, 2004), le PADT (Hajič et al., 2001), le CATiB (Habash et al., 2009) et le QADT (Dukes et al., 2010). Ces treebanks sont enrichis non seulement par des annotations morphologiques et syntaxiques (comme la voyellation, les étiquettes POS et les lemmes) mais aussi par des annotations sémantiques (comme les rôles sémantiques). Les treebanks ATB, PADT, CATiB et QADT se distinguent selon plusieurs aspects que nous détaillons dans un cadre comparatif dans ce qui suit :

6.1 Corpus source

Le projet de l'ATB a été lancé en 2001 à LDC (Maamouri et Bies, 2004). Il est composé d'articles de différentes agences de presse (France Presse, Xinhua, Al-Hayat, Ummah Press, etc.). Son corpus source a été divisé en ensembles de textes (divisions) pour les utiliser dans les recherches de TALN, puisque elles nécessitent généralement différents ensembles de textes pour les tâches d'apprentissage de modèles, de développement des techniques et d'évaluation finale (Diab et al., 2013). Il existe plus que 12 divisions de l'ATB comprenant environ 750K segments (Kulick, 2006). La division la plus utilisée est ATB3. Elle est composée de 599 documents contenant environ 350K segments.

La création de l'ATB représente un grand succès pour le traitement automatique de la langue arabe, étant donné son utilité pour un grand nombre de travaux dans différents domaines de TAL. De même, les autres treebanks ont été créés à base de l'ATB. Par exemple, le PADT

et le CATiB ont, tous deux, converti l'ATB vers leurs représentations syntaxiques en plus d'autres textes qu'ils ont annotés. En effet, en plus des données qu'il a annotées (provenant des articles de presse de Xinhua, Al-Hayat, An-Nahar), le PADT a converti les quatre premières divisions de l'ATB (ATB1, ATB2, ATB3 et ATB4) vers sa représentation syntaxique. En tout, il a annoté plus que 1,2M de segments (provenant de 1,1M de mots et représentés dans plus que 1M d'arbres) dont environ 900K segments annotés sont des divisions de l'ATB (Smrz et al., 2008). Alors que le CATiB a annoté des articles de presse (provenant de France Presse, Xinhua, Al-Hayat, Al-Asharq Al-Awsat, Al-Quds Al-Arabi, An-Nahar, Al-Ahram et As-Sabah) ainsi que ceux convertis à partir de l'ATB (ATB1, ATB2 et ATB3). Le CATiB contient alors plus que 1M de segments (provenant 841K mots et représentés dans 31,319 arbres) dont 735K segments annotés sont des divisions de l'ATB (Habash et al., 2009). Le QADT, quant à lui, traite le Coran. Le Coran est composé de 18,994 mots uniques, parmi lesquels le QADT annote 11K et les représente dans 2,500 graphes de dépendances syntaxiques (Dukes et Buckwalter, 2010).

6.2 Grammaire suivie

Vu qu'ils se basent sur des articles de presse comme corpus source, l'ATB et le PADT suivent dans leurs annotations les théories linguistiques de l'arabe standard moderne (en anglais Modern Standard Arabic, MSA) (Hajič et al., 2001 ; Maamouri et Bies, 2004). Le CATiB, quant à lui inspire ses annotations des études syntaxiques de l'arabe traditionnel, et ce dans le but de faciliter l'apprentissage des annotations sans avoir recours à des linguistes de haut niveau (Habash et Roth, 2009). Le QADT suit, également, l'arabe classique dans ses annotations. Ceci revient à la nature différente du texte source traité qui est le Coran, un texte religieux central avec le genre arabe traditionnel. Son annotation doit alors être compatible avec son genre pour ne pas impliquer le sens (Dukes et al., 2010).

6.3 Représentation Syntaxique

A la différence de l'ATB qui utilise la structure à base de constituants, le PADT, le CATiB utilisent la structure de dépendances. Le QADT offre une représentation hybridant les deux structures. Comme nous l'avons déjà mentionné dans le chapitre 1, la structure à base de constituants est une représentation arborescente dans laquelle les mots d'une phrase apparaissent sous forme de feuilles et les nœuds non terminaux représentent des catégories syntaxiques comme le syntagme nominal (Nominal Phrase (NP)) ou syntagme verbal (Verbal

Phrase (VP)). La figure 3.4 illustre la représentation en structure de syntagmes de la phrase 1 suivante :

خمسون ألف سائح زاروا لبنان و سوريا

xmswn Alf sAˆyH zArwA lbnAn wswryA (1)

Cinquante mille touristes ont visité le Liban et la Syrie

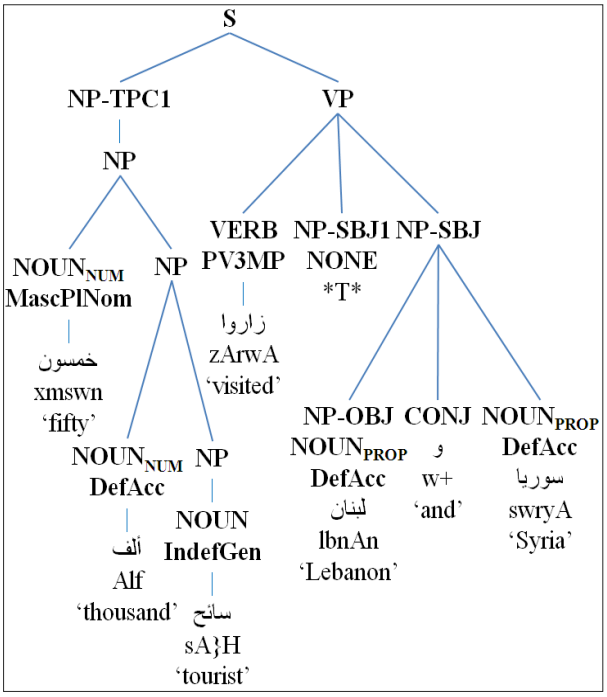


Figure 3.4: Représentation de la phrase 1 en structure de syntagmes dans l'ATB

Dans la figure 3.4 illustrant une représentation structurelle à base de constituants, les mots de la phrase 1 sont regroupés dans des segments étiquetés par des catégories syntaxiques. En effet, cette phrase étiquetée par la catégorie S est divisée aux segments de mots étiquetés respectivement par les NP-TPC1 et VP. Le segment de VP porte sur 5 mots de la phrase dont le deuxième est un mot vide. Selon cette représentation, les mots ont le même degré d'importance. Seulement les relations d'hiérarchie parent-fils et fils-frère sont considérées.

Les structures de dépendances prennent également la forme d'arbres sauf que les mots de la phrase sont les nœuds de l'arbre. En plus, plusieurs types de relations sont spécifiés. Le treebank PADT adopte ce type de représentation. Il se base, plus particulièrement, sur la description générative fonctionnelle (FGD) pour annoter son corpus (Hajič et al., 2004). La figure 3.5 illustre respectivement les représentations en structures de dépendances de la même phrase 1 aussi bien dans le PADT que dans le CATiB (Habash et al., 2009).

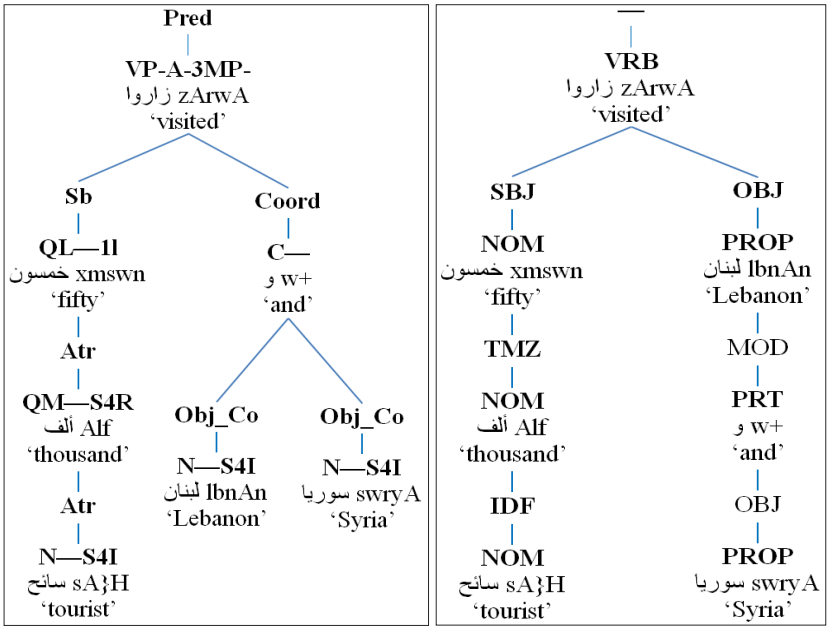


Figure 3.5: Représentation de la phrase 1 en structure de dépendances respectivement dans les treebanks PADT et le CATiB (Habash et al., 2009)

Les structures fournies par la figure 3.5 ci-dessus concernant respectivement les treebanks PADT et CATiB représentent les mêmes dépendences de façon différente. En effet, la relation entre le premier et le deuxième mot de la phrase est marquée comme une simple attribution du premier au deuxième dans le PADT. Cette relation est beaucoup plus spécifique dans le CATiB en la marquant comme un tamyiz.

Pour la représentation en structures hybrides dans le QADT, la figure 3.6 illustre une représentation d'un verset du Coran au lieu de la phrase 1 puisque le QADT se base sur le Coran comme corpus source.

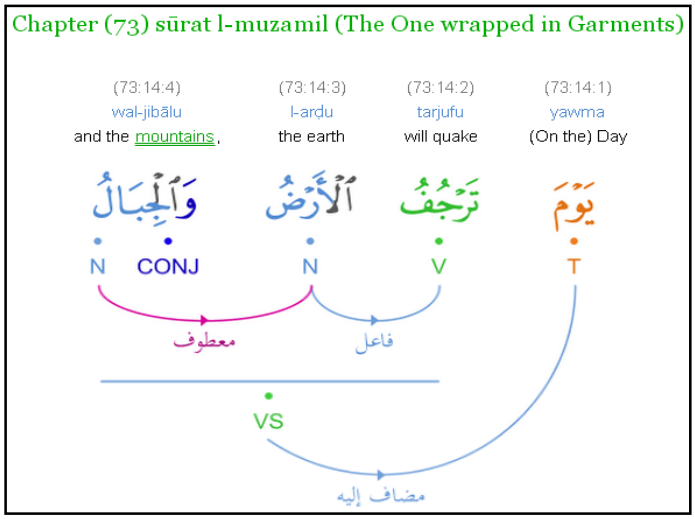


Figure 3.6: Représentation d'un verset en structure de dépendances dans le QADT

Dans l'exemple de représentation donné par la figure 3.6 ci-dessus, l'hybridation est montrée à travers le regroupement des trois derniers mots dans un segment étiqueté comme syntagme verbal et la spécification d'une relation de dépendance (Idafa) entre le premier mot de la phrase et ce segment.

6.4 Etiquettes POS

Les quatre treebanks utilisent le même schéma de segmentation, mais ils se distinguent au niveau de l'annotation en parties de discours (Part-Of-Speech pour POS) :

L'ATB utilise plus que 400 étiquettes spécifiant les différents attributs morphologiques des mots arabes comme la définition, le genre, le nombre, la personne, le mode, la voix, la déclinaison (Maamouri et Bies, 2004).

Le PADT a une morphologie plus complexe que l'ATB, même n'annote pas les pronoms vides. Par exemple, il intègre plus de distinctions au niveau de la définition, du genre et nombre des noms et adjectifs (Hajič et al., 2001).

Le CATiB utilise uniquement 6 étiquettes POS qui ne comprennent pas les pronoms vides (Habash et al., 2009).

Le QADT utilise les catégories lexicales de la grammaire traditionnelle arabe qui se base sur trois étiquettes principales qui sont le nom, le verbe et la particule. Ce treebank utilise 44 étiquettes POS pour les annotations morphologiques parmi lesquelles il annote les pronoms vides (Dukes et al., 2010).

6.5 Relations syntaxiques et sémantiques

L'ATB utilise environ 20 relations syntaxiques et sémantiques (dashtags) pour représenter les fonctions syntaxiques et sémantiques. Les Dashtags syntaxiques comprennent -PTC et -OBJ et les étiquettes sémantiques comprennent -TMP (temps) et-LOC (lieu). Certaines dashtags jouent un double rôle (sémantique et syntaxique) comme - SBJ indiquant ou bien un sujet syntaxique d'un verbe ou bien un sujet sémantique d'un nom déverbal. Les constructions Tamyiz et Idafa sont identiques dans l'ATB (Maamouri et Bies, 2004).

Le CATiB marque uniquement les fonctions syntaxiques, et l'utilisation des étiquettes syntaxiques SBJ et PTC se diffère entre le CATiB et l'ATB (Habash et al. 2009).

Le PADT utilise environ 20 étiquettes, mais avec des fonctionnalités différentes de l'ATB et du CATiB. En général, celles du PADT sont plus détaillées que le CATiB. Le PADT ne distingue pas les différents types de modificateurs nominaux, c'est à dire les adjectifs, Idafa et Tamyiz (en nombre) sont tous marqués comme « Atr » (Attribut) (Habash, 2010).

Le treebank QADT utilise les liens de dépendances syntaxiques de la grammaire traditionnelle arabe. Ils sont environ en nombre de 43 relations syntaxiques et sémantiques (Dukes et al., 2010).

6.6 Vitesse d'annotation (POS et Syntaxique)

Vu que le CATiB utilise une annotation morphosyntaxique simple, il a pu annoter 540 segments par heure, à la différence du PADT qui a une morphologie plus complexe que celle du PADT ce qui multiplie le temps d'annotation des segments. En effet, il ne peut annoter que 75 segments par heure. Toutefois, l'ATB reste plus efficace puisqu'il peut annoter son corpus source avec plus que 400 étiquettes avec une vitesse d'environ 300 segments par heure (Habash et Roth, 2009).

Malgré ces différences, il est possible de faire des conversions entre ces différentes représentations, vu que l'information linguistique est accessible à partir de l'arbre qui est représenté différemment. Le tableau suivant synthétise notre étude comparative en illustrant les différentes caractéritiques des quatre treebanls :

Treebanks	**PATB**	**PADT**	**CATiB**	**QADT**
Distribution / License	Linguistic Data Consortium	Linguistic Data Consortium	Pas disponible gratuitement (CCLS at Columbia University)	Open source (GNU general public license
Source	Articles de différentes agences de presse (France Presse, Xinhua, Al-Hayat, Ummah Press,…)	PATB1+PATB2+ PATB3+PATB4 + Autres (Xinhua, Al-Hayat, An-Nahar)	PATB1+PATB2+PATB3 + Autres (France Presse, Xinhua, Al-Hayat, Al-Asharq Al-Awsat, Al-Quds Al-Arabi, An-Nahar, Al-Ahram et As-Sabah)	Le Coran (114 chapitres ; 6236 versets ; 77,430 mots ; 18,994 mots uniques ; 12,183 stemms)
Taille de la source annotée	ATB1 - ATB12 - … (≈750K segments, le plus utilisé ATB3 : 599 documents, ≈350K segments)	>1,2M segments (≈1,1M mots, >1M arbres) [≈900K segments de PATB convertis vers PADT]	>1M segments (841K mots, 31,319 arbres) [735K segments de PATB convertis vers CATiB]	11K mots, (2,500 graphes de dépendances syntaxiques)
Grammaire suivie	Arabe standard moderne	Arabe standard moderne	Arabe traditionnel	Arabe traditionnel
Représentation Syntaxique	Structure de syntagmes	Structure de dépendances (grammaire FGD : Functional Generative Description)	Structure de dépendances (grammaire simplifiée)	Structure de dépendances
Etiquettes POS	>400 étiquettes (≈de PTB)	>>400 étiquettes	6 étiquettes	44 étiquettes
Relations	>20 relations syntaxiques et sémantiques (dashtags)	20 relations syntaxiques et sémantiques	8 liens de dépendances syntaxiques	43 liens de dépendances syntaxiques
Vitesse d'annotation	250-300 segments/heure	75 segments/heure	540 segments/heure	

Tableau 3.1: Tableau comparatif des treebanks arabes

Ce que nous pouvons constater après cette étude comparative, c'est le fait le treebank ATB répond le plus aux besoins de notre problématique. En effet, seul l'ATB représente ses annotations selon la structure de constituants. Cette structure s'adapte au comportement des grammaires de propriétés qui représentent leurs catégories grammaticales selon une structure syntaxique arborescente à base de syntagmes et non à base de dépendances (comme dans les trois autres treebanks). En plus de sa structure syntaxique, plusieurs autres aspects favorisent le choix d'utilisation de l'ATB comme source d'induction, à savoir : la richesse de ses annotations de POS et de relations syntaxiques et sémantiques, sa grammaire adaptée à l'arabe standard moderne et la pertinence syntaxique de ses documents sources (vu qu'ils sont convertis par plusieurs autres treebanks à leur représentation). Pour cela, nous nous focalisons dans ce qui suit, sur l'étude des différentes particularités du treebank ATB.

7 Enrichissement des treebanks

La direction vers l'axe de recherche d'enrichissement de treebanks est due à l'insuffisance de ces ressources linguistiques au niveau de la représentation des annotations outre le manque de traitement des phénomènes linguistiques particuliers complexes comme les relatives, les anaphores et les coordinations. L'amélioration de ces ressources permet d'obtenir de nouvelles ressources à diverses représentations plus riches tout en héritant les qualités linguistiques (robustesse, fiabilité, richesse) qui caractérisent les treebanks eux-mêmes.

Dans cet axe de recherche, il existe plusieurs travaux d'enrichissement pour différentes langues comme l'anglais, l'allemand, le danois, l'italien, l'espagnol et le turque (Oepen et al., 2002 ; Hinrichs et al., 2004 ; Çakıcı, 2005 ; Hockenmaier, 2006 ; Müller, 2010 ; Yu et al., 2010; Rauzy et Blache, 2012). Pour l'arabe, la langue qui nous intéresse le plus, il existe quelques travaux permettant l'enrichissement de l'ATB. Ces travaux portent sur l'amélioration de ce treebank par de nouvelles annotations plus riches ou bien sur sa conversion à de nouveaux formalismes par rapport à son formalisme source.

Le projet OntoNotes (Hovy et al., 2006) et le projet Proposition Bank (Propbank) pour l'arabe (Palmer et al., 2008) représentent des exemples d'extensions de treebank intégrant des annotations de niveau sémantique. La contribution d'Alkuhlani et Habash (2011) propose quant à elle l'ajout d'annotations modélisant les attributs genre et nombre fonctionnels ainsi que la rationalité. L'enrichissement a même touché le niveau sentimental en associant aux phrases de l'ATB des annotations spécifiques, et ce dans le cadre du travail réalisé par Abdul-Mageed et Diab (Abdul-Mageed et Diab, 2012). Il y a aussi le trqvail proposé par Alkuhlani et al. (2013),

mais qui enrichie le Columbia Arabic Treebank (CATiB) (Habash et al., 2009) avec les étiquettes POS et lemmes les plus complexes utilisées dans l'ATB (Maamouri et al., 2004).

Pour l'enrichissement avec l'application de nouveaux formalismes au treebank source, nous pouvons citer les exemples suivants : la contribution de Habash et Rambow (2004) permettant l'extraction d'un treebank de grammaire d'arbres adjoints (TAG), le travail de Tounsi et al. (2009) qui a généré automatiquement un treebank avec une représentation de grammaire lexicale fonctionnelle (LFG) ainsi que les deux efforts de conversion de l'ATB en des CCGbank arabes (El-taher et al., 2014).

Concernant le formalisme de GP que nous allons adopter pour l'enrichissement, il a été précédemment hybridé avec le treebank français FTB (Blache et Rauzy, 2012). La GP construite pour cette hybridation a été induite automatiquement à partir du treebank FTB. La même technique d'induction a été adoptée pour la langue chinoise dans (Blache, 2014) à partir du treebank chinois CTB pour un éventuel enrichissement.

8 Conclusion

Dans le cadre de ce chapitre, nous avons fait un aperçu sur les treebanks en spécifiant les différents aspects qui peuvent les caractériser. Après la définition précise de la notion de « treebank », nous avons décrit les différentes caractéristiques ainsi que les schémas d'annotations nécessaires à la conception de treebanks. Nous avons, également, présenté leurs méthodes et outils de développement ainsi que leurs domaines d'utilisation, en particulier le traitement du langage naturel. Le but d'exploiter ces ressources linguistiques aussi pour la langue arabe nous a mené à détailler leur étude dans ce chapitre. Dans ces trois premiers chapitres, nous avons effectué un aperçu sur les travaux existants liés à la problématique de notre thèse. En bénéficiant de cet aperçu, nous avons observé l'intérêt particulier du formalisme de grammaires de propriétés en tant qu'approche basée effectivement sur les contraintes en l'appliquant pour l'analyse. Utiliser un tel formalisme pour le traitement de la langue arabe nous parait une idée innovante et réalisable dans la mesure où elle n'a été jamais utilisée pour cette langue. Plusieurs ressources linguistiques arabes peuvent être construites à l'aide de ce formalisme : une grammaire de propriétés arabe, un treebank enrichi de propriétés, un résultat d'analyse enrichi de propriétés et un analyseur syntaxique de propriétés. Le chapitre suivant s'occupe de détailler le processus de construction de la première ressource : la grammaire de propriétés arabe en utilisant un treebank arabe : l'ATB.

Deuxième partie

Démarches proposées

Chapitre 4

Induction d’une grammaire de propriétés à partir de l’ATB

1 Introduction

Le formalisme de grammaires de propriétés se caractérise par sa robustesse et sa généralité. Il permet de représenter des contraintes, dites propriétés linguistiques, à différents niveaux d'analyse d'une manière facile, directe et indépendante. Ces propriétés peuvent être modifiées ou même supprimées à tout moment sans affecter la cohérence de l'ensemble. Il est possible également de fournir des descriptions syntaxiques pour tout type d'énoncé et à n'importe quelle position dans la phrase. Dans le but de bénéficier des qualités de ce formalisme pour la langue arabe, nous avons construit plusieurs ressources linguistiques qui l'utilisent.

La description de ces ressources occupe les trois chapitres encours. Plus particulièrement, le présent chapitre va fournir la démarche adoptée pour l'induction d'une grammaire de propriétés arabe à granularité variable à partir du treebank arabe ATB. La description de cette démarche est précédée par la justification du choix du treebank ATB en tant que source d'induction ainsi que l'étude détaillée de ce treebank pour faciliter son exploitation.

2 Raisons de choix de l'ATB

Tout d'abord, le choix de l'alternative d'induction d'une GP à partir d'un treebank revient au fait que le traitement de la langue arabe présente plusieurs défis. Ces défis ne sont pas seulement liés à certaines spécificités de l'arabe à étudier (comme l'absence des voyelles, la nature agglutinative des mots), mais aussi à des phénomènes linguistiques particuliers à traiter (comme les relatives, les anaphores et les coordinations). L'idée qui vient tout d'abord à l'esprit consiste à la construire cette GP manuellement. L'implémentation de cette directive par l'utilisation d'un corpus regroupant toutes les règles de la grammaire arabe est certainement difficile et coûteuse. Ceci requiert en effet beaucoup de temps ainsi que la collaboration de plusieurs linguistes pour développer et valider les annotations générées en termes de propriétés. Nous avons alors proposé la directive suivante : construire la GP à partir d'un corpus annoté. Les treebanks qui sont des corpus annotés manuellement (ou annotés automatiquement et révisés manuellement) formant une structure morphosyntaxique à plusieurs niveaux d'analyse (niveau mot, niveau syntagme et niveau phrase) peuvent être exploités dans ce cadre.

La sélection du treebank ATB (Maamouri et al., 2004) parmi les treebanks arabes qui sont déjà rares revient à ses qualités qui s'avèrent convenables à la construction de notre GP. Nous avons déjà décrit ces qualités dans notre contribution (Bensalem et Elkaroui, 2014) tout en présentant une étude comparative du treebank ATB avec d'autres fameux treebanks arabes. En

effet, la première qualité caractérisant ce treebank est sa représentation à base constituants, conforme à la structure syntaxique hiérarchisée de la GP à construire. Ce choix est motivé également par la richesse, la fiabilité et la compatibilité des annotations de Part-of-Speech (POS) et de relations syntaxiques et sémantiques de l'ATB. Ces annotations sont en fait élaborées et validées par des linguistes. De même, la grammaire de ce treebank est adaptée à l'arabe standard moderne (Modern Standard Arabic, MSA). Sans oublier la pertinence, la variété et la grande taille qui caractérisent ses documents sources. Cette pertinence revient à la conversion de ces documents par plusieurs autres treebanks à leur représentation comme nous l'avons déjà mentionné dans la section 6.1 du chapitre précédent. Le fait de disposer d'une ressource de ce type permet de générer automatiquement et de façon très contrôlée de nouvelles ressources dans d'autres formalismes. Des ressources à large couverture sont ainsi obtenues, héritant des qualités du treebank d'origine, étant l'ATB, tout en gagnant en temps de construction.

3 Etude de l'ATB

Avant d'expliquer les spécificités liées à l'ATB, nous présentons brièvement cette ressource linguistique. L'ATB a été lancé en 2001 dans le cadre d'un projet au LDC[13] (Maamouri et Bies, 2004). C'est un corpus composé de 23,611 phrases extraites d'articles de presse annotées manuellement. Il a été divisé en ensembles de textes (divisions) pour répondre aux besoins de recherche variés dans le domaine de TAL comme l'apprentissage et l'évaluation (Diab et al., 2013).

Doté d'une annotation très riche, l'ATB se caractérise par un ensemble de qualités servant à une meilleure induction de GP. En effet, ses annotations présentent l'avantage d'être fiables. Ceci est prouvé par son efficacité dans un grand nombre de travaux dans différents domaines de traitement de langage naturel (Habash, 2010). Son texte source a prouvé également sa pertinence dans la création d'autres treebanks arabes comme le PADT (Hajič et al., 2001) et le CATiB (Habash et Roth, 2009) qui ont converti l'ATB vers leurs représentations syntaxiques en plus d'autres textes qu'ils ont annotés. De plus, l'ATB est disponible en cinq formats différents (voir la sous-section 3.1). Une autre particularité peut être remarquée est celle de la granularité forte qui caractérise son annotation (voir la sous-section 3.2). Finalement, l'ATB a prouvé son aptitude à représenter correctement certains phénomènes particuliers de la langue

[13] LDC (Linguistic Data Consotium) : Consortium de données linguistiques https://www.ldc.upenn.edu/

arabe (voir la sous-section 3.3). Notre deuxième contribution (Bensalem et al., 2014) explique en détail les différentes caractéristiques de l'ATB.

3.1 Formats de représentation des données dans l'ATB

L'ATB est fourni sous différents formats pour étendre son utilisation pour différents besoins de recherche. Ces formats que nous citons sont en nombre de cinq. Le format *« sgm »* représente les documents sources. Par contre, le format *« pos »* affiche des informations (comme la translitération, la voyéllation et la traduction) décrivant chaque mot source sous forme de champs avant et après la séparation des agglutinations. Le format *« xml »* quant à lui affiche les annotations de l'arbre de mots sources après la séparation des agglutinations. Alors que le format PTB, il représente le corpus en deux versions (voyellée ou non) sous forme d'arborescence affichant chaque mot dans sa structure hiérarchique et devant son étiquette POS. Finalement, le format *« integrated »* affiche des informations aussi bien sur la structure arborescente que sur chaque mot source avant et après la séparation des agglutinations. Après la présentation de ces différents formats, il faut prendre une décision concernant le choix du format de l'ATB à utiliser pour l'induction de la GP.

3.2 Niveaux de granularité des catégories

L'annotation dans l'ATB est caractérisée par une granularité forte. En effet, cette annotation inclut plus que 400 étiquettes POS différentes offrant des informations morphosyntaxiques comme le cas, le mode, le genre et la définition (Maamouri et al., 2009). Parmi ces étiquettes, 22 sont syntagmatiques (des catégories syntaxiques), 20 sont des relations syntaxiques et sémantiques et 24 représentent les étiquettes POS de base. L'ATB prend compte également des pronoms vides qui peuvent apparaitre dans les phrases arabes tout en leur affectant une étiquette spécifique. Cette annotation a été toujours améliorée pour résoudre les incohérences morphosyntaxiques liées à certaines spécificités de la langue arabe (Maamouri et al., 2008 ; Kulick et al., 2010).

<table>
<tr><td colspan="2">Nature
Traits</td><td colspan="4">Nom ou Adjectif</td></tr>
<tr><td colspan="2">Type</td><td colspan="4">Nom : NUM, PROP, QUANT, VN
Adjectif : COMP, NUM, VN</td></tr>
<tr><td colspan="2">Cas syntaxique</td><td colspan="4">NOM, ACC, GEN</td></tr>
<tr><td colspan="2">Définition</td><td colspan="4">DEF, INDEF</td></tr>
<tr><td colspan="2">Déterminant</td><td colspan="4">DET</td></tr>
<tr><td colspan="2">Possédé</td><td colspan="4">POSS</td></tr>
<tr><td rowspan="2">Accord</td><td>Genre</td><td colspan="4">MASC, FEM</td></tr>
<tr><td>Nombre</td><td colspan="4">SG DU PL</td></tr>
<tr><td colspan="2" rowspan="2">Exemples</td><td>Catégorie</td><td>Mot arabe</td><td>Mot translitéré</td><td>Mot traduit</td></tr>
<tr><td>NOUN_NUM+NSUFF_FEM_SG+CASE_INDEF_NOM
DET+ADJ_COMP+CASE_DEF_ACC</td><td>أربعة
الأكثر</td><td>arobaE+ap+N
Al+>akovar+a</td><td>Quatre
Le plus</td></tr>
<tr><td colspan="2">Nature
Traits</td><td colspan="4">Verbe</td></tr>
<tr><td colspan="2">Temps</td><td colspan="4">I, C, P</td></tr>
<tr><td colspan="2">Mode</td><td colspan="4">I, JUS, SJ</td></tr>
<tr><td colspan="2">Forme</td><td colspan="4">PASS</td></tr>
<tr><td rowspan="3">Accord suffixe</td><td>Genre</td><td colspan="4">M, F</td></tr>
<tr><td>Nombre</td><td colspan="4">S, D, P</td></tr>
<tr><td>Personne</td><td colspan="4">1, 2, 3</td></tr>
<tr><td colspan="2" rowspan="2">Exemples</td><td>Catégorie</td><td>Mot arabe</td><td>Mot translitéré</td><td>Mot traduit</td></tr>
<tr><td>PV+PVSUFF_SUBJ:3MS
IV1P+IV_PASS+IVSUFF_MOOD:I</td><td>كتب
نستطيع</td><td>katab+a
na+sotaTiyE+u</td><td>a écrit
nous pouvons</td></tr>
</table>

Figure 4.1: Mots-clés indiquant les traits caractérisant des catégories lexicales de l'ATB

Les figures 4.1 et 4.2 illustrent respectivement les mots clés indiquant les différents traits caractérisant la plupart des catégories lexicales (noms, adjectifs et verbes) et syntaxiques décrites dans l'ATB (Maamouri et al., 2009).

Traits	Mots-clés
Type	SBJ, PRP, PRD, OBJ, MNR, LOC, HLN, DTV, DIR, ADV, TMP, TPC
Numérotation	1, 2, …, 25
Exemples	(**NP-ADV-5** (NOUN+CASE_DEF_ACC <izA'+a) (NP (NP (DET+NOUN+NSUFF_FEM_PL+CASE_DEF_ACC Al+Eamaliy~+At+i)) (SBAR (WHNP-4 (REL_PRON Al~atiy)) (S (VP (IV3FS+IV+IVSUFF_MOOD:Itu+naf~i*+u-) (NP-SBJ-4 (-NONE- *T*)) (NP-OBJ (IVSUFF_DO:3FS -hA)))))) إزاء العمليات التي تنفذها Envers les opérations qu'elle exécute

Figure 4.2: Mots-clés indiquant les traits caractérisant des catégories syntaxiques de l'ATB

Maintenant, si nous diminuons cette forte granularité, plusieurs sous-ensembles de catégories seront factorisés en une seule catégorie. Prenons comme exemple, le sous-ensemble {NOUN_PROP, NOUN_PROP+CASE_DEF_ACC, NOUN_PROP+CASE_DEF_NOM} qui marque les noms propres par trois étiquettes dans le cas d'une forte granularité. Si nous avons une faible granularité, ces étiquettes sont généralisées, et factorisées en une seule étiquette sous le nom NOUN_PROP. Ceci s'explique par le fait que le manque de précision dans les catégories grammaticales dû à la diminution du niveau de granularité permet de les factoriser.

3.3 Représentation de certains phénomènes particuliers de la langue arabe

Comme nous l'avons déjà mentionné dans l'introduction de cet article, la langue arabe présente plusieurs défis lors de son traitement automatique. Parmi ces défis, nous citons les phénomènes linguistiques particuliers comme les relatives, les coordinations et les anaphores. L'ATB a aussi contribué à traiter ces phénomènes en les représentant conformément à la grammaire arabe (Maamouri et al., 2009), ce qui favorise la robustesse des ressources qui peuvent l'exploiter. Dans le cas des propositions relatives, il est bien remarquable, comme le montre la figure 4.3, que la relative SBAR « التي لم تحترق» (Al~atiy lam taHotariq+o/ *qui ne sont pas brulées*) est réellement jointe au syntagme nominal « المواد الهيدروكربونية» (Al+mawAd~+i Al+hiydoruwkarobuwniy~ap+i/ *les matières hydro-carboniques*) qui la modifie.

```
(NP (NP Al+mawAd~+i:: المَوادِّ :: the+substances/materials+[def.gen.]
        Alhydrwkrbwnyp:: الهيدروكربونية :: nogloss)
    (SBAR (WHNP-2 Al~atiy:: الَّتِي ::which/who/whom_[fem.sg.] )
        (S (VP (PRT lam:: لَم :: did_not )
            ta+Hotariq+o:: تَحْتَرِقْ :: it/they/she+burn_up/be_burned+[jus.]
            (NP-SBJ-2 *T*)))))
```

Figure 4.3: Un exemple de proposition relative représenté par l'ATB (Maamouri et al., 2009)

Les coordinations, quant à elles, sont composées en arabe généralement de deux conjoints et de la conjonction qui les réunit. Plusieurs formes de coordinations sont représentées dans l'ATB selon la structure des trois composants indiqués. Concernant les anaphores, l'ATB indique uniquement ceux des catégories vides et des cas exceptionnels comme les structures écartant les syntagmes verbaux (Maamouri et al., 2009).

3.4 Améliorations et corrections subies

Dès sa création, l'ATB a connu des enrichissements continus au cours des différentes années qui suivent. Ces enrichissements consistent à intégrer différentes améliorations et corrections pour pouvoir surmonter les défis liés à certaines spécificités de la langue arabe. En effet, cette langue est caractérisée par sa morphologie complexe. Ce problème a été examiné au niveau de l'ATB par (Kulick et al., 2010) en représentant les mots ayant une forme agglutinative par des unités séparées dans une structure arborescente. Par exemple, le mot arabe « كتبه» (ktbh/ ses livres), s'il n'est pas voyellé, l'ATB le représente en deux parties tels que « ktb » (livres) est un groupe nominal et « h » (ses) est un pronom possessif. Pour réaliser cette tâche, les auteurs ont utilisé l'outil SAMA (Standard Morphological Analyzer) pour générer des solutions d'analyse morphologique pour chaque mot de l'ATB. De même, l'absence de voyelles dans la langue arabe peut générer des ambigüités. Pour surmonter cette difficulté, les auteurs de (Kulick et al., 2010) ont intégré une représentation syntaxique abstraite de la structure arborescente tout en autorisant le passage entre les différents niveaux de représentation syntaxique et tout en fournissant différents niveaux de voyellation pour chaque mot de l'ATB. Concrètement, la procédure d'annotation morphosyntaxique que les auteurs ont suivie est basée sur deux grandes étapes : la première consiste à une annotation syntaxique décomposant le texte de l'ATB en mots (appelés jetons sources). Ces mots sont intégrés dans l'outil SAMA, puis générés sous forme voyellée. La deuxième étape, quant à elle consiste à séparer ces jetons des pronoms liés durant l'annotation syntaxique. Par exemple, l'analyse du mot arabe « كتبه» par l'outil SAMA génère une solution qui le décompose en trois segments. Cette solution inclut une séquence d'informations pour chaque segment portant sur trois champs : la forme voyellée, l'étiquette Part-Of-Speech (POS) et la traduction du segment. La solution SAMA de ce mot est présentée comme suit :

[kutub, NOUN, books]	[i, CASE_DEF_GEN, def.gen]	[hi, POSS_PRON_3MS, its/his]

Les auteurs de (Kulick et al., 2010) ont cherché également des procédures spécifiques pour traiter les mots arabes ayant un caractère particulier, tels que les mots ayant une forme

agglutinative ne pouvant pas être explicitement décomposée. Le mot عما(EmA/de ce que) par exemple est une préposition suivie d'un pronom relatif. La solution proposée par SAMA est composée de deux segments. Elle inclut un « n » dans « Ean », n'ayant pas été présent dans le mot source « EmA ».

[Ean, PREP, from/about/of]	[mA, REL_PRON, what]

Par ailleurs, l'application d'une analyse statistique apprise sur le treebank arabe (ATB) et examinant des incohérences dans les annotations a généré des scores d'analyse inférieures aux prévisions. Ces incohérences résident au niveau de certaines constructions syntaxiques ou de la relation entre les étiquettes POS et les annotations syntaxiques. La résolution de ces incohérences va corriger largement les directives d'annotation. Le travail (Maamouri et al., 2008) s'inscrit dans ce cadre. En effet, il présente des corrections et des améliorations des incohérences d'annotation dans le but d'améliorer la qualité d'analyse du corpus de l'ATB. Ce travail utilise les étiquettes POS pour corriger et améliorer les directives d'annotation syntaxique. Ces corrections sont proposées aussi bien au niveau morphologique qu'au niveau syntaxique. Au niveau morphologique, les auteurs de (Maamouri et al., 2008) ont proposé de raffiner les étiquettes POS des noms et des adjectifs pour spécifier les noms quantifieurs (NOUN_QUANT), les nombres (NOUN_NUM), les adjectifs comparatifs (ADJ_COMP), les nombres ordinaux (ADJ_NUM), etc. De même, ils ont distingué des catégories d'étiquettes POS pour les différentes particules, telles que l'étiquette CONJ qui a été décomposée en quatre catégories. Et pour distinguer les pseudo-verbes des verbes, ils ont ajouté l'étiquette PSEUDOVERB pour les sœurs de la particule arabe « إنّ» (inna / que). Au niveau syntaxique, les auteurs de (Maamouri et al., 2008) se sont focalisés sur la désignation du nom par une étiquette spécifique s'il est un quantifieur dans le syntagme « idafa » pour spécifier correctement la tête sémantique du syntagme. En effet, pour le syntagme « كل مجموعة» (chaque collection), la tête sémantique n'est pas le segment « chaque » mais plutôt le segment « collection » parce que celle-là est un nom quantifieur et non pas un simple nom. Il faut alors le spécifier par l'étiquette NOUN_QUANT. Les auteurs de (Maamouri et al., 2008) ont marqué aussi les gérondifs et les participes, si ceux-ci présentent une lecture verbale, par des étiquettes regroupant toute l'unité syntaxique. Par exemple, la phrase « احتفل الفريق بفوزه بكأس الأبطال» (L'équipe a célébré son gain de la coupe des champions), ne contient pas un gérondif suivi d'un complément (« فوزه بكأس الأبطال») ce qui le caractérise par une lecture verbale plutôt qu'un simple gérondif. Cette lecture verbale est entièrement analysée.

L'ATB dans sa nouvelle forme, après les améliorations et les corrections effectuées permettant d'aligner étroitement son annotation aux catégories de la grammaire arabe traditionnelle, représente une source assez robuste offrant plusieurs spécificités servant à la réalisation d'une induction de GP réussite. L'explication du déroulement de la démarche d'induction de la GP, que nous proposons, fera l'objet de la section suivante de chapitre.

4 Induction de la GP à partir de l'ATB

Le déroulement de la démarche d'induction de la GP à partir du treebank ATB se base sur deux principales phases : l'induction d'une grammaire hors-contexte (CFG) à partir de l'ATB et l'induction de la GP partir de cette CFG. Cette démarche a été supportée par notre contribution (Bensalem et al., 2014). L'établissement de cette démarche ne peut pas être fait directement sans passer par une étape de modélisation des outils à utiliser en entrée. Dans notre cas, ceci revient à générer des formalisations spécifiques au treebank ATB, à la CFG et à la grammaire GP.

Nous commençons par présenter la description de la CFG pour une langue donnée. Cette grammaire est composée d'un ensemble de règles de production (constructions) utilisées pour générer des structures de mots. Formellement, elle est définie par le 4-uplet G= (N, Σ, P, S) où : N est un ensemble fini de symboles non terminaux, Σ est un ensemble fini de symboles terminaux, P est un ensemble fini de règles de la forme $\alpha \rightarrow \beta$ tels que $\alpha \in N$ est $\beta \in (N \cup \Sigma)^*$, et $S \in N$ est le symbole de départ. Le langage formel de G est alors défini comme $L(G) = \{w \in \Sigma^* \mid S \vdash^* w\}$. Pour chaque dérivation de S, w correspond à in arbre t_w. Dans les langues naturelles, w correspond à un phrase Sent, qui est associée à un arbre t_{Sent} s'accordant avec la grammaire G.

L'ATB de son côté est un corpus de phrases d'une langue naturelle annotées manuellement. Il peut être vu comme une suite de paires (Sent, t_{Sent}). Il est alors défini par $TB=\{(Sent, t_{Sent}) \mid S \vdash^* Sent\}$ où Sent est une suite de mots arabes donnant un sens complet. Par conséquence, une phrase $Sent \in M^*$ tel que M est l'ensemble des mots du treebank. Tant que $t_{Sent} \in \mathcal{A}$ où $\mathcal{A}$ est l'ensemble des arbres obtenus de l'analyse de chaque phrase Sent du treebank according to the grammar G. Vu que les annotations fournies par l'ATB sont étendues à plusieurs niveaux d'analyse (niveau mot, niveau syntagme, niveau phrase), la définition de l'ATB s'améliore à être un 7-uplet $TB = (M, \Psi, P, \mathcal{T}, \Omega, \mathcal{S}, \mathcal{A})$ où :

- M est un ensemble des mots du treebank.
- $\Psi = \psi_1 \times \psi_2 \times \ldots \times \psi_n$ est un n-uplet d'ensembles ψ_i de types d'informations (morphologiques, syntaxiques et sémantiques) pour les mots du corpus. La forme de Ψ est $(c_1, c_2, .., c_n)$ tel que c_i est une information d'un type défini i de n types d'informations (e.g. catégorie lexicale, translitération, traduction, …).
- P est l'ensemble de syntagmes (regroupant les mots successifs du corpus pour avoir un sens élémentaire) tels que $p \in P$ est de la forme $p \in M^*$.
- $\mathcal{T}$ est l'ensemble d'arbres élémentaires t_p obtenus de l'analyse des syntagmes $p \in P$.
- $\Omega = \omega_1 \times \omega_2 \times \ldots \times \omega_z$ est un n-uplet d'ensembles ω_i de types d'informations pour les syntagmes du corpus. La forme de Ω est $(d_1, d_2, .., d_z)$ tel que d_j est une information d'un type défini j de z types d'informations (e.g. catégorie syntaxique, relation syntaxique,…).
- $\mathcal{S}$ est l'ensemble des phrases de la forme $Sent \in M^*$.
- $\mathcal{A}$ est l'ensemble d'arbres complets obtenus de l'analyse des phrases $Sent \in \mathcal{S}$.

Dans notre cas, qui consiste à induire une CFG à partir de l'ATB, nous avons besoin uniquement de trois types d'informations pour tous les mots et syntagmes dans l'ATB. Pour chaque mot $m_i \in M$, nous avons besoin de sa catégorie lexicale $c_1(m_i) \in \psi 1$, l'arbre $t_p(m_i)$ auquel il appartient ($t_p(m_i) \in \psi 2 = \mathcal{T}$) et son ordre i par rapport aux nœuds voisins ($i \in \psi 3 = O$). Egalement, pour chaque syntagme $p_j \in P$, nous avons besoin de sa catégorie syntaxique $d_1(p_j) \in \omega 1$, l'arbre $t_p(p_j)$ auquel il appartient ($t_p(p_j) \in \omega 2 = \mathcal{T}$) et son ordre j par rapport aux nœuds voisins ($j \in \omega 3 = O$).

La GP, quant à elle, est une grammaire qui définit un ensemble de relations entre des catégories grammaticales, non pas en terme de règle de production (comme dans la CFG), mais en terme de contraintes locales (appelées propriétés). Les propriétés syntaxiques, plus particulièrement, décrivent des phénomènes linguistiques comme la précédence linéaire ($\prec$), la cooccurrence obligatoire ($\Rightarrow$), la restriction de cooccurrence ($\otimes$) , l'obligation (oblig), l'unicité (unic) et l'adjacence ($\pm$). Formellement, cette grammaire peut être définie par un 3-uplet G'= (N, Σ, R). N est un ensemble fini de catégories syntaxiques. Σ est un ensemble fini de catégories lexicales. R est un ensemble fini de propriétés syntaxiques décrivant $\forall \alpha \in N$ et dans α reliant $\forall \beta_1$ à $\beta_2 \in (N \cup \Sigma)$ selon l'une des 6 manières suivantes : $\alpha : \beta_1 \prec \beta_2$, $\alpha: \beta_1 \pm \beta_2$, $\beta_1 \in unic(\alpha)$, $\beta_1 \in oblig(\alpha)$, $\alpha: \beta_1 \Rightarrow \beta_2$, $\alpha: \beta_1 \otimes \beta_2$. Chacune de ces propriétés peut être déduite de l'ensemble P défini dans la grammaire G.

Il est maintenant possible de représenter en détail la phase d'induction de la GP à partir de l'ATB. Il faut mentionner qu'elle est basée sur l'idée d'induction de GP à partir du FTB adoptée dans (Blache et Rauzy, 2012). Cette phase s'articule autour de deux grandes étapes : l'induction de la CFG et l'induction de la GP. La phase proposée est présentée dans la figure 4.4 suivante :

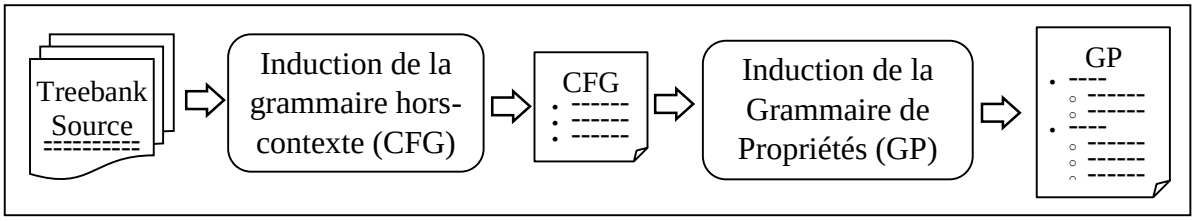


Figure 4.4 : Phase d'induction de la GP

Les deux sous-sections suivantes ont pour objectif de décrire les deux grandes étapes illustrées par la figure 4.4.

4.1 Induction de la CFG à partir de l'ATB

L'induction de notre GP ne peut pas être réalisée directement en appliquant une simple tâche d'acquisition et de manipulation des données du treebank. Il faut en effet introduire une étape intermédiaire permettant de représenter les productions décrivant les unités syntaxiques. Cette étape consiste à parcourir l'ATB et à en extraire les constructions (règles) possibles pour produire une CFG pour l'arabe à différents niveaux de granularité. Ceci est réalisé dans le cadre de trois sous-étapes primordiales : le contrôle de ses niveaux de granularité des constituants de l'ATB, la détection des constituants et l'élaboration des règles comme illustré la figure 4.5.

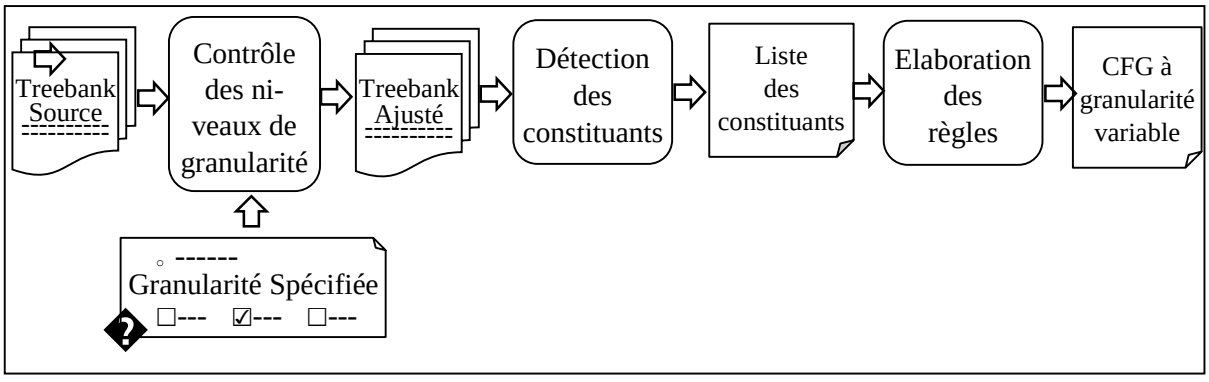


Figure 4.5 : Sous-étapes d'induction de la CFG

Nous parlons de la première sous-étape dans la dernière sous-section pour montrer son effet sur les sorties des sous-sous-étapes qui la suivent.

4.1.1 Détection des constituants

Cette sous-étape permet tout d'abord de parcourir le treebank ligne par ligne en décomposant chaque ligne en un ensemble de mots. Les mots commençant par une parenthèse ouvrante représentent les constituants. Après l'acquisition de l'ensemble des constituants, elle élimine les doublons puis trie ces constituants par ordre alphabétique. Finalement, afin de présenter quelques statistiques, cette sous-étape calcule le nombre d'occurrences de chacun de ces constituants.

4.1.2 Elaboration des règles

Il s'agit d'extraire les différentes constructions possibles (règles) pouvant être représentées dans l'ATB, ce qui permet de produire implicitement la CFG. Ceci se fait en parcourant ligne par ligne les fichiers des données source du treebank. Pour chaque ligne, trois tâches sont réalisées successivement à savoir : la récupération de l'ensemble des mots formant cette ligne, la détection des extrémités des règles en utilisant les mots récupérés et la génération des règles. La deuxième tâche permet d'affecter des positions incrémentales aux mots récupérés tout en donnant aux bornes (les parenthèses ouvrante et fermante) de chaque règle détectée la même position. A la troisième tâche, il s'agit de former les règles à générer. Chaque règle est composée d'une partie gauche consacrée à l'un des constituants du niveau syntaxique, et d'une partie droite représentant les constituants descendants de ce constituant syntaxique. Cette tâche s'étend également à l'exploration des constituants descendants dans la partie droite à la recherche d'autres règles de niveaux hiérarchiques plus fins. Après chaque parcours d'un fichier courant, les règles extraites sont accumulées à la liste des règles extraites des fichiers déjà parcourus. Cette liste est filtrée et triée. La liste finale des règles forme notre CFG. Il est possible également de calculer le nombre d'occurrences de chaque règle dans la liste pour présenter quelques statistiques.

En se trouvant devant une complexité prévue de la forte granularité caractérisant l'annotation de l'ATB, les catégories ne doivent pas être détectées d'emblée, mais le contrôle de leurs niveaux de granularité s'impose. L'emploi de ce contrôle est également motivé par le fait que nous voulons générer une grammaire à large couverture. Le fait d'offrir une certaine souplesse de présentation à cette grande masse de données favorise la qualité de la grammaire obtenue. Le contrôle à appliquer s'effectue au niveau de l'étape d'induction de la CFG vu qu'il

concerne les catégories détectées. La sous-section suivante explique les mécanismes de ce contrôle.

4.1.3 Contrôle des niveaux de granularité de la CFG

Pour pouvoir contrôler la granularité des catégories détectées, il faut en spécifier leurs traits. Ces traits peuvent être organisés selon le formalisme de hiérarchies des types. Nous rappelons qu'il considère que la catégorie grammaticale forme une structure de traits. Ces traits sont organisés sous forme hiérarchique selon leur niveau de spécification. Pour déterminer l'ensemble de ces traits dans l'ATB, il faut effectuer une étude de la structuration de ses différentes catégories grammaticales. Nous avons alors constaté que l'ATB a utilisé 22 traits dans son annotation comme indiqué dans les figures 4.1 et 4.2 de ce chapitre. La spécification de ces traits se fait en se basant sur des caractères de séparation comme « : », « - », « + », « _ » ainsi que des mots clés comme « NSUFF », « IVSUFF », « PVSUFF », « CASE », « MOOD », « PASS » et « POSS ».

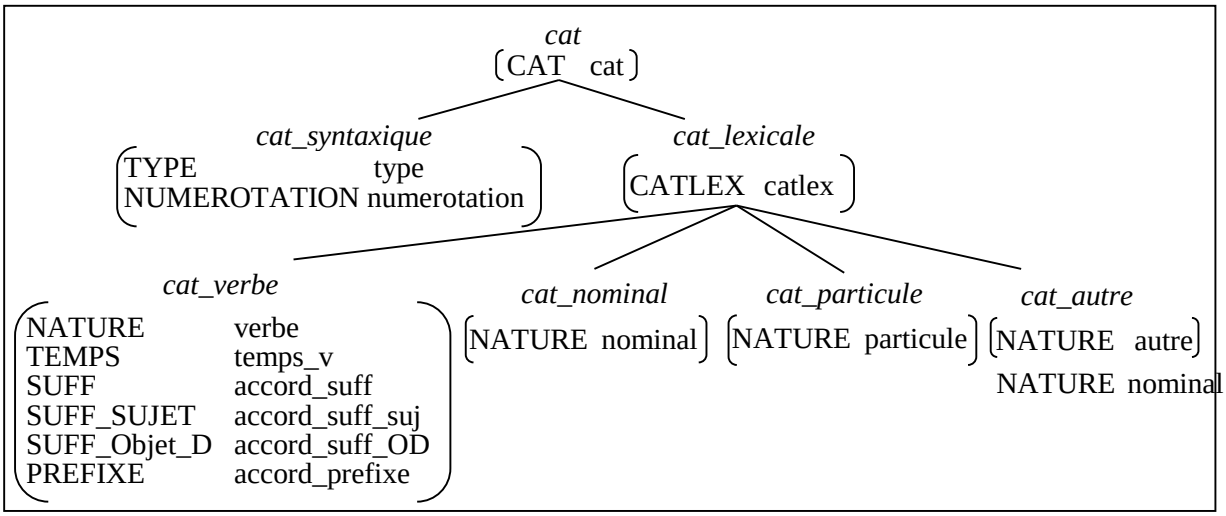


Figure 4.6: Hiérarchie de types caractérisant les catégories grammaticales dans l'ATB

La hiérarchie de types selon laquelle nous avons structuré les catégories grammaticales (syntaxiques et lexicales) de l'ATB est illustrée par la figure 4.6 ci-dessus. Les figures 4.7, 4.8 et 4.9 montrent les hiérarchies de types caractérisant des catégories lexicales bien particulières. Les racines de ces hiérarchies représentent des traits de valeurs complexes dans la figure 4.6.

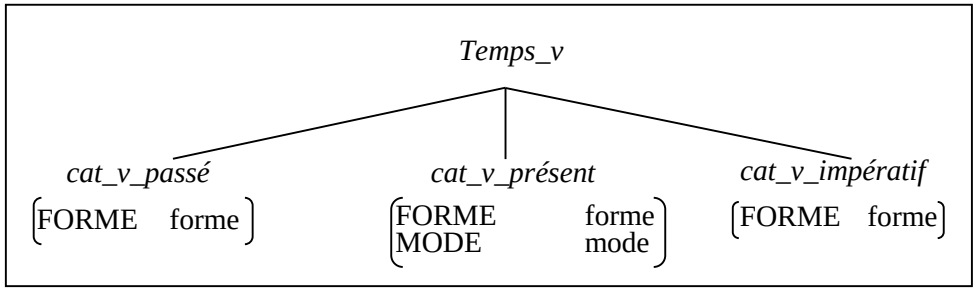


Figure 4.7: Hiérarchie de types caractérisant les verbes dans l'ATB

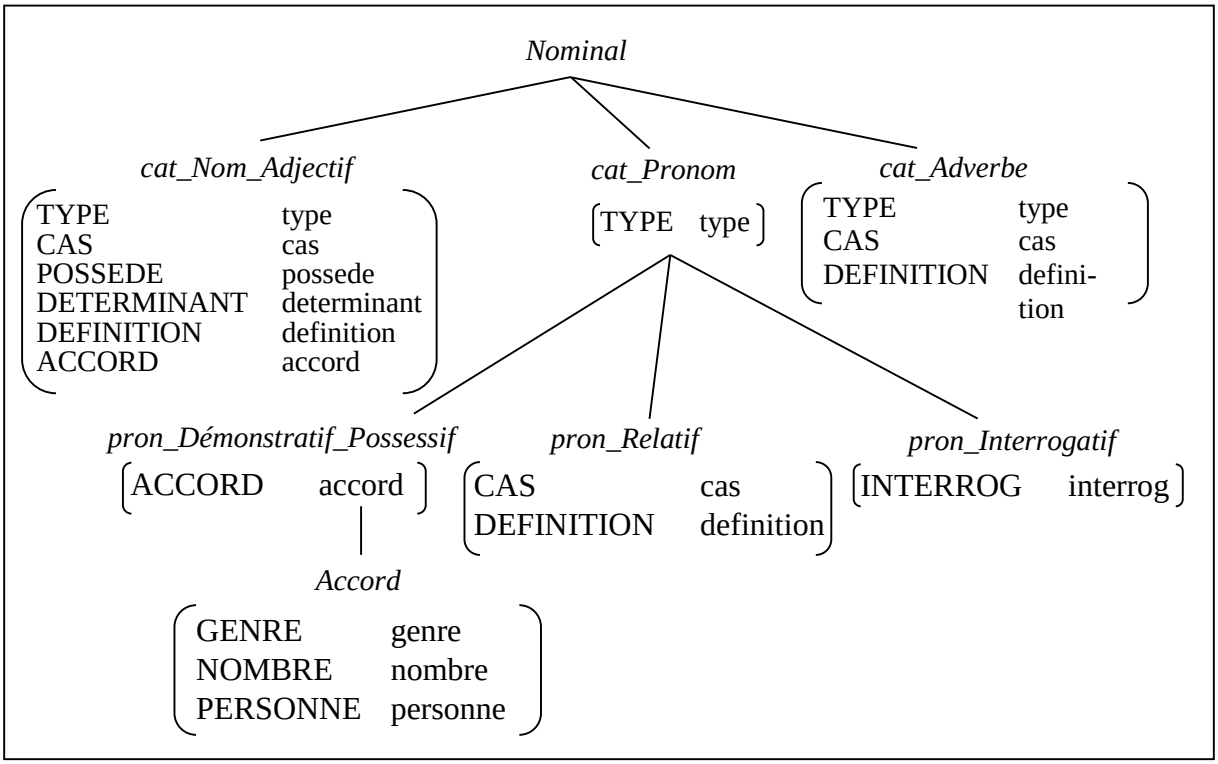


Figure 4.8: Hiérarchie de types caractérisant les nominaux dans l'ATB

Ainsi, la valeur complexe « nominal » du trait NATURE dans la figure 4.6 est représentée par le type « *nominal* » dans la figure 4.8. Ce type est décomposé en trois sous-types caractérisant respectivement les noms avec les adjectifs, les pronoms et les adverbes. Les pronoms, par exemple, sont eux-mêmes structurés selon leur type en les sous-sous-types suivants : les pronoms démonstratifs et possessifs, les pronoms relatifs et les pronoms interrogatifs.

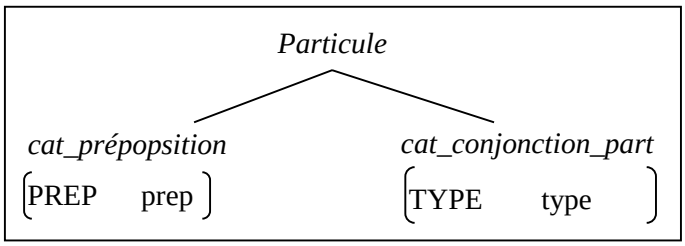
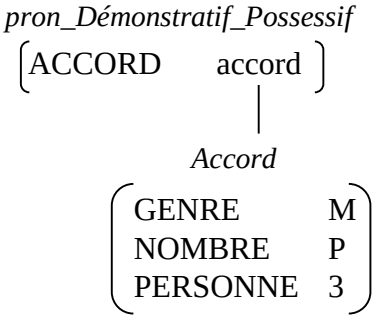


Figure 4.9: Hiérarchie de types caractérisant les particules dans l'ATB

Plus particulièrement, les pronoms démonstratifs et possessifs sont caractérisés par le trait ACCORD de valeur complexe accord. Cette valeur regroupe les traits genre, nombre et personne. Ainsi, le pronom possessif " هم " / « -hum » (leur/ leurs) étiqueté dans l'ATB avec la catégorie lexicale POSS_PRON_3MP est représenté dans la hiérarchie de types comme suit :



La variation du niveau de précision (spécification) des catégories influence le nombre de règles de la CFG et de la GP obtenue, et par conséquent leurs tailles. En effet, le fait d'ignorer un tel trait pour l'ensemble des catégories permet de les factoriser en un ensemble plus réduit. Cette factorisation peut affecter la qualité des CFG et GP obtenues. En effet, plus le niveau de granularité des catégories grammaticales est bas, plus leur nombre est réduit et plus la CFG est compacte mais abstraite et générale. Inversement, plus ce niveau est élevé, plus le nombre de catégories est grand et plus la CFG est détaillée mais précise et significative. La dégradation de la significativité dans la CFG est due à la perte d'informations au niveau des règles lorsqu'on réduit la granularité de ses catégories. Le contrôle de ce niveau de granularité est alors nécessaire pour pouvoir faire un compromis entre la quantité (taille) et la qualité de la CFG et par conséquent de la GP qui induit ses propriétés des règles de la CFG.

4.2 Induction de la GP à partir de la CFG

La CFG étant l'entrée de cette étape et la sortie de celle précédente, est utilisée pour induire automatiquement une GP. Ce qui facilite cette étape est le fait qu'aussi bien la CFG que la GP

sont structurées sous forme de constituants de niveau syntaxique auxquels nous affectons des informations de différents types. La différence réside au niveau du type de ces informations. En effet, la CFG donne pour chaque constituant syntaxique, l'ensemble de règles qui le décrit. Ces règles forment en fait des contraintes hiérarchiques. La GP, par contre, représente le constituant syntaxique à l'aide de contraintes non hiérarchiques qui sont les « propriétés ». Les propriétés sont automatiquement induites de la CFG par l'application de quelques descriptions algorithmiques pour les différents types de propriétés. Avant de montrer ces descriptions pour une catégorie syntaxique donnée définie dans la CFG, nous présentons les significations des variables utilisées.

- **t** : une catégorie syntaxique donnée définie dans la CFG.
- **RHS(t)** : ensemble de règles décrivant **t**.
- **Const(t)** : ensemble de constituants de **t**, initialisé à NIL (vide).
- **type_p** : propriété induite par type (constituance (const), linéarité (lin), adjacence (adjc), unicité (unic), obligation (oblig), exigence (exig), exclusion (excl)) entre les constituants $\mathbf{c_x}$ et $\mathbf{c_y}$ de **Const(t)**. **type_p** contient seulement $\mathbf{c_x}$ pour les types unicité et obligation (leurs propriétés sont unaires). **type_p** est vide (← NIL) et peut le rester s'il n'est pas valide ou n'est pas admis par toutes les règles de **t** dans la CFG.
- $\mathbf{c_i}$ et $\mathbf{c_j}$: paire de constituants de **Const(t).**
- **exist** : booléen, vraie si la règle traitée contient la paire $\mathbf{c_x}$ et $\mathbf{c_y}$ courante.
- **valid** : booléen, vraie si la relation est valide, elle est initialisée à « true » (vrai).
- **card** : nombre d'occurrences d'un constituant dans une règle.
- **fd1** et **fd2** : indices des constituants dans une règle, ils sont initialisés à -1.
- **add()** : méthode pour ajouter un constituant à un ensemble.
- **filter()** : méthode pour supprimer un élément en doublon d'un ensemble.
- **newProperty** : méthode pour créer une nouvelle Propriété.

Nous présentons dans les sous-sections suivantes, les algorithmes d'induction des différentes propriétés par type en se basant sur les interprétations de (Blache et Rauzy, 2012). La première sous-section décrit le premier type de propriété, soit la constituance. Elle génère l'ensemble des constituants possibles de la catégorie syntaxique à décrire. Les autres sous-sections permettent de vérifier la validité de chaque type de propriété pour une paire de constituants distincts donnée (pour les propriétés binaires) ou un constituant donné (pour les propriétés unaires) de la catégorie syntaxique à décrire. Pour générer toutes propriétés possibles des types spécifiés, chaque propriété traitée, sauf celle de constituance, doit être appliquée pour toutes les paires de constituants de cette catégorie syntaxique.

4.2.1 Propriété de constituance

Cette relation unique consiste à parcourir toutes les règles de la catégorie syntaxique à décrire et collecter tous ses constituants non dupliqués.

Algorithme 4.1: Interprétation des propriétés de constituance

```
Input : RHS(t), Const(t) ← NIL
Output : const_p
for each rhs_a in RHS(t), do
  for each c_m in rhs_a, do
    ajouter(c_m, Const(t))
  end for
end for
filtrer(Const(t))
const_p ← newProperty(t, Const(t))
return const_p
```

Selon cette description, il faut parcourir l'ensemble de règle **RHS(t)** et mettre dans l'ensemble **Const(t)** (initialement vide) chaque constituant rencontré dans chaque règle (en utilisant la méthode **ajouter()**). Il faut ensuite filtrer cet ensemble pour supprimer les constituants en doublons (en utilisant la méthode **filtrer()**). L'ensemble filtré est intégré dans la propriété de constituance.

4.2.2 Propriété de linéarité

Cette relation vérifie pour toutes les règles décrivant la catégorie syntaxique donnée la validité de la précédence linéaire entre les constituants de la paire donnée.

Algorithme 4.2: Interprétation des propriétés de linéarité

```
Input: c_i, c_j, RHS(t), exist ← false, valid ← true, lin_p ← NIL
Output: lin_p
for each rhs_a in RHS(t) and valid = true, do
  for each c_m in rhs_a and valid = true, do
    for each c_n in rhs_a and valid = true, do
      if c_m = c_i and c_n = c_j, then
        exist ← true
        if n < m, then
          valid ← false
        end if
      enf if
    end for
  enf for
end for
if valid = true and exist = true, then
  lin_p ← newProperty(t, c_i, c_j)
end if
return lin_p
```

Cette relation est valide si le premier constituant précède le second dans toutes les règles de **RHS(t)** décrivant la catégorie syntaxique **t**. Pour cela, nous considérons qu'elle est valide pour toutes les règles de **RHS(t)** tant qu'aucun contre-exemple de règle n'est trouvé (à travers la variable **valid** initialisée à "true"). Le contre-exemple consiste à trouver le second constituant précédant (ayant une position inférieure à celle de) le premier dans une règle. Par conséquence, la variable **valid** devient "false". Il faut aussi s'assurer si la paire de constituants a été déjà trouvée dans une même règle avant de vérifier si la relation est valide ou pas. Nous utilisons pour cela la variable **exist** initialisée à "false".

4.2.3 Propriété d'adjacence

Cette relation vérifie pour toutes les règles décrivant la catégorie syntaxique donnée la validité de l'adjacence gauche ou droite entre les constituants de la paire donnée.

Algorithme 4.3: Interprétation des propriétés d'adjacence

```
Input: ci, cj, RHS(t), exist ← false, valid ← true, adjc_p ← NIL
Output: adjc_p
for each rhsa in RHS(t) and valid = true, do
  for each cm in rhsa and valid = true, do
    for each cn in rhsa and valid = true, do
      if cm = ci and cn = cj, then
        exist ← true
        if n ≠ m-1 || n ≠ m+1, then
          valid ← false
        end if
      end if
    end for
  end for
end for
if valid = true and exist = true, then
  adjc_p ← newProperty(t, ci, cj)
end if
return adjc_p
```

Cette relation est valide si le premier constituant vient juste avant (ou respectivement juste après) le second dans toutes les règles de **RHS(t)** décrivant la catégorie syntaxique **t**. Alors, les deux constituants deviennent adjacents à gauche (respectivement adjacents à droite). Pour cela, comme la propriété de linéarité, nous considérons qu'elle est valide pour toutes les règles de **RHS(t)** tant qu'aucun contre-exemple de règle n'est trouvé (à travers la variable **valid** initialisée à "true"). Le contre-exemple consiste à trouver le second constituant n'ayant pas la position précédente ou suivante du premier dans une règle. Par conséquence, la variable **valid** devient "false". Il faut aussi s'assurer si la paire de constituants a été déjà trouvée dans une

même règle avant de vérifier si la relation est valide ou pas. Nous utilisons pour cela la variable **exist** initialisée à "false".

4.2.4 Propriété d'unicité (unic)

Cette relation vérifie pour toutes les règles décrivant la catégorie syntaxique donnée si le constituant donné de cette catégorie est répété dans la même règle.

Algorithme 4.4: Interprétation des propriétés d'unicité

```
Input: c_i, RHS(t), card ← 0, valid ← true, unic_p ← NIL
Output: unic_p
for each rhs_a in RHS(t) and valid = true, do
  for each c_m in rhs_a, do
     if c_m = c_i, then
       card ← card+1
     end if
   end for
   if card > 1, then
     valid ← false
   end if
end for
if valid = true, then
  unic_p ← newProperty(t, c_i)
end if
return unic_p
```

Selon cette interprétation, cette relation est valide seulement si le constituant donné $\mathbf{c_i}$ a une cardinalité (à travers la variable **card**) égale à 1 dans toutes les règles de **RHS(t).** Alors, ce constituant est considéré unique. Pour cela, nous allons supposer que $\mathbf{c_i}$ est unique tant qu'aucun contre-exemple de règle n'est trouvé (à travers la variable **valid** initialisée à "true"). Le contre-exemple consiste à avoir une valeur de cardinalité supérieure à 1. Par conséquence, la variable **valid** devient "false".

4.2.5 Propriété d'obligation (oblig)

Théoriquement, cette relation doit vérifier pour toutes les règles décrivant la catégorie syntaxique donnée si le constituant donné de cette catégorie apparait dans toutes les règles.

Algorithme 4.5: Première interprétation des propriétés d'obligation

```
Input: ci, RHS(t), valid ← true, oblig_p ← NIL
Output: oblig_p
for each rhsa in RHS(t) and valid = true, do
  valid ← false
  for each cm in rhsa and valid = false, do
     if cm = ci, then
        valid ← true
     end if
   end for
end for
if valid = true, then
  oblig_p ← newProperty(t, ci)
end if
return oblig_p
```

Selon cette interprétation, cette relation est valide seulement si le constituant donné $\mathbf{c_i}$ est toujours trouvé dans toutes les règles de **RHS(t).** Alors, ce constituant est considéré obligatoire (tête). Pour cela, nous allons supposer que cette relation est valide tant qu'elle le reste encore (à travers la variable **valid** initialisée à "true"). Cette relation reste valide si le constituant donné est trouvé dans chaque règle. Par conséquence, nous affectons la valeur "false" à la variable **valid** avant de chercher ce constituant dans chaque règle. Une fois il est trouvé, la valeur de cette variable devient "true".

En pratique, cette interprétation apporte un ensemble de propriétés d'obligation quasiment vide, car dans plusieurs cas, les constructions syntaxiques d'une même catégorie syntaxique ne se limitent à une seule catégorie obligatoire mais balancent entre deux ou plusieurs. Ainsi, dans l'ATB, à la catégorie syntaxique NP (Nominal Phrase) sont associées des constructions avec des éléments obligatoires totalement différents comme les pronoms démonstratifs (e.g : " هذه" (« h`*ihi » / cette)), les adjectifs comparatifs (e.g : "الأهم" (« Al+>aham~+u »/ le plus important)) et les suffixes compléments d'objets directs des verbes (e.g : "ـهُ" (« -hu »/lui)).

Nous avons alors ajusté cette interprétation en cherchant pour chaque construction syntaxique, le constituant considéré obligatoire indépendemment des autres constructions décrivant la catégorie syntaxique. Pour déterminer le constituant obligatoire d'une certaine construction, nous avons adopté l'interprétation proposée par (Habash et al., 2009). Ainsi, comme indiqué dans la section 6 du chapitre 3, cette interprétation se base sur une étude empirique des différentes constructions de l'ATB pour déterminer un ensemble de relations de dépendances comme le head (l'élément obligatoire), le sujet, l'objet, le modifieur, le topique et le prédicat. Le but des auteurs consiste à construire un nouveau treebank à base de dépendances

extraites de l'ATB : le treebank CATiB. Le système qu'ils ont développé pour cela, utilise seulement deux ressources construites manuellement. Ainsi, il exploite des tables de correspondances entre les étiquettes fournies par CATiB, celles utilisées dans le PTB, et celles adoptées par l'ATB. Il bénificie également d'un ensemble de règles pour extraire des relations de dépendances spécifiées par CATiB. Ces règles s'appliquent sur les différentes constructions possibles décrivant les catégories syntaxiques (XP) d'un texte annoté donné de l'ATB.

Etiquette CATiB	Etiquette PTB	Etiquette ATB
PNX	PUNC	PUNC
PROP	NNPS	NOUN_PROP+NSUFF_MASC_DU
NOM	DT+JJ	DET+ADJ
NOM	WRB	INTERROG_ADV
PRT	RP	PART
PRT	CC	CONJ
VRB	VBP	IV
VRB	VB	CV

Tableau 4.1: Extrait des tables de correspondances CATiB-ATB

Les tableaux 4.1 et 4.2 illustrent respectivement des exemples de tables de correspondances et de règles d'extraction de relations de dépendances.

Règle	XP	Construction	Relations de dépendances	Exemple
M1 SBAR 1:SBAR 2:@CC@ 3:SBAR 4:NP-SBJ => * 1:0 2:1,MOD 3:2,OBJ 4:1,SBJ	SBAR	SBAR [CC] SBAR NP-SBJ [CC] ∈ {CONJ,CONJP}	Heads : -SBAR1 pour [CC] et NP-SBJ -[CC] pour SBAR2 Modifieurs : [CC] pour SBAR1 Objet : SBAR2 pour [CC] Sujet : NP-SBJ	كتبت ونفذ حكام / *ktbt w+ nf* HkAm* « a écrit et des gouverneurs ont exécuté »
M1 VP 1:NOM 2:NP-SBJ 3:NP-PRD-ACC => NP 1:0 2:1,IDF 3:1,PRD	VP	NOM NP-SBJ NP-PRD-ACC NOM signifie les nominaux	Heads : NOM pour NP-SBJ et NP-PRD-ACC Idafa : NP-SBJ pour NOM Prédicat : NP-PRD-ACC pour NOM	كونه شاعرا / *kwn+h $AErA* « puisqu'il est conscient »

Tableau 4.2: Exemples de règles grammaticales d'extraction de dépendances à partir de l'ATB

Remarquons que dans le tableau 4.2, les relations de dépendances sont variées. Cependant, nous n'avons besoin que de l'exploitation de la relation « Head ». Cette relation spécifie les constituants desquels dépendent d'autres constituants d'une même construction. Il est possible de trouver dans cette liste, des constituants head qui dépendent d'autres constituants heads. C'est le cas notamment du constituant [CC] de la première construction qui est head de SBAR2

mais ayant lui aussi un head qui est SBAR1. Or, nous rappelons que notre but consiste à déterminer pour chaque construction son constituant obligatoire unique. Cela signifie qu'il faut sélectionner une seule relation Head. Selon notre interprétation, nous allons choisir le constituant head principal. A la différence des autres heads dans la construction, ce dernier ne dépend d'aucun autre constituant. Ainsi, comme pour la première construction, ce head (SBAR1) fait référence à l'indice le plus petit (0 dans notre cas). Chaque head extrait d'une construction est considéré comme propriété d'obligation décrivant la catégorie syntaxique annotant cette construction.

Contrairement aux autres propriétés syntaxiques, la détermination des propriétés d'obligation ne nécéssite pas la construction au préalable des règles de la CFG mais se base sur l'entrée de la CFG qui est le corpus annoté de l'ATB. L'exécution du système proposé par Habash et al. (2009) sur ce corpus permet de générer les dépendences qui caractèrisent chacun de ses mots. Soit la phrase annotée de l'ATB suivante :

Phrase	كتب وليد شقير يقول:	ktb wlyd $qyr yqwl :	Walid Chequir a écrit en disant :
Annotation	(S (VP (PV+PVSUFF_SUBJ:3MS ktb) (NP-SBJ-1 (NOUN_PROP wlyd) (NOUN_PROP $qyr)) (FRAG (VP (IV3MS+IV+IVSUFF_MOOD:I yqwl) (NP-SBJ-1 (-NONE- *)))) (PUNC :)))		

L'application du système de Habash et al. (2009) sur cette phrase nous a permis d'obtenir la liste de dépendances illustrées par la figure 4.10 suivante :

1	ktb	VRB	0	---	[WORD:ktb,LEXEME:,VRB/VBD/PV+PVSUFF_SUBJ:3MS]
2	wlyd	PROP	1	SBJ	[WORD:wlyd,LEXEME:,INDEX:1,DASHTAG:SBJ-SBJ,PROP/NNP/NOUN_PROP]
3	$qyr	PROP	2	---	[WORD:$qyr,LEXEME:,PROP/NNP/NOUN_PROP]
4	yqwl	VRB	1	MOD	[WORD:yqwl,LEXEME:,VRB/VBP/IV3MS+IV+IVSUFF_MOOD:I]
5	:	PNX	1	MOD	[WORD::,LEXEME:,PNX/PUNC/PUNC]

Figure 4.10: Résultat d'éxécution du système de Habash et al. (2009) sur la phrase de l'ATB : " كتب وليد شقير يقول:"

Dans la figure 4.10 ci-dessus, chaque ligne représente la caractérisation CATiB d'un mot de la phrase. Cette caractérisation porte sur son rang (ou indice), sa catégorie CATiB, l'indice du mot auquel il est relié, le type de la relation dépendance qu'il réalise et les catégories lexicales ATB et PTB correspondantes possibles à celles de CATiB. Remarquons que le premier mot (ktb) représente le head principal de cette phrase vu qu'il fait référence à l'indice le plus petit (0). Dans notre GP, sa catégorie lexicale dans l'ATB (PV+PVSUFF_SUBJ:3MS) doit faire l'objet d'une propriété d'obligation décrivant son parent VP. Ce parent sera également dans autre propriété d'obligation dans la description GP de son parent à lui qui est la catégorie syntaxique S. La spécification des propriétés d'obligation aux différents niveaux de l'arbre

syntaxique d'une phrase entrée de l'ATB nécessite la délimitation des diférentes constructions internes de cette phrase. Cette délimation permet d'adapter la numérotation de mots de l'ATB à celle produite par CATiB.

Algorithme 4.6 : Delimitation des catégories syntaxiques de chaque phrase de l'ATB

```
Input : phrase, nb_ouvrantes ← 0, nb_mots ← 0, lex, syn, pos, niveaux ← NIL, mots,
categs_syn ← NIL, categs_lex ← NIL, n_mot_deb_syn ← NIL, n_mot_fin_syn ← NIL
Output : phrase_delimitée
mots ← diviser(phrase, ' ')
for i ← 1 to longueur (mots) – 1, do
  if mots[i][1] = '(', then
     nb_ouvrantes ← nb_ouvrantes + 1
  end if
  if mots[i+1][1] <> '(', then //categorie lexicale
   lex ← extraire (mots[i], 2, longueur(mots[i]))
     if lex <> '-NONE-', then
        ajouter(categs_lex, lex)
        nb_mots ← nb_mots + 1
     end if
     //reemplir la chaine des parenthèses fermantes
     fermantes ← extraire (mots[i+1], indice(mots[i+1],')'),longueur(mots[i+1])
     i ← i+1
     //calcul position du mot final de la construction courante
     pos ← niveaux[nb_ouvrantes+1]
     for j ← 1 to longueur(fermantes), do
        nb_ouvrantes ← nb_ouvrantes - 1
        if j > 1, then
           if j = 2 and lex = '-NONE-', then //'-NONE-' pas compté
              n_mot_fin_syn[pos] ← n_mot_deb_syn[pos]
           else
              n_mot_fin_syn[pos_modif] ← nb_mots
           end if
        end if
     end for
  else
     syn ← extraire (mots[i], 2, longueur(mots[i]))
     ajouter(categs_syn, syn)
     if taille(niveaux) < nb_ouvrantes, then //nouveau niveau
        ajouter(niveaux, taille(categs_syn))
     else
        niveaux[nb_ouvrantes] ← taille(categs_syn)
     end if
  end if
end for
ajouter(n_mot_deb_syn, nb_mots)
ajouter(n_mot_fin_syn, 0)
phrase_délimitée ← (phrase,categs_syn,categs_lex,n_mot_deb_syn, n_mot_fin_syn)
```

L'algorithme ci-dessus réalise la délimitation d'une phrase entrée de l'ATB annotée sous forme parenthèsée. A chaque catégorie syntaxique sont associés respectivement les indices de son premier (num_mot_deb) et de son dernier mot (num_mot_fin). Il faut noter que les fonctions longueur(tableau), taille(liste), extraire(ch, p, q) et diviser(chaine, séparateur) qui sont

utilisées dans l'algorithme permettent de générer respectivement les valeur suivantes : la longueur d'un tableau, la taille d'une liste, la sous_chaine de caractère extraite d'une chaine ch à partir de la position p jusqu'à la position q et le tableau de chaines résultant de la division d'une grande chaine en fonction d'un caractère de séparation. Tandis que la procédure ajouter(liste, élément), elle permet d'ajouter un élément à la fin d'une liste.

Pour spécifier les propriétés d'obligation (heads) caractérisant chaque catégorie syntaxique, il faut exécuter le système de Habash et al. (2009) et correspondre les lignes de CATiB résultantes à celles de l'ATB. L'algorithme ci-dessous s'occupe de réaliser ces opérations :

Algorithme 4.7: Détermination des Heads des catégories syntaxiques des phrases de l'ATB

```
Input : min, fich_mapping_atb_catib, phrase_délimitée, program_catib
Output : heads_syn, n_heads_syn
lignes_catib ← exécuter(program_catib, phrase_delimitée, fich_mapping_atb_catib)
categs_syn, categs_lex,
for i ← 1 to taille(phrase_delimitée.categs_syn), do
  syn ← phrase_delimitée.categs_syn[i]
  n_deb ← phrase_delimitée.n_mot_deb_syn[i]
  n_fin ← phrase_delimitée.n_mot_fin_syn[i]
  if taille(lignes_catib) > 0, then
     min ← taille(phrase_delimitée.categs_lex)
     for j ← 0 to taille(lignes_catib), do
        champs ← diviser(lignes_catib[j], 'tabulation')
        if champs[4] < min, then //indice du mot de destination
           if syn <> 'LST' or (syn = 'LST' and champs[3]<>'PNX', then
              if champs[4] > n_deb and champs[4]  <= n_fin, then
                 min ← champs[4]
              else
                 min ← champs[1]
              end if
           end if
           if syn = 'LST' and taille(lignes_catib)=1, then
              min ← champs[1]
           end if
        end if
     end for
     if min < = taille(phrase_délimitée.categs_lex), then
        ajouter(heads_syn, phrase_délimitée.categs_lex[min])
        ajouter(n_heads_syn, min)
     else
        ajouter(heads_syn, '-NONE-')
        ajouter(n_heads_syn, -1)
     end if
  end if
end for
```

Dans l'algorithme ci-dessous, on spécifie si la catégorie syntaxique courante n'est pas marquée comme LST (une liste) qui ne contient pas des puces. Ceci est effectué pour ne pas avoir une catégorie head fausse.

4.2.6 Propriété d'exigence

Cette relation vérifie pour toutes les règles décrivant la catégorie syntaxique donnée la validité de la cooccurrence obligatoire entre les constituants de la paire donnée.

Algorithme 4.8: Interprétation des propriétés d'exigence

```
Input: ci,cj,RHS(t),fd1 ← 0,fd2 ← 0, valid ← true,
exig_p ← NIL
Output: exig_p
for each rhsa in RHS(t), do
  fd1 ← 0
  fd2 ← 0
  for each cm in rhsa , do
     if cm = ci and fd2≠m, then
        fd1 ← m
     end if
     if cm = cj and fd1≠m, then
        fd2 ← m
     end if
   end for
   if fd1≠0 and fd2=0, then
     valid ← false
   end if
end for
if valid = true, then
  exig_p ← newProperty(t, ci, cj)
end if
return exig_p
```

Cette relation est valide si, pour toutes les règles de **RHS(t)** décrivant la catégorie syntaxique **t**, l'apparence du premier constituant implique l'apparence du second dans la même règle. Cette relation n'est pas symétrique. En effet, si le second constituant apparait mais pas le premier dans la même règle, cette relation est considérée valide. Pour cela, nous considérons qu'elle est valide pour toutes les règles de **RHS(t)** tant qu'aucun contre-exemple de règle n'est trouvé (à travers la variable **valid** initialisée à "true"). Le contre-exemple consiste à trouver dans une règle le premier constituant sans le second. Nous pouvons savoir ceci en utilisant les variables **fd1** et **fd2**, qui décrivent les positions de la paire de constituants donnée dans la règle courante. Dans ce cas, **fd2** égale à 0 signifie que le second constituant n'est pas trouvé dans la règle. Par conséquence, la variable **valid** devient "false".

4.2.7 Propriété d'exclusion

Cette relation vérifie pour toutes les règles décrivant la catégorie syntaxique donnée la validité de la restriction de cooccurrence entre les constituants de la paire donnée. Cette relation est valide si, pour toutes les règles de **RHS(t)** décrivant la catégorie syntaxique **t**, l'apparence du premier constituant interdit l'apparence du second dans la même règle. A la différence de la relation d'exigence, cette relation est symétrique. En plus, même si aucun des deux constituants n'apparait dans une règle, cette relation reste valide. Pour cela, nous considérons qu'elle est valide pour toutes les règles de **RHS(t)** tant qu'aucun contre-exemple de règle n'est trouvé (à travers la variable **valid** initialisée à "true"). Le contre-exemple consiste à trouver dans une règle les deux constituants ensembles. Nous pouvons savoir ceci en utilisant les variables **fd1** et **fd2**, qui décrivent les positions de la paire de constituants donnée dans la règle courante. Dans ce cas, **fd1** et **fd2** différente de 0 signifie que les deux constituants sont les deux trouvés dans la règle. Par conséquence, la variable **valid** devient "false".

Algorithme 4.9: Interprétation des propriétés d'exclusion

```
Input: ci, cj, RHS(t), fd1 ← 0, fd2 ← 0, valid ← true,
excl_p ← NIL
Output: excl_p
for each rhsa in RHS(t), do
  fd1 ← 0
  fd2 ← 0
  for each cm in rhsa , do
     if cm = ci and fd2≠m, then
        fd1 ← m
     end if
     if cm = cj and fd1≠m, then
        fd2 ← m
     end if
   end for
  if fd1≠0 and fd2≠0, then
     valid ← false
   end if
end for
if valid = true, then
  excl_p ← newProperty(t, ci, cj)
end if
return excl_p
```

5 Conclusion

Nous avons proposé dans ce chapitre une démarche de construction d'une GP à granularité variable à partir de l'ATB, ce qui la rend une ressource à large couverture héritant des qualités de l'ATB comme sa fiabilité, sa soumission à des consensus et sa richesse en annotations de

différents types. La technique que nous avons adoptée pour construire cette ressource présente l'avantage d'être générique. En effet, elle est indépendante non seulement de toute langue, mais aussi, du formalisme source, puisque la génération des propriétés se fait directement à partir de la CFG. En plus, avec les types de propriétés définis jusqu'à présent et en excluant les propriétés d'obligation, cette technique est automatique, ce qui favorise sa réutilisabilité pour des treebanks de différentes langues et de différents formalismes sources.

Cette grammaire sera utilisée dans le chapitre suivant pour enrichir premièrement le treebank arabe ATB avec des représentations à base de propriétés, ce qui peut permettre d'améliorer la qualité du treebank et lui ajouter des informations implicites. Cet enrichissement sera appliqué également dans le résultat d'analyse de l'analyseur syntaxique Stanford Parser comme moyen d'évaluation de ce résultat. Pour optimiser la tâche d'enrichissement, plusieurs mécanismes de contrôle ont été intégrés au niveau de la détermination des unités syntaxiques à enrichir, et de l'efficacité de leurs propriétés linguistiques.

Chapitre 5

Exploitation de la grammaire de propriétés arabe pour la construction de nouvelles ressources arabes

1 Introduction

Le chapitre précédent a décrit les mécanismes d'induction d'une grammaire de propriétés à granularité variable à partir du treebank arabe ATB. Ces propriétés représentent des informations implicites dans le treebank. Les faire apparaître grâce à la grammaire de propriétés obtenue devient possible. Ceci entre dans le cadre de l'enrichissement du treebank. Cet enrichissement ne se limite pas à son amélioration par de nouvelles annotations plus riches mais permet sa conversion au formalisme des grammaires de propriétés. On aura dans ce cas une nouvelle représentation de ce treebank combinant les relations hiérarchiques qui caractérise son formalisme d'origine avec de nouvelles relations qui sont les propriétés syntaxiques. Ceci peut se faire en intégrant pour chaque unité syntaxique du treebank les propriétés syntaxiques de la GP qui peuvent la caractériser. Il faut appliquer dans ce cas plusieurs mécanismes de contrôle au niveau de la détermination des unités syntaxiques à enrichir, et de la spécification des propriétés syntaxiques à intégrer. L'enrichissement du treebank représente le premier volet d'utilisation des grammaires de propriétés. Nous proposons également comme deuxième volet d'utilisation de ce type de grammaires, l'évaluation des analyseurs syntaxiques en appliquant un enrichissement de leurs résultats avec des propriétés syntaxiques. Cette idée est née du fait que la performance d'un analyseur syntaxique n'est pas stable et peut varier en fonction du corpus donné. Or, les outils d'évaluation d'analyseurs disponibles se limitent à fournir des mesures sur tout le résultat d'analyse ou sur des syntagmes bien déterminés. L'application d'une méthode d'évaluation plus détaillée peut cerner les causes d'une telle détérioration de performances. Il s'agit en fait de vérifier la satisfaction des contraintes d'une GP sur le résultat d'analyse pour déterminer les relations syntaxiques qui n'ont pas été respectées. En passant, nous effectuons un enrichissement de la simple représentation du résultat d'analyse avec des propriétés syntaxiques de la GP, ce qui nous permet d'obtenir une représentation syntaxiquement plus significative. Notre méthode d'enrichissement et d'évaluation se focalise sur l'analyseur Stanford parser qui a marqué de meilleures performances pour plusieurs genres de corpus donnés.

Avant de dresser les approches d'enrichissement et d'évaluation proposées, nous envisageons entamer, dans la première section de ce chapitre, une étude empirique d'un ensemble de syntagmes arabes pour définir les propriétés syntaxiques qui peuvent les enrichir. Les deux autres sections abordées dans ce chapitre se focalisent sur l'explication des deux tâches d'enrichissement et d'évaluation que nous avons déjà proposées.

2 Etude empirique de l'enrichissement de syntagmes arabes avec des propriétés syntaxiques

Il faut tout d'abord vérifier la faisabilité de cette application d'enrichissement de treebank avec des propriétés syntaxiques. Nous avons pour cela conduit une étude linguistique d'un ensemble de syntagmes arabes permettant d'extraire ces propriétés pour les utiliser dans l'enrichissement de ces syntagmes. Cette étude nous permet de déduire les différentes interprétations formelles liées à chaque type de propriétés dans le formalisme de GP. La contribution (Bensalem et al, 2016) représente le support de cette étude. Avant de débuter cette étude, nous présentons dans le tableau 5.1 ci-dessous six exemples de phrases arabes pouvant regrouper les différents types de propriétés décrivant un syntagme donné.

S	ظهر ذلك ليس في تونس وحدها بل في خارجها أيضا.	أدى ذلك إلى "تحالف دولي".
T	Zahar+a *'lika layosa fiy tuwnis+a waHod+a-hA, balo fiy xArij+i-hA >ayoD+AF .	->ad~aY *'lika <ilaY " taHAluf+K duwaliy~+K " .
G	This appeared not only in Tunisia but also abroad.	This led to an "international coalition".
P	(S (VP (PV Zahar+a) (NP (PRON *'lika)) (PP (PRT (PART layosa)) (PREP fiy) (NP (NP (NOUN tuwnis+a)) (ADJP (ADJ waHod+a-) (NP (PRON -hA))))) (PUNC ,) (CONJ balo) (PP (PREP fiy) (NP (NOUN xArij+i-) (NP (PRON -hA)))) (ADVP (ADV >ayoD+AF)))) (PUNC .))	(S (VP (PV ->ad~aY) (NP (PRON *'lika)) (PP (PREP <ilaY) (PUNC ") (NP (NOUN taHAluf+K) (ADJ duwaliy~+K)) (PUNC "))) (PUNC .))
S	نجحت تونس سواء عن طريق المساعدات أو من خلال الاستثمارات.	سافر الرجل من تونس إلى مصر.
T	najaHat tuwnis+u sawA'+N Ean Tariyq+i Al+musAEad+At+i CONJ >awo min xilAl+i Al+{sotivomAr+At+i .	sAfar+a Al+rajul+u min tuwnis+a <ilaY miSr+a .
G	Tunisia succeeded, either through aids or through investments.	The man traveled from Tunisia to Egypt.
P	(S (VP (PV najaHat) (NP (NOUN tuwnis+u) (PP (PP (NP (NOUN sawA'+N)) (PREP Ean) (NP (NOUN Tariyq+i) (NP (NOUN Al+musAEad+At+i)))) (CONJ >awo) (PP (PREP min) (NP (NOUN xilAl+i) (NP (NOUN Al+{sotivomAr+At+i)))) (PUNC .))	(S (VP (PV sAfar+a) (NP (NOUN Al+rajul+u)) (PP (PP (PREP min) (NP (NOUN tuwnis+a))) (PP (PREP <ilaY) (NP (NOUN miSr+a))))) (PUNC .))
S	الخطر ليس فقط في أن الاقتصاد ينهار.	تحدث كما كان دائما.
T	Al+xaTar+u layosa faqaT fiy >an~a Al+{iqotiSAd+a ya+nohAr+u .	-taHad~av+a -ka-mA kAn+a dA}im+AF .
G	The danger is not only in the collapsing economy.	He talked as he was always.
P	(S (NP (NOUN Al+xaTar+u)) (VP (PV layosa) (NP (-NONE- *T*)) (PP (ADVP (ADV faqaT)) (PREP fiy) (SBAR (CONJ >an~a) (S (NP (NOUN Al+{iqotiSAd+a)) (VP (IV ya+nohAr+u) (NP (-NONE- *T*))))))) (PUNC .))	(S (VP (PV -taHad~av+a) (NP (-NONE- *)) (PP (PREP -ka-) (SBAR (WHNP (PRON -mA)) (S (VP (PV kAn+a) (NP (-NONE- *T*)) (NP (ADJ dA}im+AF))))))) (PUNC .))

Tableau 5.1 : Exemples de phrases arabes analysées selon l'annotation du treebank ATB

Dans le tableau 5.1 ci-dessus, les symboles S, T, G et P ont respectivement les significations suivantes : les phrases arabes, leur formes translitérées selon Buckwalter[14], leurs traductions en anglais et leurs analyses syntaxiques selon l'annotation de l'ATB (Maamouri et al, 2004). Dans ce qui suit, nous présentons quelques exemples de structures extraits de ces phrases pour exprimer l'application des différentes propriétés syntaxiques du formalisme de GP. Plus particulièrement, nous nous focalisons, pour choix arbitraire, sur les structures de syntagmes prépositionnels connus sous l'étiquette PP (Prepositional Phrase) pour décrire chaque type de propriétés.

2.1 Propriété de linéarité

La relation PREP $\prec$ NP indique qu'une préposition (PREP) doit toujours précéder le syntagme nominal (Nominal Phrase, NP) dans la langue arabe. La liste de différents exemples de structures de PP suivante prouve ceci :

	Constituants de PREP	**Constituants de NP**
1	Fiy	(**NP** (NOUN tuwnis+a)) (**ADJP** (ADJ waHod+a-) (NP (PRON -hA)))
2	Fiy	(**NOUN** xArij+i-) (**NP** (PRON -hA))
3	Ean	(**NOUN** Tariyq+i) (**NP** (NOUN Al+musAEad+At+i))
4	Min	(**NOUN** xilAl+i) (**NP** (NOUN Al+{sotivomAr+At+i))
5	Min	(**NOUN** tuwnis+a)
6	<ilaY	(**NOUN** miSr+a)

Pour une représentation arborescente plus claire, nous avons illustré en gras les constituants de chaque catégorie syntaxique spécifiée dans une certaine propriété décrivant PP. Cependant, nous pouvons trouver à partir de la même liste un contre-exemple réfutant cette relation et montrant un NP précédant une PREP, comme la partie de structure de PP " سواء عن " (sawA'+N Ean / either about) :

Constituants de NP	**Constituants de PREP**
(**NOUN** sawA'+N)	Ean

A cause d'un seul contre-exemple, cette relation ne peut pas apparaître dans la GP à construire.

2.2 Propriété d'adjacence

Un autre type de relation entre les mêmes catégories PREP et NP peut être observé dans les exemples de structures de PP présentés pour la propriété de linéarité. C'est la propriété

[14] Tableau de transliteration arabe dans le site de Tim Buckwalter site: www.qamus.org/transliteration.htm

d’adjacence. En effet, PREP se place toujours juste avant ou juste après le NP. Ces deux catégories peuvent être adjacentes si aucun contre-exemple n’est trouvé. Cependant, nous avons cet exemple de structure de PP ci-dessous. Il montre la partie gauche (Left-Hand Side, LHS) d’un exemple de règle, étant bien sûr, PP et sa partie droite (Right-Hand Side, RHS), étant la structure “"تحالف دولي "إلى” (<ilaY " taHAluf+K duwaliy~+K"/ to an “international coalition”) :

LHS	**RHS**			
PP	**PREP**	**PUNC**	**NP**	**PUNC**
1	<ilaY	“	(**NOUN** taHAluf+K) (**ADJ** duwaliy~+K)	”

Cette structure de PP sépare ces deux catégories (PREP et NP) par un symbole de ponctuation (“), la relation d’adjacence entre PREP et NP ne peut pas alors être valide.

2.3 Propriété d’unicité

C’est une relation unaire, elle ne porte alors que sur une seule catégorie. Elle induit chaque constituant unique dans la catégorie syntaxique décrite. Dans les structures de PP, nous avons plusieurs catégories uniques comme PREP, SBAR (Subordinate clause), ADVP (Adverbial phrase) and ADJP (Adjectival Phrase). Quelques exemples de structures de PP prouvant cette unicité sont montrés ci-dessous pour quelques constituants uniques dans PP :

Constituants uniques	**PREP**	**SBAR**	**ADVP**
LHS	**RHS**		
PP	(**ADVP** (ADV faqaT)) (**PREP** fiy) (**SBAR** (CONJ >an~a) (S (NP (NOUN Al+{iqotiSAd+a)) (VP (IV ya+nohAr+u) (NP (-NONE- *T*)))))		
	(**PREP** -ka-) (**SBAR** (WHNP (PRON -mA)) (S (VP (PV kAn+a) (NP (-NONE- *T*)) (NP (ADJ dA}im+AF)))))		
	(**PREP** fiy) (**NP** (NOUN xArij+i-) (NP (PRON -hA)))		

Le premier exemple de structure de PP est commun pour tous les constituants spécifiés (PREP, SBAR, ADVP) car il les montre tous uniques. Le second exemple est au contraire commun pour PREP et SBAR seulement. Le N a pu être lui aussi unique dans les structures de PP structures si nous n’avons pas l’exemple de structure de PP “سواء عن طريق المساعدات” (sawA'+N Ean Tariyq+i Al+musAEad+At+i / either through aids) illustré dans ce qui suit :

LHS	**RHS**
PP	(**NP** (NOUN sawA'+N)) (**PREP** Ean) (**NP** (NOUN Tariyq+i) (NP (NOUN Al+musAEad+At+i)))

2.4 Propriété d'obligation

Cette relation est également unaire. Elle porte sur les constituants toujours présents, et qui forment une tête dans la catégorie syntaxique décrite. A partir de la plupart des exemples de structures de PP, nous pouvons observer que PREP est un constituant obligatoire dans le PP. Cette relation ne peut pas être valide pour l'exemple de structure de PP "من تونس إلى مصر" (min tuwnis+a <ilaY miSr+a / from Tunisia to Egypt) illustré dans ce qui suit :

LHS	RHS
PP	(**PP** (PREP min) (NP (NOUN tuwnis+a))) (**PP** (PREP <ilaY) (NP (NOUN miSr+a)))

Dans cet exemple, PREP n'est pas directement un constituant obligatoire dans le sous-arbre de dérivation du PP supérieur, parce que cette structure de PP est composée de deux PP successives, chacune contient une PREP.

2.5 Propriété d'exigence

Une propriété d'exigence indique que l'apparence du premier constituant implique l'apparence du second. La relation ADVP ⇒PREP respecte cette condition dans plusieurs exemples de structures de PP, tel que la structure "فقط في أن الاقتصاد ينهار" (faqaT fiy >an~a Al+{iqotiSAd+a ya+nohAr+u / only in the collapsing economy) :

LHS	RHS
PP	(**ADVP** (ADV faqaT)) (**PREP** fiy) (**SBAR** (CONJ >an~a) (S (NP (NOUN Al+{iqotiSAd+a)) (VP (IV ya+nohAr+u) (NP (-NONE- *T*)))))

Cependant, il existe un autre exemple de structures de PP réfutant cette relation, qui est "ليس في تونس وحدها بل في خارجها أيضا" (layosa fiy tuwnis+a waHod+a-hA, balo fiy xArij+i-hA >ayoD+AF / not only in Tunisia but also abroad), illustré dans ce qui suit :

LHS	RHS
PP	(**PP** (PRT (PART layosa)) (PREP fiy) (NP (NP (NOUN tuwnis+a)) (ADJP (ADJ waHod+a-) (NP (PRON -hA))))) (**PUNC** ,) (**CONJ** balo) (**PP** (PREP fiy) (NP (NOUN xArij+i-) (NP (PRON -hA)))) (**ADVP** (ADV >ayoD+AF))

Dans cet exemple, ADVP apparait sans PREP. Si nous n'avons pas ce contre-exemple, cette relation devient valide. Cependant, sa relation symétrique (PREP ⇒ADVP) n'est pas forcément valide. En effet, la fameuse structure de PP (PREP NP) la réfute. Cette structure de PP ne permet pas de valider la relation NP ⇒PREP. Ceci est parce qu'il y a des structures de PP où apparait avec un NP, une PP au lieu d'une PREP.

La relation SBAR ⇒PREP est, au contraire, toujours valide. Alors, dans tout exemple de structure de PP, si nous avons un SBAR, nous trouvons absolument une PREP.

2.6 Propriété d'exclusion

La propriété d'exclusion est le contraire de celle d'exigence. Elle interdit la co-occurrence de deux constituants. Pour le PP, lorsque nous observons les catégories ADVP et SBAR dans plusieurs exemples arabes de structures de PP, nous constatons qu'ils n'apparaissent pas ensembles dans ces structures. Voici deux exemples prouvant cette observation, qui sont " ليس في تونس وحدها بل في خارجها أيضا" (layosa fiy tuwnis+a waHod+a-hA, balo fiy xArij+i-hA >ayoD+AF / not only in Tunisia but also abroad) et "كما كان دائما" (-ka-mA kAn+a dA}im+AF / as we was always) :

	LHS	RHS
Only **ADVP**	**PP**	(**PP** (PRT (PART layosa)) (PREP fiy) (NP (NP (NOUN tuwnis+a)) (ADJP (ADJ waHod+a-) (NP (PRON -hA))))) (**PUNC** ,) (**CONJ** balo) (**PP** (PREP fiy) (NP (NOUN xArij+i-) (NP (PRON -hA)))) (**ADVP** (ADV >ayoD+AF))
Only **SBAR**		(**PREP** -ka-) (**SBAR** (WHNP (PRON -mA)) (S (VP (PV kAn+a) (NP (-NONE- *T*)) (NP (ADJ dA}im+AF)))))

Si nous nous limitons à ces exemples, nous allons avoir une relation d'exclusion entre ADVP et SBAR (ADVP ⊗SBAR). A la différence des propriétés d'exigence, cette relation d'exclusion est symétrique. Ainsi, lorsque SBAR apparait, ADVP n'apparait pas. Cependant, en étendant notre vision à l'exemple de structure de PP "فقط في أن الاقتصاد ينهار" (faqaT fiy >an~a Al+{iqotiSAd+a ya+nohAr+u / only in the collapsing economy) où SBAR et ADVP apparaissent ensembles dans la structure de PP, cette relation devient invalide.

LHS	RHS
PP	(**ADVP** (ADV faqaT)) (**PREP** fiy) (**SBAR** (CONJ >an~a) (S (NP (NOUN Al+{iqotiSAd+a)) (VP (IV ya+nohAr+u) (NP (-NONE- *T*)))))

Cette étude linguistique nous donne une idée sur la difficulté à rencontrer pour déterminer et énumérer les différentes relations syntaxiques valides dans la langue arabe. Ces relations sont implicites dans les textes arabes. Les rendre explicites peut être utile pour plusieurs applications de TALN dans différents domaines, tels que : la résolution des ambiguïtés, l'analyse probabiliste et la construction de grammaires de dépendance.

3 Enrichissement de l'ATB avec des propriétés syntaxiques

Pour assurer la tâche d'enrichissement de l'ATB avec une représentation à base de GP, il est nécessaire de savoir dans quel cadre il faut intégrer l'ATB et la GP induite. Pour cela, il faut comprendre comment la génération de cet ATB enrichi va être réalisée. Il s'agit en fait de vérifier la satisfaction des propriétés fournies par la GP dans les différents syntagmes du treebank ATB. Cette vérification peut être vue comme un problème de satisfaction de contraintes (Constraint Satisfaction Problem, CSP). Ce problème a été expliqué d'une manière formelle dans notre contribution (Bensalem et al, 2015b). Dans ce cade, nous avons modélisé ce problème par un 5-uplet (S(ATB), S(GP), Const(ATB), Const(GP), Prop(GP)) où :

- S(ATB) = $\{p_1, p_2, \ldots p_n\}$ = P est un ensemble fini de syntagmes de l'ATB.
- S(GP) = $\{t_1, t_2, \ldots t_m\}$ = N est un ensemble fini de catégories syntaxiques de la GP.
- Const(ATB) = $\bigcup_{i=1}^{n} Const(s_i)$ où Const(s_i) = $\{c_{i1}, c_{i2}, \ldots c_{ie}\}$: ensemble de constituants (mots) de syntagme s_i de l'ATB, tel que label(c_{ix}) est sa catégorie grammaticale.
- Const(GP) = $\bigcup_{j=1}^{m} Const(t_j)$ où Const(t_j) = $\{c_{j1}, c_{j2}, \ldots c_{jf}\}$: ensemble de constituants (catégories grammaticales) de la catégorie syntaxique t_j dans la GP.
- Prop(GP) = $\bigcup_{j=1}^{m} Prop(t_j)$ où Prop(t_j)=[Prop_const(t_j), Prop_unic(t_j), Prop_oblig(t_j), Prop_lin(t_j), Prop_adjc(t_j), Prop_exig(t_j), Prop_excl(t_j)], par exemple, Prop_lin(t_j) = $\{p_{j1}, p_{j2}, \ldots p_{jg}\}$ est l'ensemble des propriétés de linéarité décrivant la catégorie syntaxique t_j dans la GP. Chaque propriété p_{jk} (avec $1<k<g$) est une relation de précédence linéaire entre deux éléments de Const(t_j) de t_j (p_{j1}=tj:$c_{jx} \prec c_{jy}$ avec c_{jx} et $c_{jy} \in$ Const(t_j)).

Pour résoudre ce problème, il faut trouver dans la GP la catégorie syntaxique qui correspond à la catégorie de chaque syntagme de l'ATB. Formellement, nous avons besoin de chercher pour chaque syntagme $p_i \in$ P de l'ATB, la catégorie syntaxique $t_j \in$ N dans la GP tel que label(p_i) = t_j. L'ensemble des propriétés Prop(t_j) décrivant la catégorie t_j sera utilisé pour enrichir le syntagme p_i par la vérification de leur satisfaction.

Il est maintenant possible de représenter en détail la phase de génération de l'ATB enrichi avec les propriétés en utilisant la GP induite. Nous rappelons qu'elle s'articule autour de quatre étapes que nous allons détailler dans les sous-sections suivantes après avoir illustré la figure 5.1 montrant le déroulement de ces étapes dans notre phase de génération d'ATB enrichi.

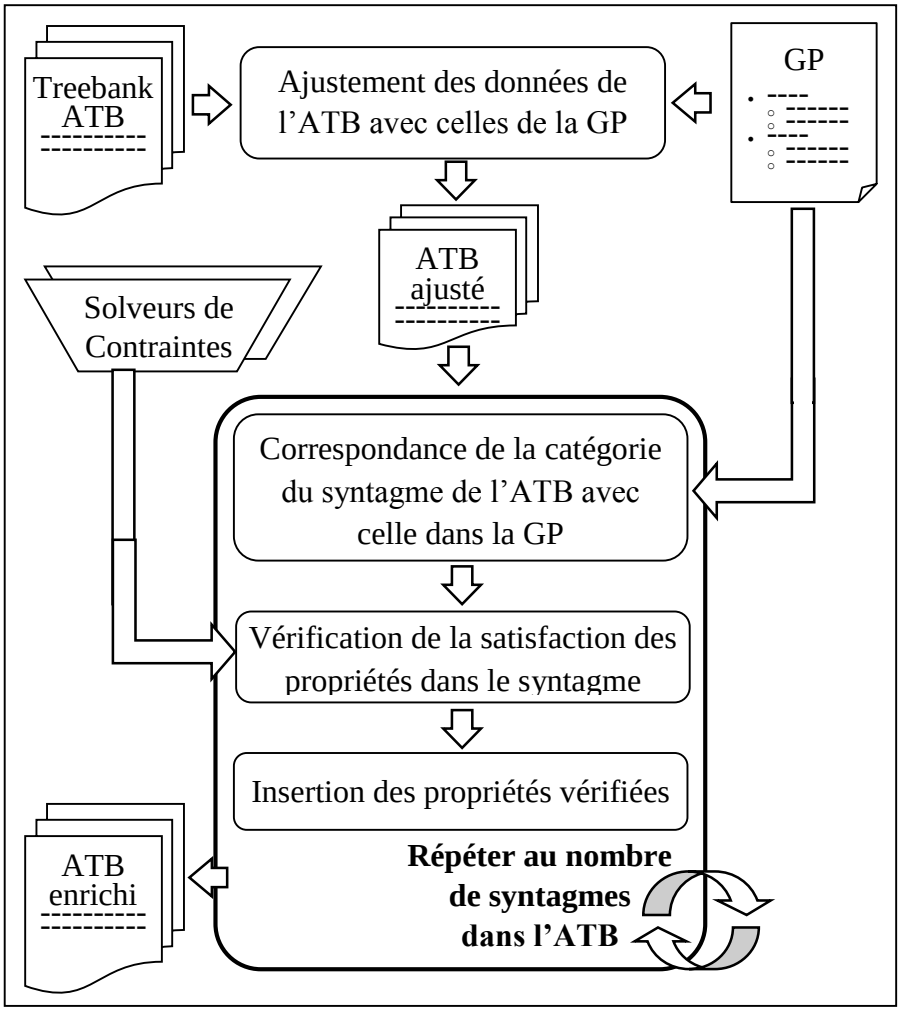


Figure 5.1 : Phase de génération de l'ATB enrichi avec des propriétés syntaxiques

Nous avons détaillé la phase d'enrichissement de l'ATB dans notre contribution (Bensalem et al., 2015a). Nous consacrons alors les sous-sections suivantes à la description des différentes étapes que nous avons appliquées à travers cette contribution pour réaliser cer enrichissement.

3.1 Ajustement des données de l'ATB selon celles de la GP

Cette étape tend beaucoup plus vers le niveau implémentation algorithmique car nous traitons ici le treebank source ATB à enrichir. Le format de l'ATB que nous avons choisi est le format PTB. Ainsi, il offre exactement les informations que nous en a besoin à savoir : la représentation arborescente et les catégories grammaticales des syntagmes et des mots pour chaque phrase. Notre but ici ne se limite pas à vérifier la satisfaction des propriétés de la GP pour les syntagmes du treebank mais s'étend à les enrichir avec cette nouvelle information (résultat de vérification de la satisfaction). Ceci génère une annotation étendue et enrichie du treebank au-delà des relations hiérarchiques entre les constituants. Par conséquent, il faut avoir

en entrée une structure de données de notre treebank source capable d'héberger cette nouvelle information. Le format parenthésé du Penn Treebank (PTB) ne fournit pas cette structure. Le format "xml" peut former cette structure, qui est à la fois accessible pour la navigation et l'enrichissement. La disposition d'un tel format ne nous a pas exemptés de la préparation d'une version simple à cause de la difficulté de manipulation de la version disponible. Par conséquent, nous avons converti le format PTB au format "xml" en établissant un processus récursif de conversion des parenthèses ouvrantes et fermantes rencontrées dans chaque document PTB en balises xml (<Paragraph> and <Category>). Les données de l'ATB sont alors organisées selon la DTD illustrée par la figure 5.2 suivante :

```
<?xml version= "1.0" encoding="UTF-8"?>
<!DOCTYPE XML_file_ATB [
<!ELEMENT XML_file_ATB (Paragraph+) >
<!ATTLIST XML_file_ATB penntree_name CDATA #REQUIRED >
<!ELEMENT Paragraph (Category+) >
<!ATTLIST Paragraph index  CDATA #REQUIRED >
<!ELEMENT Category (Category?) >
<!ATTLIST Category index CDATA #REQUIRED label CDATA #REQUIRED
value CDATA #IMPLIED >]>
```

Figure 5.2: DTD des données de l'ATB

La DTD obtenue est conforme au format PTB de l'ATB qui a été utilisée en entrée de l'étape courante. Ce format représente l'ATB sous forme d'arbres montrant chaque mot dans une structure hiérarchique à la suite de son étiquette POS.

En plus, l'ajustement des données de l'ATB ne se limite pas à l'adaptation d'un format consistent avec nos besoins d'utilisation, mais inclut également l'adéquation de l'ATB avec le niveau de granularité des catégories spécifié au début du traitement. Dans ce cas, un modèle de vérification des correspondances est intégré pour remplacer les catégories de l'ATB sous leur niveau d'origine par celles ajustées au niveau de granularité modifié.

Vu que la vérification de la satisfaction des propriétés concerne chaque syntagme de l'ATB à part, nous avons encapsulé les étapes suivantes dans un mécanisme de parcours répété au nombre de syntagmes dans l'ATB. Ainsi, l'entrée de ce mécanisme de parcours est l'ATB ajusté sous son format « xml » et la sortie, générée également en « xml » sera une nouvelle version de l'ATB enrichie avec les propriétés vérifiées satisfaites ou violées.

3.2 Correspondance des catégories du syntagme de l'ATB et de la GP

La correspondance entre l'ATB et la GP consiste à parcourir l'ATB, syntagme par syntagme, et pour chaque syntagme, à chercher le correspondant de sa catégorie dans la GP. Les propriétés décrivant le correspondant trouvé seront vérifiées et associées au syntagme courant de l'ATB comme information enrichissante. Formellement, pour faire la correspondance entre un syntagme de l'ATB $p_i \in P$ et le correspondant de sa catégorie dans la GP, il faut voir pour chaque catégorie $\alpha \in N$ dans la GP, le cas où la catégorie syntaxique de p_i label(p_i) soit égale à α (label (p_i) = α). Selon la DTD de l'ATB dans la figure 5.2, cette correspondance consiste à se pointer dans la hiérarchie xml sur une étiquette <Category> qui n'a pas l'attribut « value » spécifique aux catégories lexicales). L'étiquette doit avoir en plus une valeur dans son attribut « label » égale à la valeur de l'attribut « name » caractérisant les étiquettes <Phrase> de la GP. Ces étiquettes font référence aux catégories syntaxiques de la GP sous lesquelles sont décrits leurs constituants et propriétés syntaxiques possibles. La DTD de la GP montrant ces éléments est illustrée par la figure 5.3 suivante :

```
<?xml version= "1.0" encoding="UTF-8"?>
<!DOCTYPE GP [
<!ELEMENT GP (Phrase+) >
<!ELEMENT Phrase (Constituents+, Caracteriszation+) >
<!ATTLIST Phrase index  CDATA #REQUIRED name CDATA #REQUIRED>
<!ELEMENT Constituents (Constituent+) >
<!ELEMENT Constituent EMPTY>
<!ATTLIST Constituent index CDATA #REQUIRED name CDATA #REQUIRED >
<!ELEMENT Caracterization (Property?) >
<!ELEMENT Property EMPTY>
<!ATTLIST Property index CDATA #REQUIRED type CDATA #REQUIRED forme
CDATA #REQUIRED source CDATA #REQUIRED target CDATA #REQUIRED >]>
```

Figure 5.3: DTD de la GP utilisée

3.3 Vérification de la satisfaction des propriétés pour le syntagme

Cette étape représente le cœur de la phase de génération de l'ATB enrichi. Elle permet de vérifier la satisfaction des propriétés décrivant la catégorie de la GP qui correspond au syntagme courant de l'ATB. En se basant sur la modélisation formelle de notre problème d'enrichissement, ceci signifie que nous avons juste besoin de vérifier pour chaque syntagme p_i de l'ATB correspondant à la catégorie t_j de la GP (assurant label(p_i) = t_j), la satisfaction de toutes les propriétés de l'ensemble Prop(t_j) obtenu à partir de la GP . Pour cela, nous avons utilisé un ensemble de méthodes de vérification de la satisfaction des propriétés. Chaque méthode, appelée « solveur de contraintes » vérifie si un syntagme arabe donné caractérisé par

une certaine catégorie syntaxique satisfait une propriété donnée décrivant dans la GP cette catégorie syntaxique. La solution produite par un solveur de contraintes est le résultat de cette vérification (propriété satisfaite ou violée). Cette solution est ensuite insérée dans la description du syntagme p_i dans l'ATB. Nous présentons dans le reste de la section les différentes descriptions algorithmiques de ces solveurs de contraintes. Nous rappelons qu'elles sont inspirées des interprétations de (Blache et Rauzy, 2012). La vérification à réaliser des propriétés sera faite pour les différents types. La correspondance est supposée assurée puisqu'il s'agit de l'étape précédente. Ainsi, la catégorie du syntagme courant de l'ATB est égale à la catégorie trouvée dans la GP. Comme pour tout algorithme, nous définissons tout d'abord les significations des différentes variables utilisées dans les descriptions.

- **p** : syntagme donné de l'ATB.
- **Const(p)** : ensemble de constituants du syntagme de l'ATB **p**.
- **Const(t)** : ensemble de constituants de la catégorie de la GP **t** (tel que **t=label(p)**)
- **label(c)** : catégorie grammaticale du mot/syntagme **c**.
- **fd** : booléen, true (vrai) si **c** est trouvé.
- **nb_intersect :** nombre de constituants en intersection entre **Const(p)** et **Const(t)**.
- **verif :** chaine de caractère de valeur "+" ou "-".
- **nb_occ :** nombre d'occurrences d'un constituant dans **Const(p)**
- **type_p** : proprieté par type entre deux constituants $\mathbf{c_x}$ et $\mathbf{c_y}$ de **Const(t)**. **type_p** contient seulement $\mathbf{c_x}$ pour les propriétés unaires (d'unicité, d'obligation).
- **v_type_p :** propriété vérifiée par type (de constituance (const), de linéarité (lin), d'adjacence (adjc), d'unicité (unic), d'obligation (oblig), d'exigence (exig), d'exclusion (excl)) (ayant "+" si satisfaite, "-" sinon). Initialement, **v_type_p** est vide (← NIL) et peut le rester si les de **type_p** ne sont pas trouvés dans **p**.
- **verifProp() :** méthode pour créer une propriété vérifiée.

3.3.1 Solveur des propriétés de constituance

Ce solveur vérifie la relation de constituance entre les catégories des constituants du syntagme de l'ATB courant et les constituants de son correspondant dans la GP. Il faut s'assurer que l'intersection entre ces deux ensembles inclut tous les mots du syntagme de l'ATB.

Algorithme 5.1: Interprétation du solveur des propriétés de constituance

```
Input: Const(p), Const(t), nb_intersect ← 0, const_p
Output: v_const_p
for each ca in Const(p), do
   for each cb in Const(t), do
      if label(ca)= cb, then
         nb_intersect ← nb_intersect + 1
       end if
    end for
end for
if nb_intersect = card(Const(p), then
   v_const_p ← verifProperty(p, Const(p), "+")
else
   v_const_p ← verifProperty(p, Const(p), "+")
end if
return v_const_p
```

Cet algorithme signifie qu'il faut parcourir **Const(p)** et vérifier que la catégorie de chacun de ses éléments est inclut dans **Const(t)**. La valeur de la variable **nb_intersect** est alors incrémentée. Si la valeur finale de cette variable est égale au cardinal de **Const(p)** cardinal, la propriété est considérée satisfaite.

3.3.2 Solveur des propriétés de linéarité

Ce solveur vérifie la propriété de linéarité courante de la catégorie syntaxique de la GP. La satisfaction est assurée seulement si les deux constituants de cette catégorie formant cette relation sont aussi trouvés dans le syntagme donné et le premier constituant précède le second.

Algorithme 5.2: Interprétation du solveur des propriétés de linéarité

```
Input: Const(p), lin_p, v_lin_p ← NIL
Output: v_lin_p
for each ca in Const(p), do
  if label(ca)= lin_p.cx, then
    for each label(cb) in Const(p), do
       if a≠b and label(cb)= lin_p.cy, then
          if a>b, then
             v_lin_p ← VerifProperty(p, lin_p, "-")
          else
             v_lin_p ← VerifProperty(p, lin_p, "+")
          end if
       end if
    end for
  end if
end for
return v_lin_p
```

Cet algorithme parcoure les catégories des éléments de l'ensemble **Const(p)** à la recherche des deux constituants linéaires distincts $\mathbf{c_x}$ et $\mathbf{c_y}$ de la catégorie **t** de la GP et vérifie que la position du premier n'est pas supérieure à la position du second dans le syntagme **p**.

3.3.3 Solveur des propriétés d'adjacence

Ce solveur vérifie la propriété d'adjacence courante de la catégorie syntaxique de la GP. La satisfaction est assurée seulement si les deux constituants adjacents de cette catégorie sont également trouvés dans le syntagme donné et le premier arrive juste avant ou après le second.

Algorithme 5.3: Interprétation du solveur des propriétés d'adjacence

```
Input: Const(p), adjc_p, v_adjc_p ← NIL
Output: v_adjc_p
for each c_a in Const(p), do
 if label(c_a)= adjc_p.c_x, then
  for each c_b in Const(p), do
    if a≠b and label(c_b)= adjc_p.c_y, then
      if a≠b-1 and a≠b+1, then
        v_adjc_p ← VerifProperty(p, adjc_p, "-")
      else
        v_adjc_p ← VerifProperty(p, adjc_p, "+")
      end if
    end if
  end for
 end if
end for
return v_adjc_p
```

Cet algorithme parcoure les catégories des éléments de l'ensemble **Const(p)** à la recherche des deux constituants adjacents distincts $\mathbf{c_x}$ et $\mathbf{c_y}$ de la catégorie **t** de la GP et vérifie que le second n'a jamais arrivé indirectement avant ou après le premier dans le syntagme **p**. Nous avons utilisé le symbole "±" pour caractériser les relations d'adjacence.

3.3.4 Solveur des propriétés d'unicité

Ce solveur vérifie la propriété d'unicité courante de la catégorie syntaxique de la GP. La satisfaction est assurée seulement si le constituant de cette propriété (s'il est trouvé) n'apparait qu'une seule fois dans le syntagme donné. L'algorithme de ce solveur parcoure l'ensemble **Const(p)** à la recherche du constituant de la propriété **unic_p** décrivant la catégorie **t** de la GP et vérifie que sa cardinalité **nb_occ** n'est pas supérieure à 1.

Algorithme 5.4: Interprétation du solveur des propriétés d'unicité

```
Input: Const(p), unic_p, nb_occ ← 0, v_unic_p ← NIL
Output: v_unic_p
for each ca in Const(p), do
  if label(ca)= unic_p.cx, then
    nb_occ ← nb_occ +1
  end if
end for
if nb_occ ≥1, then
  if nb_occ = 1, then
    v_unic_p ← verifProperty(p, unic_p, "+")
  else
    v_unic_p ← verifProperty(p, unic_p, "-")
  end if
end if
return v_unic_p
```

3.3.5 Solveur des propriétés d'obligation

Ce solveur vérifie la propriété d'obligation courante de la catégorie syntaxique de la GP trouvée. La satisfaction est assurée seulement si le constituant obligatoire (tête) de cette catégorie est trouvé dans le syntagme donné.

Algorithme 5.5: Interprétation du solveur des propriétés d'obligation

```
Input: Const(p), oblig_p, fd ← false, v_oblig_p ← NIL
Output: v_oblig_p
for each ca in Const(p), do
  if label(ca)= oblig_p.cx, then
    fd ← true
    break
  end if
end for
if fd = true, then
  v_oblig_p ← verifProperty(p, oblig_p, "+")
else
  v_oblig_p ← verifProperty (p, oblig_p, "-")
end if
return v_oblig_p
```

Cet algorithme parcoure l'ensemble **Const(p)** à la recherche du constituant de **oblig_p** décrivant la catégorie **t** de la GP et vérifie s'il est trouvé. La variable **found** est égale à « true ».

3.3.6 Solveur des propriétés d'exigence

Ce solveur vérifie la propriété d'exigence courante de la catégorie syntaxique de la GP. La satisfaction est assurée seulement si, lorsque le constituant impliquant l'autre dans cette propriété est trouvé dans le syntagme donné, le second est également trouvé.

Algorithme 5.6: Interprétation du solveur des propriétés d'exigence

```
Input: Const(p), fd ← false, exig_p, v_exig_p ← NIL
Output: v_exig_p
for each c_a in Const(p), do
  if label(c_a)= exig_p.c_x, then
    fd ← false
    for each c_b in Const(p), do
      if a≠b and label(c_b)= exig_p.c_y, then
         v_exig_p ← verifProperty(s, exig_p, "+")
         fd ← true
         break
       end if
    end for
  end if
end for
if fd = false then
  v_exig_p ← verifProperty(s, exig_p, "-")
end if
return v_exig_p
```

Cet algorithme parcoure l'ensemble **Const(p)** à la recherche des deux constituants $\mathbf{c_x}$ et $\mathbf{c_y}$ de la catégorie **t** de la GP composant la relation d'exigence et vérifie si le premier est trouvé dans **Const(p)**, alors le second ne doit pas être absent dans **Const(p)**.

3.3.7 Solveur des propriétés d'exclusion

Ce solveur vérifie la propriété d'exclusion courante de la catégorie syntaxique de la GP. La satisfaction est assurée seulement si les deux constituants de cette propriété n'apparaissent pas ensembles dans le syntagme donné. L'algorithme de ce solveur parcoure l'ensemble **Const(p)** à la recherche des deux constituants $\mathbf{c_x}$ et $\mathbf{c_y}$ de la catégorie **t** de la GP composant la relation d'exclusion et marque la comme satisfaite s'ils ne sont pas trouvés ensembles ou seulement l'un des deux apparait dans le syntagme **p**. Par conséquence, dans tous les cas, il y a une propriété vérifiée à retourner. Nous avons utilisé la méthode **search()** pour chercher seulement la position de $\mathbf{c_x}$ dans les catégories des éléments de l'ensemble **Const(p)**.

Algorithme 5.7: Interprétation du solveur des propriétés d'exclusion

```
Input: Const(s), excl_p, verif ← "+" , v_excl_p ← NIL
Output: v_excl_p
a ← search(excl_p.c_x, Const(p))
if a > 0, then
  for each c_b in Const(p), do
     if a≠b and label(c_b)= excl_p.c_y, then
        verif ← "-"
        break
     end if
  end for
v_excl_p ←  verifProperty(s, excl_p, verif)
end if
return v_excl_p
```

3.4 Insertion des propriétés vérifiées dans l'ATB

Le but de cette étape consiste à juste insérer les propriétés vérifiées (satisfaites ou pas) comme caractéristiques décrivant chaque syntagme concerné dans l'ATB (sa catégorie est décrite dans la GP par ces propriétés). Cette insertion est réalisée en utilisant la nouvelle balise « xml » notée <Caracterization> fournissant une combinaison entre l'ATB et la GP. La nouvelle version enrichie de l'ATB est générée selon la DTD donnée par la figure 5.4 suivante :

```
<!ELEMENT Category (Category?, Caracterization?) >
<!ELEMENT Caracterization (Property?) >
<!ATTLIST Caracterization  nb_const CDATA #REQUIRED sat CDATA #REQUIRED >
<!ELEMENT Property EMPTY>
<!ATTLIST Property index CDATA #REQUIRED type CDATA #REQUIRED forme CDATA
#REQUIRED source CDATA #REQUIRED target CDATA #REQUIRED sat CDATA #REQUIRED>
```

Figure 5.4: Modifications de la DTD de l'ATB pour sa version enrichie

Cependant, cet enrichissement augmente remarquablement la taille de la nouvelle version de l'ATB. C'est à cause de l'augmentation exponentielle de la fréquence de la nouvelle balise <Caracterization> avec la fréquence des propriétés décrivant chaque catégorie dans la GP.

4 Enrichissement et évaluation de l'analyseur Stanford Parser par la vérification de la satisfaction des propriétés sur son résultat

Notre démarche d'enrichissement et d'évaluation de Stanford Parser se base sur trois phases principals : l'analyse syntaxique d'un texte arabe brut en utilisant l'analyseur Stanford Parser, l'enrichissement du résultat d'analyse du corpus (CA) avec des propriétés syntaxiques à l'aide d'une GP arabe de référence et l'évaluation du corpus analysé enrichi. La figure 5.5 montre le processus de cette démarche.

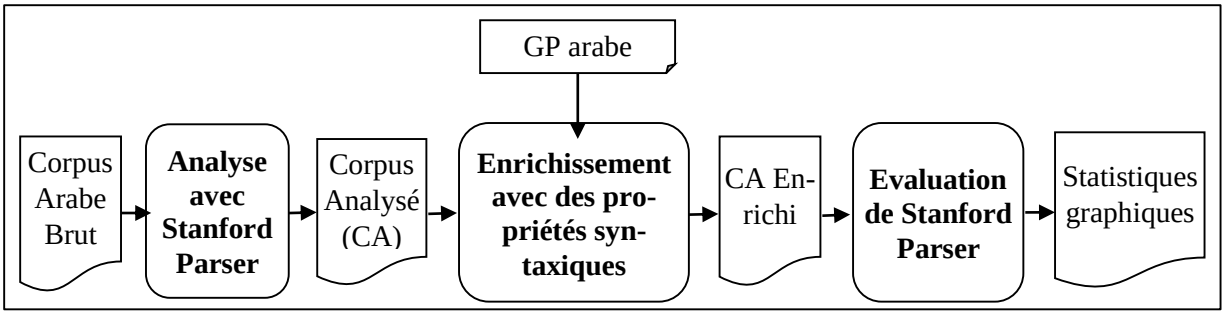


Figure 5.5: Processus d'enrichissement et d'évaluation de l'analyseur Stanford Parser

Nous avons proposé ce processus d'enrichissement et d'évaluation d'analyseurs syntaxiques dans le cadre de notre contribution (Bensalem et al, 2017). Les différntes sous-sections suivantes permettent de détailler une description de chacune des étapes de ce processus qui est illustré par la figure 5.5.

4.1 Analyse avec Stanford Parser

Avant de lancer l'analyseur Stanford Parser, le corpus en entrée doit être segmenté et annoté lexicalement. Les trois sous-étapes suivantes doivent être alors appliquées pour assurer tout l'objectif. Premièrement, l'étape de segmentation permet de diviser le corpus Arabe en une liste de phrases. Ensuite, l'étape d'analyse morphologique : pour séparer les mots agglutinés en tokens puis leurs affecter les catégories lexicales (étiquette POS) convenables. Finalement, l'étape d'analyse syntaxique : pour générer les arbres d'analyses du corpus annoté lexicalement.

4.2 Enrichissement

La méthode d'enrichissement dans cette étape ressemble beaucoup à celle de la section 5.2 sauf qu'il existe les différences suivantes :

- Les entrées à enrichir sont différentes ; l'une porte sur le treebank ATB, l'autre sur le résultat d'analyse de Stanford Parser.
- L'ajustement dans cette étape se limite à la conversion xml du résultat d'analyse et ne varie pas la granularité de ses catégories qui sont à base d'étiquetage PTB.
- La correspondance entre les catégories du résultat d'analyse et celles de la GP nécéssite une table de mapping (correspondance) entre l'étiquetage PTB utilisé par Stanford Parser et l'étiquetage ATB utilisé par la GP. Pour chercher une catégorie du corpus analyse parmi les catégories de la GP, il faut tout d'abord accéder à la table de mapping pour chercher cette catégorie et acquérir son étiquette du système ATB correspondante. Le tableau 5.2 suivant montre quelques exemples de mapping entre les deux étiquetages :

Catégories	Etiquetage PTB	Etiquetage ATB
Verbe au passé	VBD	PV
Nom propre	NNP	NOUN_PROP
Préposition	PRT	PREP
Conjonction	CC	CONJ
Adverbe	RB	ADV
Numéro dans une liste	LS	NOUN_NUM
Adjectif au superlatif	JJS	DET+ADJ_COMP
Adjectif au comparatif	JJR	ADJ_COMP
Pronom possessif	WP$	POSS_PRON
Mot étranger	FW	LATIN

Tableau 5.2 : Exemples de mapping entre les étiquetages PTB/ ATB des catégories lexicales

- Les catégories lexicales de la GP utilisée n'ont pas la granularité la plus élevée comme celles de l'ATB d'origine. Cette réduction de granularité est due à la limitation de traits

spécifiés dans les catégories à base d'étiquetage PTB trouvées dans le résultat d'analyse. Pour expliquer ceci, nous pouvons tirer deux exemples du tableau 5.2. En effet, pour différencier par exemple un adjectif au superlatif par rapport à un autre au comparatif, il suffit de garder pour chaque étiquette « ADJ » dans la GP le trait type ayant la valeur « COMP » et le trait déterminant de valeur « DET ».

Cette méthode d'enrichissement est alors composée des quatre sous-étapes illustrées par la figure 5.6. En effet, après la conversion au format « xml » du corpus analysé (CA) par Stanford Parser, nous lançons une tâche de parcours itératif des sous-étapes suivantes : correspondance de chaque catégorie syntaxique du CA avec celle de la GP, vérification des propriétés de la GP à l'aide des solveurs de contraintes et leur intégration dans le CA pour obtenir une version enrichie.

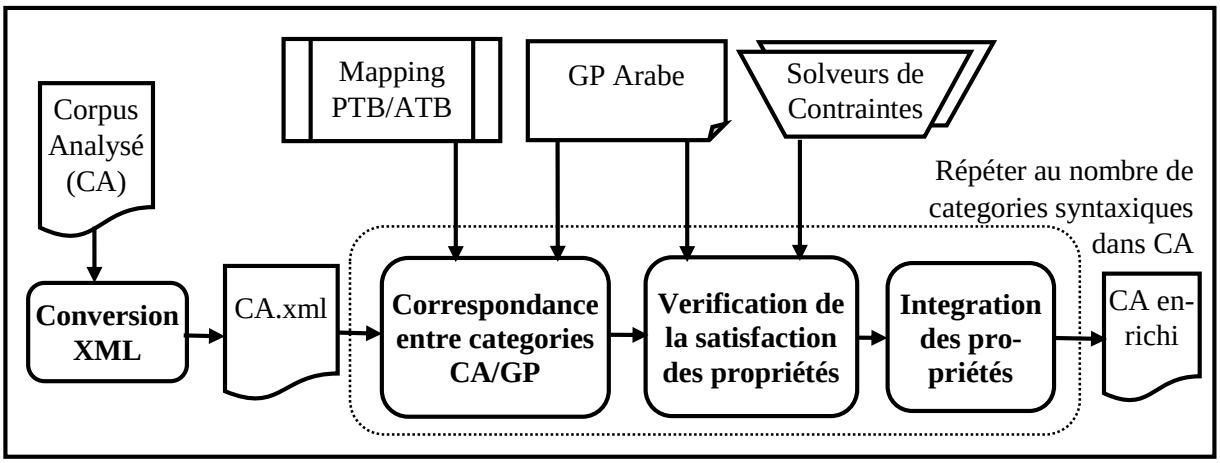


Figure 5.6: Sous-étapes de l'enrichissement du corpus analysé avec des propriétés syntaxiques

Les solveurs de contraintes que nous avons utilisés sont les mêmes adoptés pour l'enrichissement de l'ATB. Nous allons tout simplement expliquer leurs interprétations en termes d'opérations logiques et des exemples arabes significatifs au lieu de redresser leurs algorithmes. Nous allons tout d'abord définir dans ce qui suit quelques notations à utiliser.

L : liste ordonnée de constituants (catégories grammaticales étiquetées avec le système de PTB) formant la structure syntaxique courante dans le CA.
atb_tag(L) : catégorie syntaxique XP spécifiant L selon le système d'étiquetage de l'ATB.
XP : catégorie syntaxique décrite dans la GP avec différents types de propriétés syntaxiques.
$\mathbf{c_i}$ **:** constituant de L à la position i
$\mathbf{L_{i..j}}$ **:** sous-liste de L entre les positions i et j.

|S| : cardinalité de l'ensemble S.

C(XP), O(XP) et **U(XP) :** ensembles de constituants représentant respectivement les propriétés de constituance, d'obligation et d'unicité décrivant XP. Ces propriétés sont des relations unaires.

$\prec$, $\pm$, $\Rightarrow$ et **$\otimes$:** symboles des relations binaires représentant respectivement une propriété de linéarité, d'adjacence, d'exigence et d'exclusion. Chaque relation repose sur deux constituants x et y (x dans la partie gauche (lhs) et y dans la partie droite (rhs)).

Maintenant, nous donnons dans la figure 5.7 suivante une phrase arabe analysée avec Stanford Parser pour montrer l'application de solveurs de contraintes utilisés :

```
<Category index="14" label="S">
 <Category index="14:0" label="VP">
  <Category index="14:0:0" label="VBD" value="انتهى" />
  <Category index="14:0:1" label="NP">
   <Category index="14:0:1:0" label="DTNN" value="اليوم" />
   <Category index="14:0:1:1" label="ADJ_NUM" value="الاول" />
  </Category>
  <Category index="14:0:2" label="PP">
   <Category index="14:0:2:0" label="IN" value="من" />
   <Category index="14:0:2:1" label="NP">
    <Category index="14:0:2:1:0" label="DTNNS" value="المفاوضات" />
    <Category index="14:0:2:1:1" label="DTJJ" value="السورية" />
   </Category>
  </Category>
 </Category>
 <Category index="14:1" label="PUNC" value="." />
</Category>
```

Figure 5.7: Analyse Stanford de la phrase "انتهى اليوم الأول من المفاوضات السورية" (AnthY Alywm Al>wl mn AlmfAwDAt Alswryp /Le premier jour des négociations syrienne est fini)

4.2.1 Solveur des propriétés de constituance

Si ($\forall c_i \in$ L ET atb_tag(L) = XP) alors atb_tag(c_i) $\in$ C(XP)

Pour chaque catégorie c_i dans la structure L étiquetée selon ATB avec XP, l'étiquette ATB de c_i doit appartenir à l'ensemble des constituants de XP dans la GP noté C(XP). Par exemple :

L= DTNNS DTJJ ; **atb_tag(L)**=NP; **atb_tag(c_1)**= DET+NOUN ; **atb_tag(c_2)**= DET+ADJ ;
C(NP)={NOUN, DET+ADJ, ADJ_COMP, DET+NOUN, SBAR, …};
atb_tag(c_1) $\in$ C(NP) and **atb_tag(c_2) $\in$ C(NP)**

Ainsi, les étiquettes selon ATB : DET+NOUN et DET+ADJ relatifs respectivement aux constituants DTNNS et DTJJ de L sont des constituants dans la structure NP. Les propriétés de constituance de NP sont alors satisfaites.

4.2.2 Solveur des propriétés d'obligation

Si L_atb = {atb_tag(c_i) | $\forall c_i \in L$} alors |O(XP) ∩ L_atb| > 0

La cardinalité de l'intersection entre l'ensemble des constituants obligatoires O(XP) et l'ensemble des catégories (étiquetées selon ATB) de la structure L (L_atb) doit dépasser 0. Ainsi, deux constituants (DTNNS et DTJJ) de L sont marqués parmi les constituants obligatoires dans la catégorie syntaxique NP, les propriétés d'obligations de NP sont alors satisfaites.

L= DTNNS DTJJ ; **atb_tag(L)**=NP; **atb_tag(c_1)**= DET+NOUN ; **atb_tag(c_2)**= DET+ADJ ;
L_atb={ DET+NOUN, DET+ADJ}
O(NP)={NOUN, DET+ADJ, ADJ_COMP, DET+NOUN, PRON, CV…};
| **L_atb ∩ O(NP)** | = |{DET+NOUN, DET+ADJ}| = 2

4.2.3 Solveur des propriétés d'unicité

$\forall c_i \in L$, | atb_tag(c_i) ∩ U(XP) | ≤ 1

La cardinalité de l'intersection de tout constituant unique et la liste des constituants de la structure courante du corpus analysé ne doit pas dépasser 1. Par exemple :

L= VBD NP ; **atb_tag(L)**=VP; **atb_tag(c_1)**= PV ; **atb_tag(c_2)**= NP ; **L_atb**={ PV, NP};
U(VP)={NOUN.VN , ADJ.VN, CV, FRAG, PV_PASS,…};
| **atb_tag(c_1)∩ U(VP)** | = 0 ; | **atb_tag(c_2) ∩ U(VP)** | = 0

Ainsi, aucune catégorie unique dans U(VP) n'est trouvée dans L_atb. Alors, aucune propriété d'unicité ne peut être vérifiée satisfaite ou pas.

4.2.4 Solveur des propriétés de linéarité

Pour p : x ≺ y, si x=atb_tag(c_i) ET y=atb_tag(c_j), alors $c_i \in L_{1..k}$ ET $c_j \in L_{k+1..|L|}$

La propriété de linéarité x≺y est satisfaite dans la structure L si x est trouvée dans L à une position i (i ∈ [1,k]) inférieure à la position j de y dans L (j ∈ [k+1,|L|]. Par exemple,

L= VBD NP ; **atb_tag(L)**=VP; **atb_tag(c_1)**= PV ; **atb_tag(c_2)**= NP ; **L_atb** ={ PV, NP};
p: PV ≺ NP **;** PV ∈ $\mathbf{L_{1..1}}$ et NP ∈ $\mathbf{L_{2..2}}$

Ainsi, dans L la position de x=PV est égale à 1 et la position de y=NP est égale à 2, PV apparaît alors dans L avant NP. La propriété p est alors satisfaite pour L et doit être insérée dans le corpus analysé.

4.2.5 Solveur des propriétés d'exigence

Pour p : x ⇒y, si x = atb_tag(c_i) alors y ∈ L_atb

La propriété d'exigence x ⇒y est satisfaite dans la structure L si, lorsque x apparaît dans L, y fait partie aussi de L. Ainsi, aucune catégorie x dans L n'est égale à FRAG. La propriété p ne peut pas être vérifiée satisfaite ou pas dans L. Cette propriété ne sera pas alors intégrée dans le corpus enrichi.

L=DTNN ADJ_NUM ; **atb_tag(L)**=NP; **atb_tag(c_1)**= DET+NOUN; **atb_tag(c_2)**= ADJ_NUM;
L_atb={ DET+NOUN, ADJ_NUM }; **p:** FRAG ⇒NP ;
$\forall c_i \in L$, x ≠ atb_tag(c_i) : FRAG ≠ atb_tag(DTNN) et FRAG ≠ atb_tag(ADJ_NUM)

4.2.6 Solveur des propriétés d'exclusion

Pour p : x ⊗y, si x = atb_tag(c_i) alors y ∉ L_atb

La propriété d'exclusion x ⊗y est satisfaite dans la structure L si losque x est apparu dans L, y n'y est pas, et inversement. Par exemple :

L=DTNN ADJ_NUM ; **atb_tag(L)**=NP; **atb_tag(c_1)**= DET+NOUN; **atb_tag(c_2)**= ADJ_NUM;
L_atb={ DET+NOUN, ADJ_NUM }; **p:** DET+NOUN ⊗ADJ_NUM;
x = atb_tag(c_1) = DET+NOUN ; y = atb_tag(c_2) = ADJ_NUM

Ainsi, La catégorie x est trouvée dans L mais y apparait aussi. La propriété d'exclusion DET+NOUN ⊗ ADJ_NUM sera alors intégrée dans le corpus analysé en tant que propriété non satisfaite.

4.3 Evaluation

Cette étape présente un outil d'évaluation du résultat d'analyse de Stanford Parser. Les scores de cette évaluation sont basés sur les résultats produits par les solveurs de contraintes, déjà décrits dans la section précédente. Le CA enrichi, qui est normalement l'entrée de cette

évaluation, regroupe pour chaque structure syntaxique, les propriétés vérifiées satisfaites ou violées. Les pourcentages des propriétés satisfaites/violées peuvent donner un niveau d'évaluation plus détaillé du résultat d'analyse de Stanford Parser. Nous proposons de calculer ces pourcentages en fonction du nombre d'occurrences de chaque propriété satisfaite et du nombre d'occurrences de la catégorie syntaxique décrite. Nous pouvons alors fournir ces pourcentages par catégorie syntaxique, par type de propriété et par propriété. Nous notons :

p : propriété de type t ∈ T et décrit la catégorie syntaxique c ∈ C.
P : ensemble des propriétés de la GP appartenant au type t.
C : ensemble des catégories syntaxiques de la GP.
T : liste des types de propriétés syntaxiques de la GP.
$sn(p_c^t)$: nombre d'occurrences lorsque la propriété p ∈ P a été satisfaite.
$vn(p_c^t)$: nombre d'occurrences lorsque la propriété p a été violée.
$en(p_c^t)$: nombre d'occurrences total lorsque p peut être vérifiée ($sn(p_c^t) + vn(p_c^t)$).

Après l'indication des notations nécessaires, nous présentons ci-dessous les scores de notre outil d'évaluation de résultat d'analyse :

S_c^t (resp. V_c^t): pourcentage de satisfaction (resp. violation) de propriétés par type t et par catégorie syntaxique c :

$$S_c^t = \frac{\sum_{p \in P} sn(p_c^t)}{\sum_{p \in P} en(p_c^t)} \qquad V_c^t = \frac{\sum_{p \in P} vn(p_c^t)}{\sum_{p \in P} en(p_c^t)} \qquad (1)$$

S_c (resp. V_c): pourcentage de satisfaction (resp. violation) de propriétés par catégorie syntaxique c :

$$S_c = \frac{\sum_{t \in T} \sum_{p \in P} sn(p_c^t)}{\sum_{t \in T} \sum_{p \in P} en(p_c^t)} \qquad V_c = \frac{\sum_{t \in T} \sum_{p \in P} vn(p_c^t)}{\sum_{t \in T} \sum_{p \in P} en(p_c^t)} \qquad (2)$$

S^t (resp. V^t): pourcentage de satisfaction (resp. violation) de propriétés par type t :

$$S^t = \frac{\sum_{c \in C} \sum_{p \in P} sn(p_c^t)}{\sum_{c \in C} \sum_{p \in P} en(p_c^t)} \qquad V^t = \frac{\sum_{c \in C} \sum_{p \in P} vn(p_c^t)}{\sum_{c \in C} \sum_{p \in P} en(p_c^t)} \qquad (3)$$

$sp_c^t(p)$ (resp. $vp_c^t(p)$): distribution de la satisfaction (resp. violation) de la propriété p du type t, et qui décrit la catégorie syntaxique c :

$$sp_c^t(p) = \frac{sn(p_c^t)}{\sum_{p \in P} sn(p_c^t)} \qquad vp_c^t(p) = \frac{vn(p_c^t)}{\sum_{p \in P} vn(p_c^t)} \qquad (4)$$

Les différentes formules mentionnées ci-dessus se basent seulement sur les fréquences des propriétés enrichissant le résultat d'analyse. L'évaluation de ce résultat se limite sur le niveau de satisfaction de ces propriétés dans les syntagmes de ce résultat et non pas sur la structuration de ces syntagmes.

5 Conclusion

Le présent chapitre fournit deux démarches pour la construction de nouvelles ressources linguistiques en se basant sur une GP. Ces démarches se focalisent essentiellement sur l'enrichissement de ressources d'origine (un treebank et un résultat d'analyse syntaxique) avec des représentations à base de propriétés syntaxiques. Cet enrichissement permet d'obtenir de nouvelles ressources plus informatives. Son apport pour les résultats d'analyse (dans notre cas, celui de Stanford Parser) s'étend pour offrir une méthode d'évaluation détaillée de ce résultat qui définit, grâce à un ensemble de scores, les relations syntaxiques qui n'ont pas été respectées. Le principe d'utilisation de ressources linguistiques pré-créées pour générer d'autres peut être réappliqué sur les ressources nouvelles pour acquérir par exemple des lexiques syntaxiques ou encore des grammaires de dépendances. Les statistiques sur les catégories et les propriétés peuvent être bénéfiques surtout dans le développement d'analyseurs syntaxiques probabilistes. Cela participe également à l'allégement du processus d'analyse syntaxique, vu qu'il est possible de se limiter aux propriétés les plus pertinentes ou fréquentes tout en relâchant d'autres violées mais rares. L'analyse syntaxique probabilité fera alors l'objet d'un autre domaine d'utilisation des GP que nous allons détailler dans le chapitre suivant.

Chapitre 6

Construction d’un analyseur syntaxique stochastique de propriétés syntaxiques

1 Introduction

Le formalisme de grammaires de propriétés a servi dans le chapitre précédent à enrichir des corpus déjà annotés à savoir les treebanks et les résultats d'analyse de Stanford Parser. Dans ce chapitre, nous proposons de construire un analyseur syntaxique probabiliste de propriétés pour montrer l'efficacité de ce formalisme sur des corpus non annotés. Comme mentionné dans le chapitre 1, nous rappelons que l'analyse syntaxique statistique de laquelle notre analyseur fait partie, se base sur les deux composants principaux suivants :

- Le modèle d'apprentissage qui peut prendre la forme d'une grammaire ou d'un modèle de classification construits à partir d'un corpus d'apprentissage. Ce modèle est majoré d'un ensemble de statistiques estimées de ce corpus.
- L'algorithme d'analyse selon lequel un corpus de test est analysé en se référant au modèle d'apprentissage déjà construit. Le corpus de test doit être prétraité pour qu'il devienne segmenté et annoté lexicalement avant d'appliquer l'analyse syntaxique.

Le présent chapitre est consacré à l'explication de notre approche d'analyse syntaxique probabiliste de propriétés. Il est organisé comme suit : la section 2 illustre l'architecture de notre analyseur. Les deux sections suivantes expliquent respectivement le déroulement de ses deux composants principaux. La dernière section termine le chapitre par une conclusion.

2 Architecture de l'analyseur syntaxique probabiliste de propriétés

A la manière des analyseurs syntaxiques probabilistes, le nôtre utilise deux composants : le modèle d'apprentissage et l'algorithme d'analyse. Notre modèle d'apprentissage produit des règles de production que l'algorithme d'analyse va s'en servir pour générer ses analyses syntaxiques. L'apprentissage doit être dans ce cas supervisé. C'est-à-dire que le corpus d'apprentissage sera un texte annoté syntaxiquement comme les treebanks. Nous rappelons également que l'analyse syntaxique générée par l'algorithme d'analyse doit être riche de propriétés de différents types. Pour nous, outre les règles de production, les propriétés sont également des facteurs essentiels qui peuvent influencer les choix d'analyse effectués par l'algorithme d'analyse. Les règles de production obtenues de la CFG ainsi que les propriétés obtenues de la GP doivent alors faire partie du modèle d'apprentissage. Pour mesurer l'importance de ces informations, nous avons leur estimé des probabilités en fonction de leur nombre d'occurrences dans le corpus d'apprentissage. Grâce à l'introduction de ces probabilités, et comme le montre l'architecture de notre analyseur dans la figure 6.1, les

composants de notre modèle d'apprentissage seront une CFG probabiliste (PCFG) et une GP probabiliste (PGP). Dans cette architecture, l'algorithme d'analyse que nous avons choisi est l'algorithme CYK. Tandis que l'algorithme de Viterbi représente l'algorithme d'optimisation des analyses construites. Les raisons de choix de ces algorithmes sont définies respectivement dans leurs sous-sections spécifiques. Le corpus de test subit un prétraitement avec l'outil SAMA lui permettant une analyse morphologique et un étiquetage POS avant d'être utilisé par l'algorithme d'analyse CYK.

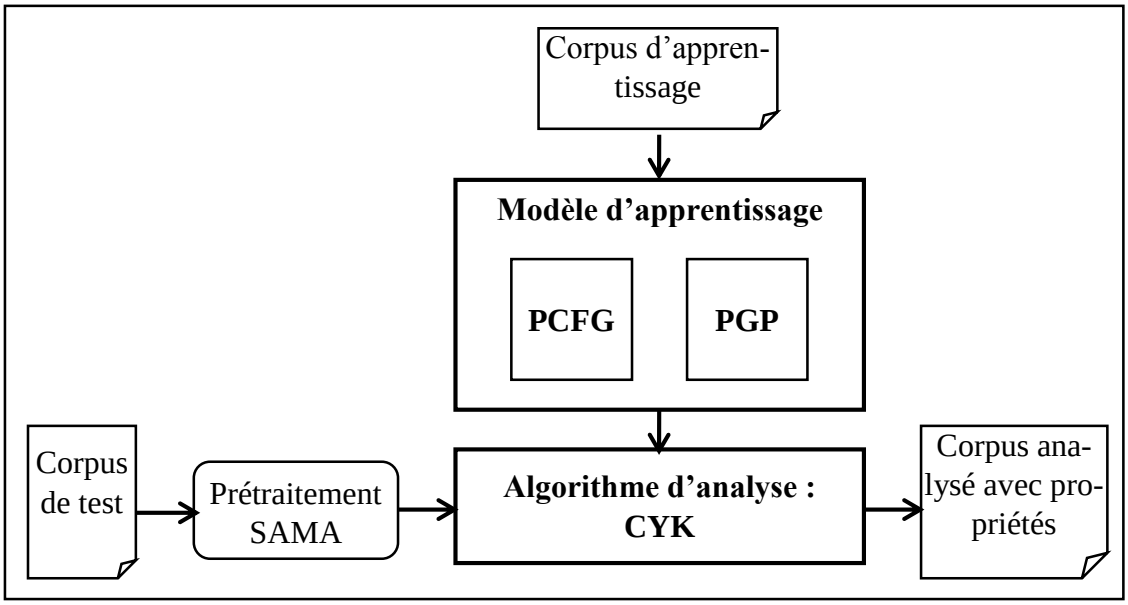


Figure 6.1: Architecture générale de notre analyseur syntaxique probabiliste de propriétés

3 Construction du modèle d'apprentissage

Comme nous l'avons déjà mentionné dans la section précédente, le modèle d'apprentissage se compose de la PCFG et de la PGP. Pour obtenir ces deux ressources, il faut établir les étapes indiquées dans ce qui suit.

3.1 Préparation des données d'apprentissage

Vu que notre objectif consiste à utiliser les données du corpus annoté pour l'analyse syntaxique, il nous paraît qu'il est préférable de bénéficier de toutes les informations fournies dans les étiquettes des catégories grammaticales. Pour cela, nous allons garder la granularité maximale de ces catégories sans aucun contrôle sur leurs traits.

Plus particulièrement, nous avons utilisé le treebank arabe ATB comme corpus d'apprentissage. Nous avons mentionné dans le chapitre 4 que ce corpus est fourni sous 5 formats : Les formats qui nous intéressent le plus sont le format PTB et le format « pos ».

Le premier format regroupe les données présentant les phrases brutes de l'ATB sous forme d'arbres d'analyse syntaxique où tous les mots sont segmentés en tokens étiquetés.

Ces arbres sont fournis sous forme parenthésée sur plusieurs articles de presse. Chaque article contient plusieurs lignes où chaque ligne donne l'analyse d'un paragraphe. Chaque paragraphe peut représenter les arbres d'analyse de plusieurs phrases.

L'analyse de ces données débute par la citation de la liste des catégories syntaxiques et lexicales pour chaque ligne d'un article. Ces données sont considérées principales vu qu'elles sont valables d'une part à l'extraction des règles de production de la PCFG, et d'autre part à l'extraction de la relation de dépendance Head par le système de (Habash et al., 2009). L'algorithme qui montre le remplissage des listes de catégories syntaxiques et lexicales d'une certaine ligne est dressé ci-dessous :

Algorithme 6.1: Remplissage des listes de catégories lexicales et syntaxiques

```
Input :
ligne, categ : String,
mots : Tableau de String,
l_tree : L_Tree
Output: ltree: L_Tree
mots ← ligne.split(" ")
i, a, b : entier
a ← 1
b ← 1
for i ← 1 to length(mots) − 1, do
   categ ← mots[i].substring(2,length(mots[i]))
   if mots[i + 1][1] != '(', then //une categorie lexicale
      categs_lex[a] ← categ
      a ← a+1
      i ← i+1
   else //une categorie syntaxique
      categs_syn[b] ← categ
      b ← b+1
   end if
end for
ltree ← new_L_Tree(ligne, categs_syn, categs_lex)
```

Les données secondaires sont fournies par le deuxième format (« pos »). Elles sont exposées sous forme d'un ensemble de listes de champs de caractéristiques caractérisant chacune un token dans l'arbre d'analyse. L'information qui nous intéresse parmi ces champs

est l'écriture arabe du token. Cette information est importante lors de l'analyse d'un corpus de test arabe. En effet, comme tout corpus brut, les mots qu'il contient sont écrits en arabe. Toutefois, ces mots ne peuvent pas être observés par les règles de la grammaire car cette dernière est apprise sur le format « penntree » qui fournit les mots selon leur écriture translitérée.

Notre solution consiste alors à enrichir chaque ligne d'un article de presse dans le format « penntree » par la liste des mots arabes correspondants aux mots écrits sous forme translitérée. Ces nouvelles informations seront utilisées par la suite pour remplacer dans les règles de production tout mot translitéré par son écriture arabe. Il s'agit en fait d'associer cette liste de mots arabes à chaque ligne dans un article de presse donné du format « penntree » en utilisant l'article de presse correspondant dans le format « pos ». L'algorithme de cette solution est donné ci-dessous.

Algorithme 6.2: Association des écritures arabes aux mots de chaque phrase

```
Input :
article_pos : Tableau de String,
article_penntree : Tableau de L_Tree,
l_tree : L_Tree,
l_pos, champs : String,
i, j, x : entier
Output : article_penntree : tableau
i ← 1
j ← 1
champ ← " INPUT STRING: "
l_tree ← article_penntree[i]
for x ← 2 to length(article_pos), do
  l_pos ← article_pos[x]
  if l_pos.startsWith(champ), then
     l_tree.mots_arabes[j] ← l_pos.extraire_information(champ)
     j ← j+1
     if j = length(l_tree.categories_lex)+1, then
        i ← i+1
        j ← 1
        if i < = length(article_penntree), then
          l_tree ← article_penntree[i]
        else
          x ← length(article_pos)+1
        end if
     end if
  end if
end for
```

Comme le montre l'algorithme ci-dessus, d'un point de vue simlificatif, nous avons considéré que les deux articles aux formats PTB et « pos » sont considérés comme des tableaux de lignes. Chaque ligne de l'article au format PTB, noté article_penntree, est une structure de plusieurs informations, appelée L_Tree. Elle est caractérisée par exemple par la liste de

catégories lexicales (notée categories_lex), la liste des mots arabes (notée mots_arabes) qui leurs correspondent. Tandis que les lignes de l'article au format « pos », noté article_pos, sont des chaines de caractères (ou String). Dans article_pos, toute ligne commençant par le champ « INPUT STRING: » contient l'écriture arabe qui d'un mot du corpus. Le remplissage de la liste de mots arabes associée à une ligne d'un article_penntree consiste à chercher ce champ pour extraire l'information qui suit, qui est bien évidemment le mot écrit en arabe.

3.1.1 Extraction des relations Heads de l'ATB

Cette étape prend en entrée le corpus ATB pour produire les relations Heads principaux spécifiques aux catégories syntaxiques trouvées dans chaque ligne d'un article de presse du format PTB. L'explication du déroulement de cette étape a été déjà mentionnée dans le chapitre 4 avec un algorithme bien déterminé lors de la création d'une GP à granularité variable. Nous allons alors nous contenter ici de donner une idée sur ses différentes sous-étapes.

3.1.2 Délimitation des catégories syntaxiques de l'ATB

Cette première sous-étape consiste à enrichir la liste de catégories syntaxiques de chaque ligne dans un article de presse de format PTB, qui a été déjà créée, par les limites (catégories et indices de début de fin) de leurs structures syntaxiques.

3.1.3 Exécution du programme d'extraction des relations de Habash

Cette deuxième sous-étape bénéficie de la sortie du système d'extraction de relations de dépendances proposé par Habash et al. (2009) pour générer leur treebank CATiB. Ce système prend en entrée le corpus à traiter et un fichier composé de règles d'extraction de relations et de correspondances entre les étiquetages de l'ATB et de CATiB. Nous rappelons que ce système peut générer plusieurs types de relations comme le Head, le sujet, l'objet, le prédicat et le modifieur. Toutefois, nous n'allons exploiter dans cette thèse que la relation Head pour générer les propriétés d'obligation. Ceci va se faire en associant les Heads principaux aux catégories syntaxiques listées pour chaque ligne annotée dans un article de presse. Cette liste est notée head_categs_syn.

3.2 Induction de la PCFG lexicalisée approfondie sous CNF

La PCFG que nous allons induire est spéciale car elle possède les caractéristiques suivantes :

- Lexicalisée : où les catégories syntaxiques sont étiquetées à la fois par leur étiquettes POS et leur constituant Head le plus profond dans l'arbre d'analyse. Ainsi la catégorie VP prend la forme VP [V] car le Head de sa structure syntaxique est la catégorie V.
- Approfondie : où les parties droites des règles ne sont pas définies seulement en fonction des nœuds descendants directs des parties gauches, mais aussi en fonction des suites de catégories lexicales descendantes indirectement de ces parties gauches.
- Formée selon la norme de Chomsky CNF (Chomsky Normal Form) (Chomsky, 1959) : où les parties droites des règles ne peuvent prendre que deux formes possibles : un mot ou exactement deux constituants.

Les algorithmes implémentant ces caractéristiques sont donnés dans les sous-secion suivantes :

3.2.1 Lexicalisation des données d'apprentissage

Pour obtenir une PCFG lexicalisée, il faut transformer les catégories syntaxiques de chaque ligne du corpus d'entrée en leur associant leurs Head. Cette transformation se fait ainsi :

Algorithme 6.3: Lexicalisation de l'ATB

```
Input : l_tree : L_Tree, mots : Tableau de String,
i, c : entier
Output : l_tree : L_Tree
mots ← l_tree.ligne.split(" ")
c ← 0
for i ← 1 to length(mots) – 1, do
   if mots[i][1] = '(' and mots[i + 1][1] = '(', then
      mots[i] ← mots[i] + "[" + l_tree.head_categs_syn[c] + "]"
      c ← c+1
  end if
end for
//Construire la ligne lexicalisée
l_tree.ligne_lex ← mots[1]
for i ← 2 to length(mots), do
    l_tree.ligne_lex ← l_tree.ligne_lex + " " + mots[i]
end for
```

3.2.2 Suppression des catégories lexicales –NONE- relatives aux mots vides

Vu que les règles, décrivant la catégorie lexicale –NONE-, portent sur des mots vides, elles ne peuvent pas être perçues par l'algorithme d'analyse car ces mots n'existent pas dans le corpus de test. Pour ignorer ces règles, il faut supprimer du corpus d'apprentissage toute catégorie étiquetée –NONE- avec le mot vide qui la suit. Cette suppression s'étend bien

évidemment pour supprimer par recurrence les catégories syntaxiques qui les encapsulent, comme le montre l'algorithme suivant :

Algorithme 6.4: Suppression des catégories lexicales -NONE- relatives aux mots vides

```
Input :
ligne : String,
i, j, pos, nb_fermantes : entier
reste : boolean
Output : ligne : String
for i ← 1 to length(mots), do
   if mots[i].contains("-NONE-") = false, then
      ligne ← ligne +" " + mots[i]
   else //ignorer la catégorie syntaxique encapsulant "-NONE-"
      ligne ← ligne + " ("
      if mots[i]= "(-NONE-", then
         nb_fermantes ← 0
         for j ← 1 to length(mots[i + 1]), do
            if mots[i + 1][j] = ')', then
               nb_fermantes ← nb_fermantes + 1
            end if
         end for
         //ajouter les parenthèses fermantes après le mot vide
         for j ← 1 to nb_fermantes, do
            ligne ← ligne + ")"
         end for
         i ← i+1 //ignorer le mot vide qui suit "-NONE-"
      end if
   end if
end for
//supprimer toute parenthèse ouvrante et fermante vide
ligne ← ligne.substring(1)//pour enlever le premier " "
ligne ← ligne.replace("()", "")
pos ← 0
reste ← true
repeat
   ligne ← ligne.replace(") )", "))")
   ligne ← ligne.replace("  ", " ")
   pos ← ligne.chercher("( )")
   if pos = 0, then
      reste ← false
   else
      ligne ← ligne.replace("( )", "")
   end if
   ligne ← ligne.replace("  ", " ")
until(reste = false)
l ← l+1
```

3.2.3 Détection des constituants

Cette sous-étape permet de détecter les listes sans doublons des constituants de type « Categorie » à savoir : la liste des catégories lexicales (notée categs_lex) et celle des catégories syntaxiques lexicalisées (notée categs_syn). Ce type associe à une catégorie son nombre

d'occurrences (noté occ) dans le corpus. Voici l'algorithme réalisant ceci pour une ligne penntree donnée.

Algorithme 6.5: Détection des catégories lexicales et syntaxiques de l'ATB lexicalisé

```
Input :
ligne, categ, head, categ_head : String
mots : Tableau de String
find, pos1, pos2 : entier
Output : categorie : Categorie
mots ← ligne.split(" ")
for i ← 1 to length(mots), do
    categ_head ← mots[i].substring(2, mots[i].length())
    head ← ""
    categ ← categ_head
    find ← 0
    pos1 ← categ_head.chercher("[")
    pos2 ← categ_head.chercher("]")
    if pos1 > 0 and pos2 > 0, then //categorie syntaxique
       head ← categ_head.substring(pos1 + 1, pos2)
       categ ← categ_head.substring(1, pos1)
       find ← chercher_dans_categs_syn(categ_head)
       if find > 0, then
          categs_syn[find].occ ← categs_syn[find].occ + 1
       else
          categorie ← construire_categ_syn(categ,head,categ_head)
          categorie.occ ← 1
          categs_syn.add(categorie)
          total_occ_categs_syn ← total_occ_categs_syn + 1
       end if
    else //categorie_lexicale
       find ← chercher_dans_categs_lex(categ)
       if find > 0, then
          categs_lex[find].occ ← categs_lex[find].occ + 1
       else
          categorie ← construire_categ_lex(categ,head,categ_head)
          categorie.occ ← 1
          categs_lex.add(categorie)
          total_occ_categs_lex ← total_occ_categs_lex + 1
       i ← i+1
       end if
    end if
end for
```

Les listes de catégories syntaxiques et lexicales ont été ensuite triées avec un tri à bulles croissant car elles font partie de la structure de la PGP.

3.2.4 Elaboration des règles de production

Comme déjà indiqué dans le chapitre 1, les règles de productions forment la CFG. Ces règles se basent sur une partie gauche (lhs) portant sur la catégorie à décrire et une partie droite (rhs) portant sur la dérivation directe de cette catégorie à une suite de catégories grammaticales.

Une analyse approfondie du corpus d'apprentissage au format PTB nous permet d'obtenir la suite de catégories lexicales sources d'une telle dérivation. La methode récursive ci_dessous permet la génération de règles de production approfondies pour une catégorie au format « xml ».

Algorithme 6.6: Génération des règles de production

```
List generate_rules_recursifs(categ_tag :Balise, categ : Fils, suite_categs_lex :
Liste de String, regles : Liste de Regle, l_tree : L_Tree, i_lecture : entier,
nombre_mots : entier)
Begin
Input :
rhs : Liste de String,
fils_tag : liste de Balise
fils : liste de Fils
sous_suite_categs_lex : Liste de String
i, pc, pr, ps : entier
regle : Regle
if categ_tag.value = NIL, then
   i_lecture ← i_lecture + 1
   // ignorer les categories -NONE- dans head_categs_syn
   while (l_tree.head_categs_syn[i_lecture] = "-NONE-"), do
      i_lecture ← i_lecture + 1
   end while
   n_head ← l_tree.n_head_categs_syn[i_lecture]
   fils_tag ← categ_tag.listChildren()//balises sous categ_tag
   for i ← 1 to length(fils_tag), do
      pc ← chercher_dans_categs_syn(fils_tag[i].label)
      if pc >= 1, then
         fils.add(i, new_Fils(pc,fils_tag,true)// si syntaxique
      else
         pc ← chercher_dans_categs_lex(fils_tag[i].label)
         fils.add(i, new_Fils(pc,fils_tag, false)
      end if
      fils[i].n_premier_mot ← nombre_mots
      sous_suite_categs_lex ← NIL
      generate_rules_recursifs(fils_tag[i], fils[i], sous_suite_categs_lex, l_tree,
i_lecture, nombre_mots)
      fils[i].suite_categs_lex.add(sous_suite_categs_lex)
      suite_categs_lex.add_elements(sous_suite_categs_lex)
      fils[i].n_dernier_mot ← nombre_mots
   end for
   total_occ_rules_syn ← total_occ_rules_syn + 1
   pr ← chercher_dans_liste_regles(true)
   if pr > 0, then
      ps ← chercher_dans_liste_suites(pr)
      if ps > 0, then
         regles[pr].occ_suites[ps] ← regles[pr].occ_suites[ps]+1
      else
         regles[pr].add_suite(ps, suite_categs_lex)
         regles[pr].add_detail_per_child(ps, fils)
         regles[pos_regle].occ_suites.add(pos_suite, 1)
      end if
      regles[pr].occ ← regles[pr].occ + 1
   else
      regle ← new_Regle(categ,fils,suite_categs_lex)
     regle.occ ← 1
     regles.add(regle)
     regle.determiner_fils_head(n_head)
else
traiter_categorie_lexicale(categ_atg, categ, suite_categs_lex , nombre_mots, regles)
end if, l_tree
return suite_categs_lex
End
```

La fonction traiter_categorie_lexicale() quant ’à elle s’occupe principalement de l’incrémentation du nombre de mots dans la phrase analysée courante et de l’ajout d’une règle lexicale dont le rhs est le mot relié à cette catégorie lexicale.

Algorithme 6.7: Ajout d'une règle lexicale

```
Traiter_categorie_lexicale(categ_tag :Balise, categ : Fils, suite_categs_lex :
Liste de String, regles : Liste de Regle, l_tree : L_Tree, nombre_mots : entier)
Begin
Input :
mot_fils : String
fils : liste de Fils
pr : entier
regle : Regle
   suite_categs_lex.add(categ_tag.label)
   nombre_mots ← nombre_mots + 1
   total_occ_rules_lex ← total_occ_rules_lex + 1
   pr ← chercher_dans_liste_regles(false)
   if pr > 0, then
      regles[pr].occ ← regles[pr].occ + 1
   else
      mot_fils ← l_tree.mots_arabes[nombre_mots]
      fils.add(pr, new_String(mot_fils)
      regle ← new_Regle(categ, fils)
      regle.occ ← 1
      regles.add(pr, regle)
   end if
End
```

3.2.5 Conversion de la PCFG à la forme CNF

Cette conversion consiste à transformer toutes règles en seulement deux formes possibles. Ainsi, et comme déjà indiqué dans le chapitre 1, nous n’avons que les règles à rhs unique X → m ou bien les règles à rhs binaire X → Y Z avec X, Y et Z appartiennent aux listes de catégories syntaxiques lexicalisées et lexicales, et m appartient à l’ensemble des mots dans le corpus. La sortie sera une PCFG selon la forme CNF, que nous appelons PCFG-CNF. Les règles de la PCFG qui respectent la forme CNF, sont directement copiées dans PCFG-CNF. C’est l’exemple des deux cas de règles suivants :

- Les règles dont le lhs est une catégorie lexicale et le rhs est un mot
- Les règles dont le rhs est une suite de deux catégories lexicales

Les autres règles ont besoin d’être reformulées ou même ignorées dans la PCFG-CNF selon plusieurs cas. Ces cas seront traités un par un dans ce qui suit. Les nombre d’occurrences marqués pour ces règles seront par conséquent mis à jour.

- **Cas des règles à rhs basé sur une seule catégorie grammaticale :** Il suffit tout simplement d’ignorer ces règles.

- **Cas des règles à rhs de taille supérieure à 1 :** On regroupe ici deux sous-cas de règles : les règles à rhs composé de deux catégories grammaticales dont l'une au moins est une catégorie syntaxique, et les règles à rhs composé de plus que deux catégories grammaticales. Les deux sous-cas de règles doivent subir un balayage des catégories de leur rhs pour remplacer toute catégorie syntaxique basée sur une seule catégorie lexicale par cette catégorie lexicale. Cela nous permet de montrer l'effet des règles du cas précèdent qui ont été ignorées et de garder la liaison avec les règles dont le lhs est une catégorie lexicale. Une catégorie syntaxique à remplacer peut être décrite par différentes catégories lexicales. Ceci nécessite la transformation de la règle d'origine qui l'encapsule en d'autres règles spécifiques remplaçant chacune cette catégorie syntaxique par une catégorie lexicale bien déterminée. Le nombre d'occurrences de la règle d'origine sera distribué aux règles spécifiques en fonction du nombre d'occurrences de leur catégorie lexicale remplaçante. Ce traitement est réalisé grâce à l'algorithme ci-dessous pour une règle donnée à rhs de taille supérieure à 1 :

Algorithme 6.8: Traitement des règles basées sur une seule catégorie grammaticale

```
Input : regle : Regle, regles_cnf : Liste de Regle_CNF, regle_cnf_gn,
regle_cnf_sp : Regle_CNF, modif : boolean
Output : regles_cnf : Liste de Regle_CNF
regle_cnf_gn ← new(regle,regle.lhs, regle.rhs, true, false)
regles_cnf.add(regle_cnf_gn)
regle_cnf_gn.occ ← regle.occ
regle_cnf_gn.suite_categs_lex ← regle.suite_categs_lex
regle_cnf_gn.fils ← regle.fils
for i ← 1 to length(regle.suite_categs_lex), do
 modif ← false
 for j ← 1 to length(regle.fils), do
  if regle.fils[j].tag <> regle.fils[j].suite_categs_lex[i] and
length(regle.fils[j].suite_categs_lex[i]) = 1, then
    regle.new_fils[j].tag ← regle.fils[j].suite_categs_lex[i]
    modif ← true
  end if
 end for
 if modif = true, then //il y a eu un remplacement
  regle_cnf_sp ← new(regle,regle.lhs,regle.new_fils,true,true)
  regles_cnf.add(regle_cnf_sp)
  regle_cnf_sp.occ ← regle.occ_suites[i]
  regle_cnf_sp.suite_categs_lex.add(regle.suite_categs_lex[i]
  regle_cnf_gn.suite_categs_lex.del(regle.suite_categs_lex[i])
  regle_cnf_sp.deplacer_details(regle_gn.fils, i)
  regle_cnf_gen.occ ← regle_cnf_gen.occ - regle_cnf_spec.occ
 end if
end for
if length(regle_cnf_gn.suite_categs_lex) = 0, then
 regles_cnf.del(regle_cnf_gn)
```

Le deuxième sous-cas de règles nécéssite encore un traitement spécifique. Il s'agit de la reformulation des règles à rhs basé sur plus que deux catégories grammaticales. Cette reformulation consiste à transformer une règle de ce type à une suite de règles en CNF, où chaque règle prend la forme X → Y Z telle que X fait référence à la catégorie Z de la règle précédente. L'algorithme suivant montre cette transformation pour une règle donnée.

Algorithme 6.9: Traitement des règles basées sur plus que deux catégories grammaticales

```
Input : rhs_ac : Tableau de String, pos, ajout : Entier, lhs_ac, categ_temp :
String, regle_cnf Regle_CNF, regles_cnf : Liste de Regle_CNF
Output : regles_cnf : Liste de Regle_CNF
for h ← 3 to max_rhs_length, do
  for r ← 1 to length(regles_cnf), do
   if length(regles_cnf[r].rhs) = h, then
    lhs_ac ← regle_cnf.lhs //lhs de la règle actuelle
    rhs_ac [0] ← regle_cnf.rhs[0]
    categ_temp ← regle_cnf.lhs+"("+regle_cnf.rhs[0]
    rhs_ac[1] ← categ_temp +")"
    gerer_nouvelle_categ() //ajout si nouvelle et gerer les occ
    regle_cnf.lhs ← lhs_ac
    regle_cnf.rhs ← rhs_ac
    ajout ← 2
    while ajout < length(regles_cnf[r].rhs), do
     lhs_ac ← rhs_ac[1]
     rhs_ac[0] ← regles_cnf[r].rhs[ajout]
     if ajout = length(regles_cnf[r].rhs) - 1, then
      rhs_ac[1] ← regles_cnf[r].rhs[ajout+1]
     else
      categ_temp ← categ_temp +","+rhs_regle[ajout]
      rhs_ac[1] ← categ_temp +")"
      gerer_nouvelle_categ()
     end if
     regle_cnf_sp←new(regle_cnf.regR,lhs_ac,rhs_ac,false,true)
     regle_cnf_sp.occ ← regle_cnf.occ
     //determiner la règle précèdente d'où vient X
     if length(regle_cnf.regA) = 0, then
      regle_cnf_sp.regX ← regle_cnf
     else
      regle_cnf_sp.regX←regle_cnf.regA[length(regle_cnf.regA)]
     end if
     //ajouter regle_cnf_sp aux regles attachées de regle_cnf
     regle_cnf.regA.add(regle_cnf_sp)
     regles_cnf.add(regle_cnf_sp)
     ajout ← ajout +1
    end while
   end if
  end for
end for
```

Les règles ont été ensuite triées par catégorie syntaxique. Vu que la PCFG est PCFG-CNF probabiliste, nous leur avons associées des probabilités calculées en fonction du nombre d'occurrences de chaque règle sous forme CNF divisé par le nombre d'occurrences total. La PCFG-CNF est générée sous le format xml tout en montrant les différentes fréquences, probabilités et nombres d'occurrences des catégories grammaticales, des règles et des suites de catégories lexicales décrivant chaque catégorie syntaxique.

3.3 Induction de la PGP lexicalisée sous CNF

Comme pour la PGP à granularité variable proposée dans le chapitre 4, la PGP lexicalisée sous CNF, notée PGP-CNF, se base sur l'extraction de différents types de propriétés syntaxiques décrivant chaque catégorie syntaxique. Toutes les interprétations algorithmiques des propriétés ressemblent à celles présentées dans le chapitre 4. Seulement l'interprétation des propriétés d'obligation nécessite un prétraitement spécifique pour déterminer les Heads des règles à rhs basé sur plus que deux catégories grammaticales. Une règle de ce type est décomposée en une suite de règles spécifiques dans la PCFG-CNF. Il faut alors déterminer les heads de ces règles pour pouvoir déduire directement les propriétés d'obligation caractérisant les différentes catégories syntaxiques. La détermination de ces Heads se fait selon plusieurs cas que nous traitons dans la méthode ci-dessous pour les règles d'une catégorie syntaxique donnée

Algorithme 6.10: Détermination des Heads dans les règles de forme CNF

```
Input :
i, j, i_head : entier, xp : Categorie,
rules_cnf : Tableau de Liste de Regle_CNF
Output : rules_cnf : Tableau de Liste de Regle_CNF
i ← xp.indice
j ← 0
while j <= length(rules_cnf[i]) and find = false  , do
// cas 1 : originale et pas lexicale
  if rules_cnf[i][j].syn=true and rules_cnf[i][j].orig=true, then
     i_head ← rules_cnf[i][j].regR.n_head //indice du fils Head
     //cas 1.1 : length(rules_cnf[i][j].regR.fils)= 2
     if length(rules_cnf[i][j].regA) = 0, then //
        rules_cnf[i][j].head ← rules_cnf[i][j].fils[i_head].tag
     else //cas 1.2 : length(rules_cnf[i][j].regR.fils) > 2
        if i_head = 1, then
           rules_cnf[i][j].head ← rules_cnf[i][j].fils[1].tag
        else
           rules_cnf[i][j].head ← rules_cnf[i][j].fils[2].tag
        end if
        //determiner les heads des règles attachees
        for k ← 0 to length(rules_cnf[i][j].regA), do
           rules_cnf[i][j].head ← rules_cnf[i][j].fils[1].tag
        end for
     end if
  end if
  j ← j +1
end while
```

La PGP intègre différentes probabilités calculées en fonction des nombre d'occurrences des propriétés décrivant une même catégorie syntaxique. Ainsi, la probabilité d'une propriété $prop_{cti}$ d'un type t caractèrisant une catégorie syntaxique c est égale à :

$$p(prop_{cti}) = \frac{occ(prop_{cti})}{\sum_{c \in C} \sum_{i \in I} occ(prop_{cti})}$$

Nous rappelons que le but de notre analyse consiste à générer le résultat optimal d'analyse syntaxique de propriétés. Nous allons alors exploiter les règles de la PCFG-CNF, ainsi que les propriétés de la PGP-CNF. Nous proposons de calculer une fonction de probabilité associée à chaque règle, qui dépend à la fois de sa probabilité et des probabilités des propriétés considérées satisfaites dans cette règle. Cette fonction sert à mesurer l'importance de la règle à laquelle elle est associée par rapport aux autres. Cette fonction peut être calculée pour une règle r_k ainsi :

$$fp\ (r_k) = \sum_{t \in T} \sum_{i \in I} w_t * \ p(prop_{cti})\ tel\ que\ lhs(r_k) = c$$

Les poids des probabilités syntaxiques notés □□ sont des valeurs numériques choisies par l'utilisateur ou bien estimées grâce à une fonction d'ajustement. La PGP résultante est présentée sous le format xml associant les différents nombres d'occurrences et probabilités aux catégories syntaxiques et à leurs constituants et propriétés.

4 Application de l'algorithme d'analyse syntaxique : CYK

Le choix de l'application de l'algorithme d'analyse syntaxique CYK est dû à l'ensemble de qualités de ce dernier conformes à nos besoins d'analyse à savoir :

- La rapidité de traitement : puisqu'il charge toutes les règles dans une table pour une seule fois, à la différence de l'algorithme d'Earley qui ne charge que les règles relatives aux catégories qu'il en a besoin durant une certaine étape.
- L'insensibilité aux règles récursives : C'est lorsque le rhs d'une règle contient une catégorie syntaxique identique à son lhs.
- La simplicité de parcours : vu qu'on n'utilise q'une liste de règles d'une manière séquentielle sans passer par des opérations de lecture, de prédiction et de complétion de l'algorithme Earley.

Avant de dresser l'algorithme d'analyse de l'algorithme CYK, il faut effectuer une étape de prétraitement du corpus de test et de la reconnaissance de ses différents constituants.

4.1 Prétraitement des données de test

Le prétraitement des données du corpus de test porte sur deux grandes tâches : l'analyse et la désambiguïsation morphologique. Il exite plusieurs outils s'occupant de ceci comme BAMA,

SAMA, MADAMIRA et Alkhalil Morpho Sys 2. L'outil SAMA a montré de meilleurs résultats, il est déjà exploité dans l'annotation lexicale du treebank ATB. Vu qu'il est payant, nous avons choisi de bénéficier du corpus de base de l'ATB enrichi avec cette annotation lexicale. Ce corpus n'est pas fourni directement mais plutôt sous forme d'une liste de mots avant et après segmentation (séparation des agglutinations). Chaque mot est caractérisé par plusieurs types d'informations comme la catégorie lexicale (étiquette POS), l'écriture voyéllée et la traduction anglaise du mot. C'est le format « pos » qui offre ces informations.

Notre idée consiste à transformer la liste de mots segmentés et annotés lexicalement de l'ATB sous format « pos » en une liste de phrases représentant notre corpus de test. Les mots dans ce format sont indexés par le paragraphe auquel il appartient ainsi que leur rang dans ce paragraphe. Notre premier but consiste à segmenter chaque paragraphe en une suite de phrases. Le caractère de séparation est normalement un signe de ponctuation comme le point (.), le point d'interrogation (?) et le point d'exclamation (!).

Algorithme 6.11: Génération du corpus de test segmenté et analysé lexicalement

```
Input : article_pos : Liste de String, x : Entier
champ_mot, champ_categ, l_pos, couple, mot, categ : String
fin_phrase, fin_couple : boolean
Output : ls_Test : Liste de L_Test, l_Test : L_Test
champ_mot ← " INPUT STRING: "
champ_categ ← "  POS: "
for x ← 1 to length(article_pos), do
  l_pos ← article_pos[x]
  if l_pos <> "", then
    if l_pos.startsWith(champ_mot), then
      mot ← l_pos.extraire_information(champ_mot)
      fin_phrase ← verifier_si_ponctuation(mot)
      couple ← "("+mot+","
    end if
    if l_pos.startsWith(champ_categ), then
      categ ← l_pos.extraire_information(champ_categ)
      couple ← couple + categ+") "
      fin_couple ← true
    end if
    if fin_couple = true, then
      l_Test ← l_Test + couple
      repeat
         x ← x + 1
      until (l_pos = "")
    end if
    if fin_phrase = true, then
      ls_Test.add(l_Test)
      l_Test ← ""
    end if
  end if
end for
```

4.2 Extraction des constituants

Etant donné le corpus de test qui contient une liste de phrases dont les mots sont segmentés et annotés lexicalement, il faut commencer par extraire ces mots ainsi que leur catégorie lexicale. Cela se fait avec une division imbriquée qui décompose chaque ligne (représentant une phrase) en un couple de mot et sa catégorie lexicale, puis chaque couple en ses deux constituants. L'algorithme illustré ci-dessous montre cette division pour une ligne donnée du corpus de test.

Algorithme 6.12: Extraction des constituants des phrases de test

```
Input : ligne : String, div, sous_div : Tableau de String,
couplesMC : Liste de CoupleMC
Output : l_test : L_Test
//Exemple d'une ligne de test : (m1,c1) (m2,c2) (m3,c3)...
div ← ligne.split(" ")
for i ← 1 to length(div), do
   sous_div ← div[i].split(",")
   coupleMC ← new_Couple_MC(i,div[i],sous_div[1], sous_div[2])
end for
l_test ← newL_Test(ligne, couplesMC)
```

4.3 Analyse avec l'algorithme CYK

L'analyse syntaxique de propriétés avec l'algorithme CYK est réalisée en fonction du corpus de test et du modèle d'apprentissage qui est composé de la PCFG-CNF et de la PGP-CNF. Le premier composant contient les règles à utiliser par l'algorithme CYK. Tandis que le deuxième contient les propriétés qui caractérisent ces règles. Comme indiqué dans le chapitre 1, l'application de l'algorithme CYK nécessite tout d'abord le remplissage d'une charte spécifique à chaque ligne de test avec toutes les règles possibles à utiliser. Ces règles sont regroupées en fonction des suites de mots à analyser dans la ligne de test. La stratégie d'analyse de cet algorithme est ascendante, ce qui fait que les suites de mots sont traitées d'une manière incrémentale en commençant par celles composées d'un seul mot et en terminant par celle regroupant tous les mots de la ligne de test. Le remplissage de la charte suit le même ordre. En effet, les règles couvrant les suites de mots les plus petites sont les premières à déterminer et ainsi de suite. Voici l'algorithme qui permet ce remplissage :

Algorithme 6.13: Remplissage de la charte de règles de l'algorithme CYK

```
Input : l_Test : L_Test, r_min : Regle_CNF,
regles_cnf, temp_L : Liste de Regle_CNF,
i, j, k, y, z, r, h : Entier,
f_y, f_z : boolean
Output : chart : Tableau[][] de Regle_CNF
for i ← 1 to length(l_Test.mots), do
  for j ← 1 to length(l_Test.mots), do
   chart[i][j]=new_List()
  end for
end for
for i ← 1 to length(l_Test.mots), do
  for r ← 1 to length(regles_cnf), do
   if regles_cnf[r].lhs= l_Test.categs[i] and
regles_cnf[r].fils[1].tag=l_Test.mots[i] and regles_cnf[r].syn=false, then
    chart[i][i].add(regles_cnf[r])
   end if
  end for
  //cas : aucune règle trouvée
  if chart[i][i].isEmpty(), then
   temp_L ← charger_regles(l_Test.categs[i])
   r_min ← chercher_regle_a_probabilite_minimale(temp_L)
   if r_min <> NIL, then
    chart[i][i].add(r_min)
   else
    temp_L ← NIL
    temp_L ← charger_regles(extraire_base(l_Test.categs[i]))
    r_min ← chercher_regle_a_probabilite_minimale(temp_L)
    chart[i][i].add(r_min)
   end if
  end if
end for
for h ← 2 to length(l_Test.mots), do// niveau dans la chart
  for i ← 1 to (length(l_Test.mots)-h), do
   j ← i+h
   for k ← i to j, do//indice séparateur entre rhs de X->Y Z
    for r ← 1 to length(regles_cnf), do
     f_y ← false
     y ← 1
     while y<=length(chart[i][k]) and f_y = false, do
      if regles_cnf[r].fils[1].tag = chart[i][k][y].lhs, then
       f_y ← true
      end if
     end while
     f_z ← false
     z ← 1
     while z<=length(chart[k+1][j]) and f_z = false, do
      if chart[k+1][j][z].orig = false, then
       //si chart[k+1][j][z] est attachée à regles_cnf[r]
       if regles_cnf[r] = chart[k+1][j][z].regX, then
        f_z ← true
       end if
      else
       if regles_cnf[r].fils[2].tag=chart[k+1][j][z].lhs, then
        f_y ← true
       end if
      end if
     end while
     if f_y =true and f_z =true, then
      chart[i][j].add(regles_cnf[r])
     end if
    end for
   end for
  end for
end for
```

Avec la charte remplie grâce à l'algorithme précédent, l'analyse d'une ligne de test donnée est réussie seulement si on arrive à trouver les règles de la chart couvrant tous les mots de cette ligne. Dans ce cas, un nombre assez important d'analyses syntaxiques de propriétés peut être généré. L'opération de sélection de la meilleure analyse se fait grâce à un algorithme d'optimisation. C'est l'algorithme de Viterbi que nous allons appliquer pour ceci. Cet algorithme permet de choisir à chaque étape et pour chaque catégorie syntaxique la meilleure analyse descendante d'elle. Cela nous permet de bénéficier de l'apport qui désigne l'algorithme de recherche locale qui est la rapidité et d'éviter à la fois son défaut qui est le risque de tomber dans un optimum local. Au cours de la construction d'une telle analyse, le choix d'une certaine sous-analyse par rapport à une autre dépend du calcul de la fonction de probabilités des règles participant à cette sous-analyse sachant les choix antérieurs. Le modèle d'apprentissage utilisé ici est un modèle génératif à base d'historique. Ainsi, à une certaine étape d'analyse, la décision de choix de la règle X → Y Z notée d_a et couvant la suite de mots ($m_i..m_j$) dépend de la probabilité conditionnelle de cette règle sachant les meilleures probabilités des règles précédemment choisies. Elle est calculée ainsi :

$$p(d_a|d_1 d_2 \dots d_{a-1}) = p(d_a) \times p(d_{a-1}|d_1 d_2 \dots d_{a-2})$$

$$= p_{[i,j]}(X \to Y\ Z) \times \max(Y, i, k) \times \max(Z, k+1, j)$$

Le but de l'algorithme d'optimisation est de maximiser la probabilité des règles couvrant la suite ($m_i..m_j$). C'est pour cette raison que les probabilités des règles précédemment choisies sont définies par la fonction max dans la formule ci-dessus. La probabilité d'une règle d_a est définie par la fonction de probabilité des règles mentionnée dans la section 3.4 de ce chapitre majorée d'un paramètre de conformité pc :

$$p(d_i) = fp(d_i) + pc$$

$$pc = \begin{cases} \dfrac{occ\left(suite_s(X \to Y\ Z)\right)}{occ(X \to Y\ Z)} & si\ (c_i \dots c_j) = suite_s(X \to Y\ Z) \\ 0\ sinon \end{cases}$$

Comme le montre la formule ci-dessus, le paramètre pc est positif si la suite de catégories lexicales de la ligne de test ($c_i..c_j$), qui est couverte par la sous-analyse, est identique à une certaine suite de catégories lexicales caractérisant la règle choisie. Dans ce cas, il est égal à la part de cette suite par rapport à toutes les suites de la même règle.

L'algorithme suivant calcule les probabilités conditionnelles de toutes les règles de la chart :

Algorithme 6.14: Algorithme de Viterbi appliqué sur la chart de l'algorithme CYK

```
Input :
chart : Tableau[][] de Regle_CNF
categories : Tableau de Categorie
i, j, k, c, h, s : Entier, p_max, pc : reel
l_Test : L_Test
Output :
choix_max : Tableau[][][] de Choix{pr : Reel, reg : Regle_CNF, sp : Entier}
for i ← 1 to length(chart), do
  for j ← 1 to length(chart), do
    for t ← 1 to length(categories), do
      choix_max[i][j][t]= new_Choix(0, NIL, 0)
    end for
  end for
end for
for i ← 1 to length(chart), do
  for c ← 1 to length(chart[i][i]), do
    p_max ← 0
    choix ←
    if chart[i][i][c].fp > p_max, then
      choix_max[i][i][chart[i][i][c].indice].reg ← chart[i][i][c]
      choix.pr ← chart[i][i][c].fp
    end if
  end for
end for
for h ← 2 to length(chart), do// niveau dans la chart
  for i ← 1 to (length(chart)-h), do
    j ← i+h
    for k ← i to j, do//indice séparateur entre rhs de X->Y Z
      for c ← 1 to length(chart[i][j]), do
        pc ← 0
        s ← chercher_pos_suite(l_Test.categs)
        if s > 0, then
          pc ← chart[i][j][c].suite_categs_lex[s].occ/chart[i][j][c].occ
        end if
        p_choix ← (chart[i][j][c].fp + pc) *
choix_max[i][k][chart[i][j][c].fils[1].indice].p_max *
choix_max[k+1][j][chart[i][j][c].fils[2].indice].p_max
      if choix_max[i][j][chart[i][j][c].indice].fp <p_choix, then
        choix_max[i][i][chart[i][i][c].indice].reg ← chart[i][j][c]
        choix_max[i][i][chart[i][i][c].indice].pr ← chart[i][j][c].fp
        choix_max[i][i][chart[i][i][c].indice].sp ← k
      end if
    end for
  end for
 end for
end for
end for
```

5 Conclusion

Dans le présent chapitre, nous avons proposé une méthode d'analyse syntaxique probabiliste de propriétés. Pour cela, nous avons utilisé un modèle d'apprentissage génératif à

base d'historiques conforme à l'algorithme d'analyse CYK. Cet algorithme est combiné avec l'algorithme de Viterbi grâce aux probabilités fournies par le modèle d'apprentissage. Ce modèle a été construit à partir du corpus d'entrée ATB, et il a subi plusieurs traitements pour le préparer pour la tâche d'analyse d'un corpus de test. Nous citons par exemple : l'analyse approfondie, la lexicalisation et la conversion au format CNF.

Pour concrétiser notre méthode d'analyse, nous l'avons implémenté et abouti à un ensemble d'expérimentations à illustrer et à discuter dans le chapitre 9. Nous mentionnons que cette méthode d'analyse n'est pas encore supportée par une publication.

Troisième Partie

Réalisation et évaluation

Chapitre 7

Expérimentation et évaluation de la grammaire de propriétés induite à partir de l'ATB

1 Introduction

Toutes les présentations fournies jusqu'à maintenant dans le présent rapport restent encore dans le cadre théorique. Pour pouvoir tester la faisabilité des différentes études linguistiques réalisées et évaluer la rentabilité des démarches proposées dans la résolution des problèmes en question, il faut concrétiser ces démarches par le passage à un niveau d'analyse beaucoup plus profond. Il s'agit en fait d'aboutir à la tâche d'implémentation des différents algorithmes proposés par ces démarches, puis à leur expérimentation pour en tirer les interprétations nécessaires. Le présent chapitre commence par la présentation des expérimentations obtenues de l'application de la démarche d'induction de la GP à granularité variable qui a été décrite dans le chapitre 4. Nous rappelons que nous avons supporté cette démarche par les deux contributions (Bensalem et Elkaroui, 2014) et (Bensalem et al., 2014). L'illustration des expérimentations sur cette démarche sera dans la section 3 pour les entrées de notre démarche et dans la section 4 pour ses sorties. Dans cette section, un jeu de la granularité des catégories est également appliqué pour déterminer son effet sur la validité des propriétés induites, sur le degré de précision des catégories et sur les tailles des sorties.

2 Caractéristiques des entrées

Pour cette section et pour toutes les autres sections du chapitre, nous allons utiliser quelques symboles caractérisant quelques entêtes dans les différentes tables. Les significations de ces symboles sont illustrées dans le tableau 7.1 ci-dessous :

Symboles	**Significations**	**Symboles**	**Significations**
XP	Catégorie syntaxique	Const	Ensemble des propriétés de constituance
#	Nombre d'occurrences	Unic	Ensemble des propriétés d'unicité
∑	Total	Oblig	Ensemble des propriétés d'obligation
#C	Cardinal de constituants possibles	Lin	Ensemble des propriétés de linéarité
#R	Cardinal de règles de production	Exig	Ensemble des propriétés d'exigence
#P	Cardinal de propriétés	Excl	Ensemble des propriétés d'exclusion

Tableau 7.1 : Significations des symboles des en-têtes des tables

Rappelons tout d'abord du déroulement de la démarche d'induction de la GP : l'ATB est en premier lieu ajusté en fonction du niveau de granularité exigé par l'utilisateur. Ensuite, la CFG est induite de l'ATB ajusté. Finalement, la GP est induite de la CFG obtenue. La

détermination des propriétés d'obligation, en particulier, nécessite l'application d'un ensemble de règles élaborées par des ressources externes (Habash et al., 2009). Dans ce cas, l'ATB ainsi que ces règles représentent les entrées de cette démarche. Les sous-sections suivantes sont consacrées à la présentation de leurs différentes caractéristiques.

2.1 Treebank ATB

Nous avons testé notre analyseur sur la version 3.1 de la division 2 de l'ATB (notée ATB2v1.3) qui inclut 501 articles de l'agence de presse UmmahArabic News, contenant 144.199 segments avant la fragmentation des clitiques. Les deux formats de l'ATB que nous avons besoin d'utiliser sont le format PTB et le format « pos ». Comme nous l'avons déjà mentionné dans le chapitre Le premier format prend la forme d'arbres syntaxiques des phrases de l'ATB où les mots sont voyellés et écrit sous forme translitérée. Nous avons adopté ce format pour exécuter les règles de (Habash et al, 2009) pour générer les relations Head. Le tableau 7.2 ci-dessous donne quelques statistiques sur les catégories grammaticales trouvées dans la version voyéllée de l'ATB sous le format PTB.

	#C	**#**
Catégories syntaxiques	461	189711
Catégories lexicales	380	184251
∑	**841**	**373962**

Tableau 7.2 : Nombres d'occurrences des catégories grammaticales dans l'ATB

Nous pouvons remarquer à travers le nombre d'occurrences important de catégories syntaxiques dans l'ATB que les catégories lexicales sont regroupées à l'intérieur de plusieurs constructions imbriquées. Nous avons constaté ceci à cause de l'abondance du nombre de catégories lexicales (228) contre environ la moitié pour les catégories syntaxiques (132). Ce rapport est anormalement inversé au niveau du nombre d'occurrences au profit des catégories syntaxiques. Pour avoir plus de détails sur ces catégories, nous pouvons montrer leurs répartitions dans l'ATB à travers la figure 7.1 suivante :

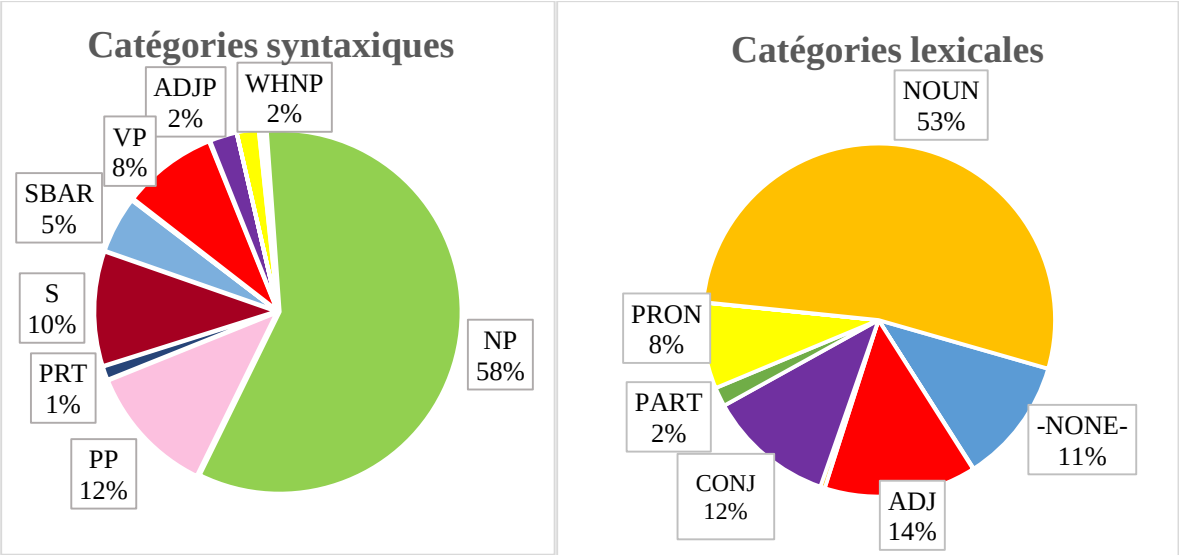


Figure 7.1: Répartitions des principales catégories syntaxiques et lexicales dans le corpus ATB

Selon les répartitions des catégories grammaticales illustrées par la figure 7.1 ci-dessus, il est remarquable que l'ATB est très riche en syntagmes nominaux étiquetés par la catégorie NP. Cette catégorie occupe plus que la moitié de l'ensemble des catégories syntaxiques. La catégorie lexicale NOUN n'occupe pas cette part parmi les catégories lexicale ce qui nous permet de constater qu'elle n'est pas un constituant obligatoire dans toutes les constructions du NP. Tandis que la catégorie syntaxique PP et son constituant habituel PREP ont presque la même importance dans leurs répartitions. Nous avons remarqué également que l'ATB utilise un nombre assez important de la catégorie –NONE- qui représente les pronoms cachés. Ces pronoms sont des mots ayant une existence virtuelle pour garder un certain ordre de présentation des mots d'une structure donnée.

2.2 Règles de (Habash et al., 2009)

Les règles de (Habash et al., 2009) nous permettent à générer plusieurs relations de dépendances entre les mots du corpus ATB. La relation qui intéresse dans le présent travail est la relation Head. Cette relation participe d'une part à la lexicalisation du corpus ATB sous son format PTB et d'autre part à l'induction des propriétés d'obligation décrivant chaque catégorie syntaxique dans le cadre de la PGP. Après avoir montré la correspondance (mapping en anglais) entre l'étiquetage de l'ATB et celui du PTB, Habash et al (2009) ont fourni un ensemble de règles d'extraction de relations de dépendances à partir d'exemples de constructions syntaxiques. Ils ont distingué trois types de règles : les règles principales (notées M1), les règles

étendues ajoutées une seule fois (notées A1) et les règles étendues ajoutées plusieurs fois (notées An). Le tableau 7.3 ci-dessous donne les fréquences des règles et des relations pouvant être extraites.

Types de règles	#	Types de relations								
		HEAD	**SBJ**	**OBJ**	**PRD**	**MOD**	**IDF**	**TMZ**	**TPC**	**Σ**
M1	223	**223**	23	94	10	139	18	10	1	518
A1	26	**26**	3	2	5	12	6	1	1	56
An	24	**24**	1	2	0	21	0	0	0	27
Σ	273	**273**	27	98	15	142	24	11	2	601

Tableau 7.3: Fréquences des règles et des relations à extraire dans (Habash et al., 2009)

Selon le tableau 7.3 ci-dessus, les relations Head sont définies à chaque règle élaborée.

3 Caractéristiques des sorties

Comme nous l'avons déjà mentionné dans section précédente, l'induction de la GP n'est pas une tâche directe, mais elle nécessite, avant d'obtenir la GP, la génération des sorties intermédiaires suivantes : l'ATB ajusté en fonction du niveau de granularité des catégories choisi et la CFG. Les résultats des expérimentations de ces trois sorties seront discutés respectivement dans les trois sous sections suivantes :

3.1 Treebank ATB ajusté

Grâce à la variation de la granularité des variables, plusieurs versions ajustées de l'ATB peuvent être générées. Ainsi le tableau 7.4 ci-dessous montre les fréquences des constituants de chaque catégorie syntaxique en leur fixant respectivement la granularité minimale et maximale.

XP	#	Min	Max	XP	#	Min	Max	XP	#	Min	Max
NP	110748	**299**	**2361**	**PRT**	2292	13	14	**PRN**	65	10	15
PP	22100	**22**	**248**	**ADVP**	539	6	69	**LST**	56	2	2
S	19358	138	266	**NAC**	221	18	53	**SQ**	51	12	19
VP	15947	342	439	**FRAG**	178	22	29	**CONJP**	37	3	3
SBAR	9524	**47**	**382**	**WHADVP**	136	3	34	**INTJ**	11	1	1
WHNP	4574	**3**	**66**	**UCP**	132	19	64	**X**	5	4	6
ADJP	3665	88	191	**SBARQ**	68	19	35	**WHPP**	3	3	6
									∑	**841**	**4303**

Tableau 7.4: Fréquences des constituants par catégorie dans leurs granularités maximales et maximales

En observant le tableau 7.4 ci-dessus, nous pouvons remarquer qu'il existe une grande différence en nombre de constituants entre la granularité minimale et celle maximale. Encore, plus les catégories encapsulant ces constituants sont abondantes dans le corpus ATB, plus il existe des variantes de leurs constituants dans leur granularité maximale. C'est le cas notamment des constituants des catégories NP, PP, SBAR et WHNP. Le nombre de traits enrichissant ces constituants est élevé. Pour expliquer cette idée, nous avons choisi de montrer dans le tableau 7.5, ci-dessous, la variation du choix des traits dans deux constituants de la catégorie NP, à savoir : les noms et les adjectifs. Ils sont étiquetés dans l'ATB respectivement par les catégories lexicales NOUN et ADJ.

Ligne	Det	Type	Def	Gre	Nbre	Cas	Poss	#C
1	0	0	0	0	0	0	0	**601**
2	0	0	1	0	0	0	0	**605**
3	1	0	1	0	0	0	0	613
4	0	1	0	1	1	0	0	613
5	1	0	1	0	0	1	1	625
6	0	1	0	1	1	1	1	637
7	1	1	1	1	1	1	1	**841**

Tableau 7.5: Fréquences des constituants des catégories grammaticales de la variation du choix des traits des catégories NOUN et ADJ

Dans le tableau 7.5 ci-dessus, le jeu de variation aléatoire de la granularité des catégories NOUN et ADJ montre que le nombre de variantes de ces catégories peut perdre plus que son un-quart en enlevant ses traits morphologiques (841 contre 601). Cette perte est également perçue dans la ligne 6 lorsqu'on ignore les deux traits Déterminant (Det) et Définition (Def). Le premier trait est le plus important car dans la ligne 2, l'intégration du deuxième trait Def dans la structure de base de la catégorie n'a apporté que 4 variantes supplémentaires (605 contre 601). Dans la section suivante, nous montrons l'effet de ce jeu de variation de la granularité sur le nombre de règle de la CFG induite de l'ATB ajusté.

3.2 CFG

La CFG représente la deuxième sortie dans notre démarche. Comme nous l'avons mentionné dans le chapitre 4, elle est composée d'ensembles des règles de production décrivant les catégories syntaxiques. Ces règles se basent sur des suites de catégories grammaticales dérivant une catégorie syntaxique donnée. Alors, tout comme les catégories, la CFG est affectée par la variation de la granularité de ces catégories grammaticales. Le tableau 7.6 ci-dessous

montre les fréquences des règles de production décrivant chaque catégorie syntaxique à granularités minimales et maximales.

XP	#	Min	Max	XP	#	Min	Max	XP	#	Min	Max
NP	**110748**	**538**	**4824**	**PRT**	**2292**	**4**	**14**	**PRN**	65	16	20
PP	**22100**	**59**	**263**	**ADVP**	539	7	68	**LST**	56	2	2
S	**19358**	**723**	**1230**	**NAC**	221	18	53	**SQ**	51	25	26
VP	**15947**	**1056**	**6675**	**FRAG**	178	51	56	**CONJP**	37	2	2
SBAR	**9524**	**88**	**380**	**WHADVP**	136	2	34	**INTJ**	11	1	1
WHNP	**4574**	**6**	**64**	**UCP**	132	64	88	**X**	5	5	5
ADJP	3665	91	593	**SBARQ**	68	29	51	**WHPP**	3	2	3
									∑	**2789**	**14452**

Tableau 7.6: Fréquences des règles dans leurs granularités minimales et maximales

Le tableau 7.6 montre que les syntagmes verbaux (VP) représentent la catégorie la plus riche en règles de production bien qu'elle n'est pas la plus fréquente. Ce sont les syntagmes nominaux (NP) qui apparaissent dans l'ATB presque 7 fois le nombre d'occurrence des VP. La fréquence de leurs règles représente presque la moitié dans le cas de la granularité minimale des catégories (538 contre 1056). Cet écart diminue dans le cas de la granularité maximale (4824 contre 6675). Cette idée s'applique également pour les propositions subordonnées (SBAR), les syntagmes prépositionnels (PP), les syntagmes de particules (PRT), les phrases (S) et les syntagmes WHNP. En appliquant une vue générale, le nombre de règles est divisé par 7 en choisissant d'appliquer une granularité minimale. Ceci indique l'importance des traits qui enrichissent la structure des catégories. Nous montrons dans le tableau 7.7 ci-dessous, le changement du nombre de règles avec la variation du choix de traits que nous avons appliqué pour les adjectifs et les noms.

Ligne	Det	Type	Def	Gre	Nbre	Cas	Poss	#R
1	0	0	0	0	0	0	0	11410
2	0	0	1	0	0	0	0	**12002**
3	1	0	1	0	0	0	0	**12829**
4	0	1	0	1	1	0	0	**12268**
5	1	0	1	0	0	1	1	12849
6	0	1	0	1	1	1	1	12628
7	1	1	1	1	1	1	1	14452

Tableau 7.7: Fréquences des règles de production avec la variation du choix des traits des catégories NOUN et ADJ

Comme pour les fréquences des constituants encapsulés dans une catégorie syntaxique, la présence du trait Det permet de les obtenir en plusieurs variantes et par conséquent en plusieurs variantes de règles. Ceci est observé à travers l'augmentation du nombre 12829 dans la ligne 3 par rapport à 12002 dans la ligne 2. Même les différences entre les lignes 3 et 4 montrent que l'absence de ce trait a un grand effet sur le changement de la taille de la CFG et par conséquent de celle de la GP. En effet, plus le niveau de granularité des catégories est élevé, plus ces grammaires sont complexes et volumineuses mais leurs propriétés sont de plus en plus fidèles à la langue, et inversement. En effet, la règle "PP → PREP NP" qui décrit les PP s'applique à toutes ses variantes. La diminution de la granularité de ses PP ne va pas toucher à la pertinence de la CFG. Tandis que la règle "PP → PUNC PREP SBAR PUNC", elle ne concerne que quelques variantes (e.g. PP-TMP et PP-PRP) mais pas les autres (e.g. PP-TPC, PP-MNR et PP-PRD). La généralisation de cette règle à toutes les PP dans le cas du choix de la granularité minimale entraîne un manque de précision dans la CFG et par conséquent dans la GP. La fidélité de ces grammaires aux données de l'ATB d'origine est alors à discuter.

3.3 GP

La GP qui est induite de la CFG peut hériter les différences de fréquences en variant leur granularité. Le tableau 7.8 ci-dessous nous montre ces fréquences pour les propriétés aux granularités minimales et maximales.

XP	#	Min	Max	XP	#	Min	Max	XP	#	Min	Max
NP	1960734	395	122446	**PRT**	11460	14	119	**PRN**	708	58	94
PP	299848	138	1755	**ADVP**	3831	45	163	**LST**	308	6	6
S	201533	211	19038	**NAC**	4039	70	355	**SQ**	718	84	222
VP	338242	377	95120	**FRAG**	4115	155	435	**CONJP**	116	9	9
SBAR	125338	169	4569	**WHADVP**	408	5	88	**INTJ**	22	2	2
WHNP	22583	19	210	**UCP**	2651	98	463	**X**	34	16	25
ADJP	36966	146	6532	**SBARQ**	1176	85	606	**WHPP**	18	9	18
									∑	**2111**	**252275**

Tableau 7.8: Fréquences des propriétés par catégorie dans leurs granularités maximales et maximales

Le tableau 7.8 ci-dessus, montre des différences des catégories pivot indiqués pour la CFG. En effet, la VP n'est plus la catégorie la plus fréquente comme avec les règles de production.

Le NP récupère sa place aussi bien pour la granularité minimale que maximale en termes du nombre de propriétés. Il est décrit par environ la moitié des propriétés. Globalement la variation de la granularité affecte largement la taille de la GP. Son nombre de propriétés pour une granularité maximale se multiplie par plus que 100. Ceci ne concerne pas bien entendu le PP qui est malgré son grand nombre d'occurrences dans l'ATB, il se caractérise par un nombre de propriétés relativement faible. Ceci revient à l'écart assez faible entre le nombre de ses règles de productions à leur granularité minimale par rapport celles à la granularité maximale (59 contre 263). Ceci s'explique par le fait que la plupart des règles de la PP partagent pratiquement les mêmes structures syntaxiques. Pour cette catégorie, nous illustrons respectivement, dans les figures 7.2 et 7.3, des extraits des GP aux granularités maximales et minimales.

PP-PRP-PRD	Const	{PREP, NP, SBAR}
	Unic	{**PREP, NP**, SBAR}
	Oblig	{PREP}
	Lin	{**PREP ≺ NP**, PREP ≺ SBAR}
	Exig	{**NP ⇒ PREP**, SBAR ⇒ PREP}
	Excl	{**NP ⊗ SBAR**}

Figure 7.2: Extrait de la GP à granularité maximale (description de la catégorie PP-PRP-PRD)

La figure 7.2 montre différents types de propriétés décrivant la catégorie PP-PRP-PRD. Ainsi Les propriétés d'unicité (Unic) considèrent que tous les constituants sont uniques dans les règles de productions décrivant PP-PRP-PRD. La propriété d'obligation (Oblig) indique que la catégorie PREP est le seul constituant obligatoire. Les propriétés d'exigence (Exig) et de linéarité (Lin) montrent que les catégories NP et SBAR nécessitent la présence de PREP mais ne peuvent pas la précéder. Les propriétés d'exclusion, quant à elles, indiquent que ces catégories ne peuvent pas apparaître ensemble. Ces propriétés sont induites de deux règles de production, l'une dérive une PREP suivie d'un NP, l'autre dérive une PREP suivie d'un SBAR.

Dans la figure 7.3, qui montre un extrait de la GP avec la granularité minimale, la catégorie PP-PRP-PRD est regroupée au sein d'une seule catégorie syntaxique qui est PP. Dans ce cas, plusieurs propriétés sont généralisées ou encore ignorées. Nous citons alors les nouveaux changements que subit cette catégorie. Premièrement, les constituants PREP et NP ne sont plus uniques dans leurs règles de production. Deuxièmement, la propriété de linéarité "PREP ≺ NP"

est également ignorée. Troisièmement, la propriété d'exigence " NP ⇒ PREP " n'est plus mentionnée, donc NP a été trouvé sans PREP dans des règles de production d'autres variantes de PP. Ceci est pareil pour la propriété d'exclusion entre NP et SBAR. Les propriétés qui ont été gardées sont l'obligation de PREP et la précédence linéaire de PREP par rapport à SBAR, outre les propriétés de constituance qui sont évidemment gardées. Bien que la catégorie originale PP-PRP-PRD n'apparaisse que cinq fois dans l'ATB (soit 0,002%), la diminution du nombre de propriétés pour plusieurs catégories non fréquentes dans le but de les généraliser peut entrainer un manque de précision remarquable.

PP	Const	{-NONE-, ADVP, PREP, NP, SBAR, CONJ, PP, PUNC, CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO}
	Unic	{-NONE-, ADVP, SBAR, CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO}
	Oblig	{-NONE-, ADVP, PREP, NP, SBAR, PP, PUNC, S, TYPO}
	Lin	{PREP ≺ {SBAR, CONJ, FRAG, PRON, S, UCP}; ADVP ≺ {NP, SBAR}; CONJ ≺ {ADVP, NP}; NP ≺ {SBAR, CONJ}; PP ≺ { ADVP, NAC}; PRT ≺ {NP, PREP, SBAR}; SBAR ≺ PP ; PUNC ≺ ADVP; PRON ≺ NP; S ≺ PUNC ; TYPO ≺ NP}
	Exig	{SBAR ⇒ PREP ; CONJ ⇒ PP; CONJP ⇒ PP; NAC ⇒ PP; FRAG⇒PREP; PRON ⇒ PREP; PRON ⇒ NP; S ⇒ PREP; UCP⇒ PREP; PRT ⇒ PREP; TYPO ⇒ NP}
	Excl	-NONE- ⊗ {ADVP, PREP, NP, SBAR, CONJ, PP, PUNC, CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO} ADVP ⊗ {CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO} PREP ⊗ {CONJP, NAC, TYPO} ; PRON ⊗ {S, UCP, PRT, TYPO}; NP ⊗ {CONJP, NAC, FRAG S, UCP}; S ⊗ {UCP, PRT, TYPO} SBAR ⊗ {CONJP, NAC, FRAG, PRON, S, UCP, TYPO}; -NONE- ⊗ {ADVP, PREP, NP, SBAR, CONJ, PP, PUNC, CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO}; UCP ⊗ {PRT, TYPO}; CONJ ⊗ {CONJP, NAC, FRAG, PRON, S, UCP, PRT, TYPO}; PP ⊗ {FRAG, PRON, S, UCP, PRT, TYPO}; PUNC ⊗ {NAC, FRAG, PRON, S, UCP, TYPO}; NAC ⊗ { FRAG, PRON, S, UCP, PRT, TYPO}; CONJP ⊗ {NAC, FRAG, PRON, S, UCP, PRT, TYPO}; FRAG ⊗ {PRON, S, UCP, PRT, TYPO}; PRT ⊗ TYPO

Figure 7.3 : Extrait de la GP à granularité minimale (description de la catégorie PP)

La spécification de la granularité peut concerner certaines catégories parmi d'autres. Nous avons déjà choisi de contrôler la granularité des noms et des adjectifs et nous avons interprété les résultats obtenus au niveau des catégories et des règles de production. C'est le tour maintenant des propriétés pour voir l'effet de cette variation sur leurs fréquences. Le tableau 7.9 montre ce jeu de variation de granularité sur les différents types de propriétés.

Ligne	**Traits**							**#P**						
	Det	**Type**	**Def**	**Gre**	**Nbre**	**Cas**	**Poss**	**Const**	**Unic**	**Oblig**	**Lin**	**Exig**	**Excl**	∑
1	0	0	0	0	0	0	0	3067	2703	1552	6216	2453	122346	138337
2	**0**	0	1	0	0	0	0	3205	2814	1661	6507	2491	126970	**143648**
3	**1**	0	1	0	0	**0**	0	3424	3019	1861	6811	**2561**	137930	**155606**
4	**0**	1	0	1	1	0	0	3314	2910	1763	6649	2543	131554	**148733**
5	1	0	1	0	0	**1**	1	3496	3064	1908	7062	**2624**	142229	160383
6	0	1	0	1	1	1	1	3618	3194	2018	7139	2656	158008	176633
7	1	1	1	1	1	1	1	4303	3810	2610	8266	3011	234578	256578

Tableau 7.9: Fréquences des propriétés avec la variation du choix des traits des catégories NOUN et ADJ

Dans le tableau 7.9 ci-dessus, le trait "Det" prouve encore son importance à travers l'augmentation totale du nombre de ces propriétés des lignes 2 et 4 à la ligne 3 dans laquelle il est choisi dans la variation des granularités des catégories NOUN et ADJ. Un autre trait, qui est le trait "Cas", parait important pour les propriétés d'exigence grâce à sa présence dans la ligne 5 à la différence de la ligne 3. Globalement, les propriétés d'unicité et d'exclusion sont les plus fréquentes pour toutes les variations de granularité proposées. Il faut en plus mentionner que quelque soit le niveau de granularité fixé, aucune propriété d'adjacence ne peut être induite de l'ATB. Nous pouvons alors constater qu'il n'est pas nécessaire d'avoir une condition sur l'ordre direct entre les constituants d'une même construction. Ceci prouve en d'autres termes la variété de la structuration des règles syntaxiques dans la langue arabe.

4 Conclusion

Le présent chapitre s'est occupé de la présentation des différentes expérimentations liées aux catégories, aux règles et aux propriétés obtenues en variant leur granularité. Ces informations sont générées dans le cadre de l'ATB ajusté et des grammaires CFG et GP. Les tailles de ces sorties dépendent du niveau de granularité des catégories qu'elles décrivent,

puisque toute catégorie peut être caractérisée par différents traits morphologiques. Le jeu de variation du choix de ces traits nous a indiqué leur degré d'importance dans la création de nouvelles variantes de cette catégorie. Plus le niveau de granularité des catégories est élevé, plus les grammaires CFG et GP sont complexes mais leurs propriétés et règles de plus en plus fidèles à la langue et à l'ATB d'origine, et inversement. La GP à granularité variable obtenue sera dans le chapitre suivant exploitée pour générer de nouvelles ressources héritant les qualités de l'ATB d'origine.

Chapitre 8

Expérimentations et évaluation de l'ATB enrichi et du résultat de Stanford Parser enrichi

1 Introduction

La concrétisation d'une certaine méthode ne se fait que si elle expérimentée sur un corpus de test. Le présent chapitre vise à réaliser cet objectif dans deux sections indépendantes respectivement sur les deux méthodes d'enrichissement et d'évaluation proposées dans le chapitre 5. Ces expérimentations sont accompagnées d'un ensemble de discussions pour comprendre les liaisons entre les résultats obtenus, apprécier les apports de ces méthodes, fixer leurs insuffisances et chercher des perspectives à appliquer. Nous rappelons que le chapitre 5 a fourni deux domaines d'utilisation de la GP arabe, à savoir : l'enrichissement de l'ATB et l'évaluation et l'enrichissement des résultats d'analyse de Stanford Parser. Nous rappelons que nos contributions (Bensalem et al, 2015a ; Bensalem et al., 2015b ; Bensalem et al., 2016 ; Bensaem et al., 2017) représentent les supports des deux méthodes d'enrichissement et d'évaluation adoptées. Ces méthodes se basent sur la vérification de la satisfaction des propriétés syntaxiques de la GP perçues entre les constituants des différentes unités syntaxiques. Ces propriétés sont intégrées dans les architectures de ces unités syntaxiques pour obtenir une version enrichie.

2 Expérimentations et évaluation sur l'ATB enrichi

Nous allons emprunter les différentes significations utilisées dans les en-têtes des tables et figures du chapitre précèdent, tout ajoutant la liste suivante :

Symboles	**Significations**
E	Nombre d'occurrences des propriétés évaluées (vérifiées) satisfaites ou violées
V	Nombre d'occurrences des propriétés violées
ΣE	Total des propriétés évaluées
ΣV	Total des propriétés violées

Tableau 8.1: Significations des en-têtes des tableaux

2.1 Caractéristiques des entrées

Nous avons testé notre tâche d'enrichissement sur la même version de l'ATB (notée ATB2v1.3) qui a été utilisée pour induire la GP à granularité variable expérimentée dans le chapitre précédent. Nous citons quelques fréquences sur cette ressource à enrichir et sur la GP utilisée pour l'enrichissement qui nous semblent nécessaires pour ce chapitre.

2.1.1 GP utilisée pour l'enrichissement

Pour la GP que nous avons utilisé pour extraire ses propriétés, elle est induite à partir de la moitié du treebank ATB pour rendre le corpus d'étude différent à celui de test. Cette GP a été réduite à la granularité la plus faible pour les catégories grammaticales de l'ATB qu'elle a décrites. Pour plus de précision, le tableau 8.2 ci-dessous affiche en termes de nombre d'occurrences des caractéristiques de cette GP.

	#C	**#**
Catégories syntaxiques	21	189711
Catégories lexicales	39	184251
Propriétés	3081	6792698

Tableau 8.2 : Caractéristiques de la GP arabe utilisée

2.1.2 Treebank ATB à enrichir

Pour l'ATB à enrichir, nous avons trouvé premièrement qu'il est composé de 841 catégories grammaticales dont 348 sont syntaxiques (regroupées en 21 groupes). Le tableau 8.3 illustre la répartition des groupes de catégories syntaxiques de l'ATB en fonction de leur fréquence d'apparition, le nombre de leurs constituants possibles ainsi que le nombre des règles de production qui les caractérisent.

XP	**#**	**# C**	**# R**	**XP**	**#**	**# C**	**# R**	**XP**	**#**	**# C**	**# R**
NP	110748	299	4824	PRT	2292	13	14	PRN	65	10	20
PP	22100	22	263	ADVP	539	6	68	LST	56	2	2
S	19358	138	1230	NAC	221	18	53	SQ	51	12	26
VP	15947	342	6675	FRAG	178	22	56	CONJP	37	3	2
SBAR	9524	47	380	WHADVP	136	3	34	INTJ	11	1	1
WHNP	4574	3	64	UCP	132	19	88	X	5	4	5
ADJP	3665	88	593	SBARQ	68	19	51	WHPP	3	3	3
									∑	**841**	**14452**

Tableau 8.3: Répartition des catégories syntaxiques dans l'ATB à la granularité minimale

D'après le tableau 8.3, nous remarquons que la catégorie syntaxique la plus fréquente dans l'ATB décrit le syntagme nominal (NP). Elle contient même un grand nombre de constituants possibles avec un nombre de règles de production assez important qui couvre le $\frac{1}{3}$ des règles.

2.2 Caractéristiques de la sortie : treebank ATB en version enrichie

La version enrichie du treebank ATB est également fournie sous le format XML. A partir des résultats obtenus concernant les propriétés de la GP intégrées dans l'ATB, nous avons remarqué plusieurs points que nous citons dans ce qui suit. Tout d'abord, la dominance du NP dans l'ATB n'influence pas excessivement la répartition des propriétés dans l'ATB enrichi. En effet, selon le tableau 8.4 qui montre les occurrences des propriétés décrivant les dix catégories syntaxiques les plus fréquentes, le NP occupe la première position au niveau de la répartition des propriétés d'unicité, (40%), et d'exclusion (83%). Cependant, cette position est affectée au syntagme verbal (VP) pour les propriétés de linéarité (62%) et à la proposition subordonnée (SBAR) pour les propriétés d'exigence. Cette position est partagée entre le NP et le VP pour les propriétés de constituance et d'obligation. Les fréquences de ces propriétés sont relativement faibles pour la catégorie PP qui occupe 27% du nombre d'apparition dans l'ATB. Nous pouvons noter aussi que la disposition d'un nombre assez élevé de propriétés d'unicité pour la plupart des catégories syntaxiques implique la nécessite d'avoir un constituant unique dans chaque syntagme arabe. En plus, l'ordre linéaire est important non seulement au VP mais aussi à la SBAR.

XP	**Constituance**		**Unicité**		**Obligation**		**Linéarité**		**Exigence**		**Exclusion**		∑	
	#P	**#**	**#P**	**#**	**#P**	**#**	**#P**	**#**	**#P**	**#**	**#P**	**#**	**#P**	**#**
NP	**29**	176822	22	**21686**	**19**	163585	50	1237	6	125	404	**44742192**	531	45216395
PP	16	45789	12	1840	9	45510	17	1795	15	1947	106	2342600	176	2461581
S	22	34622	11	539	12	34283	45	3807	12	89	99	1916442	202	2009140
VP	**28**	50749	19	16694	**18**	50215	104	**26536**	16	1493	196	3125612	382	3287246
SBAR	18	19785	13	5238	8	19060	30	9053	11	**4913**	129	1228596	210	1296169
WHNP	5	4578	5	4574	3	4574	2	4	2	4	8	36592	26	54903
ADJP	17	6182	10	88	5	3784	17	68	12	88	87	318855	149	332730
PRT	4	2292	4	2290	4	2292	0	0	0	0	66	151272	87	160438
ADVP	8	561	6	559	4	539	3	21	4	22	12	6468	38	8709
NAC	11	572	9	426	5	358	9	189	10	204	33	7293	78	9263
∑	211	166805	162	54721	140	325558	346	43096	139	9279	1288	53891401	1879	54836574

Tableau 8.4: Répartition des propriétés par type et par catégorie syntaxique dans l'ATB enrichi

Indépendamment de la répartition de propriétés donnée, nous avons obtenu pour un tel texte arabe, des informations définies implicitement qui sont importantes et variées. Ainsi, pour un VP arabe, nous avons la propriété de linéarité IV $\prec$ PP, qui exige qu'un syntagme Prépositionnel (PP) ne doit jamais précéder un verbe au présent (IV). De même, nous avons la propriété d'exigence ADJ $\Rightarrow$ NP, qui nécessite la présence d'un NP si un adjectif (ADJ) existe.

En se focalisant sur la répartition des types de propriétés, il est remarquable que les propriétés d'obligation occupent une grande part, soit 54%, sans compter les propriétés d'exclusion qui sont très abondantes.

En plus, pour un aperçu global de tous les types de propriétés, nous pouvons remarquer leur fréquence élevée en comparaison avec d'autres treebanks enrichis de propriétés syntaxiques (e.g. the FTB) (Blache et Rauzy, 2012). Ce grand nombre peut être plus grand si nous étendons notre travail d'enrichissement à la granularité la plus élevée des catégories grammaticales de l'ATB. Ceci reflète la richesse et la variété des structures syntaxiques arabes.

Treebanks	**ATB**	**FTB**
Uniqueness	54721	38007
Obligation	325558	32602
Linearity	43096	27367
Requirement	9279	11022
Exclusion	53891401	89293
$\sum$	**54324055**	**198291**

Tableau 8.5: Répartition des propriétés par types dans les treebanks

Comme pour les catégories syntaxiques, il est possible de prendre une idée sur la répartition des propriétés par leur type comme présenté par la figure 8.1. A travers cette figure, nous avons remarqué que plusieurs catégories peuvent être répétées mais pas absentes dans les syntagmes arabes. Cette interprétation est déduite à partir de la grande différence perçue entre les fréquences des propriétés d'unicité et celles d'obligation. Il est également clair que nous avons une grande abondance des propriétés d'exclusion. Ceci nécessite d'être ajusté par la description de nouvelles interprétations utilisant des heuristiques pour ces types de propriétés. Cependant, les propriétés d'adjacence ne peuvent pas avoir d'autres conceptions car elles portent sur une information d'ordre qui ne peut être définie qu'automatiquement.

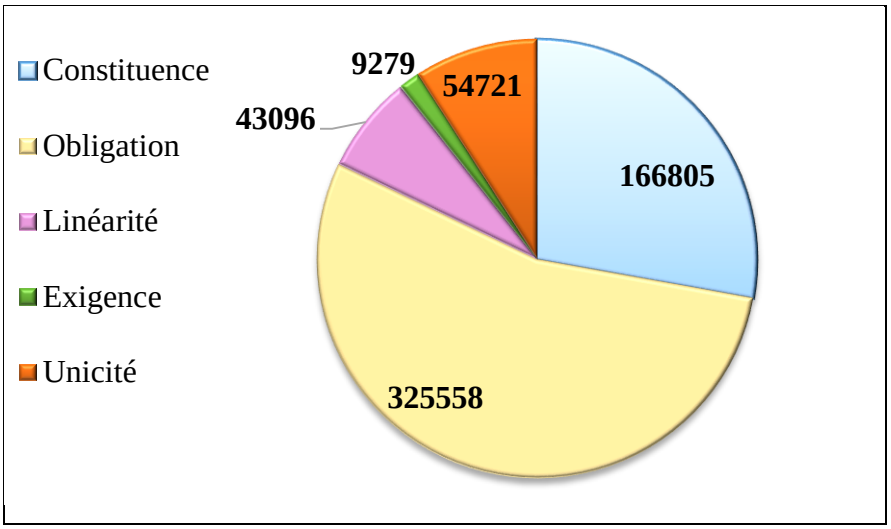


Figure 8.1: Répartition des propriétés de l'ATB par type

Comme déjà mentionné dans la section précédente, la vérification de la satisfaction des propriétés permet de distinguer celles ayant été satisfaites de celles violées. Nous donnons dans le tableau 8.6, comme exemple, les taux de satisfaction des propriétés relatives aux catégories VP.

Etat de propriété	**Unicité**		**Linéarité**		**Exigence**		**Exclusion**	
	#	**%**	**#**	**%**	**#**	**%**	**#**	**%**
Satisfaite	15932	99.91	26529	99.99	1491	99.87	3125590	99.99
Violée	15	0.09	7	0.01	2	0.13	22	0.01

Tableau 8.6 : Taux de satisfaction des propriétés décrivant la catégorie VP par type

Selon les résultats obtenus, les propriétés violées sont en nombre négligeable en les comparants avec celles satisfaites. Si l'ATB est considéré comme une ressource à large couverture, la GP utilisée dans la tâche d'enrichissement hérite aussi cette richesse.

La répartition des propriétés peut être détaillée à des niveaux plus fins par la description à part de chaque propriété d'un certain type et décrivant une certaine catégorie syntaxique. Cette description nous permet de déterminer le degré de l'importance de chaque propriété. Cette information précise peut ne pas être accessible en se limitant à la répartition globale par type de propriétés ou par catégorie syntaxique. Nous pouvons, ainsi, ne pas considérer les propriétés d'un certain type comme bloc homogène. Nous présentons dans la figure 8.2, par exemple, la répartition des propriétés décrivant la catégorie VP pour déterminer celles les plus fréquentes. Les axes des abscisses des schémas montrés concernent les indices des propriétés. Tandis que

les axes des ordonnées, ils indiquent leurs fréquences. Nous pouvons considérer dans ce cas que les propriétés les plus fréquentes possèdent les poids les plus élevés. L'information de poids qui porte sur le nombre d'occurrences de la propriété peut être considérée comme information pertinente. Il est maintenant possible de spécifier un seuil sur les poids des propriétés pour distinguer celles fortes de celles faibles.

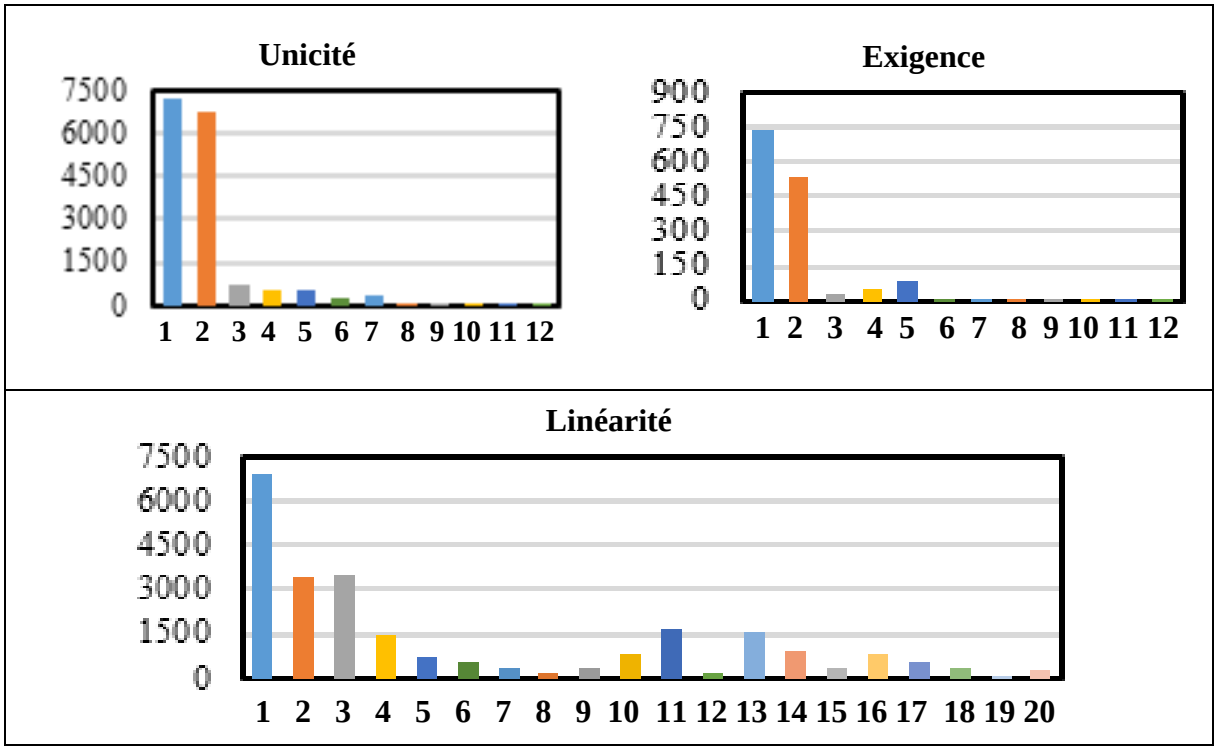


Figure 8.2: Répartition de quelques propriétés de la catégorie VP

Avec un aperçu de la figure 8.2, nous pouvons par exemple choisir de fixer que la propriété de linéarité ou d'unicité forte doit avoir un poids d'au moins 1500 occurrences. Alors que le poids de la propriété d'exigence forte doit dépasser les 500 occurrences. Le tableau 8.7 nous donne les propriétés fortes décrivant les VP.

	Unicité	**Linéarité**	**Exigence**
Indice	1, 2	1, 2; 3; 11, 13	1, 2
Propriété	PV, IV	PV $\prec$ {NP, PP}; IV $\prec$ PP; PRT $\prec$ {NP, IV}	NOUN $\Rightarrow$ NP; ADJ $\Rightarrow$ NP

Tableau 8.7: Propriétés fortes de la catégorie VP

En utilisant ces résultats, nous pouvons mesurer automatiquement les poids des propriétés dans une certaine construction. Cette information peut être inclue dans la GP pour alléger le

processus d'analyse syntaxique. En effet, nous pouvons vérifier seulement la satisfaction des propriétés fortes. Les autres peuvent être relaxées (ignorées). Cette information est également efficace pour évaluer la difficulté de traitement dans les systèmes cognitifs puisque la violation d'une propriété forte implique une difficulté plus grande.

3 Expérimentations et évaluation sur le corpus analysé de Stanford Parser enrichi

Nous présentons dans ce qui suit quelques résultats obtenus des expérimentations réalisées sur un corpus analysé de Stanford Parser.

3.1 Caractéristiques des entrées

Deux entrées sont nécessaires pour notre méthode d'évaluation et d'enrichissement d'analyses syntaxiques sont le résultat d'analyse d'un corpus sur Stanford Parser et la GP que nous avons utilisée pour l'enrichissement.

3.1.1 Résultat d'analyse de Stanford Parser sur un corpus

Nous avons évalué l'analyseur syntaxique Stanford Parser (v3.6) sur 100 phrases obtenues de contes arabes destinés aux enfants[15]. Les phrases contiennent en moyenne 9 mots. Le résultat d'analyse nous a donné les arbres syntaxiques de ces phrases. Nous avons obtenu 877 catégories syntaxiques encapsulant 1047 catégories lexicales. La répartition de ces catégories est montrée par la figure 8.3 :

[15] From the Arabic book "زهرة بابنج للعصفورة " (chamomile flower to the bird) of Talal Hassan : http://www.awu-dam.org/book/02/child02/105-t-h/105-t-h.zip

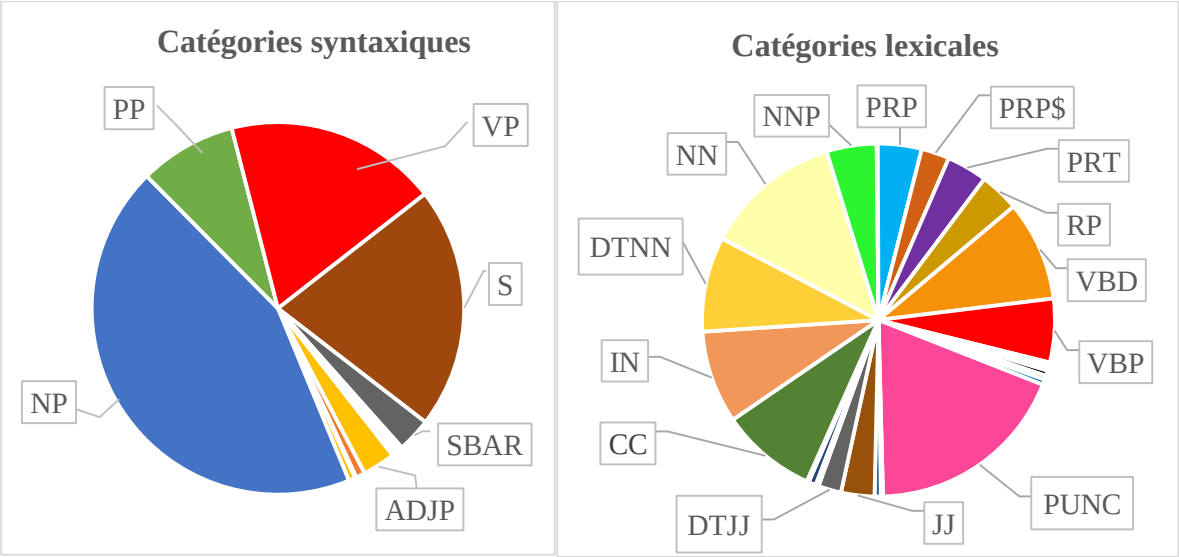


Figure 8.3: Répartition des catégories dans le corpus de contes d'enfants

Comme le montre la figure 8.3, les syntagmes nominaux (NP) sont dominants par rapport aux autres catégories syntaxiques avec 384 occurrences. Ils plus fréquents que les syntagmes verbaux (VP) et les phrases (S) car ce corpus analysé contient des phrases simples destinées aux enfants. Une phrase simple est étiquetée par une S qui encapsule un VP. Ce VP est composé généralement d'un verbe et de plusieurs NP ou PP (syntagme prépositionnel). Nous justifions ceci par la somme des fréquences 96 et 61 relatives respectivement aux verbes au présent (VBP) et au verbe au passé (VBD) qui est proche de la fréquence des VP (162). Les fréquences des propositions de subordination (SBAR), utilisées dans les phrases complexes, sont en plus faibles. Nous savons bien que les contes d'enfants (CE) se basent également sur les ponctuations. C'est pour cette raison que la fréquence de catégories lexicales la plus élevée est occupée par l'étiquette de ponctuation (PUNC).

3.1.2 GP utilisée pour l'enrichissement

Pour la GP arabe utilisé, elle est induite du corpus ATB (ATB2v1.3). Nous lui avons réduit la granularité de ses catégories grammaticales pour avoir une meilleure correspondance (mapping) entre les étiquettes du PTB (utilisées par Stanford Parser) et celles de l'ATB (utilisées par la GP). Cette GP se caractérise par les éléments suivants :

	Fréquence	**Nombre d'occurrences**
Catégories syntaxiques	21	189711
Catégories lexicales	39	184251
Propriétés	3081	6792698

Tableau 8.8: Caractéristiques de la GP arabe utilisée

3.2 Caractéristiques de la sortie

Pour évaluer le corpus de CE analysé, nous avons vérifié la satisfaction des propriétés de la GP arabe. En passant, nous avons intégré ces propriétés dans corpus analysé pour l'obtenir en une version enrichie. La figure 8.4 suivante montre un extrait du corpus de CE analysé par Stanford Parser et enrichie avec des propriétés syntaxiques :

```
<Category index="16:0:1:2" label="NP" pos-mapping="35">
 <Category index="16:0:1:2:0" label="DTNN" pos-mapping="29" value="الضبع" />
 <Category index="16:0:1:2:1" label="DTJJ" pos-mapping="10" value="العجوز" />
 <Caracterization nb_const="2" sat="+">
  <Property index="16:0:1:2:0" type="obligation" source="NP" target="DTNN" sat="+"/>
  <Property index="16:0:1:2:2" type="exclusion" source="DTNN" target="UH" sat="+"/>
  <Property index="16:0:1:2:3" type="exclusion" source="DTJJ" target="UH" sat="+" />
 </Caracterization>
</Category>
```

Figure 8.4: Extrait du corpus de CE analysé et enrichie avec des propriétés de la GP

Dans la figure 8.4, l'étiquette « characterization » encapsule les propriétés de la GP qui peuvent être vérifiées sur les constituants « DTNN » et « DTJJ » de la catégorie « NP ». Toutes ces propriétés ont été vérifiées comme satisfaites dans ce NP. Le paramètre « sat » est alors marqué par la valeur « + ». Nous présentons dans le tableau 8.9, les nombres d'occurrences de toutes les propriétés vérifiées par type et par catégorie syntaxique.

Type	**ADJP**	**ADVP**	**INTJ**	**NAC**	**NP**	**PP**	**VP**	**S**	**SBAR**	**UCP**	**SQ**	**SBARQ**	**WHADVP**	**WHNP**	**Σ**
Const	25	7	1	5	384	74	162	184	26	1	1	1	3	3	877
Unic	2	7	1	6	87	2	19	6	19	2	1	2	3	3	160
Oblig	39	7	1	5	538	148	390	410	21	3	1	3	3	3	**1572**
Lin	6	0	0	1	14	3	**166**	38	19	1	1	3	0	0	252
Exig	0	0	0	1	3	2	8	1	**14**	1	2	1	0	0	33
Excl	51	7	0	6	1009	149	871	411	71	3	2	4	0	6	**2590**
Σ	123	28	3	24	**2035**	378	**1616**	**1050**	170	11	8	14	9	15	5484

Tableau 8.9: Nombres d'occurrences des propriétés vérifiées sur le corpus analysé de CE

Selon le tableau 8.9, la moitié des propriétés vérifiées décrivent la catégorie syntaxique la plus fréquente qui est NP (2035/5484). Cependant, nous ne pouvons pas observer cette dominance pour la totalité des types de propriétés. En effet, la plupart des propriétés décrit les VP (65%) et la plupart des propriétés d'exigence décrit les SBAR (42%). Les 14 propriétés de linéarité qui ont été vérifiées dans les NP, correspondent seulement à 7 parmi 56 formes de

propriétés et les 3 propriétés d'exigence correspondent seulement à une 1 parmi 15 formes de propriétés. Ceci est dû au caractère simple et redondant des constructions de NP dans le corpus de CE destiné aux enfants. Les propriétés d'exclusion sont les plus vérifiées dans ce corpus occupant la moitié de l'ensemble car la GP souffre de la surgénération de ce type de propriétés. Les propriétés d'obligation sont également fréquentes (1572) et dépasse le nombre d'occurrence des constituants (877). Ceci montre que plusieurs constituants sont marqués plusieurs fois comme élément obligatoire (Head). Ceci prouve en plus la variété des constructions de l'ATB duquel la GP a été induite.

Les propriétés sont vérifiées comme satisfaites ou violées. Pour plusieurs catégories syntaxiques et types de propriétés, toutes les propriétés vérifiées ont été satisfaites dans ce corpus. Pour cela, nous proposons de montrer dans le tableau 8.10 ci-dessous seulement les nombres d'occurrences des propriétés violées (V) parmi celles vérifiées (E).

	ADJP			**VP**			**SBAR**			**UCP**			**SQ**			**ΣV**	**ΣE**
	Propriété	**V**	**E**	**Propriété**	**V**	**E**	**Propriété**	**V**	**E**	**Propriété**	**V**	**E**	**Propriété**	**V**	**E**		
Unic	-	0	2	-	0	19	IN ∈ Unic(SBAR)	1	19	-	0	2	-	0	1	1	160
Lin	-	0	6	-	0	166	-	0	19	ADJP≺SBAR	1	1	-	0	1	1	252
Exig	-	0	0	**NAC⇒UH** **UCP⇒UH** **VBN⇒UH** **VN⇒UH**	1 1 2 2	8	-	0	14	**SBAR ⇒UH**	1	1	**PP ⇒UH**	1	2	8	33
Excl	JJ⊗ADJ_NUM	2	51		0	871	-	0	71	-	0	3	-	0	2	2	2590
Σ	2		123	6		1616	1		170	2		11	1		8	12	5484

Tableau 8.10 : Nombre d'occurrences des propriétés violées dans le corpus de CE analysé

Comme montré par le tableau 8.10, la plupart des propriétés violées décrit les VP. Ces derniers contiennent des catégories exigeant la présence de l'interjection « UH ». C'est le cas aussi des SBAR dans les propositions juxtaposées (UCP) et les PP dans les phrases interrogatives (SQ). En se basant sur les résultats des tables 8 et 9, nous pouvons calculer les scores d'évaluation (1 ; 2 ; 3 ; 4) que nous avons proposés dans le chapitre 5. Par exemple :

Pour le score 1, les pourcentages de satisfaction/violation des propriétés d'exigence qui décrivent un VP sont respectivement :

$$S_{VP}^{req} = \frac{\sum_{p \in P} sn(p_{VP}^{req})}{\sum_{p \in P} en(p_{VP}^{req})} = \frac{2}{8} = \mathbf{25\%} \text{ et } V_{VP}^{req} = \frac{\sum_{p \in P} vn(p_{VP}^{req})}{\sum_{p \in P} en(p_{VP}^{req})} = \frac{6}{8} = \mathbf{75\%} \text{ (score 1)}$$

Pour le score 2, les pourcentages de satisfaction/violation de toutes les propriétés qui décrivent un VP sont respectivement :

$$S_{VP} = \frac{\sum_{t\in T}\sum_{p\in P} sn(p_{VP}^{t})}{\sum_{t\in T}\sum_{p\in P} en(p_{VP}^{t})} = \frac{155}{161} = \mathbf{99.6\%} \text{ et } V_{VP} = \frac{\sum_{t\in T}\sum_{p\in P} vn(p_{VP}^{t})}{\sum_{t\in T}\sum_{p\in P} en(p_{VP}^{t})} = \frac{6}{161} = \mathbf{0.4\%}$$

Pour le score 3, pourcentages de satisfaction/violation des propriétés d'exigence décrivant toutes les catégories syntaxiques sont respectivement :

$$S^{req} = \frac{\sum_{c\in C}\sum_{p\in P} sn(p_{c}^{req})}{\sum_{c\in C}\sum_{p\in P} en(p_{c}^{req})} = \frac{25}{33} = \mathbf{75.76\%} \text{ and } V^{req} = \frac{\sum_{c\in C}\sum_{p\in P} vn(p_{c}^{req})}{\sum_{c\in C}\sum_{p\in P} en(p_{c}^{req})} = \frac{8}{33} = \mathbf{24.27\%}$$

Pour le score 4, les répartitions de la satisfaction/violation de la propriété NAC⇒NP qui décrit un VP est respectivement :

$$sp_{VP}^{req}(NAC \Rightarrow NP) = \frac{sn(NAC \Rightarrow NP)}{\sum_{p\in P} sn(p_{VP}^{req})} = \frac{1}{2} = \mathbf{50\%} = vp_{VP}^{req}(NAC \Rightarrow NP)$$

Ces scores ne remplacent pas les outils d'évaluation Evalb et GramRelEval qui mesurent le rappel et la précision d'un certain résultat d'analyse. Ils considèrent en fait que le corpus est correctement analysé mais ils vérifient si les relations entre les unités syntaxiques sont correctes.

4 Conclusion

Au cours de ce chapitre, nous avons présenté différentes expérimentations aussi bien sur l'enrichissement de l'ATB que sur l'évaluation et l'enrichissement des résultats d'analyse de Stanford Parser. Les résultats que nous avons obtenus sont encourageants et significatifs.

Comme perspectives, nous pouvons enrichir d'autres treebanks et évaluer des résultats d'autres analyseurs syntaxiques. Cela nécessite la préparation d'une table de mapping entre l'étiquetage de la GP et celui du système à enrichir. Nous pouvons également comparer les versions enrichies de résultat d'analyse avec des analyseurs construits à base de GP. Le corpus de référence peut être l'ATB enrichi et la comparaison peut être en termes de nombre de propriétés syntaxiques décrites. Nous pouvons aller plus loin en intégrant dans la GP des descriptions des propriétés morphologiques et sémantiques. Ceci peut alléger encore le processus d'analyse

Chapitre 9

Expérimentation et évaluation de l’analyseur syntaxique stochastique de propriétés syntaxiques

1 Introduction

La démarche d'analyse syntaxique probabiliste de propriétés, déjà décrite dans le chapitre 6, se base sur deux composants principaux : le modèle d'apprentissage et l'algorithme d'analyse. Le modèle d'apprentissage de son tout se base sur deux sous-composants : la PCFG lexicalisée approfondie et la PGP lexicalisée. Après avoir mis en œuvre cette démarche, nous visons à travers le présent chapitre à l'expérimenter sur un corpus d'entrée et un corpus de test et discuter les résultats obtenus. Ces discussions aideront à l'améliorer et à surmonter ses insuffisances. Notre chapitre présente dans les trois sections suivantes respectivement les caractéristiques des entrées de notre démarche, celles du modèle d'apprentissage construit et celles du résultat d'analyse avec l'algorithme CYK.

2 Caractéristiques des entrées

Pour lancer notre analyseur syntaxique probabiliste de propriétés, il nous faut utiliser les trois entrées suivantes : le corpus d'apprentissage, le corpus de test et les règles de (Habash et al, 2009).Ces dernières sont les ressources externes que nous avons appliquées pour l'induction de notre GP. Nous les avons déjà décrites dans le chapitre 7 qui est destiné à montrer les expérimentations réalisées de cette induction. Les sous-sections suivantes du présent chapitre se seront alors limitées à la présentation de différentes caractéristiques des autres entrées.

2.1 Corpus d'apprentissage : le Treebank ATB

L'ATB, qui représente notre corpus d'apprentissage, a été décrit dans le chapitre 7. Nous ajoutons dans cette sous-section un petit détail concernant les formats de l'ATB à utiliser. Il s'agit des deux formats "penntree" et "pos". Le premier contient les arbres syntaxiques d'un corpus brut comme pour l'induction de la GP à granularité variable expérimentée dans le chapitre 7. Tandis que le deuxième fournit plusieurs informations lexicales, syntaxiques et sémantiques des différents mots du corpus. Nous allons l'utiliser pour extraire l'écriture arabe de chaque mot du corpus car il est donné en écriture translitérée dans le format "penntree". Cela est nécessaire pour que chaque mot du modèle d'apprentissage à construire puisse être reconnu lors de la phase d'analyse avec l'algorithme CYK, puisque le corpus de test entré dans cette phase est fourni avec des mots écrits en arabe. Nous rappelons dans le tableau 9.1, ci-dessous des fréquences générales des variantes des catégories grammaticales trouvées dans l'ATB, tout en adaptant les symboles des entêtes des tables utilisés dans les deux chapitre précédents.

Catégories syntaxiques						Catégories lexicales					
XP	**#C**	**#**	**XP**	**#C**	**#**	**C**	**#C**	**#**	**C**	**#C**	**#**
ADJP	**13**	3665	S	22	19358	-NONE-	1	14932	FOREIGN	1	9
ADVP	**61**	539	SBAR	64	9524	ABBREV	1	74	INTERJ	1	12
CONJP	1	37	SBARQ	2	68	ADJ	97	18108	LATIN	1	28
FRAG	2	179	SQ	1	51	ADV	4	539	VERB	**81**	**15699**
INTJ	1	11	UCP	9	132	CONJ	2	15070	PUNC	1	16351
LST	1	56	VP	2	15947	PART	11	2288	PREP	**1**	**21803**
NAC	10	221	WHADVP	19	136	PRON	32	10252			
NP	167	110748	WHNP	24	4574	DET	1	12			
PP	55	22100	WHPP	3	3	DIALECT	2	19			
PRN	2	65	X	1	5	NOUN	141	68628			
PRT	1	2292	∑	**461**	**189711**	TYPO	2	427	∑	**380**	**184251**
									∑	**841**	**373962**

Tableau 9.1: Fréquences des variantes des catégories grammaticales dans l'ATB

Nous pouvons remarquer à partir de le tableau 9.1 illustré ci-dessus, que l'ATB se base sur des catégories grammaticales très variées. Cette variation n'est pas relative à leur nombre d'occurrences. Ainsi, les catégories syntaxiques ADVP sont très variées (61) par rapport aux catégories ADJP (13) bien que ces dernières sont plus fréquentes. De même, dans les catégories lexicales, les PREP sont très fréquentes mais elles n'ont qu'une seule variante alors que les verbes (VERB) qui sont moins fréquents sont très variés. Globalement, cette variation de catégories grammaticales de l'ATB favorise sa robustesse, ce qui peut justifier son choix comme corpus d'apprentissage pour notre analyseur.

2.2 Corpus de test : le Treebank ATB

Notre corpus de test est un extrait de l'ATB. C'est le corpus que nous allons analyser avec l'algorithme CYK. Il a été traité à priori lexicalement en séparant ses agglutinations et en affectant des catégories lexicales à ses mots. Nous avons utilisé le format "pos" de l'ATB pour assurer ce prétraitement. Notre corpus de test regroupe 400 phrases de différentes longueurs (ou nombres de mots). Pour réaliser les expérimentations nécessaires, nous avons divisé ce corpus en fonction de la longueur par phrase. Chaque division contient 100 phrases de l'ATB dont la longueur appartient à un intervalle de 10 mots de différence. Ainsi, la première division est composée des phrases dont la longueur est limitée à 10 mots. La deuxième division regroupe les phrases dont la longueur est comprise entre 10 et 20, et ainsi de suite. La dernière version englobe les phrases dont la longueur ne dépasse pas les 40 mots.

Le tableau 9.2 suivant montre les différentes fréquences des phrases, catégories lexicales et mots dans l'ATB globalement et plus particulièrement dans notre extrait pour montrer son niveau de couverture :

Longueur	ATB						Notre Extrait			
	Phrases		Catégories lexicales		Mots		Catégories lexicales		Mots	
	#	%	#	%	#	%	#	%	#	%
[1, 10]	2031	**26%**	4086	33%	266	**2%**	407	66%	106	**17%**
[11, 20]	1562	**20%**	5800	24%	298	**1%**	777	51%	134	**9%**
[21, 30]	1245	**16%**	6748	**21%**	291	1%	1182	**46%**	150	6%
[31, 40]	986	**13%**	7032	**20%**	302	1%	1517	**43%**	168	5%
[41, 50]	749	10%	6937	21%	288	1%	-	-	-	-
[51, 60]	480	6%	5978	23%	281	1%	-	-	-	-
[61, 70]	296	4%	4875	25%	257	1%	-	-	-	-
[71, +∞]	423	5%	7177	19%	302	1%	-	-	-	-
Σ	**7772**	**100%**	**220080**				**42033**			

Tableau 9.2 : Fréquences des variantes des catégories grammaticales dans l'ATB

Selon le tableau 9.2 ci-dessus, le nombre de mots de la plupart des phrases de l'ATB (75%) ne dépasse pas 40, ce qui justifie notre choix d'extraits limité à des divisions couvrant au maximum ce nombre. Bien qu'il soit quasi-arbitraire, ce choix semble bénéfique de sorte qu'il marque une couverture de catégories lexicales et de mots arabes largement élevée par rapport à celle de l'ATB entier. En effet, pour toutes les divisions, cette couverture atteint le double ou plus par rapport à celle de l'ATB entier pour les catégories lexicales (e.g. 46% contre 21% et 43% contre 20% respectivement pour les divisions [21,30] et [31, 40]). Tandis que la couverture des mots est multipliée même par 8 par rapport à celle de l'ATB pour les divisions [1, 10] et [11, 20].

3 Caractéristiques du modèle d'apprentissage construit

Le modèle d'apprentissage construit est composé d'une PCFG lexicalisée approfondie sous forme CNF ainsi que la PGP lexicalisée sous forme CNF. Nous rappelons alors de sa démarche de construction déjà mentionnée dans le chapitre 6. L'entrée principale est l'ATB sous son format « penntree ». Cette entrée est premièrement lexicalisée en utilisant les relations Heads extraites par les règles de (Habash et al., 2009). Elle est également minorée des catégories lexicales –NONE- et des catégories syntaxiques qui les encapsulent. Ensuite, elle est analysée

pour l'extraction des règles qui la décrivent. Cette analyse est réalisée d'une manière approfondie pour pouvoir générer les suites de catégories lexicales relatives à chaque règle extraite. Un ensemble de statistiques sur les règles et les catégories rendent la CFG probabiliste. La PCFG obtenue est ajustée selon la forme CNF pour qu'elle puisse être utilisée par l'algorithme d'analyse CYK. Les résultats des expérimentations de ces trois sous-étapes seront discutés respectivement dans les trois sous sections suivantes :

3.1 ATB lexicalisé minoré des catégories –NONE-

Comme nous l'avons déjà mentionné dans le chapitre 6, l'étape de lexicalisation consiste à associer aux étiquettes des catégories syntaxiques de l'ATB leur constituent lexical obligatoire (Head). Cette étape est réalisée après l'extraction des relations Heads de (Habash et al., 2009). Nous avons à la fois ignoré les catégories des mots vides -NONE- et toutes les catégories qui les encapsulent. Le tableau 9.3 montre les fréquences des catégories de l'ATB avant et après la réalisation de cette étape.

Catégories grammaticales	**#C**		**#**	
	Avant	**Après**	**Avant**	**Après**
Catégories syntaxiques	461	2897	189711	174912
Catégories lexicales	380	379	184251	169319
∑	**841**	**3276**	**373962**	**343866**

Tableau 9.3: Fréquences des catégories grammaticales avant et après la lexicalisation et la suppression des catégories -NONE-

A partir du tableau 9.3, nous pouvons remarquer que le nombre d'occurrences de toutes les catégories grammaticales a été affecté à cause de la non prise en compte de la catégorie -NONE- et de ses catégories syntaxiques qui l'encapsulent. Ainsi, environ 8% des occurrences des catégories syntaxiques et 11% de celles lexicales ont été supprimées de la nouvelle version de l'ATB. En plus, le nombre de catégories syntaxiques s'est multiplié par plus que 6 fois en leur associant la nouvelle information Head. Ces catégories sont alors de plus en plus variées. Pour observer cette variation, nous montrons dans les quatre figures suivantes respectivement les répartitions des nouvelles catégories syntaxiques principales dans l'ATB.

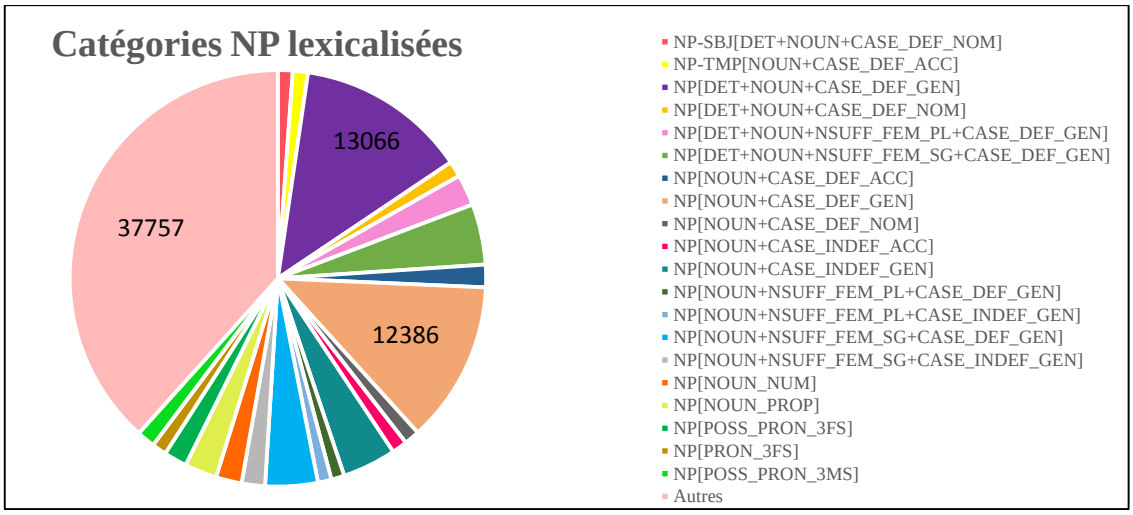


Figure 9.1: Répartition des catégories NP lexicalisées dans l'ATB

Selon la figure 9.1, les variantes les plus fréquentes de la catégorie NP lexicalisée sont NP[DET+NOUN+CASE_DEF_GEN] et NP[NOUN+CASE_DEF_GEN], ce qui indique l'emploi énorme des noms définis au génitif dans les constructions de l'ATB. Cependant, le nombre d'occurrences total de ces deux variantes occupe seulement environ 26 %. Ceci revient à la variation de la répartition des catégories NP lexicalisées dans l'ATB. En effet, plus que 38 % de leurs occurrences dans l'ATB sont occupées par différentes catégories peu fréquentes mais très nombreuses dans le corpus. Le nombre d'occurrences de l'une ne dépasse pas 1000. Elles représentent cependant plus que 98,7% de la liste des catégories. Pour toutes les figures, nous avons encapsulé ce type de catégories dans une étiquette appelée "Autres". Les fréquences des catégories relatives à cette étiquette sont citées dans le tableau 9.4 ci-dessous.

XP	**#C**	**#**	**Etiquette "Autres"**		
			#C	**# par constituent**	**#**
NP	1655	93906	1635	<1000	37757
PP	78	22013	71	<200	391
S	299	19355	290	<500	3736
VP	128	15945	114	<200	1864

Tableau 9.4 : Fréquences des catégories syntaxiques encapsulées dans l'étiquette « Autres »

Nous allons exploiter les fréquences indiquées par le tableau 9.4 dans les trois figures suivantes. C'est pour cette raison que chaque ligne du tableau est relative à une catégorie syntaxique lexicalisée présentée dans l'une de ces figures. En effet, chaque ligne fournit la catégorie lexicalisée concernée, le nombre de variantes de cette catégorie, le nombre d'occurrences total de toutes ces variantes et les mêmes informations pour l'étiquette "Autres" pour pouvoir montrer sa part dans toute la liste des variantes d'une catégorie lexicalisée. Les

informations relatives à l'étiquette "Autres" portent alors sur le nombre de variantes encapsulées dans cette étiquette, le nombre d'occurrences maximal par variante et le nombre d'occurrences total de ces variantes.

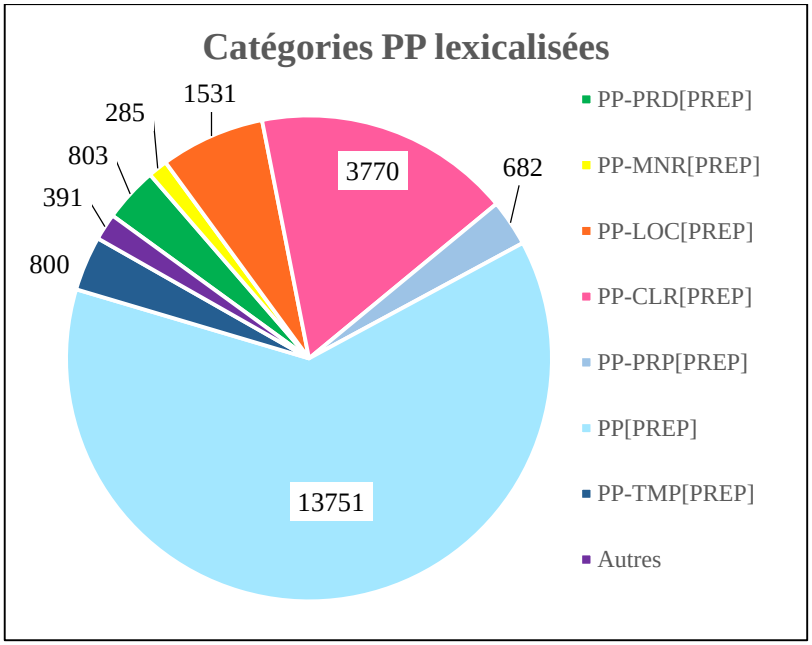


Figure 9.2: Répartition des catégories PP lexicalisées dans l'ATB

A travers la figure 9.2, les catégories PP lexicalisées ne sont pas très variées. Elles partagent le même constituent Head (PREP), et elles sont dominées par la catégorie PP à granularité minimale. Mêmes les variantes de PP encapsulées dans l'étiquette "Autres" ne forment pas ensembles une grande part (seulement 4%), bien qu'elles sont nombreuses selon le tableau 9.4 en représentant 91% de toutes les variantes.

Dans la figure 9.3, les catégories S lexicalisées annotant les phrases sont réparties de manière plus ou moins équitables. En effet, à part la variante dominante S[PV+PVSUFF_SUBJ:3MS] avec 22% du nombre d'occurrences total, quatre autres variantes ont presque les mêmes pourcentages variant entre 10% et 13%. Ces variantes représentent les phrases basées sur les verbes au passé et au présent pour les genres féminins et masculins singuliers.

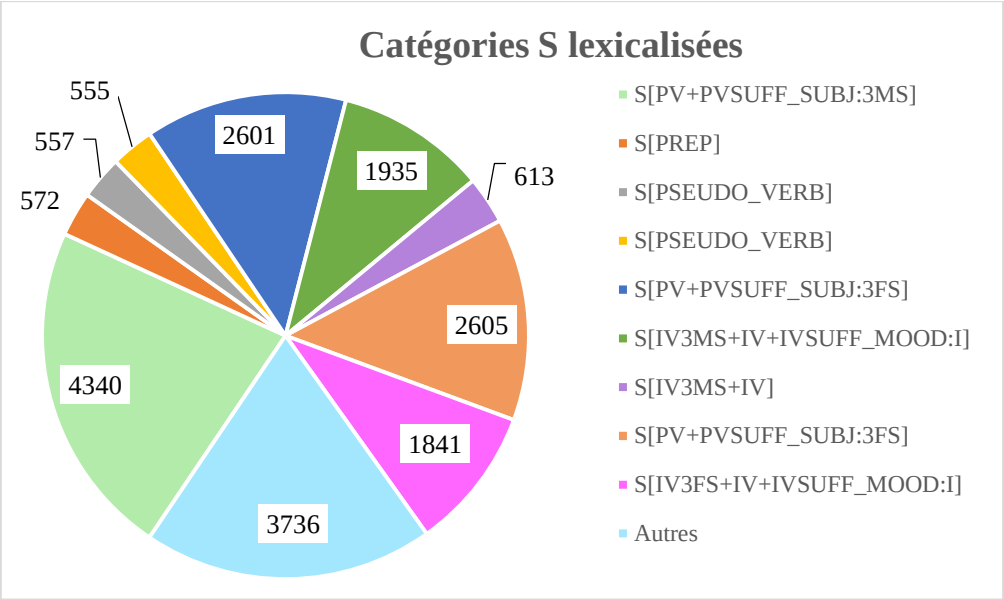


Figure 9.3: Répartition des catégories S lexicalisées dans l'ATB

L'étiquette "Autres", dans la figure 9.3, occupe une part assez importante relativement comparable à celles de la variante de S la plus dominante. Elle regroupe également presque 97% des variantes de S.

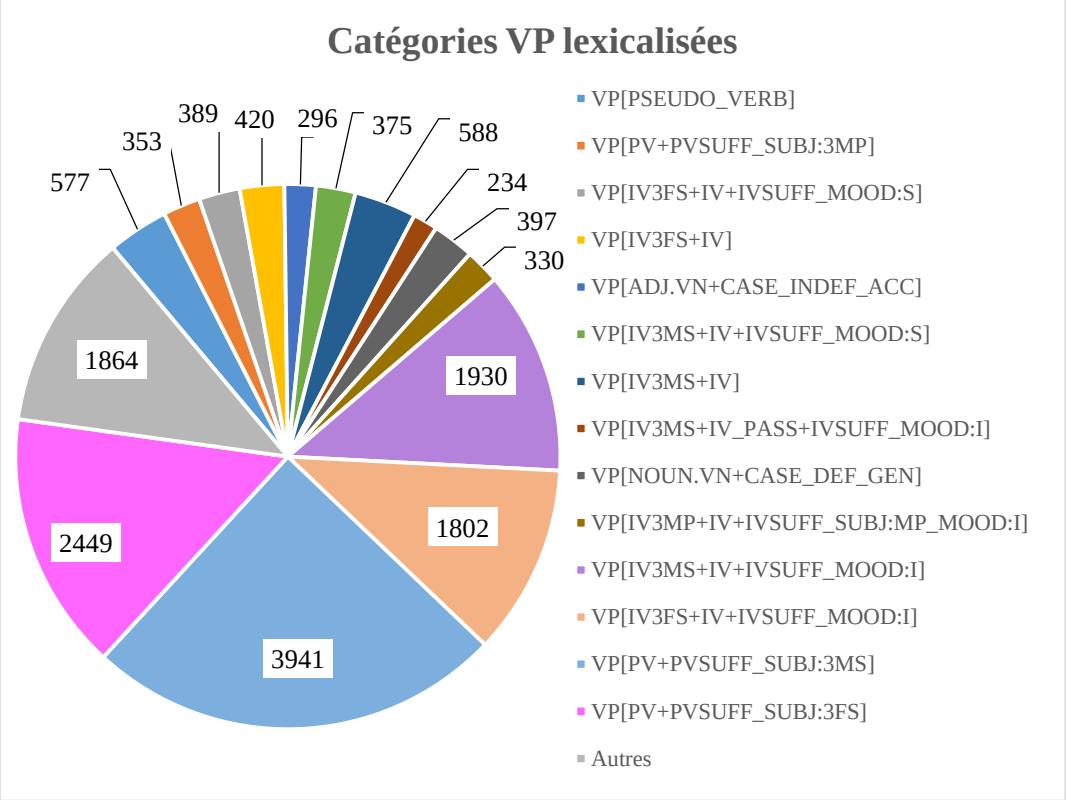


Figure 9.4: Répartition des catégories VP lexicalisées dans l'ATB

Dans la figure 9.4 qui montre la répartition des catégories VP lexicalisée, la variante de VP la plus dominante marquant 25% des occurrences est VP[PV+PVSUFF_SUBJ:3MS]. Elle représente les syntagmes verbaux basés sur les verbes conjugués au passé avec le masculin singulier. Ceux basés sur les verbes conjugués au passé avec le féminin singulier sont un peu moins fréquents marquant 15%. L'étiquette "Autres" dans les catégories VP n'a pas la même part que celle dans les NP. Bien qu'elle regroupe 89% des variantes de VP, elle n'occupe que 12% des occurrences. Deux autres variantes sont également importantes en termes de nombre d'occurrences (plus de 11%) et qui sont les syntagmes encapsulant des verbes conjugués au présent avec le singulier. Les 25% autres occurrences sont partagées par un groupe équitablement réparti de 10 variantes des VP.

H1	**DET+NOUN+CASE_DEF_GEN**	**H7**	NOUN+NSUFF_FEM_SG+CASE_DEF_GEN
H2	DET+NOUN+NSUFF_FEM_PL+CASE_DEF_GEN	**H8**	NOUN_PROP
H3	DET+NOUN+NSUFF_FEM_SG+CASE_DEF_GEN	**H9**	**PREP**
H4	NOUN+CASE_DEF_ACC	**H10**	PV+PVSUFF_SUBJ:3FS
H5	**NOUN+CASE_DEF_GEN**	**H11**	PV+PVSUFF_SUBJ:3MS
H6	NOUN+CASE_INDEF_GEN	**H12**	SUB_CONJ

Tableau 9.5: Liste des principaux Heads dans l'ATB

Grâce à l'étape de lexicalisation, nous pouvons prendre une idée sue les catégories lexicales Heads les plus fréquents dans l'ATB en jetant l'œil sur leur répartition entre les catégories syntaxiques qui les encapsulent. Le tableau 9.6 illustre les catégories lexicalisées en fonctions de leurs Heads. Avant d'interpréter les résultats de cette table, nous montrons dans le tableau 9.5 la liste des Heads marqués par des symboles pour gagner en espace dans le tableau 9.6.

Heads / **XP**	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**	**H8**	**H9**	**H10**	**H11**	**H12**
NP-ADV[H4]	0	0	0	2151	0	0	0	0	0	0	0	0
NP[H1]	**13066**	0	0	0	0	0	0	0	0	0	0	0
NP[H2]	0	2390	0	0	0	0	0	0	0	0	0	0
NP[H3]	0	0	4636	0	0	0	0	0	0	0	0	0
NP[H5]	0	0	0	0	**12386**	0	0	0	0	0	0	0
NP[H6]	0	0	0	0	0	4114	0	0	0	0	0	0
NP[H7]	0	0	0	0	0	0	4015	0	0	0	0	0
NP[H8]	0	0	0	0	0	0	0	2466	0	0	0	0
PP-CLR[H9]	0	0	0	0	0	0	0	0	3770	0	0	0
PP[H9]	0	0	0	0	0	0	0	0	**13751**	0	0	0
SBAR[H12]	0	0	0	0	0	0	0	0	0	0	0	2801
S[H10]	0	0	0	0	0	0	0	0	0	2605	0	0
S[11]	0	0	0	0	0	0	0	0	0	0	4340	0
VP[H10]	0	0	0	0	0	0	0	0	0	2449	0	0
VP[H11]	0	0	0	0	0	0	0	0	0	0	3941	0

Tableau 9.6: Répartition des catégories lexicales Heads entre les catégories syntaxiques

Le tableau 9.6 n'est pas complet. Il ne montre que les catégories Heads qui ont une fréquence dépassant 2000. Les plus fréquents sont les noms définis au génitif avec ou sans déterminant et les prépositions. Nous pouvons remarquer, en plus, qu'il existe des Heads associés à plusieurs catégories lexicalisées. C'est le cas des prépositions, des conjonctions et des verbes conjugués au passé avec le masculin singulier. Il est à noter aussi que la plupart des catégories lexicalisées fréquentes sont génériques, c'est à dire que leur granularité est minimale.

3.2 PCFG lexicalisée approfondie

La tâche d'induction d'une PCFG lexicalisée approfondie est réalisée à la base de l'ATB lexicalisé et minoré des catégories -NONE-. Son objectif consiste à générer la liste des règles de production décrivant les catégories syntaxiques et lexicales. Ces règles de production sont différentes à celles de la PCFG expérimentée dans le chapitre 7 vu qu'elles portent sur des catégories lexicalisées. Le tableau 9.7 suivant montre ces différentes en termes de fréquences.

XP	#R		#	#*	La règle la plus fréquente
	Avant	Après			
NP	4824	10650	110748	6039	NP[DET+NOUN+CASE_DEF_GEN] -> DET+NOUN+CASE_DEF_GEN
PP	263	817	22100	3417	PP[PREP] -> PREP NP[NOUN+CASE_DEF_GEN]
S	1230	4255	19358	1587	S[PV+PVSUFF_SUBJ:3MS] -> VP[PV+PVSUFF_SUBJ:3MS]
VP	6675	8683	15947	212	VP[PV+PVSUFF_SUBJ:3MS] -> PV+PVSUFF_SUBJ:3MS SBAR[SUB_CONJ]
SBAR	380	1512	9524	407	SBAR[IV3FS+IV+IVSUFF_MOOD:I] -> S[IV3FS+IV+IVSUFF_MOOD:I]
WHNP	22	42	4574	571	WHNP-1[REL_PRON] -> REL_PRON
ADJP	593	759	3665	303	ADJP[DET+ADJ+NSUFF_FEM_SG+CASE_DEF_GEN] -> DET+ADJ+NSUFF_FEM_SG+CASE_DEF_GEN
PRT	**14**	**14**	**2292**	1051	PRT[NEG_PART] -> NEG_PART
ADVP	5	11	539	162	ADVP[ADV] -> ADV
NAC	53	77	221	26	NAC[REL_PRON] -> CONJ SBAR[REL_PRON]
FRAG	56	64	178	88	FRAG[NOUN_NUM] -> NOUN_NUM PUNC
WHADVP	17	26	136	21	WHADVP-2[REL_ADV] -> REL_ADV
UCP	88	113	132	4	UCP[NOUN+CASE_DEF_GEN] -> NP[NOUN+CASE_DEF_GEN] PUNC CONJ SBAR[SUB_CONJ]
SBARQ	51	66	68	2	SBARQ[IV1P+IV+IVSUFF_MOOD:I] -> S[IV1P+IV+IVSUFF_MOOD:I]
PRN	20	42	65	7	PRN[NOUN+CASE_DEF_ACC] -> S[NOUN+CASE_DEF_ACC]
LST	**2**	**2**	**56**	49	LST[NOUN_NUM] -> NOUN_NUM PUNC
SQ	26	40	51	**4**	SQ[PV+PVSUFF_SUBJ:3FS] -> PRT[INTERROG_PART] VP[PV+PVSUFF_SUBJ:3FS] PUNC
CONJP	2	2	37	34	CONJP[CONJ] -> CONJ CONJ
INTJ	**1**	**1**	**10**	10	INTJ[INTERJ] -> INTERJ
X	**5**	**5**	**5**	1	X[LATIN] -> LATIN
WHPP	**3**	**3**	**3**	1	WHPP-1[PREP] -> PREP WHNP[REL_PRON]
∑	**14452**	**27184**	**174912**[16]		

Tableau 9.7: Fréquences des règles de la PCFG lexicalisée par catégorie syntaxique

[16] Nombre d'occurrences des catégories après la suppression de celle encapsulant les constituants -NONE-

A partir du tableau 9.7 ci-dessus, nous pouvons remarquer que le nombre de règles de production pour la plupart des catégories syntaxiques a largement augmenté après la lexicalisation en passant au double, triple ou même au quadruple. Cette augmentation revient à la différence de Heads affectés à une même variante d'une catégorie syntaxique. L'avantage de cette lexicalisation est qu'elle donne une information sur le contenu de la catégorie syntaxique. Pour les catégories les moins fréquentes dans l'ATB, le nombre de règles n'a pas été affecté pour l'une des deux raisons suivantes : leurs règles n'apparaissent qu'une seule fois comme pour X et WHPP, ou bien leurs règles sont très peu nombreuses comme pour INTJ, X, LST et PRT. Il est à noter que la règle donnée pour décrire les X et les WHPP n'est qu'un exemple car toutes leurs règles ont les mêmes occurrences.

A partir du même tableau, nous pouvons bénéficier du nombre d'occurrences de la règle la plus fréquente (marqué par le symbole #*) décrivant chaque catégorie syntaxique pour prendre une idée sur la répartition des règles de cette catégorie. Par exemple, la règle la plus fréquente de la catégorie SQ n'apparait que 4 fois dans l'ATB, alors que qu'il existe 39 autres règles occupant le reste des occurrences (39). Cette répartition est totale car toutes les autres règles n'ont qu'une seule occurrence. Notre PCFG lexicalisée contient également des catégories lexicales dont la partie gauche est une catégorie lexicale et la partie droite est un mot arabe. Le tableau 9.8 suivant montre quelques fréquences qui concernent ce type de règles.

XP	**#C**	**#**	**σ**	**#C***	**XP**	**#C**	**#**	**∑**	**#C***
ABBREV	1	74	3.246	10	FOREIGN	1	9	0.000	1
ADJ	97	**18108**	**9.929**	190	INTERJ	1	12	0.707	6
ADV	4	539	15.082	65	LATIN	1	28	0.481	3
CONJ	2	15070	2538.405	9900	VERB	81	**15699**	**10.217**	307
PART	11	2288	**109.013**	496	PUNC	1	16351	1946.269	5956
PRON	32	10252	**174.975**	913	PREP	1	**21803**	1127.237	4938
DET	1	12	0.000	12	NOUN	141	**68628**	**10.892**	677
DIALECT	2	19	0.235	2	TYPO	2	427	25.800	252
					∑	**379**	**169319**		

Tableau 9.8: Fréquences des règles lexicales dans l'ATB

Selon le tableau 9.8, les catégories lexicales les plus occurrentes dans l'ATB sont les noms (NOUN) et les prépositions (PREP). A la différence des PREP, les NOUN ont un très grand nombre de variantes. C'est le cas aussi des adjectifs (ADJ) et des verbes (VERB). Elles ont également le même degré de dispersion mesuré par l'écart type (σ). Plus sa valeur est faible,

plus les données sont dispersées dans l'ATB. Cette valeur est déjà nulle pour les catégories n'ayant qu'une seule dérivation comme DET ou sont équitablement réparties comme FOREIGN. La valeur de l'écart-type est très élevée pour les conjonctions (CONJ), les ponctuations (PUNC) et les prépositions (PREP) car leurs règles les plus fréquentes accaparent la plupart des occurrences.

Puisque la PCFG lexicalisée obtenue est basée sur une analyse approfondie, les suites de catégories lexicales représentant chaque règle peuvent être observées, le tableau 9.9 suivant montre quelques fréquences liées à ces suites.

XP	#	#Suite	σ	#Suite*	Head
NP	110748	31159	45.354	6039	DET+NOUN+CASE_DEF_GEN
PP	22100	11177	12.215	866	PREP
S	19358	18054	0.416	33	PV+PVSUFF_SUBJ:3MS
VP	15947	14807	0.568	44	PV+PVSUFF_SUBJ:3MS
SBAR	9524	8903	0.239	6	PV+PVSUFF_SUBJ:3MS
WHNP	4574	42	**141.604**	571	**REL_PRON**
ADJP	3665	1493	12.183	303	DET+ADJ+NSUFF_FEM_SG+CASE_DEF_GEN
PRT	2292	14	**297.757**	1051	**NEG_PART**
ADVP	539	11	55.021	162	ADV
NAC	221	179	0.285	4	SUB_CONJ
FRAG	178	77	9.955	88	NOUN_NUM
WHADVP	136	26	5.048	21	REL_ADV
UCP	132	124	0.154	2	NOUN+NSUFF_FEM_SG+CASE_DEF_GEN
SBARQ	68	66	0.123	2	SUB_CONJ
PRN	65	60	0.219	2	ADJ+CASE_INDEF_NOM
LST	56	2	29.698	49	NOUN_NUM
SQ	51	51	0.000	1	ADV
CONJP	37	2	23.334	34	CONJ
INTJ	10	1	0.000	10	INTERJ
X	5	5	0.000	1	LATIN
WHPP	3	3	0.000	1	PREP

Tableau 9.9 : Fréquences des suites de catégories lexicales par catégorie syntaxique

A travers les fréquences liées aux suites de catégories lexicales illustrées par le tableau 9.9, nous pouvons prendre une idée sur le degré de dispersion (σ) de ces suites pour chaque catégorie syntaxique. Ainsi, les catégories NP, WHNP et PRT sont très peu dispersées. Les Heads de leurs suites les plus fréquentes (NOUN, PRON et PART) sont également peu dispersées comme

le montre le tableau 9.8 précédant. Les catégories les plus dispersées sont X, S, SBAR, UCP, SBARQ, PRN et NAC. Cela est déjà montré souvent par le faible nombre d'occurrences de chacune de leur suites les plus fréquentes (#Suite*). Les autres suites encapsulées dans la même catégorie et définies par la valeur de l'entête #Suite ont alors une part importante du nombre total des occurrences de cette catégorie.

3.3 PCFG lexicalisée approfondie sous CNF

Notre PCFG lexicalisée a pris la forme CNF pour qu'elle puisse être utilisée par l'algorithme CYK. Nous rappelons que les règles de cette forme ne peuvent être que deux types : des règles à partie droite binaire sans description des mots ou bien unaire décrivant un seul mot. Dans ce cas notre PCFG doit subir un ensemble de traitements balançant entre la suppression d'un ensemble de règles et l'ajout de règles intermédiaires, ou encore la modification de la partie droite des règles originales. Le tableau 9.10 suivant montre l'effet de ces modifications sur notre PCFG.

XP	**#R**		**#R ajoutées**	**#R supprimées**	**#R modifiées**
	Avant	Après			
ADVP	11	408	397	7	4
CONJP	2	37	35	**0**	2
FRAG	64	284	220	8	56
INTJ	1	11	10	1	**0**
LST	2	56	54	1	1
NAC	77	220	143	**0**	77
NP	10650	105895	95245	1015	9635
PP	817	22356	21539	3	814
PRN	42	100	58	9	33
PRT	14	2292	0	0	**0**
S	4255	25601	21346	252	4003
SBAR	1512	10082	8570	159	1353
SBARQ	66	160	94	3	63
SQ	40	119	79	**0**	40
UCP	113	24663	24550	1	112
VP	8683	30684	22001	33	8650
WHADVP	26	126	100	26	**0**
WHNP	42	2442	2400	38	4
WHPP	3	3	0	0	3
X	5	36	31	2	3
∑	27184	229694	202510	1634	25550

Tableau 9.10: Fréquences des règles syntaxiques de la PCFG lexicalisée sous CNF

Dans le tableau 9.10, le nombre de règles de la PCFG lexicalisée a augmenté après leur conversion à la forme CNF, sauf celui des règles décrivant la catégorie WHPP. Ceci s'explique par l'absence de règles à supprimer ou à ajouter. La suppression des règles concerne celle syntaxiques décrivant une seule catégorie grammaticale dans leurs parties droites. La modification concerne les autres règles syntaxiques. Cette modification est de deux types, ou bien un remplacement dans la règle courante de toute catégorie syntaxique par son unique catégorie lexicale qui l'encapsule dans une règle déjà supprimée, ou un étiquetage de la plupart de la partie droite de la règle courante par une nouvelle catégorie pour assurer la forme binaire exigée par la CNF. Tandis que l'ajout des règles, il concerne celles décrivant les catégories nouvellement créées. La modification d'une règle exige l'ajout de nouvelles autres règles lorsque la partie droite de cette règle est composée de plus que deux catégories grammaticales.

En se basant sur les définitions précédentes, nous pouvons expliquer la raison pour laquelle la catégorie WHPP n'a pas de règles à supprimer ou à ajouter car toutes leurs parties droites sont composées de deux catégories grammaticales. Elle a par contre des règles à modifier car au moins l'une des catégories de chacune de leurs parties droite est une catégorie syntaxique encapsulant une seule catégorie lexicale. Cette catégorie syntaxique doit être remplacée par cette catégorie lexicale. Il est à noter que les catégories NAC, SQ et CONJP n'ont pas également des règles à supprimer puisqu'aucune d'elles n'est unaire. Les catégories INTJ, PRT et WHADVP n'ont pas par contre des règles à modifier puisqu'aucune d'elles ne contienne dans sa partie droite une catégorie syntaxique encapsulant une seule catégorie lexicale. Les catégories qui ont été très affectées par la suppression, la modification et l'ajout de règles sont NP, VP, S et SBAR. Il est à noter que dans la PCFG lexicalisée sous forme CNF, les règles lexicales ne sont pas affectées puisqu'elles respectent cette forme. Globalement, le nombre des règles après la conversion de la PCFG lexicalisée à la forme CNF se sont multipliées par plus que 8 malgré la suppression de plusieurs règles originales.

3.4 PGP lexicalisée sous CNF

La PGP lexicalisée sous CNF représente le deuxième composant de notre modèle d'apprentissage. Celui-ci permet de caractériser chaque règle du premier composant par les différents types de propriétés qu'elle peut exprimer entre les unités de sa partie droite. Ces propriétés entre dans le calcul de la probabilité de cette règle dans toute la PCFG. Les propriétés sont de six types : la constituance, l'unicité, l'obligation, la linéarité, l'exigence et l'exclusion.

Le tableau 9.11 suivant illustre les fréquences de ces propriétés par catégorie syntaxique et par type de propriétés.

XP	Constituance		Unicité		Obligation		Linéarité		Exigence		Exclusion		∑	
ADJP	190	1263	145	1222	125	119	357	878	84	1068	5819	3658	6720	8208
ADVP	69	7	69	7	65	3	4	4	5	6	20	1	232	28
CONJP	3	3	2	2	2	2	1	1	2	2	2	0	12	10
FRAG	29	286	22	282	19	25	60	145	34	272	300	92	464	1102
INTJ	1	0	1	0	1	0	0	0	0	0	0	0	3	0
LST	2	2	2	2	2	1	1	1	1	2	0	0	**8**	**8**
NAC	53	183	52	183	38	63	41	100	39	172	185	126	408	827
NP	2361	17581	2052	17148	1566	1571	2500	12906	1252	15008	115076	205499	124807	269713
PP	248	1326	222	1318	105	98	196	1048	215	1223	1017	39132	**2003**	**44145**
PRN	15	126	11	125	10	13	9	67	12	117	52	20	109	468
PRT	14	0	14	0	14	0	0	0	0	0	91	0	133	0
S	266	11559	242	11434	83	306	671	7012	297	10472	17745	135889	**19304**	**176672**
SBAR	382	2138	345	2128	265	268	306	1612	369	1856	3284	18714	4951	26716
SBARQ	35	275	31	270	8	19	64	139	53	262	450	124	641	1089
SQ	19	198	16	191	10	15	43	100	36	182	117	138	241	824
UCP	64	11904	53	11782	34	342	74	7194	95	10810	207	136027	**527**	**178059**
VP	439	15841	421	15708	160	139	3929	11859	497	13625	90113	427009	**95559**	**484181**
WHADVP	34	0	34	0	34	0	0	0	0	0	20	0	122	0
WHNP	66	8	66	8	62	4	4	4	6	8	72	0	276	32
WHPP	6	6	6	6	3	3	3	3	6	6	0	0	**24**	**24**
X	6	67	3	66	3	1	3	33	8	66	8	4	31	237
∑	**4302**	**62773**	**3809**	**61882**	**2609**	**2992**	**8266**	**43106**	**3011**	**55157**	**234578**	**966433**	256575	1192343

Tableau 9.11: Fréquences des propriétés syntaxiques par catégorie syntaxique et par type

Selon le tableau 9.11 ci-dessus, le nombre de propriétés décrivant les catégories syntaxiques a largement augmenté pour la plupart des cas. Ceci revient à l'augmentation du nombre de catégories non originales, ayant été crées pour adapter les règles de la PCFG lexicalisée à la forme CNF. Les catégories qui n'ont pas été majorées par d'autres non originales n'ont pas vécu ces changements de valeurs. C'est le cas notamment des catégories WHPP et LST. Les catégories qui ont été par contre très affectées sont les VP, S, PP et UCP.

En termes de types, les propriétés d'obligation sont les moins affectées à cause de leur quasi-absence dans la description des catégories non originales, où le Head représente, dans la plupart des cas, le premier composant de la règle traitée. En plus, les propriétés de constituance,

d'unicité et d'exigence marquent une augmentation considérable dans leurs parts au profit des propriétés d'exclusion, vu que l'interprétation du type de ces dernières se base la détermination des cas où deux constituants d'une catégorie syntaxique n'apparaissent jamais ensembles dans les règles qu'elle décrit. Dans ce cas, plus le nombre de constituants est élevé, plus il existe des non apparitions des couples de ces constituants, alors que les catégories non originales se caractérisent par un nombre réduit de constituants.

4 Caractéristiques des résultats d'analyse avec l'algorithme CYK

L'algorithme d'analyse représente le deuxième composant de notre analyseur syntaxique de propriétés. Cet algorithme effectue une stratégie d'analyse ascendante de chaque phrase dans le corpus de test pour générer son arbre d'analyse syntaxique. Les structures syntaxiques de cet arbre sont décrites en plus par des propriétés syntaxiques. Pour effectuer ses choix d'analyse, cet algorithme utilise les règles, les propriétés et leurs probabilités obtenues à partir du modèle d'apprentissage. Pour évaluer les arbres d'analyse générer par notre analyseur, nous avons réalisé un ensemble d'expérimentations aussi bien sur notre analyseur et les poids de ses propriétés que sur l'analyseur de l'état de l'art, Stanford Parser à qui nous avons comparé nos résultats. Les deux sous-sections suivantes se focalisent respectivement sur la présentation de ces différentes expérimentations.

4.1 Expérimentations d'un analyseur état de l'art : Stanford Parser

Nous allons commencer à présenter et interpréter les résultats d'évaluation d'un analyseur syntaxique, état de l'art, à savoir : Stanford Parser. Ceci revient à évaluer les analyses qu'il produit sur un corpus de test donné. Comme nous l'avons déjà mentionné dans la section 2.2 de ce chapitre, notre corpus de test est divisé en 4 parties en fonction du nombre de mots (ou la longueur) de ses phrases. L'évaluation des résultats d'analyse des phrases de ces différentes parties se base sur une comparaison entre les analyses fournies par Stanford Parser et celles fournies par le treebank de référence ATB.

Pour réussir cette comparaison, la série de recommandations suivante doit être appliquée :

- Ajuster les mots arabes du résultat d'analyse de Stanford Parser à leurs écritures translitérées comme dans l'ATB.
- Ajuster l'ATB à l'étiquetage du PTB adopté par les résultats de Stanford Parser,

- Utiliser la mesure d'évaluation ParsEval pour comparer les arbres syntaxiques des phrases de Stanford Parser à ceux de l'ATB,
- Enrichir l'ATB ajusté avec les propriétés d'une GP à traits reconnus par l'étiquetage PTB,
- Enrichir les arbres d'analyse de Stanford Parser avec les propriétés de la même GP.
- Utiliser les mesures de rappel et précision pour comparer les propriétés décrites.

Selon ces recommandations, les comparaisons seront faites à deux niveaux : le niveau des arbres d'analyse et le niveau des propriétés. Elles portent également sur deux étiquetages de catégories différents : celui de l'ATB et celui du PTB. Le premier caractérise l'ATB et notre analyseur, l'autre est utilisé par Stanford Parser. Les résultats d'évaluation de l'analyseur Stanford Parser au niveau des arbres d'analyse sont présentés dans le tableau 9.12 ci-dessous :

Longueur	**Précision**		**Rappel**		**F-mesure**		**Cross-brackets**		**Temps (sec)**
	basique	**cumulée**	**basique**	**cumulé**	**basique**	**cumulée**	**basique**	**cumulée**	
[1, 10]	**82,24**	-	80,96	-	81,59	-	0,84	-	**12**
[11, 20]	78,80	80,52	76,98	78,97	77,88	79,74	3,79	2,31	25
[21, 30]	72,80	77,95	72,25	76,73	72,53	77,33	6,75	3,79	63
[31, 40]	70,08	75,98	69,75	74,99	69,92	**75,48**	9,35	5,18	**110**

Tableau 9.12: Degrés de performance des arbres d'analyses générés par Stanford Parser

Le tableau 9.12 montre que plus la longueur des phrases analysées par Stanford Parser augmente, plus les valeurs des mesures de rappel et de précision diminuent. Les meilleures valeurs marquées ne dépassent pas les 82, 24 %. Ces valeurs se détériorent d'environ 12 % en traitant des phrases de longueur comprise entre 30 et 40 mots. Ces valeurs se stabilisent pour un corpus cumulant toutes les phrases. La valeur du score F-mesure qui rassemble les mesures de rappel et de précision pour ce corpus atteint 75,48%. Tout comme les valeurs de rappel et de précision, les valeurs des mesures cross-brackets sont affectées négativement de manière très remarquable proportionnellement avec l'augmentation de la longueur des phrases dans le corpus. Nous rappelons que cette mesure calcule le pourcentage des structures syntaxiques dans le résultat d'analyse de Stanford Parser qui se chevauchent avec celles du corpus de référence. Par conséquent, plus cette valeur est faible, plus le corpus est correctement segmenté selon le corpus de référence. Le temps d'exécution de l'analyse des phrase s'incrémente également de manière exponentielle avec la longueur des phrases en se multipliant par environ 9 fois le temps d'analyse d'une phrase composée d'au plus 10 mots (12/110). En observant le rapport entre les valeurs

de mesure de la précision et du rappel pour un même corpus de phrases, nous pouvons remarquer que ces valeurs sont très rapprochées pour la plupart des corpus, ce qui indique que l'analyseur Stanford Parser est pertinent autant qu'il est précis.

Pour les mesures de rappel (Rap.) et de précision (Préci.) concernant les propriétés et qui sont indiquées dans les recommandations ci-dessus, elles nous permettent de calculer le degré de similarité entre les propriétés de l'ATB et celles correspondantes données par l'analyseur à évaluer.

	Constituance		**Unicité**		**Obligation**		**Linéarité**		**Exigence**		**Exclusion**		**Σ**	
	Préci.	**Rap.**	**Préci.**	**Rap.**	**Préci.**	**Rap.**	**Préci.**	**Rap.**	**Préci.**	**Rap.**	**Préci.**	**Rap.**	**Préci.**	**Rap.**
[1, 10]	72,55	71,83	64,47	**80,33**	66,67	**67,15**	28,83	28,07	0,00	0,00	63,64	63,73	65,82	66,10
[11, 20]	75,29	**73,82**	64,80	65,13	67,96	66,85	39,64	**35,05**	10,53	9,52	65,43	**65,01**	68,01	66,95
[21, 30]	69,86	70,01	64,69	71,13	61,63	62,45	34,98	33,26	**33,33**	**27,91**	60,00	62,12	62,53	63,72
[31, 40]	65,39	65,53	60,64	66,13	58,78	59,51	24,01	22,78	18,52	17,24	57,59	58,65	59,22	59,90

Tableau 9.13: Degrés de performance des propriétés syntaxiques décrivant les analyses de Stanford Parser

A partir du tableau 9.13 dressé ci-dessus, nous pouvons remarquer que les valeurs de précison et de rappel pour les propriétés ne sont pas toujours proches comme pour les arbres syntaxiques. Des fois, la précision dépasse d'une manière remarquable le rappel comme pour les propriétés de linéarité pour le corpus [11, 20] et les propriétés d'exigence pour le corpus [21, 30]. Des fois, nous trouvons le contraire, c'est-à-dire que le rappel prend la première position. C'est le cas notamment des propriétés d'unicité pour tous les corpus sauf [11, 20] et les propriétés d'exclusion pour le corpus [21, 30]. Ceci indique que la pertinence de l'analyseur Stanford Parser domine sa précision.

Nous avons observé également que les propriétés unaires (constituance, unicité et obligation) sont beaucoup plus représentées dans le résultat Stanford Parser. Par contre, il y a des cas où aucune propriété ne peut être observée. C'est le cas des propriétés d'exigence pour le corpus [1, 10]. Ceci est dû au fait que l'ATB emploie dans plusieurs cas des règles dont les catégories syntaxiques annotent une seule catégorie lexicale (e.g. NP → NOUN, ADJP → ADJ). Les propriétés sur ces constituants uniques ne peuvent être que de forme unaire et seront souvent satisfaites. Le regroupement de ces deux catégories lexicales n'est pas donc fait directement mais à travers les catégories syntaxiques NP et ADJP. Les propriétés binaires vont

porter sur ces catégories syntaxiques contrairement aux annotations de Stanford Parser qui regroupent ces catégories lexicales directement dans une catégorie syntaxique.

4.2 Comparaison de notre analyseur avec Stanford Parser

Vu que nos analyses ont le même étiquetage de catégories et la même écriture translitérée que ceux illustrés par l'ATB, la comparaison entre nos analyses et l'ATB peut être faite directement. Cette comparaison est également à deux niveaux ; les arbres syntaxiques et les propriétés. Au niveau des arbres syntaxiques, elle est réalisée également grâce aux mesures d'évaluation de ParsEval. Le tableau 9.14 suivant indique les résultats obtenus de cette évaluation aussi bien pour l'analyseur Stanford Parser que notre analyseur.

Longueur	**Précision**		**Rappel**		**F-mesure**		**Cross-brackets**		**Temps (sec)**	
	Stanford	**Notre**	**Stanford**	**Notre**	**Stanford**	**Notre**	**Stanford**	**Notre**	**Stanford**	**Notre**
[1, 10]	82,24	**86,81**	**80,96**	70,2	**81,59**	77,62	0,75	2,26	12	7
[11, 20]	78,8	75,52	76,98	67,47	77,87	71,26	3,79	13,11	25	26
[21, 30]	72,8	65,23	72,25	59,45	72,52	62,20	6,75	17,58	63	84
[31, 40]	70,08	61,68	69,75	52,18	69,91	56,53	9,35	20,43	110	193

Tableau 9.14 : Degrés de performance des arbres d'analyses générés par Stanford Parser et par notre analyseur (Notre)

Le tableau 9.14 montre que notre analyseur se caractérise par une précision plus élevée que celle de Stanford Parser en évaluant les analyses du corpus [1, 10]. Le rappel et le cross-brackets de nos analyses sont par contre plus faibles que Stanford Parser. Ceci indique que notre analyseur génère presque le vrai volume d'annotations. Mais, ce ne sont pas toutes des annotations correctes. En plus, pour tous les corpus, les valeurs de cross-brackets élevées montrent que notre regroupement des catégories lexicales en segments n'est pas correct, ce qui n'est pas le cas de Stanford Parser. Le temps d'analyse par phrase du corpus [1, 10] par notre analyseur est plus faible que celui de Stanford Parser. Ce temps augmente d'une manière exponentielle pour finir par dépasser celui de Stanford Parser d'environ 44%. Il est à noter que pour notre analyseur, ce temps est consacrée à la fois à la génération de l'arbre d'analyse de la phrase et implicitement la détermination des propriétés syntaxiques qui décrivent les différentes catégories. Tandis que l'analyseur Stanford Parser fait ces deux tâches d'une manière séparée. L'enrichissement de ses arbres d'analyse avec des propriétés prend encore plus de temps.

Pour évaluer les propriétés que notre analyseur peut générer pour notre corpus de test, nous avons enrichi les phrases de référence (tirées de l'ATB) avec les propriétés de la GP à la

granularité maximale. La technique d'enrichissement utilisée est celle que nous avons décrit dans (Bensalem et al., 2015a ; Bensalem et al., 2015b ; Bensalem et al., 2016). Le tableau 9.15 suivant illustre les résultats d'évaluation de notre analyseur en comparaison avec Stanford Parser au niveau de la description des propriétés.

Type	**Constituence**				**Unicité**				**Obligation**			
	Précision		**Rappel**		**Précision**		**Rappel**		**Précision**		**Rappel**	
Corpus	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre
[1, 10]	71,34	79,95	68,67	68,41	64,47	76,36	**80,33**	71,48	66,67	**80,05**	67,15	**79,84**
[11, 20]	75,29	76,47	73,82	65,59	64,8	73,81	65,13	72,28	67,96	78,32	66,85	76,01
[21, 30]	69,86	74,55	70,01	64,24	64,69	70,69	71,13	68,47	61,63	75,13	62,45	75,67
[31, 40]	65,39	72,82	65,53	61,38	60,64	69,64	66,13	65,11	58,78	74,07	59,51	73,09
Type	**Linéarité**				**Exigence**				**Exclusion**			
	Précision		**Rappel**		**Précision**		**Rappel**		**Précision**		**Rappel**	
Corpus	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre	Stanf.	Notre
[1, 10]	28,83	**49,54**	28,07	**46,14**	0,00	**44,73**	0,00	**42,47**	63,64	**77,42**	63,73	**75,44**
[11, 20]	39,64	46,98	35,05	42,39	10,53	38,69	9,52	37,22	65,43	72,09	65,01	69,46
[21, 30]	34,98	41,75	**33,26**	35,07	33,33	32,81	27,91	29,48	60,00	68,71	62,12	61,29
[31, 40]	24,01	35,32	22,78	26,49	18,52	26,94	17,24	24,36	57,59	61,82	58,65	57,56

Tableau 9.15 : Degrés de performance des propriétés décrivant les résultats de notre analyseur (Notre) en comparaison avec ceux de Stanford Parser (Stanf.)

Selon le tableau 9.15 ci-dessus, nous pouvons remarquer qu'à la différence de son évaluation au niveau des arbres syntaxiques, notre analyseur apporte une performance plus élevée pour la plupart des types de propriétés que celle de Stanford Parser. Même, les propriétés d'exigence, elles sont assez représentées pour le corpus [1, 10] et elles décrivent correctement un bon nombre de catégories syntaxiques. Les propriétés binaires sont également mieux correctement décrites que dans les analyses de Stanford Parser. Ces constatations peuvent être expliquées par les deux raisons suivantes : Premièrement, les arbres de notre analyseur sont construits à la base de règles dont les probabilités sont estimées sur les fréquences de propriétés dans le corpus d'apprentissage. Deuxièmement, ces règles prennent les formes des règles appliquées pour générer les phrases de référence. En effet, à la différence de Stanford Parser, notre analyseur emploie des règles encapsulant une seule catégorie syntaxique tout comme l'ATB duquel nous avons extrait les phrases de référence. Les propriétés binaires seront plus observées dans les arbres de notre analyseur que dans ceux de Stanford Parser.

5 Conclusion

Pour conclure, notre analyseur syntaxique probabiliste de propriétés offre de nouvelles informations en plus des relations hiérarchiques représentées par les arbres d'analyses des autres analyseurs. Stanford Parser forme déjà un exemple auquel nous avons comparé notre analyseur. L'évaluation de ce dernier n'est donc pas liée seulement à la structuration des mots du corpus, mais elle repose sur la description des propriétés relatives à chaque structure syntaxique construite. Cet analyseur utilise un modèle d'apprentissage très informatif exploitant outre les règles de production, des informations implicites à savoir les suites de catégories lexicales, les Heads associés à chaque catégorie syntaxique et finalement les propriétés syntaxiques. L'utilisation des propriétés syntaxiques pour générer des arbres d'analyses semble être une idée bénéfique vu qu'elle apporte des résultats encourageants. La description de ces propriétés dans les résultats d'analyse apporte encore plus d'avantage surtout lorsqu'il s'agit d'une analyse à la base de ces propriétés et non pas un simple enrichissement d'une analyse déjà générée. Ceci peut optimiser les analyses à construire en permettant d'intégrer une pondération des propriétés ou d'ajouter plusieurs autres types de propriétés.

Conclusion générale

Nous avons présenté dans ce mémoire de thèse l'ensemble des contributions que nous avons appliquées pour obtenir de nouvelles ressources syntaxiques arabes. Les ressources existantes arabes sont fournies sous forme de représentations structurelles à base de constituants ou à base de dépendances. Les premières représentent la phrase sous forme d'arbres syntaxiques où les feuilles sont les mots de la phrase et les nœuds sont hiérarchiquement organisés pour annoter les mots ou les segments de mots. Ces représentations ne fournissent ni des informations sur l'ordre des unités dans une structure syntaxique, ni les unités essentielles de cette structure, ni les dépendances entre elles. L'autre type fournit ces relations mais seulement entre les mots de la phrase, sans donner une liaison structurelle regroupant ces mots. Il n'existe donc pas des nœuds intermédiaires annotant les segments de mots. Dans les deux cas de représentations des informations restent manquantes, alors qu'elles sont implicitement présentes.

Le formalisme de grammaires de propriétés offre une nouvelle représentation qui combine les informations des deux représentations indiquées et leur ajoute d'autres implicites. Il décrit, en effet, les catégories des structures syntaxiques en utilisant différents types de propriétés. Ces propriétés expriment différentes relations syntaxiques pouvant exister entre les différentes unités de cette structure. Ces relations peuvent être unaires portant par exemple sur la constituance, l'unicité et l'obligation d'apparition (ou Head), ou bien binaires, portant ainsi sur la précédence linéaire, l'obligation de cooccurrence (ou exigence) et l'interdiction de cooccurrence (ou exclusion). Ce sont des contraintes indépendantes les unes des autres. La vérification de la satisfaction de ces contraintes se fait d'une manière directe sur les catégories sans avoir besoin de construire une structure locale des informations syntaxiques comme le font d'autres approches basées sur les contraintes. Nous avons alors utilisé ce formalisme pour construire de nouvelles ressources linguistiques arabes à la base d'autres existantes.

La grammaire de propriétés pour l'arabe n'était pas disponible. C'était donc notre première ressource à construire. Pour ce faire, il nous a fallu tout d'abord de choisir les ressources linguistiques existantes qui conviennent aux besoins de création de cette grammaire. Une étude des différentes spécificités de la langue arabe, qui sont traitées par ces ressources était nécessaire. Le treebank arabe ATB était la ressource principale à exploiter pour intégrer le

formalisme de grammaires de propriétés vu ses plusieurs qualités comme la représentation structurelle hiérarchique adaptable au formalisme de grammaires de propriétés, la richesse de ses catégories en informations morphosyntaxiques variées et la validation de ses annotations par des linguistes. La deuxième ressource linguistique utilisée était les règles de Habash et al., (2009) permettant de générer le treebank CATiB. Elles nous ont permis de générer les propriétés d'obligation. La grammaire de propriétés résultante hérite les qualités des treebanks d'origine et fournit en plus de nouveaux types d'informations. Nous avons exploité cette grammaire pour construire plusieurs autres ressources. Ainsi, nous avons pu générer un treebank enrichi de propriétés syntaxiques à la base du treebank ATB, ainsi que des résultats d'analyses de Stanford Parser enrichis et évalués. Nous avons également construit un analyseur syntaxique probabiliste de propriétés en intégrant dans son modèle d'apprentissage une version améliorée de la grammaire de propriétés déjà construite. Cette version se base sur une analyse approfondie de l'ATB et sur une lexicalisation de ses catégories.

La technique que nous avons adoptée pour induire notre grammaire à partir du treebank ATB consiste à induire, à partir de l'ATB, un modèle intermédiaire, qui est la grammaire à contexte libre. Cette grammaire est composée de toutes les règles de production induites automatiquement de l'ATB. Nous avons appliqué des interprétations par type de propriétés sur ces règles pour les générer automatiquement. Cette technique d'induction automatique présente l'avantage d'être générique. En effet, elle est indépendante non seulement de toute langue, mais aussi, du formalisme source, puisque la génération des propriétés se fait directement à partir de la grammaire à contexte libre.

Nous avons appliqué différents mécanismes de contrôle pour obtenir ces produits. Ces mécanismes sont liés à plusieurs critères : la réduction de la complexité des composants de la grammaire de propriétés produite, l'interprétation des propriétés, la vérification de leur satisfaction et leur intégration dans l'ATB et dans le résultat de Stanford Parser. Ces mécanismes concernent également l'exploitation des propriétés pour évaluer Stanford Parser ainsi que leur représentation dans l'algorithme d'analyse.

Parmi les formalismes que nous avons appliqués dans le cadre de ces mécanismes, il y a le formalisme d'hiérarchie de types, qui nous a facilité le développement d'un outil de contrôle de la granularité de notre grammaire de propriétés. Ceci nous a permis aussi de prendre une idée sur le degré de l'importance de chacun des traits morphosyntaxiques fournis dans la structure des catégories de l'ATB. Plus un trait est important, plus sa présence implique la création de

nouvelles variantes de sa catégorie. Plus le niveau de granularité des catégories est élevé, plus les grammaires de propriétés et à contexte libre sont complexes mais leurs propriétés et règles de plus en plus fidèles à la langue et à l'ATB d'origine, et inversement. C'est le cas notamment des traits "cas syntaxique" et "définition" caractérisant les noms et les adjectifs. Leur présence dans la structure des catégories améliore largement le nombre de règle de productions et le nombre de propriétés décrites.

La lexicalisation des catégories syntaxiques de l'ATB dans le modèle d'apprentissage de notre analyseur syntaxique nous a permis également de prendre une idée sur le contenu des structures syntaxiques dans l'ATB en associant le constituant obligatoire principal (ou Head) à chaque catégorie syntaxique caractérisant une certaine structure syntaxique. Nous avons collecté des informations comme l'ensemble des Heads les plus fréquents dans l'ATB. Cela peut être utile dans la phase d'analyse d'un texte brut en imposant un certain Head dans une structure dans le but d'orienter cette analyse dans le bon sens. L'analyse approfondie, appliquée dans le cadre du modèle d'apprentissage, est un autre mécanisme de contrôle permettant de citer les différentes suites de catégories lexicales possibles relatives à chaque règle de production. L'opportunité de choisir une telle règle est relié non pas seulement à son nombre d'occurrences dans le corpus d'apprentissage, mais aussi à la présence de l'une de ses suites de catégories lexicales dans le texte arabe à analyser. L'induction des propriétés à partir des règles de production adaptées à la forme CNF, a nécessité un traitement spécial. Ceci revient à l'adoption d'interprétations des propriétés appliquées spécifiquement à des règles virtuelles créées selon les exigences de cette forme pour garder la même information fournie par les règles d'origine.

Pour l'enrichissement de ressources avec des représentations syntaxiques, nous avons intégré des méthodes de contrôle, appelées solveurs de contraintes. Ces solveurs nous ont permis de vérifier la satisfaction des propriétés d'une catégorie syntaxique dans chacune des structures syntaxiques de la ressource à enrichir. Un autre contrôle a été réalisé sur la correspondance entre la catégorie syntaxique caractérisée par les propriétés enrichissantes et celle décrivant la structure syntaxique à enrichir. Cela nous a permis de prendre en compte le cas où la granularité de notre grammaire est différente de celle de la ressource à enrichir, et ce par la variation de la granularité des catégories de notre grammaire en fonction de la granularité des catégories de la ressource à enrichir. En plus, l'utilisation des tables de correspondance entre l'étiquetage des catégories de notre grammaire et celui des catégories des résultats d'analyse de Stanford Parser, nous a aidés à réussir l'enrichissement de ces résultats. A ces

résultats plus particulièrement, nous avons adopté un mécanisme de contrôle de leur performance en proposant un ensemble de mesures d'évaluation calculées sur les fréquences des différentes propriétés de notre grammaire.

Dans la continuité du présent travail, diverses perspectives s'ouvrent tout en méritant une étude plus approfondie. Nous présentons celles d'entre-elles qui nous semblent les plus prometteuses. Tout d'abord, vu que nos techniques d'induction, d'enrichissement et d'analyse de propriétés sont automatiques, nous pouvons les réutiliser pour des treebanks de différentes langues. La technique d'enrichissement peut être également appliquée sur les résultats d'autres analyseurs. Cela nécessite tout simplement la disponibilité d'une table de correspondance entre l'étiquetage des catégories de notre grammaire et ceux des autres analyseurs. Une comparaison avec notre analyseur de propriétés, en termes de nombre de propriétés décrites, peut être utile.

Dans le but d'offrir une représentation très précise de l'information syntaxique, l'ensemble des relations présentées dans notre grammaire peut toujours être enrichi ou modifié. En effet, le nombre volumineux et anormale des propriétés d'exclusion pourra être réduit en proposant de nouvelles interprétations prenant en comptes un ensemble de facteurs comme la position dans la structure, la catégorie de la structure, ses occurrences, la symétrie entre les unités en relation exclusion possible et leurs occurrences. Il est également possible d'ajouter des interprétations à de nouveaux types de propriétés syntaxiques comme l'accord et les dépendances. Les propriétés peuvent couvrir en plus des niveaux d'analyse (e.g. sémantique et morphologique). La disposition du treebank dans ses différents formats peut aider à extraire des informations comme la traduction anglaise d'un mot arabe. Ceci offre la possibilité de se lancer dans les axes d'analyses sémantique et morphologique des mots arabe.

Nous envisageons également de profiter des propriétés syntaxiques pour construire des modèles d'apprentissage à base de classification. Ces propriétés peuvent être les critères selon lesquelles les représentations syntaxiques sont classées. Il faut pour cela représenter les propriétés de manière à faciliter sa quantification. Nous pouvons aller plus loin en transformant notre analyseur de propriétés à une version hybride. Cette version combine un ensemble de règles préparées manuellement par des experts avec les règles du modèle d'apprentissage, qui sont induites automatiquement à partir de l'ATB. Nous souhaitons par ceci renforcer le traitement des phénomènes linguistiques particuliers dans notre analyseur probabiliste de propriétés syntaxiques.

Bibliographie

Abdelali, A., Darwish, K., Durrani, N., Mubarak, H. (2016) . Farasa: A Fast and Furious Segmenter for Arabic.

Abeillé, A. (1988). Parsing French with Tree Adjoining Grammar: Some Linguistic Accounts. Proceedings of the 12th International Conference on Computational Linguistics, pp. 7-12. Budapest, Hungary.

Abeillé, A. (2003). Treebanks: Building and Using Parsed Corpora, Kluwer Academic Publishers, Dordrecht.

Abeillé, A., Clément, L., Toussenel, F. (2003). Building a Treebank for French. In: Abeillé 2003, 165-187.

Afonso, S., Bick, E., Haber, R., Santos, D. (2002). Floresta Sintá(c)tica, a Treebank for Portuguese. In: Proceedings of the Third International Conference on Language Resources and Evaluation. Las Palmas, Spain, 1698-1703.

Al-Daoud, E., Basata, A. (2009). A Framework to Automate the Parsing of Arabic Language Sentences, in The International Arab Journal of Information Technology, Vol. 6, 192 No. 2.

Attia, M., Shaalan, K. , Tounsi, L.,Van Genabith, J. (2012). Automatic Extraction and Evaluation of Arabic LFG Resources, Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), European Language Resources Association (ELRA)

Attia, M. (2006). 'An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks'. The Challenge of Arabic for NLP/MT Conference. The British Computer Society, London.

Ayed, R., Bounhas, I., Elayeb, B., Ben Saoud, N. B., Evrard, F. (2014) Improving Arabic Texts Morphological Disambiguation Using a Possibilistic Classifier. NLDB 2014: 138-147.

Bar-Hillel, Y. (1953). A Quasi-Arithmetical Notation for Syntactic Description. Language 29(1): 47-58.

Beesley, K. (2001). Finite-State Morphological Analysis and Generation of Arabic at Xerox Research. Status and Plans.

Bensalem, R.B., Kadri, N., Haddar, K., Blache, P. (2017). Evaluation and enrichment of Stanford Parser using an Arabic Property Grammar.April, Hungary.

Bensalem, R.B., Haddar, K., Blache, P. (2016) A Property Grammar-Based Method to Enrich the Arabic Treebank ATB. Chapter in the book Knowledge Discovery, Knowledge Engineering and Knowledge Management, Springer Verlag, January.

Bensalem, R.B., Haddar, K., Blache, P. (2015) A Formal Modeling Method to Enrich the Arabic Treebank ATB with Syntactic Properties. In: the proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD) , November, Portugal.

Bensalem, R.B., Haddar, K., Blache, P. (2015). Enrichment of the Arabic Treebank ATB with Syntactic Properties in Proceedings of CicLing, April, Egypt.

Bensalem, R.B., Elkaroui, M., Haddar, K., Blache, P. (2014). Building an Arabic Linguistic Resource from a treebank: The Case of Property Grammar. In the proceedings of Text, Speench and Dialogue (TSD'14), pp.240-246 , September, Czech Republic.

Bensalem, R.B., Elkaroui, M., Haddar, K., Blache, P. (2014). Induction d'une grammaire de propriétés à granularité variable à partir du treebank arabe ATBRaja Bensalem Bahloul, Marwa Elkarwi in in recital (atala), Juillet, France.

Bick, E. (2003). Arboretum, a Hybrid Treebank for Danish. In: Nivre et Hinrichs , 9-20.

Bikel, D. (2004). Intricacies of Collins‟ Parsing Model. Computational Linguistics, 30:4 (479-511).

Bikel, D.M., Chiang, D. (2000) Two statistical parsing models applied to the Chinese Treebank. In Proceedings of the Second Workshop on Chinese Language Processing, 1-6.

Blache, P. (2000). Le rôle des contraintes dans les théories linguistiques et leur intérêt pour l'analyse automatique : les Grammaires de Propriétés Conférence TALN 2000, Lausanne, France.

Blache, P. (2001) Les Grammaires de Propriétés : Des contraintes pour le traitement automatique des langues naturelles. Hermès science publications, 228 pages.

Blache, P., Rauzy, S. (2012). Hybridization and Treebank Enrichment with Constraint-Based Representations. In: Proceedings of LREC-2012.

Blache, P. (2014). A Chinese Constraint Grammar Extracted from the Chinese Treebank. In: Proceedings of APCLC.

Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R. and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In Proceedings of the 5th DARPA Speech and Natural Language Workshop, 31–37.

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., Strzalkowski, T. (1991). A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In: Proceedings of the DARPA Speech and Natural Language Workshop. Pacific Grove, CA, 306-311.

Bloomfield, L.(1933) . Language. Holt, Rinehart and Winston, New York, New York, USA.

Bod, R., Scha, R., Sima'an, K. (2003). Data-Oriented Parsing. CSLI Publications, University of Chicago Press, Illinois, USA.

Bod, R.(1998). Beyond Grammar: An Experience-based Theory of Language. Cambridge University Press, Cambridge, England.

Boella, M., Romani, F.R., Al-Raies, A., Solimando, C., Lancioni, G. (2011). The SALAH Project: Segmentation and Linguistic Analysis of ḥadīṯ Arabic Texts. In: Salem M.V.M.,

Böhmová, A., Hajič, J., Hajičová, E., Hladká, B. (2003). The PDT: A 3-level Annotation Scenario. In: Abeillé 2003a, 103-127.

Bolc, L. (1983). The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks. Springer-Verlag, Berlin, Germany.

Boudchiche, M., Mazroui, A., Ould Abdallahi Ould Bebah, M., Lakhouaja, A., Boudlal, A. (2016). AlKhalil Morpho Sys 2: A robust Arabic morpho-syntactic analyzer, Journal of King Saud University – Computer and Information Sciences, DOI: 10.1016/j.jksuci.2016.05.002.

Boudlal, A., Lakhouaja, A., Mazroui, A., Meziane, A., Bebah, M. O. A. O., & Shoul, M. (2010). Alkhalil morpho sys1: A morphosyntactic analysis system for arabic texts. In International Arab conference on information technology (1-6). Benghazi Libya.

Boukedi, S., Haddar, K., (2014). HPSG Grammar Treating of Different Forms of Arabic Coordination. Research in Computing Science 86: 25-41

Brants, T., Skut, W., Uszkoreit, H. (2003). Syntactic Annotation of a German Newspaper Corpus. In: Abeillé 2003a, 73-87.

Brants, S., Dipper, S., Hansen, S., Lezius, W., Smith, G. (2002). The TIGER Treebank. In: Hinrichs et Simov 2002, 24-42.

Brants, T., Plaehn, O. (2000). Interactive Corpus Annotation. In: Proceedings of the Second International Conference on Language Resources and Evaluation. Athens, Greece, 453-459.

Briscoe, E., Carroll, J. (1997). Automatic Extraction of Subcategorization from Corpora. In: Proceedings of the Fifth Conference on Applied Natural Language Processing. Washington, DC, 356-363.

Buckwalter, T. (2002). Buckwalter Arabic Morphological Analyzer Version 1.0 Philadelphia, Linguistic Data Consortium

Cahill, A., McCarthy, M., Van Genabith, J., Way, A. (2002). Evaluating F-structure Annotation for the Penn-II Treebank. In: Hinrichs et Simov 2002 , 43-60.

Chang ,P. , Tseng, H., Jurafsky, D. , Manning, C. D. (2009) . Discriminative Reordering with Chinese Grammatical Relations Features. In Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation.

Charniak, E., Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In ACL'05, 173–180

Charniak, E.(1996) . Tree-bank Grammars. Technical Report CS-96-02. Brown University, Providence, Rhode Island, USA, 1031-1036.

Charniak, E. (2000). A maximum-entropy-inspired parser. In The Proceedings of the North American Chapter of the Association for Computational Linguistics, pages 132–139.

Chen, K., Luo, C., Chang, M., Chen, F., Chen, C., Huang, C., Gao, Z. (2003). Sinica Treebank. In: Abeillé 2003, 231-248.

Chomsky, N. (1965). Aspects of the Theory of Syntax. Cambridge, MA: MIT Press.

Chomsky, N. (1959) . On certain formal properties of grammars. Information and Control, 2(2):137-167

Čmejrek, M., Curˇı´n, J., Havelka, J., Hajič, J., Kubonˇ , V. (2004) . Prague Czech-English Dependency Treebank: Syntactically Annotated Resources for Machine Translation. In: Proceedings of the IV International Conference on Language Resources and Evaluation. Lisbon, Portugal, 1597-1600.

Collins, M. (1999). Head-driven Statistical Models for Natural Language Parsing. PhD Thesis, University of Pennsylvania.

Collins, M. (1997) . Three generative, lexicalized models for statistical parsing. In The Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, San Francisco. Morgan Kaufmann.

Copestake, A. (2002). «Implementing Typed Feature Structure Grammars ». CSLI Publications, Stanford University.

De Marneffe, M. C., MacCartney, B., Manning, C. D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In LREC.

Debusmann, R., Smolka, G. (2006). Multi-dimensional Dependency Grammar as Multigraph Description. Proceedings of the 19th International FLAIRS Conference, pp. 740-745. Melbourne Beach, Florida, USA.

Dechter, R. (1990). Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. In Artificial, Intelligence, volume 41, pages 273-312.

Diab, M. T., Habash, N., Rambow, O., Roth, R. (2013). LDC Arabic Treebanks and Associated Corpora: Data Divisions Manual. Columbia University, Technical Report, Center for Computational Learning Systems

Dickinson, M., Meurers, W. D. (2003), Detecting Inconsistencies in Treebanks. In: Nivre et Hinrichs 2003, 45-56.

Dukes, K., Atwell, E., Sharaf A. B. (2010). Syntactic Annotation Guidelines for the Quranic Arabic Dependency Treebank. Language Resources and Evaluation Conference (LREC). Valletta, Malta.

Dukes, K., Buckwalter, T. (2010). A Dependency Treebank of the Quran using traditional Arabic grammar. Institute of Electrical and Electronics Engineers

Earley, J. (1970). An Efficient Context-free Parsing Algorithm. Communications of the ACM, 13(2): 94-102.

Garside, R., Leech, G., Varadi, T. (1992). Lancaster Parsed Corpus. A Machine-readable Syntactically Analyzed Corpus of 144,000 Words. Available for Distribution through ICAME.

Garey, M., Johnson, D. (1979). Computers and Intractability. W.H.Freeman.

Gaschnig, J. (1977). A General Backtrack Algorithm That Eliminates Most Redundant Tests, Proceedings 5th International Joint Conference on Artificial Intelligence, 457.

Gazdar, G., Klein, E., Pullum, G., Sag, I. (1985). Generalized Phrase Structure Grammar. Harvard University Press, Cambridge, Massachusetts, USA.

Geman, S., Johnson, M. (2002). Probabilistic Grammars and their Applications. In Smelser, N., J., Baltes, P., B. (eds.): International Encyclopedia of the Social & Behavioral Sciences. Pergamon, Oxford, UK.

Gildea, D. (2001). Corpus variation and parser performance. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 167–202.

Glennie, A., E.(1960) . On the Syntax Machine and the Construction of a Universal Compiler. Tech Report No. 2, Contr. NR 049-141. Carnegie Institute of Technology, Pittsburgh, Pennsylvania, USA.

Graff, D., Maamouri, M., Bouziri, B., Krouna, S., Kulick, S., Buckwalter, T. (2009) Standard Arabic Morphological Analyzer (SAMA) Version 3.1. Linguistic Data Consortium

LDC2009E73.

Green, S., DeNero, J. (2012). A class-based agreement model for generating accurately inflected translations. In ACL.

Green, S., De Marneffe, M. C. Bauer, J., Manning, C. D. (2011). Multiword Expression Identification with Tree Substitution Grammars: A Parsing tour de forcewith French.. In EMNLP 2011.

Green, S., Manning, C. (2010). Better Arabic Parsing: Baselines, Evaluations, and Analysis. In Proceedings of the International Conference on Computational Linguistics (COLING) (349-402). Beijing, China.

Habash, N., Roth, R., Rambow, O., Eskander, R., Tomeh, N. (2013). Morphological analysis and disambiguation for dialectal Arabic. In HLTNAACL.

Habash, N., Roth, R. (2009). CATiB: The Columbia Arabic Treebank. In the Proceedings of the Conference of the Association for Computational Linguistics (ACL'09). Suntec, Singapore.

Habash, N., Faraj, R., Roth, R. (2009) .Syntactic Annotation in the Columbia Arabic Treebank. In Proceedings of the 2nd International Conference on Arabic Language Resources and Tools (MEDAR), Cairo, Egypt.

Habash, N., Sadat, F. (2006). Arabic preprocessing schemes for statistical machine translation. In NAACL, Short Papers.

Haddar, K., Boukedi, S., Zalila, I. (2010) Arabic language and its specification in TDL. International Journal on Information and Communication Technologies, Serials Publications, Advances in Arabic Language Processing, 3 (3), 52-64.

Hajic, J., Smrz, O., Zemanek, P., Snaidauf, J., Beska, E. (2004). Prague Arabic Dependency Treebank: Development in Data and Tools. Proceedings of the NEMLAR international conference on Arabic Language Resources and Tools.

Hajič, J., Vidová-Hladká, B., Pajas, P. (2001). The Prague Dependency Treebank: Annotation Structure and Support. In: Proceeding of the IRCS Workshop on Linguistic Databases, 105–114.

Hajič, J. (1998). Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In: Issues of Valency and Meaning. Prague: Karolinum, 106-132.

Hajičová, E. (1998). Prague Dependency Treebank: From Analytic to Tectogrammatical Annotation. In: Proceedings of the First Workshop on Text, Speech, Dialogue. Brno, Czech Republic, 45-50.

Hall, J., Nilsson, J., Nivre, J., Eryiğit, G. (2007a). Single Malt or Blended? A Study in Multilingual Parser Optimization. In Proceedings of the Shared Task Session of EMNLP-CoNLL (933-939). Prague.

Hall, J., Kübler, S., McDonald, R. (2007b). The CoNLL 2007 Shared Task on Dependency Parsing. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (915-932).

Hammouda, N.G., Haddar, K. (2016). Integration of a Segmentation Tool for Arabic Corpora in NooJ Platform to Build an Automatic Annotation Tool. In: Barone L., Monteleone M., Silberztein M. (eds) Automatic Processing of Natural-Language Electronic Texts with NooJ. NooJ . Communications in Computer and Information Science, vol 667. Springer, Cham

Han, C., Han, N., Ko, S. (2002). Development and Evaluation of a Korean Treebank and its Application to NLP. In: Proceedings of the Third International Conference on Language Resources and Evaluation. Las Palmas, Spain, 1635-1642.

Haralick, R. M., Elliott, G.L. (1980). Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, Elsevier, Volume 14, Issue 3, 263-313.

Hays, D. (1964) Dependency Theory: a Formalism and Some Observations. Language, 40(4): 511-525.

Hindle, D. (1994). A Parser for Text Corpora. In: Zampolli, A. (ed.), Computational Approaches to the Lexicon. New York: Oxford University Press, 103-151.

Hinrichs, E., Simov, K. (2002). Proceedings of the First Workshop on Treebanks and Linguistic.

Hinrichs, E. W., Bartels, J., Kawata, Y.,Kordoni, V., Telljohann, H. (2000). The Tübingen Treebanks for Spoken German, English and Japanese. In: Wahlster, W. (ed.), Verbmobil: Foundations of Speech-to-Speech Translation. Berlin: Springer, 552-576.

Hockenmaier, J.,Steedman, M. (2002). Acquiring Compact Lexicalized Grammars from a Cleaner Treebank. In: Proceedings of the Third International Conference on Language Resources and Evaluation. Las Palmas, Spain, 1974-1981.

Johnson, M. (1998). PCFG models of linguistic tree representations. Computational Linguistics 24, 613–632.

Johnson, M., Geman, S., Canon, S., Chi, Z., Riezler, S. (1999). Estimators for stochastic “unification-based” grammars. In The Proceedings of the 37th Annual Conference of the Association for Computational Linguistics, 535–541, College Park, Maryland.

Karlsson, F. (1990). Constraint Grammar as a Framework for Parsing Running Text. Proceedings of the 13th Conference on Computational Linguistics - Volume 3, 168-173. Helsinki, Finland.

Kasami, T. (1965) .An Efficient Recognition and Syntax-analysis Algorithm for Context-free Languages. Scientific Report AFCRL-65-758. Air Force Cambridge Research Lab, Bedford, Massachusetts, USA.

Keskes, I., Benamara, F., Belguith, L.H. (2013) Segmentation de textes arabes en unités discursives minimales. Actes de la conférence Traitement Automatique de la Langue Naturelle TALN. Les Sables d'Olonne, France.

Khoufi, N. , Aloulou, C. Belguith, L. H. (2016) . Toward hybrid method for parsing Modern Standard Arabic. SNPD : 451-456

Khoufi,N., Aloulou, C. , Belguith, L. H. (2014a). "Chunking Arabic Texts Using Conditional Random Fields," Proc. The IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA) 428-432.

Khoufi, N., Aloulou, C., Belguith, L. H. (2014b). Supervised learning model for parsing Arabic language. CoRR abs/1410.8783.

Kingsbury, P., Palmer, M. (2003). PropBank: The Next Level of TreeBank. In: Nivre et Hinrichs 2003 2003, 105-116.

Klein, D., Manning, C. D. (2003a). Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL), 423–430.

Klein, D., Manning , C. D. (2003b). Fast Exact Inference with a Factored Model for Natural Language Parsing. In Advances in Neural Information Processing Systems 15 (NIPS 2002), Cambridge, MA: MIT Press, 3-10.

Kroch, A., Taylor, A. (2000). Penn-Helsinki Parsed Corpus of Middle English. URL: *http://www.ling.upenn.edu/mideng/.

Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG Treebank Tool. In: Nivre et Hinrichs 2003, 217-220.

Kübler, S.,Nivre, J., Hinrichs, E., Wunsch, H. (2004). Proceedings of the Third Workshop on Treebanks and Linguistic Theories. Tübingen, Germany.

Kučera, H.,Francis,W. (1967). Computational Analysis of Present-day American English. Providence, RI: Brown University Press.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: a survey. In Artificial Intelligence Magazine, volume 13,32-44.

Kunz, K., Hansen-Schirra, S. (2003). Coreference annotation of the TIGER treebank. In: Nivre et Hinrichs 2003, 221-224.

Laurière J.-L. (1978). A Language and a Program for Stating and Solving Combinatorial Problems, Arti_cial Intelligence 10 :1.

Lehmann, S., Oepen, S., Regnier-Prost, S., Netter, K., Lux, V., Klein, J., Falkedal, K., Fouvry, F., Estival, D., Dauphin, E., Compagnion, H., Baur, J., Balkan, L., Arnold, D. (1996). TSNLP - Test Suites for Natural Language Processing. In: Proceedings of the 16th International Conference on Computational Linguistics. Copenhagen, Denmark, 711-716.

Levy, R., Manning, C. D. (2003). Is it harder to parse Chinese, or the Chinese Treebank? ACL 2003, 439-446.

Lhioui, C., Zouaghi, A., Zrigui., M. (2015) Realization of Minimum Discursive Units Segmentation of Arab Oral Utterances. International Journal of Computational Linguistics and Applications, vol. 7, no. 1, 2016, pp. 31–50.

Liang, P., Petrov, S., Jordan, M., Klein, D. (2007). The infinite PCFG using hierarchical Dirichlet processes. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 688–697.

Lin, D. (1998). A Dependency-based Method for Evaluating Broad-coverage Parser. In: Journal of Natural Language Engineering 4, 97-114.

Linard, A., Morin, E., Daille, B. (2016) Extraction de lexiques bilingues à partir de corpus comparables spécialisés à travers une langue pivot. Actes de la conférence conjointe JEP-TALN-RECITAL 2016, volume 2 : TALN, Paris, France.

Maamouri, M. (2010). LDC Standard Arabic Morphological Analyzer (SAMA) Version 3.1 LDC2010L01. Web Download. Philadelphia: Linguistic Data Consortium.

Maamouri, M., Bies, A., Krouna, S., Gaddeche, F., Bouziri, B (2009). Penn Arabic Treebank guidelines v4.8. Technical report, LDC, University of Pennsylvania

Maamouri, M., Bies, A., Buckwalter, T., Mekki, W. (2004). The Penn Arabic Treebank: Building a Large-scale Annotated Arabic Corpus. In Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools, 102-109, Cairo, Egypt.

Maamouri, M., Bies, A. (2004). Developing an Arabic Treebank: Methods, Guidelines, Procedures, and Tools. In: Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages. Geneva, Switzerland, 2-9.

Marcus, M. P., Kim, G., Marcinkiewics, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In: Proceedings of the Human Language Technology Workshop. Plainsboro, NJ, 114-119.

Marcus, M. P., Santorini, B., Marcinkiewics, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. In: Computational Linguistics 19, 313-330.

Marton, Y., Habash, N., Rambow, O. (2013). Dependency Parsing of Modern Standard Arabic with Lexical and Inflectional Features. Computational Linguistics, 39:1, 161-194.

Matsuzaki, T., Miyao, Y. ,Tsujii, J. (2005). Probabilistic CFG with latent annotations. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL), 75–82.

Miltsakaki, E., Prasad, R., Joshi, A., Webber, B. (2004). The Penn Discourse Treebank. In: Proceedings of the IV International Conference on Language Resources and Evaluation. Lisbon, Portugal, 2237-2240.

Mohri, M., Roark, B. (2006). Probabilistic Context-Free Grammar Induction Based on Structural Zeros. In: Proceedings of the Seventh Meeting of the Human Language Technology conference- North American Chapter of the Association for Computational Linguistics (HLT-NAACL), New York.

Monroe, W., Green, S., Manning, C. D. (2014). Word Segmentation of Informal Arabic with Domain Adaptation. In ACL.

Montemagni, S., Barsotti, F., Battista, M., Calzolari, N., Corazzari, O., Lenci, A., Zampolli, A., Fanciulli, F., Massetani, M., Raffaelli, R., Basili, R., Pazienza, M. T., Saracino, D., Zanzotto, F., Nana, N., Pianesi, F., Delmonte, R. (2003). Building the Italian Syntactic-semantic Treebank. In: Abeillé 2003a, 189-210.

Nivre, J. (2008). Treebanks. In : Joakim Nivre. Corpus Linguistics : An International Handbook. Berlin : Mouton de Gruyter , 225-241.

Nivre, J., Hall, J., Nilsson, J., Chanev, A. (2007a). MaltParser: A Language Independent System for Data-driven Dependency Parsing. Natural Language Engineering, 13:2 (95-135).

Nivre, J. (2006). Two Strategies for Text Parsing. SKY Journal of Linguistics, 19: 440-448,. Sirine Boukedi, Kais Haddar

Nivre, J., Hinrichs, E. (2003). Proceedings of the Second Workshop on Treebanks and Linguistic Theories. Växjö, Sweden: Växjö University Press.

Nivre, J. (2003). Theory-supporting treebanks. In: Nivre et Hinrichs , 117-128.

Oepen, S., Flickinger, D., Toutanova, K., Manning, C. D. (2002). LinGO Redwoods: A Rich and Dynamic Treebank for HPSG. In: Hinrichs et Simov 2002, 139-149.

Oflazer, K., Say, B., Hakkani-Tür, D. Z., Tür, G. (2003). Building a Turkish Treebank. In: Abeillé 2003a, 261-277.

Othman, E., Shaalan, K., Rafea, A., (2003). A chart parser for analyzing modern standard Arabic sentence, in Proceedings of the MT summit IX workshop on machine translation for Semitic languages: issues and approaches, revue 37-44

Pasha, A., Al-Badrashiny, M., Diab, M., El Kholy,A., Eskander,R., Habash,N., Pooleery, M.,Rambow, O., Roth, R. M (2014). Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In LREC-2014, Reykjavik, Iceland.

Petrov, S. (2009). Coarse-to-Fine Natural Language Processing. PhD Thesis. University of California, Berkeley.

Petrov, S., Barrett, L., Thibaux, R., Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, 433–440.

Pollard, C. (1996). The Nature of Constraint-based Grammar. Proceedings of the 10th Pacific Asia Conference on Language, Information and Computation. Seoul, Korea.

Pollard, C., Sag, I. (1994). Head-driven Phrase Structure Grammar. University of Chicago Press and CSLI Publications, Chicago, Illinois, USA.

Rafea, A., Shaalan, K. (1993). Lexical Analysis of Inflected Arabic words using Exhaustive Search of an Augmented Transition Network, Software Practice and Experience, Vol. 23(6), 567-588, John Wiley & sons, U.K

Rafferty, A., and Christopher D. Manning. 2008. Parsing Three German Treebanks: Lexicalized and Unlexicalized Baselines. In ACL Workshop on Parsing German.

Resnik, P. (1992). Probabilistic Tree-adjoining Grammar as a Framework for Statistical Natural Language Processing. Proceedings of the 14th Conference on Computational Linguistics, pp. 418-424. Nantes, France.

Riezler, S., King, M., Kaplan, R., Crouch, R., Maxwell, J. (2002). Parsing the Wall Street Journal Using a Lexical-functional Grammar and Discriminative Estimation Techniques. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 271-278.

Rocio, V., Alves, M. A., Lopes, J. G., Xavier, M. F., Vicenter, G. (2003). Automated Creation of a Medieval Portuguese Treebank. In: Abeillé 2003a, 211-227.

Sampson, G. (2003). Thoughts on Two Decades of Drawing Trees. In: Abeillé 2003, 23-41.

Sampson, G. (1995). English for the Computer. The SUSANNE Corpus and Analytic Scheme. Oxford: Oxford University Press.

Sasaki, F., Witt, A., Metzing, D. (2003). Declarations of Relations, Differences and Transformations between Theory-specific Treebanks: A New Methodology. In: Nivre et Hinrichs 2003 2003, 141-152.

Schabes, Y. (1992). Stochastic Lexicalized Tree-Adjoining Grammars. Proceedings of the 14th Conference on Computational Linguistics - Volume 2, pp. 425-432. Nantes, France.

Schuster, S., Manning, C. D. (2016). Enhanced English Universal Dependencies: An Improved Representation for Natural Language Understanding Tasks. In LREC.

Sgall, P., Hajičová, E., Panevová, J. (1986). The Meaning of the Sentence in Its Pragmatic Aspects. Dordrecht: Reidel.

Simov, K., Osenova, P., Kolkovska, S., Balabanova, E., Doikoff, D., Ivanova, K., Simov, A., Kouylekov, M. (2002). Building a Linguistically Interpreted Corpus of Bulgarian: The BulTreeBank. In: Proceedings of the Third International Conference on Language Resources and Evaluation. Las Palmas, Spain, 1729-1736.

Smrz, O., Bielicky, V., Kourilova, I., Kracmar, J., Hajic, J., Zemanek, P. (2008). Prague Arabic Dependency Treebank: A Word on the Million Words. In: Proceedings of the Workshop on Arabic and Local Languages (LREC 2008), 16–23, Marrakech, Morocco.

Stamou, S., Andrikopoulos, V., Christodoulakis, D. (2003). Towards Developing a Semantically Annotated Treebank Corpus for Greek. In: Nivre et Hinrichs 2003 , 225-228.

Steedman, M. (1985). Combinators and Grammars. In Oherle, R., Bach, E., Wheeler, D. (eds.): Categorial Grammars and Natural Language Structures. Foris, Dordrecht, Holland.

Taylor, A., Marcus, M., Santorini, B. (2003). The Penn Treebank: An Overview. In: Abeillé 2003, 5-22.

Teleman, U. (1974). Manual för grammatisk beskrivning av talad och skriven svenska. Lund: Studentlitteratur.

Tounsi, L., Van Genabith, J. (2010). Arabic Parsing Using Grammar Transforms. LREC.

Tounsi, L., Attia, M., Van Genabith, J. (2009). Automatic Treebank-Based Acquisition of Arabic LFG Dependency, Structures, Proceedings of the EACL Workshop on Computational Approaches to Semitic Languages, pages 45–52, Athens, Greece. c 2009 Association for Computational Linguistics

Toutanova, K., Manning, C. D., Shieber, S. M., Flickinger, D., Oepen, S. (2002). Parse Disambiguation for a Rich HPSG Grammar. In: Hinrichs et Simov 2002, 253-263.

Ule, T., Simov, K. (2004). Unexpected Productions May Well Be Errors. In: Proceedings of the Fourth International Conference on Language Resources and Evaluation. Lisbon, Portugal, 1795-1798.

Vilnat, A., Paroubek, P., Monceaux, L., Robba, I., Gendner, V., Illouz, G., Jardino, M. (2003). EASY or How Difficult Can It Be to Define a Reference Treebank for French. In: Nivre et Hinrichs 2003, 229-232.

Wallis, S. (2003), Completing Parsed Corpora. In: Abeillé, 2003, 61-71.

Wouden, T., van der/Hoekstra, H., Moortgat, M., Renmans, B., Schuurman, I. (2002). Syntactic Analysis in the Spoken Dutch Corpus. In: Proceedings of the Third International Conference on Language Resources and Evaluation. Las Palmas, Spain, 768-773.

Xue, N., Xia, F., Chiou, F.-D., Palmer, M. (2004). The Penn Chinese Treebank: Phrase Structure Annotation of a Large Corpus. In: Natural Language Engineering 11, 207-238.

Yamada, H., Matsumoto, Y. (2003). Statistical Dependency Analysis with Support Vector Machines. In Proceedings of the International Conference on Parsing Technologies (IWPT) (195-206). Nancy, France.

Yngve, V., H. (1959). The Feasibility of Machine Searching of English Texts. Proceedings of the International Conference on Scientific Information, pp. 975-995. Washington, DC, USA.

Younger, D., H. (1967). Recognition and Parsing of Context-free Languages in Time n3. Information and Control, 10(2): 189-208.