Vers une analyse syntaxique à granularité variable

Tristan Vanrullen

Vers une analyse syntaxique à granularité variable

THÈSE

présentée et soutenue publiquement le 12/12/2005

pour l'obtention du

Doctorat de l'Université de Provence – Aix-Marseille 1

(spécialité informatique)

par

Tristan VanRullen

Composition du jury

Rapporteurs :	Eric Villemonte de la Clergerie Eric Wehrli
Examinateurs :	Patrick Paroubek Monique Rolbert Laurent Romary
Directeur :	Philippe Blache

Numéro d'inventaire _ _ _ _ _ _

Laboratoire Parole et Langage d'Aix-en-Provence — CNRS UMR 6057

Ecole Doctorale de Mathématiques et d'Informatique de Marseille

Mis en page avec la classe thlpl.

Résumé

Il est souhaitable qu'une analyse syntaxique -en traitement automatique des langues naturelles - soit réalisée avec plus ou moins de précision en fonction du contexte, c'est-à-dire que sa granularité soit réglable. Afin d'atteindre cet objectif, nous présentons ici des études préliminaires permettant d'appréhender les contextes technique et scientifique qui soulèvent ce problème. Nous établissons un cadre pour les développements à réaliser et pour leur évaluation. Nous choisissons un formalisme d'analyse par satisfaction de contraintes (celui des Grammaires de Propriétés) ayant l'avantage de permettre l'utilisation des mêmes ressources linguistiques avec un degré de précision réglable. Nous introduisons une reformulation mathématique du formalisme des Grammaires de Propriétés et nous définissons une mesure (la densité de satisfaction), qui permet de contrôler la granularité de l'analyse. Puis nous décrivons un ensemble d'outils modulaires (LPLSuite) et de ressources (lexique et sous-lexiques DicoLPL) développés pour permettre une analyse syntaxique et susceptibles d'être embarqués dans des applications de haut niveau. Nous présentons et évaluons ensuite plusieurs analyseurs syntaxiques dans ce formalisme, le dernier (SeedParser) étant destiné à mettre en œuvre une véritable analyse à granularité variable. L'évaluation de ces outils est l'objet d'une étude approfondie. Enfin, nous présentons quelques applications développées à l'aide de nos outils.

Mots-clés: Analyse syntaxique automatique, robustesse, granularité variable, Grammaires de Propriétés

Remerciements

Je remercie Philippe Blache pour sa direction, ses indications et son soutien sans faille tout au long de ma progression. Il m'a intéressé aux Grammaires de Propriétés et à l'analyse syntaxique. La pluridisciplinarité de son approche ainsi que celle qui règne au sein du Laboratoire Parole et Langage m'a aidé à envisager la portée du présent travail dans une perspective large, qui englobe des domaines très divers de la linguistique formelle à l'informatique. Je remercie le CNRS d'avoir permis mon travail au sein de ce laboratoire où règne une atmosphère très productive pour une recherche très riche. L'équipe avec laquelle j'ai travaillé est humainement, scientifiquement et techniquement remarquable.

Je souhaite aussi remercier Eric Villemonte de la Clergerie et Eric Wehrli pour avoir relu cette thèse. Les remarques précises qui m'ont été faites ont permis de la corriger et de l'enrichir.

Je remercie les membres de mon jury, Patrick Paroubek, Laurent Romary, Monique Rolbert, Eric Villemonte de la Clergerie et Eric Wehrli de me faire l'honneur d'assister à ma soutenance.

Je remercie Marie Laure Guénot pour son aide précieuse et son soutien amical. La modélisation et la conception de Grammaires de Propriétés n'auraient pas été possibles sans sa collaboration et ses connaissances linguistiques.

Je remercie ensuite tous ceux qui ont participé de près à l'ensemble de ce travail, scientifiquement et techniquement. En particulier, Stéphane Rauzy, Jean-Marie Balfourier, Emmanuel Bellengier, Christel Portes et Jean-François Maeyhieux ont permis à l'ensemble des outils de voir le jour et de devenir une véritable plateforme pour le traitement automatique des langues naturelles.

Je remercie encore les amis et parents qui m'ont aidé à achever la rédaction de ce document par leur soutien, leur présence et leurs remarques qui m'ont été très utiles.

Par dessus tout, je tiens à remercier Pascale Vardanega, la personne qui m'a accompagné et soutenu tout au long de ce travail, le jour comme la nuit, et qui m'a permis de persévérer malgré les moments difficiles, les doutes et les deuils.

Enfin, je remercie les trois personnes qui m'ont donné le plus envie d'achever cette tâche. Adrien et Tristan pour leur patience, leur aide et les sacrifices qu'ils ont accepté de faire. Lancelot, pour son infinie sagesse, son regard profond sur le monde et sa promptitude à ramener les questions essentielles au premier plan.

Sans les personnes que j'ai citées et sans beaucoup d'autres que je n'oublie pas, ce travail n'aurait pas abouti.

"Le langage est l'équivalent pour la bouche vide du rêve pour les yeux fermés."
Pascal Quignard, Vie secrète

Je dédie ce travail à Pascale, Lancelot, Tristan et Adrien.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des définitions

Introduction générale

Pour une analyse syntaxique basée sur les contraintes

Le domaine scientifique de la linguistique formelle, plus ancien que celui du traitement automatique des langues naturelles, doit de plus en plus répondre aux attentes de ce dernier en lui offrant des modèles, des théories et des formalismes susceptibles de supporter les traitements toujours plus complexes que requièrent les progrès en traitement automatique et dont les ordinateurs sont de plus en plus capables du fait de l'accroissement de leurs capacités calculatoires et mémorielles.
Réciproquement, les théories linguistiques questionnent les tentatives informatiques : après la grande vague de positivisme scientifique qui a entouré l'éclosion de l'Intelligence Artificielle, des obstacles très nombreux se sont opposés au traitement immédiat des phénomènes linguistiques les plus délicats.
Il n'est pas difficile, connaissant les faibles possibilités de calcul des premiers systèmes informatiques, de comprendre les simplifications massives apportées aux notions et aux modèles de la linguistique des années 1950 et les schématisations excessives des processus mis en jeu dans la compréhension et la production des langages.
Le développement des relations entre linguistique et informatique se fait progressivement, au prix de heurts épistémologiques entre les concepts appartenant respectivement à chaque domaine. Le domaine interdisciplinaire correspondant au terrain de rencontre de ces deux domaines, désigné par le nom de Traitement Automatique des Langues Naturelles (ou encore TALN), devient un domaine scientifique à part entière, doté de ses concepts et de ses objets, de ses théories, de ses expérimentations et de ses techniques. Les recherches d'aujourd'hui dans ce domaine suivent deux grandes directions :

- La première consiste en une recherche applicative, où des théories linguistiques et informatiques sont employées et mises en pratiques pour répondre à des besoins techniques et pratiques.
- La seconde est une recherche théorique où les anciens modèles sont confrontés aux nouveaux besoins. Leurs limites et leurs possibilités sont explorées. De nouvelles théories sont encore avancées.

Une circulation des efforts existe entre ces deux directions : l'expérimentation autour d'une théorie nécessitant le développement d'applications qui la supportent, les résultats des expériences mettant à leur tour en cause (ou validant) les théories et nécessitant alors leur correction. Cette circulation est caractéristique d'un processus empirique, signe de jeunesse du domaine scientifique concerné. Èmerge de ce processus la question particulière de l'évaluation des formalismes et des résultats expérimentaux.

Chacune des directions données ci-dessus a cependant ses propres particularités. La première pouvant pour des raisons pratiques effectuer des simplifications extrêmes aux théories qu'elle adoptera (par exemple la programmation d'outils dédiés à des tâches spécifiques nécessitant peu de connaissance linguistique). La seconde pouvant se passer au contraire d'application pratique pendant un long moment (par exemple la collecte d'informations menant à l'élaboration de théories guidées par l'usage linguistique).

Malgré les grandes disparités observables entre ces directions, de grands courants théoriques dominent les évolutions de tout le domaine du TALN. Nous les étudierons dans la première partie de cette étude. Du plus ancien mouvement théorique, inauguré par Chomsky sur les bases de la théorie des langages formels et en révision des approches grammairiennes utilisées depuis Port Royal (voir [Chomsky, 1965] et [Chomsky, 1968]), aux théories les plus récentes, la linguistique formelle a beaucoup évolué en réponse aux problèmes qu'elle rencontrait. Les premières approches étaient de fait motivées par des considérations trop réductrices (d'un point de vue psycholinguistique par exemple), trop normatives, etc.
Comme nous le verrons plus loin, une théorie doit permettre l'extraction d'informations répondant à des critères toujours plus divers, plus fins, plus indépendants ou au contraire plus interconnectés, etc.

Les nombreux problèmes rencontrés par les théories et les formalismes actuels seront abordés dans la suite de cette étude. Pour les résumer brièvement, nous faisons le constat des limites de l'approche dite *générative* qui place l'étude des langues sous le seul angle de leur production ; nous constatons aussi les impasses que connaissent les approches qui prennent en compte un seul phénomène ou un seul domaine de la linguistique sans considérer ses relations avec les autres domaines linguistiques. Une prise en compte de ces problèmes par l'usage de représentations et de contraintes suffisamment souples, par la formalisation de techniques robustes est nécessaire. C'est la position d'un courant d'approches basées sur les contraintes. Nous en adoptons certaines idées pour défendre une analyse syntaxique au sens large. Dans les développements ultérieurs, nous choisirons le formalisme des Grammaires de Propriétés, qui se place exactement dans l'optique que nous avons énoncée. Cette présentation nous permettra ensuite d'aborder la problématique qui entoure notre travail.

Vers une analyse syntaxique à granularité variable

Certaines applications en Traitement Automatique des Langues Naturelles s'appuient sur des techniques d'analyse superficielle de la syntaxe (typiquement celles traitant des données volumineuses), d'autres comptent sur l'analyse profonde (par exemple la traduction automatique, le dialogue homme-machine, la fouille de données textuelles *etc.*).

Les techniques employées dans ces deux cas sont tout à fait différentes. La première se fonde habituellement sur des méthodes stochastiques ou sur l'emploi de règles avec des heuristiques simplificatrices,la seconde sur des techniques symboliques. Cependant, ceci peut constituer un problème pour des applications qui auraient besoin d'une analyse peu profonde la plupart du temps et dans certaines situations d'une information plus détaillée. C'est typiquement le cas pour les systèmes de synthèse vocale. De telles applications se fondent habituellement sur des analyseurs peu profonds afin de calculer les groupes intonatifs sur la base d'unités syntaxiques (ou plus précisément sur des 'chunks'). Mais dans certains cas, une information aussi

superficielle n'est pas assez précise. Une solution consisterait alors à employer une analyse profonde pour quelques constructions. Aucun système mettant en application une telle approche n'existe. Cela est dû en particulier au fait qu'un tel système exigerait deux traitements différents, le second refaisant en fait la totalité du travail. De fait, il est difficile d'imaginer dans le cadre génératif classique de mettre en application une technique d'analyse capable de calculer des 'chunks', et dans certains cas, des expressions plus complexes avec une organisation hiérarchique.

La question que nous allons nous poser découle de ce constat : il existe un manque dans l'analyse syntaxique automatique d'aujourd'hui. Ce manque concerne la notion beaucoup employée de *granularité*. Ce terme nous interroge car il est communément utilisé pour désigner les différences entre les approches profondes et superficielles (d'une granularité fine à une granularité grossière), entre les approches hiérarchisées et celles qui produisent des structures plates. Entre les approches multimodales et les approches unimodales, enfin entre les approches modulaires et non modulaires.

Nous allons commencer par chercher à définir le terme de granularité variable en observant les principaux courants en analyse syntaxique.

Nous présenterons ensuite un formalisme fondé sur les contraintes, qui constitue une réponse possible à ce problème. Cette approche permet l'utilisation d'une même ressource linguistique (c.-à-d. une grammaire unique) qui puisse être employée entièrement ou partiellement par l'analyseur. Cette approche se fonde sur le fait que

1. toute l'information linguistique est représentée au moyen de contraintes et
2. les contraintes sont de type régulier. L'idée consiste alors à mettre en application une technique qui puisse se servir de quelques contraintes dans le cas de l'analyse peu profonde, et de leur ensemble entier pour l'analyse profonde.

Dans ce formalisme (celui des Grammaires de Propriétés), les contraintes sont organisées en différents types. Régler la granularité de l'analyse peut alors être guidé par le choix des types de contrainte devant être satisfaits, mais aussi par l'évaluation en cours d'analyse de la quantité de contraintes satisfaites par rapport à la quantité de contraintes présentes dans la grammaire. Nous devrons reformuler certaines caractéristiques de ce formalisme, afin de préciser comment la granularité de l'analyse peut devenir réglable dans des parseurs qui s'appuient sur lui.

Dans l'étude que nous menons, la problématique de l'analyse à granularité variable nous servira de fil conducteur, mais ne sera sans doute pas complètement résolue. Nous proposons une approche particulière, sans ignorer que d'autres possibilités peuvent être développées. Les aspects linguistiques comme ceux liés à l'algorithmique et à la technologie interviennent différemment et interfèrent souvent entre eux. Nous développons intégralement une approche, de sa formalisation à sa mise en œuvre, avec l'effort considérable que demande la conception d'une plateforme répondant à tous les critères (de modularité et de généricité, notammment) que nous allons définir pour une analyse syntaxique à granularité variable.

Organisation de notre étude

La première partie de notre étude introduit aux principales approches actuelles en les questionnant sur le sens qu'elles accordent à la notion d'*analyse syntaxique*.
Nous proposons tout d'abord une réflexion autour du mot *syntaxe*, qui est envisagé dans sa position centrale, au carrefour des représentations linguistique (linguistique computationnelle) et informatique (traitement automatique des langues naturelles).

Nous observons ensuite les approches actuelles en opposant celles qui conçoivent le processus d'analyse syntaxique comme une *reconnaissance* et celles qui l'envisagent comme une *analyse*. Les relations qu'entretiennent la linguistique formelle et l'informatique sont considérées formalisme par formalisme. Nous en dégageons un ensemble de critères permettant de cerner les difficultés et les attentes de l'analyse syntaxique automatique aujourd'hui. La problématique qui en ressort fait état de lacunes et de besoins spécifiques liés à des secteurs du TALN en développement, tels que la synthèse vocale, l'analyse multimodale, la correction automatique etc.
Dans cette première partie, nous présentons ensuite notre choix de développer, dans le cadre du formalisme des Grammaires de Propriétés, des solutions aux problèmes soulevés. Nous y établissons une modélisation logico-mathématique du formalisme et de ses grammaires en accord avec les critères d'analyse présentés au plus tôt.

Dans la seconde partie de cette étude, nous proposons un ensemble d'outils et d'applications correspondant à nos besoins. Nous abordons tout d'abord l'ensemble des critères techniques qui ont servi de guide à nos développements, tels que la généricité, l'évolutivité, la modularité, la réutilisabilité, la robustesse, et la complexité. Les outils mis en place pour réaliser la tâche complexe qu'est l'analyse syntaxique sont ensuite présentés un par un dans leur ordre d'usage : un dictionnaire du Français, un segmenteur, un étiqueteur et un fréquenceur. Leur ensemble forme la plateforme que nous appellerons LPLSuite. Des techniques de développement de ressources lexicales -telles que des dictionnaires noyaux- sont proposées.

Une troisième partie est consacrée aux analyseurs syntaxiques basés sur les Grammaires de Propriétés. Nous y comparons diverses tentatives de développement et mettons au point un analyseur capable d'analyse à granularité variable.

Dans la quatrième partie, nous évaluons l'ensemble des outils, ce qui constitue un nouveau problème -à la fois sur le plan théorique et sur le plan technique : comment obtenir des informations qualitatives et quantitatives sur nos outils et ressources, avec ou sans données de référence ou éléments de comparaison ?
Les évaluations portent aussi bien sur la qualité et la complexité des algorithmes que sur la valeur des résultats obtenus à l'aide de ces outils. Nous proposons des expériences d'évaluation pour chacun de ces problèmes.

Enfin, nous présentons dans la dernière partie les applications développées autour de ces outils. La synthèse vocale, le traitement de corpus, la relation linguiste/informaticien avec un logiciel de développement et de centralisation des tâches et de développement de grammaires en environnement contrôlé. Une application d'aide à la communication pour personnes handicapées est aussi présentée pour mettre en évidence l'intérêt pratique de la plateforme LPLSuite.
Le bilan de ces implantations clôture notre étude. Nous y faisons l'inventaire des évolutions attendues pour ces outils.

L'étude qui commence ici a été effectuée en ayant en permanence à l'esprit les questions suivantes :

- comment intégrer simultanément les données de la linguistique formelle et de l'informatique, face aux attentes toujours plus grandes en termes de rapidité, de robustesse, de précision, de qualité et de cohérence dans le domaine de l'analyse syntaxique ?

– Les domaines scientifiques qui ont pour objet *l'enchaînement d'informations dans un message, son analyse et son élucidation (sa caractérisation, sa description)* ont face aux *langages naturels* des attentes grandissantes que peinent à combler les techniques, les heuristiques et les formalismes de l'informatique. Est-ce à cause d'un trop haut degré de simplification ? Est-ce par l'effet de l'usage de descriptions trop mathématisées, totalement rationalisées, que les analyses ne réussissent qu'en excluant de leur succès les analyses de contextes métaphoriques, lyriques, abstraits, erronés ? Les apports des sciences cognitives, de la pragmatique, de la prosodie, et d'encore bien d'autres sciences, doivent être pris en compte, leurs demandes doivent être acceptées et traitées par les applications à venir. Ce dernier guide sera le cahier des charges des futures applications de haut niveau. Notre travail ici se doit de garder une porte ouverte pour accueillir ces évolutions.

Première partie

Pour la définition d'un modèle d'analyse à granularité variable

Lorsqu'on parle d'*analyse syntaxique*, nous faisons référence à la fois à un domaine de la linguistique et à un domaine plus récent de l'informatique. Des similitudes et des divergences de sens notables doivent être différenciées au sein de ces deux domaines afin d'éviter les amalgames qui troublent la compréhension de ce terme .

Premièrement, l'analyse syntaxique en tant que domaine linguistique, définit son cadre d'étude avec le texte rédigé en langue naturelle comme objet -éventuellement la transcription écrite de messages oraux. La syntaxe elle-même est une notion que recouvrent beaucoup de théories historiquement marquées par des transformations philosophiques et épistémologiques liées à l'évolution des cultures et des langues. Pour définir la syntaxe et l'analyse syntaxique au sens linguistique, il faut à la fois retracer l'histoire de ces notions et dresser un tableau des théories actuelles.

Deuxièmement, du point de vue informatique, les termes de syntaxe et d'analyse syntaxique s'appliquent à un objet plus vaste que celui de la linguistique : il s'agit de l'information brute ou complexe (au sens de la théorie de l'information). La syntaxe se donne pour objet la description de la structure et de l'organisation de messages porteurs d'information. Dans cette acception, la naissance de cette étude est beaucoup plus récente que celle de la linguistique. Marquée cependant par le très long préalable à son apparition qu'est l'histoire des mathématiques et de la logique. La théorie de l'information, puis la théorie des langages, ont adopté le terme d'analyse syntaxique pour désigner la reconnaissance d'informations complexes dans des trames d'informations. On employait déjà le terme de *grammaire* dans l'approche rigoureuse initiale qu'exigeait l'informatique face à des données construites telles que les énoncés de programmes informatiques ou les bases de données, fichiers formatés etc.. Avec la complexification des données et des informations à traiter, cette approche s'est peu à peu assouplie. L'informatique en tant qu'outil de traitement de données, appliqué à des domaines de moins en moins rigoureusement organisés, a dû assouplir ses techniques d'analyse et les rendre plus robustes. Le message traité étant de moins en moins grammatical, de moins en moins normalisé, de nouvelles formalisations voient le jour et leur ressemblance avec les formalismes de la linguistique n'est pas fortuite.

L'idée d'appliquer l'informatique à l'étude des langues est aussi ancienne que l'informatique. Les tentations positivistes des premiers temps dans ce domaine ont peu à peu fait place à une floraison de techniques et de théories opérationelles dans des sous-domaines très étroits de l'ensemble des langues dites naturelles.

Inspirées de la théorie des langages, les premières approches des langues naturelles ont eu des résultats retentissants (voir [Chomsky, 1957]) à tel point qu'elles questionnaient et bousculaient les théories linguistiques admises, comme le montre le débat de Royaumont entre Jean Piaget et Noam Chomsky [Piaget *et al.*, 1979].

L'analyse syntaxique informatique appliquée au domaine des langues naturelles, à cause de son origine, a longtemps été une technique stricte, peu robuste face à l'agrammaticalité, coûteuse en calculs et en mémoire. C'est pour ces quelques raisons que ce domaine de l'informatique qui prit le nom de Traitement Automatique des Langues Naturelles, fut dans ses débuts une entreprise réductrice face à son sujet d'étude (les langues naturelles au sens le plus large). Entreprise réductrice car les premiers temps virent naître des programmes simples, traitant des données réduites (discrétisées) au maximum. Les structures reconnues étant des simplifications parfois sans rapport avec ce que la connaissance linguistique avait jusqu'alors apporté. Ce n'est qu'avec l'accroissement récent en puissance de calcul que les besoins en TALN et les possibilités offertes par les théories linguistiques ont pu commencer à s'accorder. Un jeu continu de question/réponse a lieu entre ces deux domaines afin d'améliorer les traite-

ments d'une part et d'affiner les théories d'autre part. L'expérimentation sur de vastes corpus participe à la vérification des hypothèses et met en évidence de nouveaux problèmes.

Les liens établis entre l'informatique et la linguistique formelle ont fait naître un domaine d'étude à leur intersection : vu depuis l'informatique, ce domaine est couramment appelé Traitement Automatique des Langues Naturelles (TALN), tandis que du point de vue linguistique, on le désigne par le nom de Linguistique Computationelle. Ce domaine est une pièce à deux faces. Selon que l'on s'intéresse à certains critères informatiques ou selon qu'on observe son rapport avec les théories de la linguistique formelle, le sens des termes employés peut varier complètement. C'est le cas pour les mot *syntaxe* et *analyse syntaxique* comme nous venons de le voir. C'est encore le cas pour l'*évaluation* des résultats expérimentaux. Des convergences de sens, des acceptions communes aux deux domaines forment cependant le noyau terminologique du TALN. Nous conserverons à l'esprit tout au long de cette étude qu'il faut toujours mettre en parallèle les deux points de vue pour saisir un phénomène relevant du TALN.

Loin de pouvoir prendre en compte toutes les études menées jusqu'ici, nous allons dans un premier temps effectuer un tour d'horizon de l'opposition qui existe entre linguistique formelle et analyse automatique en observant pas à pas l'évolution des approches. Nous chercherons à classer les principaux courants fondés sur des théories et des techniques de plus en plus complexes. Le premier chapitre de cette partie est consacré à cette classification, au fil de laquelle nous pourrons discerner les caractéristiques, les critères d'analyse, et les limites des techniques décrites.
A partir de ces premières études, nous introduirons les besoins, les objectifs et la problématique caractérisant notre propre travail. En commençant par une réflexion sur la notion même de syntaxe, puis en mettant en évidence la nécessité d'améliorer d'anciennes approches ou de proposer de nouvelles techniques.
Enfin, nous choisirons un formalisme répondant à ces besoins, celui des *Grammaires de Propriétés*, et nous développerons sa formulation en termes logiques et mathématiques.
Nous pourrons alors aborder l'implantation des outils présentée dans la seconde partie.

Chapitre 1

Courants et besoins en analyse syntaxique

Nous ébauchons ici un parallèle entre l'histoire de la linguistique formelle et celle du traitement automatique des langues naturelles. Nous commençons par mettre en évidence une opposition nette entre deux conceptions de l'analyse syntaxique automatique : la *reconnaissance* et l'*analyse*.

– Une représentation idéalisée des informations linguistiques caractérise les débuts du TALN. D'une part, les programmes d'analyse n'étaient pas adaptés à des textes complexes, d'autre part, l'analyse consistait essentiellement en une *vérification de la bonne formation* du message par rapport aux règles qui décrivent le langage. Il s'agit de ce point vue de vérifier l'appartenance d'une chaîne à un langage, c'est à dire de *reconnaître* un mot du langage. La question de l'analyse syntaxique est dans ce sens une question fermée :
La chaîne analysée appartient-elle au langage ?

– Par la suite, des nécessités dépassant celle de la reconnaissance ont établi ce que nous entendons par *analyse* : décrire une chaîne reconnue, donner des informations concernant sa position parmi les autres chaînes reconnues, indiquer les liens qu'entretiennent entre eux les éléments du message, gérer les chaînes partiellement reconnues, etc.. La question qui se pose dans ce cas est dès lors une question ouverte :
Quelles informations pouvons nous donner sur la chaîne analysée ?
C'est seulement à ce niveau d'interrogation que les objets de la linguistique formelle et de l'informatique linguistique peuvent se rejoindre, se confronter et collaborer.

Un premier regroupement des domaines majeurs du TALN nous permet d'observer cette opposition (théorie linguistique *versus* approche informatique) en caractérisant leurs possibilités théoriques et en les rapprochant des formalismes de la linguistique auxquels ils s'associent.

La figure 1 schématise la portée des deux grandes familles d'approche en analyse syntaxique abordées dans ce chapitre. Elle met en évidence les différences de *couverture* qui existent entre l'approche statistique et l'approche symbolique face à des textes allant d'une rédaction très normée à une écriture quelconque en passant par des domaines spécifiques (texte médical, scientifique ...). La liste n'est pas exhaustive : nous laissons hors de ce schéma les problèmes complexes liés aux textes -poétiques par exemple- où les processus de rationalisation échouent,

où la structuration ambiguë, métaphorique, où l'organisation même du texte peut nécessiter des techniques d'analyse particulières. Cette figure met en évidence le fait qu'une approche faisant appel à des règles grammaticales prédéfinies (approche symbolique, la plupart du temps) s'oppose à une approche faisant appel à un apprentissage préalable (approche statistique, la plupart du temps). Les techniques que nous allons présenter ici appartiennent soit à la première, soit à la seconde, soit fonctionnent de façon hybride. Le fait d'employer des règles s'applique bien aux domaines très normés de la langue, mais les techniques basées sur l'apprentissage automatique peuvent mieux rendre compte de phénomènes anormaux et s'avérer plus robustes face à des types de texte plus ou moins agrammaticaux.

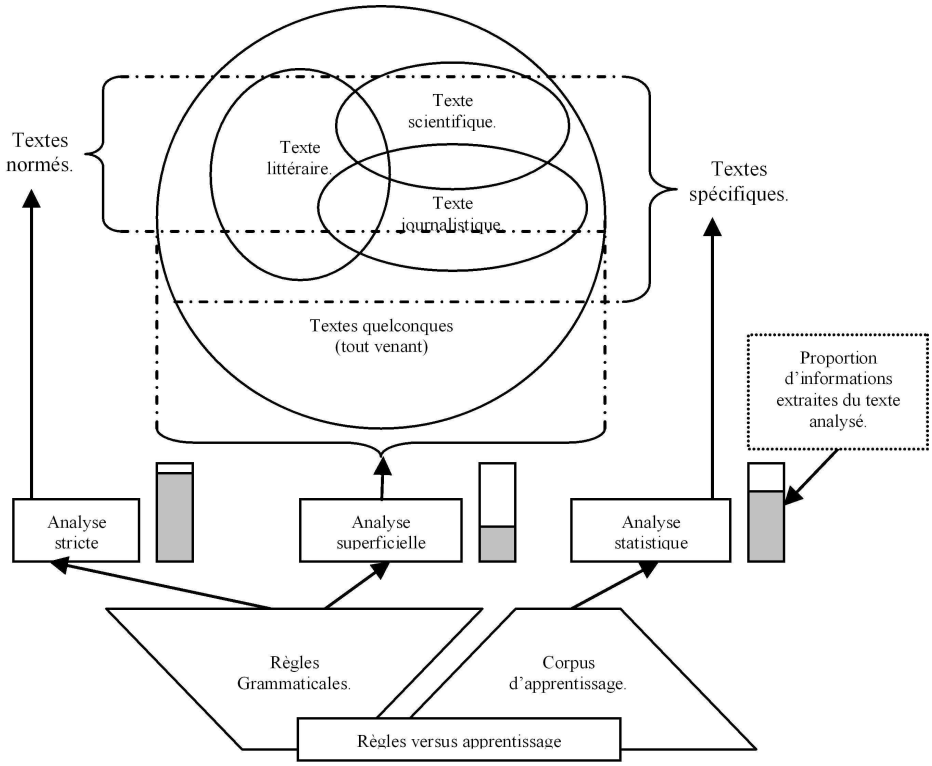


Fig. 1 – Les types d'analyse et leur portée.

De nombreux travaux récents donnent un aperçu assez exhaustif des approches que nous allons observer ici. Citons par exemple l'article [Abeillé and Blache, 1999] ou le volume de la revue TAL consacré aux évolutions en analyse syntaxique ([Villemonte de la Clergerie, 2002]). Nous ici choisissons de présenter la progression des techniques et des formalismes en observant la complexification de leurs données, de leurs besoins et de leurs algorithmes, plutôt que de suivre une liste exhaustive chronologique des techniques et des formalismes.

1.1 Premières approches symboliques

Les débuts du TALN sont caractérisés par **l'approche symbolique**, avec des techniques pour lesquelles les éléments du message sont manipulés par le truchement d'un symbole qui les représente (une catégorie englobante ou bien directement la forme lexicale de ces éléments). Les symboles sont employés dans une grammaire pour faire référence aux éléments du message. La

caractéristique principale des approches symboliques, par opposition à celles décrites ci-après, est la possibilité qu'elles offrent de retracer les étapes qui auront permis d'associer entre eux les symboles de la grammaire. Cette *trace de l'analyse* est la liste des symboles et des règles de la grammaire, qui rend lisible et reproductible le passage de la chaîne linguistique analysée à l'ensemble des descriptions résultant de l'analyse (la succession des règles appliquées pour effectuer l'analyse). En démonstration automatique, comme en TAL, cette trace est cruciale pour la compréhension des phénomènes analysés. Les travaux scientifiques en linguistique computationnelle, tels que le développement de grammaires ou la vérification de théories, en ont besoin autant que du résultat de l'analyse. L'analyse de l'entrée est dans cette approche aussi importante que le "*comment*" de cette analyse.

Dans les années 1950 à 1960, les techniques reposent sur les grammaires transformationnelles de Chomsky ([Chomsky, 1957] et [Chomsky, 1965]). Les grammaires sont génératives ou dérivationelles et se placent dans l'approche symbolique. Elles sont caractérisées par leur fonctionnement basé sur la *reconnaissance*.

On décide si un message est bien formé en *reconnaissant* l'appartenance de ses éléments au modèle décrit par la grammaire. Les règles de cette grammaire associent des symboles terminaux et des symboles non-terminaux via une relation de constituance. Un symbole non terminal est donné comme axiome (point de départ de la reconnaissance). Chaque symbole non-terminal est défini par une règle de constituance qui définit de quels symboles terminaux ou non-terminaux il peut être constitué. Ces règles sont dites non-contextuelles car un symbole non-terminal est toujours défini en dehors de tout contexte.

Des analyseurs sont construits à partir de ce formalisme. Nous retenons par exemple celui de Kuno[1], dont dériveront les ATN (réseaux de transitions, [Woods, 1970] et [Woods, 1980]). Au delà des grammaires de Chomsky[2], d'autres systèmes de règles peuvent être considérés comme caractéristiques de l'analyse par règle. Dans ces grammaires, le symbole non-terminal est aussi appelé *syntagme*. On peut citer parmi les autres techniques basées sur des règles de dérivation, les grammaires contextuelles, qui diffèrent des grammaires de Chomsky par le seul fait que les symboles non-terminaux peuvent se définir en fonction de leur environnement. Les limites de l'analyse hors contexte ont vite été ressenties. On voit rapidement se développer d'autres formalismes. Par exemple [Colmerauer, 1975] et les grammaires de métamorphose, ou encore les Definite Clause Grammars (DCG) de [Pereira and Warren, 1980]. Si le dernier s'avère difficile à maintenir manuellement, on trouve encore aujourd'hui des techniques permettant de traduire des grammaires GPSG en grammaires DCG, comme dans [Wilcock 1996]. Quant aux grammaires de métamorphoses, qui sont à l'origine du langage Prolog, fonctionnant par preuve sur des clauses de Horn, elles ont connu un réel essor avec les améliorations apportées au Lambda Calcul. La grammaire GNF (Grammaire noyau du Français, [Sabatier and Pasero, 1998]) est développée sur ses bases et est encore le sujet d'études très approfondies.

Les grammaires transformationnelles sont assez proches des grammaires de dérivation. le terme dérivation emprunté à la théorie des automates (application de l'étoile de Kleene) se rapproche aisément de celui de transformation (axiomes définis par des catégories terminales ou non terminales). D'autre part, le grammaires transformationelles imposent de fait une précédence linéraire entre les constituants qu'elles décrivent. On arrive rapidement à une critique des grammaires non-contextuelles transformationnelles en leur reprochant d'amalgamer ces deux phénomènes (dérivation et précédence). Les grammaires de métamorphose souffrent du

[1] [Kuno and Oettinger, 1962]

[2] voir [Chomsky, 1957], [Chomsky, 1965] et [Chomsky, 1968]

même amalgame, ce qui les rend beaucoup plus fragiles (sensibles à l'échec pour des entrées mal formées). Mais avant que ces critiques n'aboutissent à de nouveaux formalismes et de nouvelles techniques, les grammaires transformationelles évoluent beaucoup. Avec GB (Government and Biding) et la syntaxe X-barre ([Chomsky, 1981]), de nouvelles techniques sont proposées. Des analyseurs fondés sur GB sont toujours employés aujourd'hui avec succès (par exemple [Wehrli, 1988]). Les grammaires de constituants mènent peu à peu à l'idée d'unification (voir [Stabler, 1992], [Laezlinger and Wehrli, 1991] et [Berwick *et al.*, 1991]).

L'ensemble des techniques employées jusqu'au milieu des années 1970 correspond étroitement avec le domaine de la compilation (voir [Aho and Ullman, 1972] par exemple). Les algorithmes fondés sur les grammaires transformationelles ont depuis été maintes fois transformés et réemployés pour d'autres applications. Citons par exemple le célèbre algorithme de Earley ([Earley, 1970]) repris par [Shieber, 1985] et [Schabes, 1994], et encore [Hopcroft and Ullman, 1979], [Kasami, 1965], [Younger, 1967] et [Tomita, 1987]. Ces algorithmes étant assez généraux (ne portant pas précisément toujours sur le domaine du TAL, mais plus généralement sur l'analyse syntaxique au sens de la théorie des automates), certaines propriétés des langues dites naturelles ont pu être exploitées pour améliorer ces techniques lors de leur utilisation en TAL. Des heuristiques de reconnaissance de coins-gauches par exemple sont proposées dans [Rosenkrantz and Lewis, 1970] ou [Pereira and Shieber, 1987]. On verra plus tard d'autres heuristiques s'appliquer à des algorithmes plus généraux (comme par exemple la reconnaissance de *coins de tête* dans [Van Noord, 1997]).

Les techniques et formalismes symboliques que nous avons présentés jusqu'ici sont le résultat d'une première tentative de décrire de façon formelle et mathématisée les grammaires des langues naturelles. Leur complexification est dûe aux difficultés rencontrées. La distinction observée entre dérivation et précédence conduira à l'élaboration des premiers formalismes basés sur les contraintes.

A part l'approche proposée depuis Chomsky, il faut noter l'apparition de l'approche basée sur les grammaires de dépendances. Cette dernière est l'héritière des travaux de Tesnière (voir [Tesnière, 1934] et [Tesnière, 1959]). De nombreux travaux ont été fournis dans ce domaine, citons par exemple ceux de [Mel'Čuk, 1988], de [Giguet, 1997] ou [Karlsson *et al.*, 1995]. Pour les grammaires de dépendance, la notion de constituant importe moins que la relation entre les constituants. La syntaxe est envisagée plus comme une description des relations que comme une description des constructions. Dès 1965, Gaifman s'est interrogé sur l'équivalence formelle entre les grammaires de constituants et les grammaires de dépendance (voir [Gaifman, 1965]). Nous verrons dans la partie consacrée à l'approche basée sur les contraintes, que le mariage de ces deux modèles est possible.

Parallèlement à l'évolution des grands courants liés aux grammaires transformationnelles et aux grammaires de dépendances, dès les années 1980, d'autres techniques voient le jour, dans le but annoncé d'analyser avec plus d'acuité et de robustesse des corpus tout-venant. Ces dernières techniques sont motivées en grande partie par des nécessités industrielles pour lesquelles l'application est primordiale, mais ne sont pas pour autant dénuées de fondements théoriques. Les techniques symboliques basées sur les transducteurs ont l'avantage d'être rapides (voir [Appelt, 1987], [Roche, 1993] et [Aït-Mokhtar and Chanod, 1997]) et sont encore très employées. On verra aussi apparaître un grand nombre de techniques numériques, basées sur l'apprentissage direct des grammaires à partir de corpus.

1.2 Approches numériques

Face à des problèmes nouveaux de plus en plus nombreux, qui concernent directement le traitement informatique de données linguistiques dans le cadre de la linguistique computationelle, dans lesquels la trace importe moins que le résultat de l'analyse et où la rapidité est plus importante que la qualité des informations produites, de nouvelles approches sont apparues. Nous n'en ferons ici qu'une description succincte.

Les problèmes linguistiques concernés sont ceux où une information linguistique au sens large (sons, textes, dialogues, discours, grands corpus) doit être parcourue efficacement et rapidement, afin de fournir des données analytiques à son propos. Ces données produites peuvent-être une réponse à une requête, une reconnaissance aussi bien qu'une description de l'input.
Dans ces approches, on utilise d'une part un apprentissage préalable puis on met en œuvre des modèles statistiques pour répondre efficacement à un input.

Les modèles statistiques utilisés pour atteindre l'objectif énoncé ont beaucoup évolué et constituent une part essentielle du TALN.
Les résultats obtenus grâce aux techniques stochastiques étant de plus en plus fins et de plus en plus robustes, la notion de reconnaissance a pris son essor plus dans ce domaine qu'avec les techniques symboliques. Il s'agit en effet de reconnaître des motifs appris, ce qui rend ces systèmes plus proches de *l'usage* que les systèmes à base de règles.
Du point de vue théorique, les formalismes utilisés appartiennent quasiment totalement au domaine des mathématiques statistiques (modèles de prédiction) mais peu à peu les théories linguistiques font leur entrée en tant que références et ressources pour guider les heuristiques.

Les applications sont essentiellement destinées à des usages à large couverture. Reconnaissance vocale, étiquetage, catégorisation à tous les niveaux que traitent habituellement l'approche symbolique.

Le premier courant dans ce domaine est basé sur les réseaux de Markov, dans lesquels les symboles à reconnaître sont pré-appris sous forme de chaînes de probabilité. On pourra grâce à cet apprentissage connaître la probabilité d'un événement en fonction d'un ensemble d'événements déjà produits. La résolution d'ambiguïté se fait en maximisant une fonction de probabilité de chaque événement en contexte.

Le principe de l'analyse statistique à partir des modèles de Markov a une portée immense sur le traitement automatique des langues naturelles. Il repose sur un vaste apprentissage de corpus (parfois annoté manuellement), au cours duquel la probabilité d'occurrence d'un item dans son contexte est mémorisée. Quel que soit l'item (la transition entre deux étiquettes de catégorie) , la réalisation de l'analyse dépend de la maximisation d'une fonction de score.

Pour une suite d'éléments analysés, la meilleure interprétation (association d'un symbole w_i à chaque élément de rang i) est celle qui rend maximale la somme des produits élément par élément de la probabilité p_i où i désigne l'indice d'un élément de l'entrée analysée et p_i la probabilité d'avoir le symbole w_i en position i :

$$\sum_{(p_i)} \Pi_{i=1}^{n} \frac{Prob(p_i|w_i).Prob(p_i|p_{i+1}.p_{i+2})}{Prob(p_i)}$$

Church, en 1988 ([Church, 1988]), utilise un algorithme fondé sur les chaines de Markov, dans lequel il détecte les groupes nominaux en plaçant soit des parenthèses, soit rien, sur les

transitions entre mots. L'inconvénient de son système est qu'il ne vérifie pas le bon parenthésage, ce qui le rend plus souple, mais aussi plus risqué.

On pourra encore citer les approches de [Bod, 1998] et de [Rajman, 1995]

Parmi de nouvelles approches statistiques, celle qui emploie des réseaux de neurones est la plus opaque au sens linguistique. Comme dans celle de Church, il est impossible de connaître la trace des raisonnements qui mèneraient de la grammaire à l'analyse, mais l'apprentissage ne peut plus du tout être guidé par des théories linguistiques. D'autre part, le modèle statistique employé dans les couches du réseau de neurones est régi uniquement par des motivations computationnelles. Rien ne garantit, dans ces techniques, la capacité à répondre à tous les inputs. Dans les approches précédentes, il est possible de démontrer la robustesse d'une technique et de prouver la décidabilité d'un système. Ce n'est pas le cas dans cette dernière technique qui s'avère cependant de plus en plus efficace et utile face aux problèmes de la compréhension textuelle. Dans le cadre de l'analyse syntaxique, cette approche comme la précédente réalise une reconnaissance et non une analyse au sens que nous avons défini.

De plus en plus utilisées, les techniques stochastiques envahissent notre environnement informatique à cause de leur grande couverture, de leur rapidité et de l'amélioration progressive des procédés et des théories. Les limites à l'approche non-symbolique sont cependant de plus en plus sensibles. Des approches mixtes, mariant statistiques et règles sont favorisées pour certains traitements où la connaissance linguistique peut être efficace et rapidement accessible (vérification des accords par exemple). La portée et le succès de ces techniques est croissant, d'autant plus que leurs algorithmes sont assez généraux et assez portables d'un problème à un autre (la reconnaissance de signal, de formes ; l'étiquetage morpho-syntaxique, sémantique ; la catégorisation et la prédiction).

Signalons aussi l'existence de plus en plus prépondérante de systèmes mariant l'approche symbolique et l'approche numérique. Il est souvent possible de guider l'analyse approfondie à partir de données simples obtenues à l'aide de techniques stochastiques (voir par exemple [Klavans and Resnik, 1996]). L'opération inverse consistant à inclure une analyse context-free au sein de systèmes statistiques a aussi été envisagée dans [Carroll and Weir, 1997] et [Manning and Carpenter, 1997].

L'angle de vue que propose l'opposition entre reconnaissance et analyse en TALN nous a permis d'aborder dans un premier aperçu les techniques et approches symboliques et numériques. Nous pouvons à présent étudier pas à pas la complexification formelle des théories et des techniques menant à l'analyse symbolique que nous entendons développer ici. Le choix d'une technique symbolique nous parait essentiel si on garde à l'esprit que l'expertise linguistique n'est pas à écarter du TALN. Notre trajet nous conduit donc vers les approches symboliques basées sur les contraintes.

1.3 La notion de contraintes dans les approches symboliques

Les *grammaires d'unification*, basées sur la reconnaissance et l'unification de patrons, telles FUG, LFG, DCG, GPSG, HPSG, puis la *théorie de l'optimalité* et les *Grammaires de Propriétés* considèrent l'analyse comme une succession d'unifications entre la chaîne analysée (*input*) et les patrons (arbres, atomes et structures de traits) de la grammaire.

Dans ces derniers formalismes, la représentation du modèle linguistique est plus complexe que dans les premières grammaires dérivationnelles, car apparaît la notion de traits et la reconnaissance devient un processus d'unification. L'emploi des contraintes, apparaît peu à peu, d'abord comme une fonctionnalité qui soutient le processus d'analyse et d'unification basé sur les règles, puis comme l'unique base de calcul (et s'oppose dans ce cas à l'emploi de règles dérivationelles). Les principes mêmes de la dérivation -la concaténation et la substitution- deviennent des contraintes parmi d'autres dans ces derniers modèles dès l'apparition du formalisme DI/PL.

L'évolution du TALN est marquée au sein des approches symboliques par l'apparition des techniques et des formalismes basés sur les contraintes. L'interprétation du terme "contrainte" étant sujette à des discussions ultérieures, nous en donnons pour l'instant une première description. En analyse par contraintes, la grammaire est un ensemble de clauses logiques pouvant être satisfaites en fonction de l'existence d'éléments linguistiques dans la chaîne qui sera analysée. La nuance entre les termes de contrainte et de règle réside dans l'interprétation qui est faite de ces mots au moment de l'analyse. Les règles qui échouent conduisent à l'échec d'un sous-espace d'analyse. L'approche par contraintes, quant à elle, donne lieu à la caractérisation des phénomènes tels que les écarts à la norme, les erreurs, les constructions incomplètes etc., mais reste robuste même en cas d'échecs locaux.

Le caractère spécifique de l'approche par contraintes est la satisfaisabilité. En effet, une fois l'étape de reconnaissance et d'unification effectuée, la satisfaction comme la violation des contraintes amène le processus d'analyse à répondre non seulement à la question de l'appartenance de la chaîne au langage, mais aussi à cette question plus fine : comment le langage que décrit la grammaire accepte-t-il la chaîne analysée ? On ne cherche plus alors à savoir si la chaîne est *grammaticale*, mais on donne l'ensemble des contraintes qu'elle satisfait ou viole comme *trace de l'analyse*. L'intérêt d'une telle approche est la plus grande robustesse des traitements qu'elle autorise. L'information linguistique produite par l'analyse est non-seulement constituée des symboles du vocabulaire non-terminal de la grammaire, mais aussi de l'ensemble des contraintes évaluées.

Avec l'emploi des grammaires lexicalisées (utilisation de valeurs de traits), LFG voit le jour (voir [Bresnan and Kaplan, 1982]). LFG reste un formalisme légèrement sensible au contexte.

La complexification progressive des outils et formalismes symboliques que nous avons présentés précédemment, ainsi que les carences en termes de couverture (la théorie d'une langue doit pouvoir décrire l'usage réel de cette langue) et de robustesse, ont permis l'apparition de formalismes pour lesquels les phénomènes tels que la dépendance et la dérivation doivent être séparés. C'est ainsi que le formalisme DI/PL est né. Avec la généralisation des structures de traits, l'emploi de principes et de contraintes sur les objets décrits par le formalisme, GPSG (voir [Gazdar *et al.*, 1985]) voit le jour. Schieber proposera une révision de l'algorithme d'Earley permettant une analyse rapide dans DI/PL avec GPSG.

Les opérations avec GPSG semblaient insuffisantes dans bien des situations, ce qui a permis au formalisme TAG ([Joshi, 1987]) de se développer. Bien que GPSG ait fortement influencé TAG, l'idée qui fonde ce dernier formalisme réside dans l'intégration complète de la grammaire au lexique sous forme d'arbres lexicalisés.

De nouveaux travaux inspirés de GPSG et de TAG donnent naissance à HPSG ([Pollard and Sag, 1987]). On peut dès lors parler d'analyses et de théories basées sur les contraintes. Joshi remarque l'équivalence forte entre HPSG et TAG ([Joshi *et al.*, 1991]), et de nombreuses plateformes de développement de grammaires et d'analyseurs voient le jour en intégrant la possibilité de convertir des grammaires GPSG ou HPSG vers des grammaires context-free

(comme dans [Phillips and Thompson, 1985]).

Un foisonnement de formalisations marque les années 1980, mais les grands courants se sont stabilisés. Les problèmes nouveaux rencontrés par les formalismes basés sur les contraintes sont liés à la robustesse de l'analyse et à la couverture grammaticale, plus qu'à la représentabilité de la langue (qui était le problème des grammaires transformationnelles).

Remarquons aussi que parmi les approches basées sur les contraintes, différentes techniques sont souvent surajoutées pour gérer les questions ayant trait à la dépendance. Le mariage difficile entre grammaires de constituants et grammaires de dépendance est proposé dans [Bès and Blache, 1999]. C'est à partir de ces derniers travaux que les Grammaires de Propriétés voient le jour. Nous les présenterons en détail à la fin de cette partie.

Du point de vue computationnel, il est intéressant de remarquer que les techniques et les algorithmes se complexifient progressivement, parallèlement à l'accroissement du formalisme auquel ils se réfèrent. L'analyse par contraintes n'est plus du tout possible en temps polynômial, sauf dans des situations très particulières. L'unification est un processus assez lent. Il découle de ces problèmes pratiques, que de nombreuses techniques stochastiques ou symboliques mais basées sur des simplifications et des heuristiques continueront longtemps à être utilisées en TAL. Nous ne souhaitons pas réduire le champ de l'analyse syntaxique à la seule caractérisation de phénomènes superficiels. Le propos de notre travail est ici lié à la robustesse mais sans perte d'information au prix de la rapidité. Nous verrons plus loin dans la partie consacrée au développement d'analyseurs fondés sur les Grammaires de Propriétés, que la complexité des systèmes basés sur les contraintes est très difficilement calculable, mais reste mesurable.

1.4 Résumé

Les relations qu'entretient la linguistique computationelle avec les formalismes linguistiques sont d'une part orientées par les besoins opérationnels et les capacités réelles des outils utilisés, d'autre part guidés par un souci d'adhésion aux modèles théoriques. Avec l'évolution des outils, le TALN a peu à peu pu se détacher des contingences matérielles pour proposer réellement ses propres formalismes en énonçant de véritables théories linguistiques et en s'attachant à un souci de cohérence conceptuelle, quitte à affronter des problèmes de plus en plus difficiles sans économie de temps ou de mémoire. L'algorithmique employée dans les nouvelles approches cherche cependant à vérifier un ensemble de critères toujours incontournables.

La table 1 ci-dessous donne un résumé des principales approches symboliques que nous avons décrites précédemment. On peut y comparer l'évolution des paradigmes théoriques, des formalismes et des techniques. De la reconnaissance à l'analyse, des systèmes à automates jusqu'aux approches basées sur les contraintes, la progression met en évidence une complexification algorithmique.

Pour continuer l'étude de cette complexification et définir la problématique de notre étude, nous allons à présent observer en détail certains points caractéristiques en analyse syntaxique aujourd'hui.

Paradigme théorique	**Formalisme**	**Techniques, ressources et algorithmes**	Décidabilité	Générativité	Reconnaissance/ Analyse Informativité		Emboitement	Normativité/ Robustesse	**Notes**	**Complexité algorithmique**
Théorie Standard Etendue	Théorie des langages, Grammaires de Chomsky type 2 et 3 : grammaires context-free et grammaires régulières	Automates à états finis	+	++	R	-	selon la gr.	N+ R-	Syntagmes ([Chomsky, 1965])	Linéraire à polynomiale
Théorie Standard Etendue	Théorie des langage, grammaires de Chomsky type 1 : grammaires sensibles au contexte	Automates à piles	+	+	R	-	selon la gr.	N+ R-	Syntagmes	de polynomiale à exponentielle selon les approches et les heuristiques
Théorie Standard Etendue	Théorie des langage, grammaires de Chomsky type 0 : grammaires non contraintes	Automates à piles	-	+	R	-	selon la gr.	N+ R-	Syntagmes	de polynomiale à exponentielle selon les approches et les heuristiques
Grammaires transformationelles ([Kuno and Oettinger, 1962])	Augmented Transition Networks ([Woods, 1970])	Réseaux de transition	-	-	R	-	+	N+ R-	Syntagmes	
Grammaires de métamorphoses (COL 75)	Théories des langages, Logique.	Lexique-Grammaire ([Gross, 1975]) Programmation fonctionnelle (Lisp) et logique (Prolog)	-	-	R	-	+	N+ R-	Syntagmes et traits	
Grammaires de clauses définies ([Pereira and Warren, 1980])	CDG, Théories des langages, Logique		-	-	R	-	+	N+ R-	Difficiles à maintenir	
Héritage des grammaires dérivationelles context free, grammaires d'unification ([Bresnan and Kaplan, 1982])	Grammaires Lexicales Fonctionnelles, non transformationelles	LFG et théorie des automates	-	-	R	+	+	N+ R-	Syntagmes , fonctions grammaticales, structures de traits Equivalence avec des Gr. de Chomsky type 1	
Héritage des grammaires dérivationelles context free, grammaires d'unification.	GPSG ([Gazdar *et al.*, 1985]), formalisme DI/PL		-	-	A	++	++	N- R+	Structures de traits, partage de traits entre syntagmes. Principes. Contraintes sur les objets. Equivalence avec des Gr. de Chomsky type 2	
Héritage des grammaires dérivationelles contextuelles, grammaires d'unification.	TAG ([Joshi, 1987]) Grammaires d'Arbres Adjoints	Combinaison (substitution et adjonction) de structures de traits	-	-	A	++	++	N- R+	La grammaire est placée dans le lexique sous forme d'arbres lexicalisés. Contraintes sur les objets. Equivalence avec des Gr. de Chomsky type 1	
Héritage de GPSG	HPSG ([Pollard and Sag, 1987])	Analyse ascendante (bottom-up parsing)	-	-	A	++	++	N- R+	Contraintes sur les objets. Equivalence avec des Gr. de Chomsky type 1	
	Théorie de l'optimalité (OT) ([Prince and Smolensky, 1993])	Generer puis choisir la proposition optimale.	-	+	A	++	++	N- R+		
Héritage de HPSG	Grammaires de Propriétés ([Blache, 2001])	Analyse ascendante (bottom-up parsing)	-	-	A	++	++	N- R+		complexité théorique exponentielle, mais polynomiale dans les situations réelles

TAB. 1 – Comparaison des théories et de leurs applications

Chapitre 2

Discussion des critères déterminants en analyse syntaxique

Avant d'aborder le choix du formalisme des Grammaires de Propriétés, nous voulons discuter les points critiques en analyse syntaxique. Nous entendons par *point critique* un terme ou une notion qui peut actuellement être considérée comme discriminante et caractéristique d'une approche en analyse syntaxique. Au delà de la classification des systèmes par type d'approche (symbolique ou numérique), il est important de situer dans les techniques ce qui les rendent spécifiques et différentes des autres. Nous pensons que ces points sont caractéristiques de la situation actuelle dans le domaine de l'analyse syntaxique et qu'ils peuvent servir de *critères déterminants* pour orienter les choix de conception d'une approche plus robuste et plus souple.

2.1 Autour du mot *syntaxe*

La rigueur qu'exige le travail de recherche ne doit pas être contrarié par les contraintes qu'imposent les applications. De fait, les critères que nous avons énumérés ci-dessus montrent à quel point chaque choix théorique et algorithmique aura des conséquences sur les résultats. Un programme deviendrait vite un gadget si nous oubliions le contexte qui l'a vu naître.

Afin de contrôler ces variables à chaque étape de nos développements, il est nécessaire de se donner des directions épistémologiques pour nos démarches. Ce cadre, déplaçable pour tout projet scientifique trouve un sens dans la définition de l'objet étudié.

L'objet de notre étude est l'analyse syntaxique automatique, au sens le plus ouvert.

De manière formelle, prenons un peu de recul par rapport à cet objet.

1. Une théorie scientifique, susceptible d'évoluer, se place derrière chaque formalisme que nous pourrions adopter.
2. Une axiomatique qui ne doit pas rester implicite guide les évolutions de plusieurs théories ayant le même objet. Cette axiomatique est la supposition de cet objet. Elle admet une logique qui est indépendante des évolutions des théories sous-jacentes. Cette logique et ses théorèmes sont le seul contexte dans lequel nous pouvons affirmer qu'un programme est valide.

3. Notre démarche doit rester autant que possible indépendante de la théorie, mais non de l'axiomatique. Si plusieurs tendaces théoriques s'opposent et se contredisent, un choix s'avère nécessaire parmi celles-ci pour permettre des développements pratiques.

D'autre part, du point de vue informatique, le terme syntaxe ne désigne pas le même objet qu'en linguistique. En TALN, le double héritage théorique (linguistique et informatique) nous amène à considérer le mot syntaxe avec une extrême précision. Nous posons le présupposé suivant qui en donnera le contour pour nos travaux.

La syntaxe est caractérisée par ses éléments que sont :
– Les *syntagmes*, repérés comme parties du *texte*. Un syntagme est lui-même constitué de *mots* (composés ou non), auxquels sont associées des *catégories*.
– Les *catégories syntaxiques*, associées à chaque syntagme ou à chaque mot.

Nous pourrions adopter une autre axiomatique de la syntaxe (par exemple en définissant le syntagme à partir de mots simples et en ne reconnaissant pas les mots composés), ce qui rendrait immédiatement caduques tous les choix effectués ci-dessous et dans la suite de nos travaux. Une révision de cette base nécessiterait la refonte totale des théories et outils développés ici.

Dans cette acception, l'analyse syntaxique qui pourra être effectuée couvrira non seulement la syntaxe au sens linguistique, mais aussi toute forme d'analyse d'informations pouvant être transcrites (analyse prosodique, morphologique, sémantique, pragmatique, discursive, etc.) Cet élargissement du sens découle de l'approche informatique inaugurée par la théorie des langages et la compilation des programmes. Les objets manipulés en deçà de ce terme, syntagmes et catégories -quoique ces termes soient empruntés au vocabulaire de la linguistique-, sont à définir pour chaque type d'analyse voulue. Nous proposons donc les précisions suivantes :

Toute théorie du texte qui en suggérerait un découpage en *groupes d'informations* dotés d'une *catégorie* sera appelée dans nos travaux une *syntaxe*.

Ainsi, la syntaxe et les autres domaines de la linguistique n'ont pas forcément à être en concurrence au moment du traitement du texte, mais peuvent simultanément être considérés comme des théories fondées sur une axiomatique du découpage textuel en unités pertinentes, que par abus de langage nous appellerons syntaxe.

La superposition des unités obtenues par deux analyses dans le cadre de deux théories fondées par une même axiomatique syntaxique n'est pas garantie, c'est le cadre dans lequel nous devons travailler.

2.2 Des critères formels aux critères applicatifs

Au delà des différences de conception que nous avons entrevues jusqu'à présent, un certain nombre de critères d'analyse se retrouvent au sein de chaque approche. Ces critères sont fournis d'une part par les théories linguistiques et d'autre part par les formalismes informatiques qui

les manipulent. Avant de classer les techniques face aux théories qui les gouvernent, nous avons tout intérêt à considérer ces critères, à les définir, à les classer eux-mêmes, afin qu'ils nous servent de cadre dans l'étude qui suit.

2.2.1 Informativité

Les théories des langues, syntaxes et grammaires (linguistique) et les théories des langages formels (informatique) se donnent des objets d'étude et définissent les propriétés de ces objets. L'informativité peut alors devenir une notion pertinente en tant que critère d'analyse pour les deux raisons suivantes :

- Parmi ces objets, certains sont posés *a-priori* -donnés comme principes, présupposés ou axiomes - (comme par exemple, l'idée de *mot*) et leur interprétation peut différer d'une théorie à l'autre.

- Ces théories offrent des axiomatisations et des représentations qui peuvent aller de l'extrêmement schématique à une vision complexe de la syntaxe en passant par des chemins soit formels (rationnels, expérimentables) soit notionels (flous, se croisant et se recouvrant parfois, interprétables).

Par *informativité* d'une théorie, nous entendons alors son pouvoir descriptif, son vocabulaire et l'organisation de ses éléments...

Il sera nécessaire, lorsqu'on parlera d'une analyse syntaxique, de situer son niveau d'informativité. Les applications informatiques doivent être en adéquation avec la structure informative de la théorie qu'elle formalise.
Vérifier cette adéquation suppose la connaissance des représentations informatiques associées à chaque objet de la théorie. C'est par cette vérification que sera possible la classification des algorithmes en termes de couverture théorique. En effet, c'est en reconnaissant la dimension et la portée des notions posées comme axiomes dans les théories que nous pourrons reconnaître la généralité ou la spécificité des techniques et des formalismes étudiés.
Une hiérarchie des termes et des notions entre théories doit être envisagée et décrite ; travail qui appartient aux linguistes et qui ne sera ici que partiellement élaboré : les objets manipulés seront comparés d'une théorie à l'autre en questionnant les termes fondamentaux que sont

- la langue : quelle(s) langue(s) sont couvertes par une théorie ? La théorie admet-elle les intersections, les emplois de plusieurs langues ? Le formalisme proposé admet-il des écarts à la norme ? Tous les faits d'une langue sont-ils couverts ?
- le mot : mot simple, composé, acronyme, chaîne de caractères, suite de signes, idéogramme, transcription orale ...
- la ponctuation : quel est son rôle ? est-elle un signe syntaxique ? ...
- la catégorie syntaxique : syntagme, paradigme, catégorie plus générale (au sens informatique), le chunk (parenthésage sans catégorisation) ...
- les traits : sous-catégories ou ensembles de traits, hiérarchisés ou non, disjoints ou à intersection non-vide ...
- la phrase : limitée par des marques observables non syntaxiques, bornée ou non bornée ...
- la hiérarchisation : les structures proposées peuvent-elles être emboîtées ? Quelles contraintes pèsent sur leur imbrication ?

- la syntaxe : la syntaxe elle même ne prend pas le même sens du point de vue linguistique (où elle concerne la structure des faits de langue) et du point de vue informatique (où elle concerne plus largement la structure de suites d'informations).
- la modalité : les objets analysés sont-ils consécutifs ? Certaines approches récentes permettent l'analyse simultanée de plusieurs messages cooccurents (geste et parole par exemple).

2.2.2 Pouvoir génératif / pouvoir analytique

Une notion qui oriente et alimente les débats entre linguistes et à présent entre linguistes et informaticiens est la *générativité* d'une théorie linguistique. Cette notion découle de l'affirmation suivante : la théorisation d'une langue doit être à même de décrire ses productions. Une grammaire doit donc, si on se positionne *a priori* dans une acceptation directe de cette idée, être générative, c'est à dire permettre de produire toutes les phrases bien formées d'une langue.
Envisager les choses de cette manière amène à construire des grammaires pour lesquelles la production et l'analyse de faits de langue dépend d'un seule et même ensemble d'objets et de règles.
D'autres courants admettent une idée différente, selon laquelle la génération et l'analyse peuvent ne pas relever du même processus ni d'une même grammaire.
Le fait que l'analyse suppose la génération n'est pas surprenant. Lorsque nous questionnerons les formalismes et les techniques, nous observerons une complexification des représentation qui rend l'analyse possible tout en interdisant la génération. Il nous semble possible de classer les approches d'analyse en fonction de leur pouvoir génératif.

2.2.3 Déterminisme et ambiguïté

En informatique, la représentation d'une langue sera dite déterministe si les analyses qu'elle permet aboutissent à la construction d'une et une seule description pour une une chaîne donnée. Une grammaire déterministe ne permet donc qu'une interprétation et s'oppose en cela à l'idée qu'une langue puisse être ambiguë. On trouve beaucoup plus de théories des langages informatiques dotées de grammaires déterministes que parmi les théories linguistiques, mais l'héritage des premières tentatives dans le domaine du TALN a maintenu la notion de déterminisme au sein des critères d'analyse. De fait, atteindre une seule interprétation est beaucoup moins coûteux que chercher toutes les interprétations. On peut à présent considérer des grammaires non-déterministes dont l'application informatique peut être rendue déterministe. La complexification des formalismes peut aussi ne plus autoriser d'interprétation unique, sauf au prix d'un travail de filtrage, d'heuristiques plus ou moins théoriquement motivées. Tous ces éléments sont à prendre en compte en étudiant les techniques d'analyse. Permettent-elles une analyse avec maintien de l'ambiguïté ou bien permettent-elles une analyse déterministe ?

2.2.4 Grammaticalité

L'analyse que permet une théorie est -comme nous avons pu l'observer dans l'étude comparative entre reconnaissance et analyse- intimement liée à la notion de grammaticalité. Celle-ci permet de classer les techniques en deux familles elles-mêmes divisibles en deux classes complémentaires :

- par rapport à l'écart à la grammaire : celles qui le tolèrent et celles qui ne le permettent pas,
- par rapport à la modalité de la réponse : celles qui déterminent si un message est grammatical et celles qui donnent des informations grammaticales sur un message donné.

Cette classification permet de différencier les approches, de montrer leurs limites. nous la retiendrons par la suite parmi les critères d'analyse.

2.2.5 Complexité syntaxique

Pour la seconde classe de la seconde famille citée ci-dessus, au delà de la question de grammaticalité, il faudra chercher à différencier plus finement de quelle manière le degré de grammaticalité peut être quantifié. D'une part au sein d'une approche, d'autre part comparativement entre des approches. Cette question se place parmi les critères d'analyse et a aussi un rôle essentiel a jouer dans le processus d'évaluation des analyseurs, dont nous parlerons dans la prochaine partie.

2.2.6 Couverture théorique

Entre la théorie et son application, les données, les objets et les traitements peuvent ne pas correspondre exactement : on assiste souvent, au moment de la programmation, à des simplifications ou à des généralisations. Il s'agit ici d'observer attentivement ces transformations, de percevoir les recouvrements, les lacunes et les transgressions entre le modèle applicatif et le modèle théorique.
Il peut se produire que l'algorithme mis en œuvre, le programme, ses données et ses structures, puissent fonctionner dans plusieurs modèles théoriques. Il faut considérer dans ce cas que l'algorithme implanté est suffisamment général pour englober plusieurs tendances théoriques. C'est une particularité notable des algorithmes que de permettre le traitement d'objets différents, mais similaires -appartenant à des modèles parfois totalement disjoints. Ainsi, le même algorithme probabiliste peut s'appliquer aux données climatologiques, boursières, phonétiques, syntaxiques etc.
Nous devons donc considérer cette couverture théorique comme une caractéristique majeure des applications étudiées. Dans les développements ultérieurs, cette considération deviendra un critère essentiel permettant la description des algorithmes aussi bien qu'un critère d'évolutivité.

2.2.7 Traçabilité

Plus près de l'application informatique elle-même se pose la question de la trace qu'elle laisse après son exécution . Traiter des données signifie produire une *sortie* à partir d'une *entrée*. Dans cette *sortie*, nous distinguerons :

le *produit* la donnée résultant directement du traitement de l'*entrée*

la *trace* la donnée constituée par l'historique des transformations appliquées à l'*entrée*

La seconde n'est pas toujours accessible, ce qui rend alors la première opaque. Les approches dites symboliques permettent d'accéder à cette trace. Mais avec la complexification des algorithmes, cette dernière devient de plus en plus difficile à conserver (soit à cause de son volume, soit à cause du coût calculatoire nécessaire à sa conservation). En ce qui concerne l'analyse syntaxique, les approches présentées plus bas devront être observées à travers ce critère.

2.2.8 Terminaison, calculabilité et décidabilité

Faisant appel aux théories linguistiques impliquées dans le calcul, aux caractéristiques des modèles de représentation et aux algorithmes mis en jeu, les notions de décidabilité, de calculabilité et de terminaison sont centraux dans l'évaluation des techniques mises en place. Pour résumer ce problème, lorsque nous développons un analyseur syntaxique, il est nécessaire de savoir si le calcul se terminera dans un temps raisonnable quelle que soit l'*entrée* présentée au programme. Démontrer qu'un calcul termine dans un temps acceptable consiste en deux étapes :

décidabilité et calculabilité : Il faut évaluer le problème lui-même et le classer parmi les problèmes simples, difficiles ou NP-difficiles (théorème de Cook). Cette démonstration relève du calcul de complexité (voir plu bas).

terminaison : Il faut analyser les processus du programme en déterminant quand une récursion ou une itération infinie peut se produire. Cette dernière opération s'effectue en fonction de la catégorie des algorithmes étudiés et de la structuration des données.

2.2.9 Complexité informatique

La complexité est le critère le plus important au sens informatique, une fois vérifiée la cohérence entre le programme et le modèle. L'entreprise de conception de projet place souvent ces deux critères sur le même plan. L'un n'allant pas sans l'autre. Par complexité, il faut entendre coût calculatoire des algorithmes mis en œuvre. Dans la notion de coût, en informatique, on doit distinguer :

le coût théorique (CT) celui-ci se divise en :

– le coût théorique en nombre d'opérations élémentaires $CTO(n)$ qu'un algorithme effectuera pour une entrée de taille n. Ce nombre est évalué mathématiquement en dehors de toute mise en pratique.

– le coût théorique en nombre d'unités de mémoire $CTM(n)$ qu'un algorithme allouera pour une entrée de taille n. Ce nombre est évalué mathématiquement en dehors de toute mise en pratique.

le coût effectif (CE) celui-ci se divise à son tour en :

– le nombre d'opérations élémentaires $CEO(n)$ qu'un algorithme effectuera pour une entrée de taille n. Ce nombre est évalué statistiquement par la mise en pratique de l'algorithme sur des données concrètes.

– le coût effectif en nombre d'unités de mémoire $CEM(n)$ qu'un algorithme allouera pour une entrée de taille n. Ce nombre est évalué statistiquement par la mise en pratique de l'algorithme sur des données concrètes.

L'évaluation quantitative du temps de calcul n'est pas une bonne indication, car elle dépend du matériel utilisé. On parle pourtant de *temps* par abus de langage, pour désigner la complexité d'un algorithme. Les fonctions de complexité calculées ou mesurées se ramènent à de grandes familles de fonctions : les fonctions linéaires, polynômiales, exponentielles, factorielles .... Il est essentiel pour connaître la faisabilité d'un calcul, de savoir évaluer cette complexité, de savoir la comparer, et pour les algorithmes à grande variabilité, de la donner en termes de moyenne et d'*extrema*. Ce critère est primordial lorsqu'on procède à l'évaluation des algorithmes au

sens informatique. Du point de vue de l'analyse syntaxique, ce critère est souvent antagoniste à la qualité des résultats. Le critère d'évaluation qui correspond à cette qualité linguistique des traitements se compose de plusieurs critères déjà abordés ici : la cohérence théorique, l'informativité du modèle. Il faut à cela ajouter un critère plus délicat et qui concerne les relations qu'entretiennent les théories et les grammaires entre elles : l'évaluabilité des résultats, que nous décrirons dans la quatrième partie de cette étude. Les problèmes posés par l'analyse syntaxique en termes de complexité sont très intéressants [Barton *et al.*, 1987], mais à prendre avec précaution. Quand on parle de complexité, c'est en fonction de la longueur de la phrase à analyser. La taille de la grammaire (facteur constant) pouvant en pratique jouer un rôle beaucoup plus important, de même que les temps d'accès au lexique.

2.2.10 Robustesse

Quel que soit le système utilisé, il est difficile de prédire si celui-ci sera susceptible de fournir une analyse correcte ou bien d'échouer pour une entrée donnée. La robustesse d'un système qualifie sa capacité à répondre dans des conditions quelconques. Celle-ci peut s'évaluer en établissant une classification des types d'entrée. Des textes littéraires, journalistiques, spécifiques à des domaines (médicaux, administratifs, juridiques etc.) ou encore des transcriptions de l'oral, des mails etc. présentent des différences telles que la robustesse des systèmes qui les analysent peut y être comparée. Nous évaluerons ainsi nos analyseurs en comparant leurs scores de rappel et de précision par type de corpus dans la quatrième partie. Dans sa thèse, Nuria Gala présente des travaux directement liés au problème de la robustesse (voir [Pavia, 2003]). Son choix opérationnel a consisté en une construction de grammaires modulaires (où chaque module est un système de transducteurs) permettant de résoudre au cas par cas les difficultés résiduelles rencontrées durant l'analyse. Ainsi, certains phénomènes particuliers tels que les titres, les expressions parenthésées etc. reçoivent un traitement spécifique. Nous sommes nous aussi très conscients de la nécessité d'avoir un analyseur robuste, mais sans faire le choix préalable d'un système de résolution. Même si le choix d'un formalisme est nécessaire, nous chercherons à définir autant que possible un analyseur robuste pour lequel une seule grammaire est utilisée plutôt qu'un ensemble de modules grammaticaux. L'interaction entre les modules suppose de fait une précédence dans l'ordonnancement des tâches qui nous semble incompatible avec une analyse par contraintes pour laquelle chaque phénomène ne prime pas sur un autre.

2.3 Granularité

L'ensemble des questions qui se sont posées jusqu'à présent nous amène à définir enfin le coeur de notre problématique. Entre robustesse et richesse d'information, il semble n'y avoir aujourd'hui qu'une équation de relation inversement proportionnelle. Plus les analyseurs sont robustes, et moins ils sont informatifs. La notion de granularité, qui est un mot sans doute trop employé, trop flou ou encore trop porteur de sens, nous interpelle et demande à être précisée.

Les algorithmes d'analyse syntaxique que nous allons étudier plus loin diffèrent du point de vue théorique, en informativité et en couverture. Les différences que nous pouvons observer entre les niveaux d'information qu'offrent les théories se répercutent sur la conception des algorithmes. Il en découle une variabilité de la qualité des analyses entre les approches. La notion de *granularité* désigne bien cette idée qu'une approche peut être plus ou moins fine

qu'une autre. La mesure de ce *niveau de granularité* peut se définir en fonction de la richesse des objets et de la capacité d'emboitement des structures que les algorithmes manipulent. On pourra alors distinguer :

- la granularité en **niveau d'imbrication** : opposant par exemple les structures plates et superficielles des *chunkers* aux analyses hiérarchisées et profondes qu'offrent les *grammaires syntagmatiques*.
- la granularité en termes de **diversité des paradigmes interprétatifs** : opposant les techniques qui n'offrent qu'une information par élément analysé à celles qui peuvent proposer pour un même élément un ensemble d'informations liées à des phénomènes linguistiques différents (par exemple une information syntagmatique, une information sémantique et une fonctionnelle).
- la granularité en termes de **multiplicité des interprétations** au sein du même paradigme interprétatif : opposant les techniques qui n'offrent qu'une information par élément analysé à celles qui peuvent proposer pour un même élément un ensemble d'informations liées à des phénomènes linguistiques différents (par exemple une information syntagmatique, une information sémantique et une fonctionnelle).

Certaines applications ont besoin la plupart du temps d'une analyse peu profonde et dans certaines situations d'une information plus détaillée. C'est typiquement le cas pour les systèmes de synthèse vocale. De telles applications se fondent habituellement sur des analyseurs peu profonds afin de calculer les groupes intonatifs sur la base d'unités syntaxiques (ou plus précisément sur des 'chunks'), en accord avec les recommandations d'auteurs de tels systèmes comme Allen, S. Hunnincutt, R. Carlson et B. Granström (voir [Allen *et al.*, 1979] et [Allen *et al.*, 1987]) :

Un système de synthèse vocale demande des choses plutôt inhabituelles à un analyseur. " Il doit avoir une large (quoique légère autant que possible) couverture des textes, sans restriction, plutôt qu'une analyse profonde d'un domaine restreint. L'échec de l'analyse est inacceptable dans un système TTS."

Dans certains cas, une information trop superficielle n'est pas assez précise. En synthèse vocale, mais aussi pour le dialogue homme-machine ou la recherche d'informations dans de grands corpus. Une solution consisterait alors à employer une analyse approfondie pour quelques constructions qui le nécessiteraient. Aucun système mettant en application une telle approche n'existe. C'est dû en particulier au fait que ceci exigerait deux traitements différents, le second refaisant en fait la totalité du travail. De fait, il est difficile d'imaginer dans le cadre génératif classique de mettre en application une technique d'analyse capable de calculer des 'chunks', et dans certains cas, des expressions plus complexes avec une organisation hiérarchique.

Notre problème est celui-ci : comment développer un outil d'analyse tantôt superficiel, tantôt profond, basé sur les mêmes ressources linguistiques, pouvant s'adapter à de nouvelles ressources et pouvant intégrer plusieurs interprétations pour une même entrée ? Découpons ce problème en deux parties :

- Il nous faut un *outil d'analyse à granularité variable*. Cet outil permettra de réaliser une analyse plus ou moins profonde en fonction du contexte. Pour cela, il nous faudra définir les critères qui permettent de détecter un contexte requérant une analyse approfondie ou au contraire une analyse superficielle.
- Il nous faut un outil capable d'effectuer une *sélection de granularité entre différentes*

techniques. Cet outil permettra de gérer des interprétations concurrentes pour un même texte. Il peut dans ce cas s'agir de gérer l'ambiguité au sein du même paradigme interprétatif (par exemple en règlant le niveau d'ambiguité toléré pour une interprétation syntaxique) ou encore entre paradigmes interprétatifs (par exemple un contour syntaxique et un contour prosodique).

2.3.1 Granularité variable

Le sujet de cette thèse est d'envisager les possibilités d'une évolution des analyseurs dans le sens d'une sélection automatique de granularité en fonction du contexte et des besoins. Nous posons comme problématique la recherche des possibilités en termes de variation de granularité.
La position de ce problème est à la fois théorique (comment déterminer localement le niveau d'information requis ? Quelles motivations et quelles théories pour caractériser ce besoin ?) que pratique (comment définir un algorithme permettant cette sélection de granulatité ?). Cette question nous conduit directement à envisager d'utiliser le formalisme des Grammaires de Propriétés, basé sur l'analyse par satisfaction de contraintes et qui permet aussi bien de représenter des grammmaires syntaxiques que sémantiques ou encore d'autres phénomènes linguistiques.

Chapitre 3

Les Grammaires de Propriétés

Nous avons à présent une idée précise des différences qui existent entre les courants du TALN. Notre objectif computationel nous oblige à écarter les techniques générativistes classiques et à rechercher une approche dans laquelle la souplesse n'est pas un mécanisme adjoint, mais une propriété intrinsèque du formalisme. La théorie choisie doit répondre au mieux aux critères énumérés précédemment, notamment celui portant sur la granularité, qui est au centre de notre problématique. Les approches basées sur les contraintes, qui correspondent le mieux à notre attente, diffèrent beaucoup dans leur formalisation ainsi que dans les particularités techniques qui correspondent à leur mise en œuvre. Nous allons décider ici du choix des Grammaires de Propriétés en considérant en détail les caractéristiques qui distinguent cette approche des autres théories basées sur les contraintes. Nous définissons dans cette perspective un ensemble de termes et de modèles permettant la description de ces grammaires. Au delà de ce choix, nous précisons les conditions d'utilisation de cette théorie à des fins pratiques (ce point sera à nouveau abordé lorsque nous décrirons les algorithmes et les techniques d'implantation).

Nous proposons ici une présentation progressive des Grammaires de Propriétés. Tout d'abord, nous présentons les *concepts et notions* qui entourent le problème de l'analyse avec les GPs. Ensuite, nous élaborons une *modélisation* très précise de ce formalisme. Enfin, nous proposons une réflexion et des développements visant à choisir une *représentation* des GPs.

Comme la présentation de ce formalisme nécessite une illustration aussi précise que possible pour rendre compréhensibles certains problèmes et choix décrits au fil de ce chapitre, nous incitons le lecteur à se référer aux exemples de grammaire donnés en annexes A à E ainsi qu'aux figures 52 à 56 de la troisième partie.

3.1 Concepts fondamentaux

Les Grammaires de Propriétés permettent de représenter l'état d'un système de contraintes portant sur des éléments de l'énoncé analysé. Si nous voulons définir précisément ce système de contraintes, nous devons au préalable nous donner un vocabulaire de ces grammaires.
Pour commencer, une grammaire de propriétés, telle que nous allons la définir est un ensemble de catégories.

Exemple 1. L'écriture $G = \{P, N, GN, V, GV\}$ définit une grammaire G dont les catégories sont P, N, GN et GV. Ces catégories doivent être précisées par la suite.

3.1.1 Catégories

Les éléments de l'énoncé analysé sont supposés étiquetés, c'est à dire qu'une étiquette de catégorie leur a été attribuée. Éventuellement, plusieurs étiquettes peuvent être associées au même mot de l'énoncé, ou aucune si le mot est inconnu. L'ensemble des catégories sur lesquelles peuvent porter les GPs dépend de l'usage que l'on souhaite en faire : ainsi, des catégories syntaxiques au sens de la linguistique structuraliste, des syntagmes, ou encore des catégories morphologiques, phonétiques, prosodiques, sémantiques, pragmatiques etc. L'idée de catégorie manipulée dans les GPs a un sens très large et rejoint celle de classe d'objets (voir à ce propos Russel (ensembles et éléments) et Gross (classes d'objets)). Les notions de *catégories terminales* et de *catégories non terminales* ne sont pas présentes dans le formalisme des GPs de la même façon que dans les grammaires génératives : ces dernières ont besoin d'une distinction fondamentale et initiale entre les catégories directement associées aux éléments lexicaux, tandis que les premières n'ont pas besoin de cette distinction pour être opérantes. Une catégorie, dans les Grammaires de Propriétés est un nom auquel sont associés d'une part un ensemble de traits possibles -regroupés par type de traits- et d'autre part un ensemble de contraintes portant sur d'autres catégories. Ces contraintes, aussi appelées propriétés, explicitent les liens éventuels entre une catégorie et les autres catégories de la grammaire. Une catégorie est implicitement terminale si elle n'est pas définie par des contraintes sur d'autres catégories. Elle sera dite non-terminale si elle se définit à partir de contraintes sur d'autres catégories. Rien n'empêche cependant une catégorie non-terminale des Grammaires de Propriétés d'être l'étiquette immédiate d'un élément du texte analysé.
Pour résumer, une catégorie des Grammaires de Propriétés est définie par deux ensembles : l'ensemble de ses propriétés (regroupées par type de propriété) et l'ensemble de ses traits possibles (regroupés par types de traits).

Exemple 2. L'écriture $GN = [\{\{Oblig : N\}, \{Prec : (Det, N)\}, \{Dep : (Det, N)\}\}, \{\{Genre : M, F\}, \{Nombre : S, P\}\}]$ définit une catégorie GN dont les types de propriétés sont Oblig, Prec et Dep et les types de traits sont Genre et Nombre.

3.1.2 Types de traits et traits

Étant donné que la notion de catégorie dans les Grammaires de Propriétés est une notion extrêmement large, dépendant de l'usage qu'on souhaite en faire, nous ne pouvons donner la liste exhaustive de l'ensemble des traits possibles pour toutes les catégories. Nous pouvons juste donner une définition assez générale de ce que nous entendons par trait. Pour une catégorie donnée, pour une grammaire donnée, un type de traits est un ensemble de valeurs pouvant être associées à la catégorie afin de la spécifier ou de la sous-catégoriser. Ainsi, un nom masculin ou féminin reste un nom, mais dont un type de traits nommé genre peut recevoir la valeur masculin ou féminin. La grammaire fait usage de traits pour éviter la multiplication des catégories et des écritures. Il est important de noter qu'au sein d'un même type de traits, les valeurs sont supposées mutuellement exclusives : dans notre exemple sur le genre des noms, il ne sera pas possible de sous-catégoriser un nom avec simultanément les traits de genre masculin et féminin. Il est toutefois possible d'introduire dans la grammaire des traits composites, dont le rôle doit permettre de couvrir les indéterminations, les invariances et les neutralités. Pour résumer, un type de traits est un ensemble nommé de traits mutuellement exclusifs, chacun d'eux étant repéré par un couple (valeur, description)

Exemple 3. L'écriture $\{Genre : M, F\}$ définit un type de traits appelé genre dont les valeurs de traits possibles sont M et F.

3.1.3 Types de propriété et propriétés

Les Grammaires de Propriétés font appel à des contraintes sur les catégories pour définir des catégories. Un type de propriété est un ensemble repéré par son nom, servant à regrouper les contraintes basées sur le même mécanisme de satisfaction. Les contraintes (appelées propriétés) sont des fonctions de vérité portant sur les catégories de la grammaire, dont la valeur (Vrai ou Faux) est calculable au cours de l'analyse. On dira qu'une propriété est satisfaite ou non satisfaite en fonction de la sémantique associée au type de propriété à laquelle elle correspond. Les Grammaires de Propriétés, telles que définies par Philippe Blache (cf [**?**]) font habituellement appel à six types de propriétés : la *linearité* (precédence linéaire), la *dependance* (correspondance de traits entre catégories), *obligation* (ensemble des tetes possibles), l'*exclusion* (restriction de cooccurrence), l'*exigence* (obligation de cooccurrence) et l'*unicité* (unicité d'une catégorie) :

Exemple 4. Obligation (notée Oblig) : Spécifie les têtes possibles d'un syntagme. Une de ces catégories (et une seule) doit etre réalisée (présente dans le syntagme analysé).
Oblig(SN) = {N, Pro}
Unicité (notée Uniq) : Ensemble des categories qui ne peuvent pas être répétées dans un syntagme.
Uniq(SN) = {Det, N, SA, SP, Sup, Pro}
Exigence (notée $\Rightarrow$) : Cooccurrence entre catégories.
N[com] $\Rightarrow$ Det
Exclusion (notée $\neq$) : restriction de cooccurrence entre ensembles de catégories.
SA $\neq$ Sup (dans un SN, un superlatif ne peut pas apparaître si un SA est présent)
Linearité (notée $<$) : précédence linéaire.
(Dans un SN) : Det $<$ N
Dependance (notée $\rightarrow$) : Relations de dépendance entre categories.

(dans un SN) : Det.genre $\rightarrow$N.genre (les traits de genre du déterminant et du nom doivent s'accorder).

3.1.4 Caractérisation

L'énoncé (le texte, la chaîne d'informations, l'input, l'entrée de l'analyse) est observé au travers du crible de la grammaire : le processus de satisfaction de contraintes associé à une grammaire est appelé *caractérisation*. La *caractérisation* est aussi le résultat de ce processus, c'est à dire l'ensemble des contraintes satisfaites par l'énoncé. Une contrainte peut être exprimée en fonction de la disponibilité d'une catégorie, ou en fonction de son indisponibilité. Par exemple, une contrainte d'exigence entre un Det et un N ne peut être satisfaite que si un Det et un N sont réalisés. Dans le même exemple, si un Det est présent et pas un N, alors la contrainte est non satisfaite. Si par contre un N est présent et pas un Det, on ne peut rien dire. Il y a dans la sémantique de chaque propriété une notion d'évaluabilité qui précède celle de satisfaisabilité. Nous verrons plus loin comment formaliser ce phénomène.

3.1.5 P+, P-, P0

La caractérisation est constituée de trois ensembles :

– le premier est l'ensemble des contraintes satisfaites, noté P^+,
– le second est l'ensemble des contraintes non satisfaites, noté P^-,
– le dernier est l'ensemble des contraintes non concernées par l'énoncé, noté P^0.

Nous élargissons la notion de caractérisation en y intégrant l'ensemble des catégories telles qu'au moins une des propriétés qui les définissent appartient à P^+. Nous développons ci-dessous le terme de *Projection des catégories*. La technique habituelle qui consiste à construire des catégories sur les catégories de l'énoncé ou sur les catégories déjà construites est un mécanisme obligatoire dans les grammaires syntagmatiques. On appelle ce mécanisme la projection dans les grammaires de dépendance. Dans la plupart des approches que nous avons étudiées, effectuer une analyse syntaxique consiste en l'élaboration d'un arbre ou d'un graphe qui relie entre elles des catégories. Dans le cadre des Grammaires de Propriétés, seule la caractérisation est considérée comme résultat du processus d'analyse. On passe cependant par une forme de construction, puisque l'ensemble des contraintes satisfaites à un niveau donné de l'analyse "active" les catégories qui sont définies par ces contraintes. Ces catégories actives satisfont à leur tour d'autres contraintes de la grammaire qui activent encore de nouvelles catégories. La question qui se pose autour de l'activation de ces catégories est cruciale. Lorsque notre formalisation aura été précisée, nous aurons une idée plus précise des possibilités computationnelles face à cette question. En attendant, nous pouvons nous donner deux orientations pour aborder la question de la construction : celle dans laquelle la Caractérisation des catégories est une contrainte comme une autre et celle dans laquelle elle est un processus non contraint, mais gouverné par un principe.

3.1.6 Projection des catégories non contrainte

Les Grammaires de Propriétés ne proposent pas de réponse à la question de la caractérisation des catégories. On peut alors supposer qu'une catégorie est "activée" dès lors qu'une seule des contraintes qui la définissent est satisfaite. On parlera alors de caractérisation des catégories non contrainte puisque sous cet angle de vue, un principe de construction gouverne l'analyse. Ce point de vue n'appartient pas à la théorie des Grammaires de Propriétés et n'est pas motivé linguistiquement. Il s'avèrera utile de le considérer dans la perspective computationnelle, mais de garder à l'esprit sa distance avec la théorie.

3.1.7 Projection des catégories contrainte

Si nous considérons la notion de constituance, qui n'appartient pas non plus au formalisme des GPs, une autre possibilité de caractérisation des catégories pourrait être dirigée par un type de propriétés appelé "constituance". Si une contrainte de constituance et une seule est satisfaite, un syntagme peut être caractérisé. L'idée de "noyau" se cache derrière l'expression de la constituance. Cette idée est aussi discutable que la première. La propriété d'obligation, dont nous avons déjà parlé peut jouer le rôle de contrainte de caractérisation des catégories (de constituance). Néanmoins, il faudra alors garder à l'esprit que cette technique ne fait pas partie du formalisme des GPs et n'est pertinente que dans une approche computationnelle.

3.1.8 Projection des catégories

Un mécanisme de construction de syntagmes (projection), adjoint au mécanisme d'analyse (caractérisation) permet de produire des arbres ou des graphes d'analyse syntaxique à la manière habituellement employée par la communauté scientifique dans le domaine du TALN. Nous mettons en avant la possibilité offerte par les Grammaires de Propriétés de le permettre avec les deux manières que nous avons données.

Dans la formalisation que nous élaborons, nous optons pour la caractérisation des catégories non contrainte. Cela peut concrètement être mis en place grâce à un un marqueur booléen dans la définition des types de propriétés : ce marqueur exprime pour chaque type de propriétés la possibilité de caractériser un syntagme lorsqu'une une propriété de ce type est satisfaite. Ce marqueur n'alourdit en rien la grammaire et permettra à différents types d'analyseurs d'en tenir compte.

3.1.9 Propagation des traits

Les approches basées sur les contraintes, telles que GPSG et HPSG, utilisent la notion de traits. Lors du processus de construction (ou de projection), des réponses différentes sont données par ces approches à la question de la propagation des traits entre une catégorie constituante et une catégorie projetée.
Dans ces approches et plus particulièrement dans HPSG, des principes gouvernent la propagation des traits. (Principe des traits de tête, principe des traits de pied, etc.). Les Grammaires de Propriétés ne font pas appel à de tels principes. La question est toujours ouverte, quoi que des propositions aient été apportées : la couverture de ces grammaires est assez vaste et la notion de traits n'est pas totalement fermée. On peut imaginer des traits syntaxiques, prosodiques, sémantiques etc. La propagation de ces traits entre un élément de l'énoncé et un élément construit suppose d'une part la connaissance des types de traits possibles pour les catégories de la grammaire et la représentation d'un processus de propagation aussi libre que possible. On doit pouvoir, en rédigeant une grammaire, ne permettre que le passage des traits de tête, ou bien permettre l'apparition de nouveaux traits en fonction de ceux des constituants (immédiats ou non) du syntagme construit.
Comme la théorie des Grammaires de Propriétés ne fournit pas de réponse explicite à cette question, nous nous donnons deux possibilités aussi libres que possible : la propagation non conditionnelle et la propagation conditionnelle. Comme pour la construction, il s'agit encore de mécanismes motivés par des perspectives computationnelles.

3.1.10 Propagation non conditionnelle

Lorsqu'un syntagme est construit, il "hérite" de tous les traits de ses constituants. Ce fonctionnement par défaut ne peut être mis à mal que lorsqu'on accepte des constructions dont les constituants ont des traits en contradiction (par exemple, le syntagme nominal *le chaise* est en contradiction sur le trait de genre). L'analyseur doit assumer cette éventuelle contradiction puisque les Grammaires de propriétés sont tolérantes à la violation de contraintes.

3.1.11 Propagation conditionnelle

Lorsqu'un syntagme est construit, de nouveaux types de traits peuvent lui être affectés en fonction des traits de ses constituants. On parle alors de propagation conditionnelle, car l'expression de cette propagation est une implication de la forme :

Si *(expression de vérité sur les valeurs de traits des constituants)* ***Alors*** *(affectation de valeurs de traits au syntagme construit)*

Cette possibilité est importante et est actuellement envisagée pour le développement de grammaires sémantiques.

3.2 Formalisation des Grammaires de Propriétés

A ce stade de la présentation des Grammaires de Propriétés, nous pouvons relever suffisamment de caractéristiques dans le fonctionnement des propriétés pour élaborer une formalisation rigoureuse en utilisant une notation mathématique.

3.2.1 Nécessité et écueils d'un langage mathématique englobant

Les propriétés ne sont pas hiérarchisées, mais sont organisées comme des ensembles de contraintes ayant le même fonctionnement, même si le sens associé à chaque type de propriété diffère. Un ensemble de caractéristiques et d'attributs minimal permettant d'exprimer le processus de satisfaction de tous les types de propriétés peut être défini. La notation mathématique, faisant appel à la logique des propositions, ainsi qu'aux fonctions sur les propositions nous permettra de préciser le contexte de satisfaction pour les propriétés. En observant le fonctionnement de la précédence linéaire, nous nous trouvons face à la nécessité de comparer les positions des éléments de l'énoncé. Nous allons donc aussi utiliser les fonctions sur les entiers pour formaliser les GPs.
Les possibilités offertes par une telle formalisation dépasseront la description des six propriétés habituelles et permettront d'en introduire de nouvelles, telles que la contiguité, l'interdiction etc. L'idée fondamentale des GPs est que toute contrainte portant sur les constituants d'une catégorie peut faire partie de l'ensemble des propriétés. En utilisant le formalisme que nous allons définir, nous allons nous doter d'un outil capable de décrire les propriétés habituelles plus un certain nombre d'autres qui peuvent se révéler intéressantes, mais nous nous fermerons à d'autres possibilités. Le problème qui se pose est le même que celui d'une programmation dédiée à la résolution d'un problème. En extrapolant suffisamment, nous pouvons programmer un outil capable de résoudre des problèmes de la même classe, mais pas forcément tous les problèmes de la même classe. En d'autres mots, la liste des caractéristiques logico-mathématiques permettant de décrire des Grammaires de Propriétés n'est pas complète. Nous ne prétendons pas faire le tour de toutes les contraintes pouvant porter sur un énoncé, mais de permettre l'expression d'un ensemble assez vaste d'entre elles. Plusieurs questions se posent autour des choix de formalisation que nous allons faire ici. Tout d'abord, existe-t-il une justification ou une motivation cognitive aux choix d'un type de propriété plutôt qu'un autre ? Nous ne pourrons pas répondre à cette question. Des expériences sont actuellement menées à ce propos pour

décider de la qualité et de l'importance relative de certaines propriétés telles que la dépendance et la précédence linéaire. Deuxièmement, est-il possible de réaliser une liste exhaustive des contraintes que nous pourrons définir à l'aide de notre formalisation ? Là encore, après avoir défini notre approche, il apparaîtra à la fois impossible et inutile de dresser une telle liste. Tout au plus, nous pourrons mieux voir quelles sortes de contraintes ne peuvent pas être définies à l'aide de notre modèle. Par exemple, avec les limites que nous allons poser, il sera impossible de créer des propriétés de constituance indirecte (vérifier si un groupe nominal contient un groupe contenant un adjectif, par exemple). Ces limites apparaîtront clairement lorsque notre modèle sera défini.

3.2.2 Un objectif computationnel

Un autre point important qui gouverne le processus de formalisation que nous entreprenons est la visée computationnelle. En effet, nous souhaitons implanter un analyseur syntaxique basé sur les Grammaires de Propriétés, dans lequel la spécification sémantique des propriétés est séparée de la liste des contraintes formant la grammaire, et encore dans lequel la grammaire est séparée totalement du programme qui effectuera l'analyse. Nous verrons plus loin comment atteindre cet objectif. Il faut pour l'instant garder à l'esprit que l'objectif computationnel va gouverner l'introduction d'un certain nombre de notions qui permettront de séparer la grammaire du programme. Les choix que nous effectuons dans notre travail auront un effet réducteur sur les possibilités théoriques des Grammaires de Propriétés. Nous chercherons à minimiser cet effet en rendant aussi général que possible le modèle que nous définissons.

3.2.3 Plan de la formalisation

Les propriétés sont exprimées en supposant un énoncé analysé et en faisant référence aux catégories de la grammaire. Ces catégories sont à leur tour définies comme des listes de propriétés et des listes de traits. Les propriétés qui pourront apparaître dans ces listes doivent être décrites et explicitées de manière cohérente. Nous allons aborder la formalisation de ces trois niveaux de représentation (énoncé, grammaire et spécification sémantique de la grammaire). Pour chacun de ces niveaux, il nous a paru utile de proposer aussi bien une représentation formelle et mathématisée (en cherchant ainsi à donner une modélisation aussi fidèle que possible du formalisme des Grammaires de Propriétés), qu'une représentation BNF et DTD (visant ainsi une intégration du modèle formel aux outils que nous développerons ensuite). Ces niveaux de représentation assez distincts cohabiteront tout au long de notre formalisation pour permettre une transition vers les développements techniques aussi fidèle que possible aux Grammaires de Propriétés. De cette façon, les GPs et même la spécification sémantique des GPs pourront être représentées sous forme de fichiers XML indépendants des outils qui le utiliseront. La visée computationnelle et la fidélité au formalisme sont deux contraintes antagonistes qui doivent se rencontrer dès le niveau de la modélisation.

Le plan de notre formalisation propose tout d'abord de définir l'énoncé à analyser. Celui-ci sera modélisé à l'aide d'une notation mathématique ensembliste, puis sous forme d'une DTD. L'énoncé que nous allons définir est en réalité bien plus qu'un simple texte brut : il s'agit plutôt d'une représentation étiquetée d'un texte brut. La DTD correspondante sera donc à terme celle employée pour l'étiquetage morphosyntaxique.

Nous abordons ensuite la formalisation des Grammaires de Propriétés en suivant la même procédure : définitions mathématiques, représentation BNF de ces définitions, puis DTD. La

DTD des grammaires servira à produire des fichiers XML contenant des grammaires correctement écrites.

Enfin, nous donnons selon le même procédé la formalisation de la spécification sémantique des Grammaires de Propriétés. La DTD ainsi obtenue servira à construire des fichiers XML décrivant la sémantique des grammaires.

3.2.4 Formalisation de l'énoncé analysé

Un *énoncé*, comme l'illustre la figure 2, est une suite d'informations atomiques (des *tokens*), où chaque token est associé à une liste d'étiquettes de catégories, avec pour chacune d'elles un ensemble de valeurs de traits. Nous appellerons *item* l'ensemble composé d'un token et de sa liste d'étiquettes.
Un *item* est un vecteur d'informations hétérogènes supposé décrire l'information contenue dans l'énoncé : non seulement l'information elle même (le token pouvant être un mot, le phonème ou toute autre information analysable), mais aussi un ensemble d'informations linguistiques (les étiquettes).

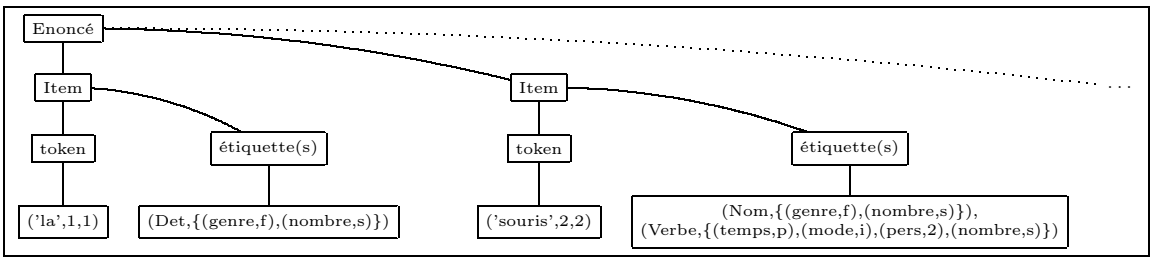


FIG. 2 – Un exemple d'énoncé

Cette formalisation permet de représenter un texte étiqueté de façon rigoureuse, en tolérant un nombre variable d'étiquettes pour chaque élément. La notion de phrase étant pour l'instant laissée de côté, nous considérons l'ensemble d'un texte comme un seul énoncé.

Definition 1 (Un énoncé).

$\mathcal{E}$	$= \{\mathcal{I}_i \mid i \in [1, Card(\mathcal{E})]\}$	*Ensemble d' Items*

L'extrait de DTD fourni par la figure 3 explicite le fait qu'un texte étiqueté (un *énoncé*) est une succession d'*items*. Des informations relatives au dictionnaire de référence, à la version du désambiguiseur et de l'étiqueteur sont fournies avec la première balise. Le nom du fichier contenant l'énoncé est stocké avec la balise ENONCE.

```
<!ELEMENT TEXTE_ETIQUETE (ENONCE)>
<!ATTLIST TEXTE_ETIQUETE
    Dictionnaire  CDATA #REQUIRED
    Desambiguiseur CDATA #REQUIRED
    Etiqueteur CDATA #REQUIRED>
<!ELEMENT ENONCE (ITEM*)>
<!ATTLIST ENONCE
    Fichier CDATA #REQUIRED>
```

FIG. 3 – Extrait de DTD donnant la spécification des énoncés étiquetés

Definition 2 (Un item).

$\mathcal{I}$	$= (\mathrm{t},$	*(La portion d'informations analysée (un token),*
	$\{\varepsilon_{\mathrm{t},i} \mid i \in \mathbb{N}\})$	*l'ensemble d' étiquettes associées à* t*)*

Definition 3 (Un token).

t =	(m, r_{deb} *et* r_{fin})	*(L'information elle même (un mot ...),* *le rang (position dans l'énoncé) du token,* *noté avec deux valeurs entières (*r_{deb} *et* $r_{fin} \in [0..\infty[$*)*

L'extrait de DTD fourni par la figure 4 explicite le fait qu'un *item* est un *token* associé à plusieurs étiquettes. Un token est juste représenté par un ensemble d'attributs (type, token, numéro d'énoncé, numéro du token). Remarquons dès à présent que la représentation mathématisée des énoncés, telle que nous la décrivons ici, est plus forte que celle contenue dans les extraits de DTD fournis. En effet, la DTD qui nous sert d'illustration n'est que celle permettant de décrire un texte étiqueté avant toute analyse syntaxique. La formulation mathématique que nous tentons de déployer a une visée plus large puisque l'*énoncé* qu'elle décrit sera aussi ce qui évoluera au cours de l'analyse syntaxique. Ainsi, un *token* dans un texte étiqueté ne nécessite pas d'information sur ses indices de début et de fin, mais seulement un numéro de token (numForme) alors que dans la perspective d'une analyse syntaxique, la notion de frontières gauches et droites de syntagmes imposera d'indexer aussi les tokens avec un rang de début et de fin. Notons aussi que la DTD fournit plus d'informations que ce que nous avons représenté mathématiquement (type du token par exemple) car le repérage des ponctuations et des nombres est intéressant dès l'étape d'étiquetage, avant toute analyse.

```
<!ELEMENT ITEM (ETIQUETTE*)>
<!ATTLIST ITEM
    typeToken CDATA #REQUIRED (numérique, ponctuation, mot ou autre)
    token CDATA #REQUIRED
    numForme CDATA #IMPLIED
    sur CDATA #IMPLIED>
```

FIG. 4 – Extrait de DTD donnant la spécification des items

Definition 4 (Une étiquette).

$\varepsilon_{\mathbf{t}}$ =	$(\gamma,$ $\{(\nu_{\theta,i}, \upsilon_{\theta,i}) \mid \theta \in \Theta_\gamma, i \in [0, Card(\Theta_\gamma)]\})$	*(une catégorie appartenant à la grammaire* Γ*,* *un ensemble de couples* *(nom de type de trait,valeur de trait)* *associées à* γ*)*

Les étiquettes données aux tokens font référence aux catégories de la grammaire et à leurs traits possibles. Nous travaillerons dans un cadre où ces étiquettes sont supposées connues pour chaque token avant l'analyse. Rien ne nous empêchera, cependant, de construire un analyseur qui effectuera le travail de désambiguisation après chargement d'un énoncé non désambiguisé. L'exemple de DTD présenté à la figure 5 utilise deux attributs : un numéro d'index qui fait référence à une entrée du lexique contenant à la forme, la catégorie et les informations de traits correspondant à l'information décrite dans la définition 4. La catégorie est ajoutée pour rendre les fichiers lisibles en dehors de tout accès au lexique.

```
<!ELEMENT ETIQUETTE EMPTY>
<!ATTLIST ETIQUETTE
    Index CDATA #REQUIRED
    Categorie CDATA #REQUIRED>
```

FIG. 5 – Extrait de DTD donnant la spécification des étiquettes

Comme nous venons de le voir, la représentation mathématisée et la représentation informatique sont assez différentes, mais restent cohérentes. Voyons à présent comment formaliser les Grammaires de Propriétés elles-mêmes.

3.2.5 Formalisation des grammaires

Les grammaires supposent qu'un énoncé sera analysé. Il doit y avoir une correspondance exacte entre l'ensemble catégories et des valeurs de traits employé pour décrire l'énoncé et l'ensemble des catégories et des valeurs de traits employés dans la grammaire. La vérification de cette correspondance appartient au concepteur des outils et ne peut pas être résolue au sein de la présente formalisation.

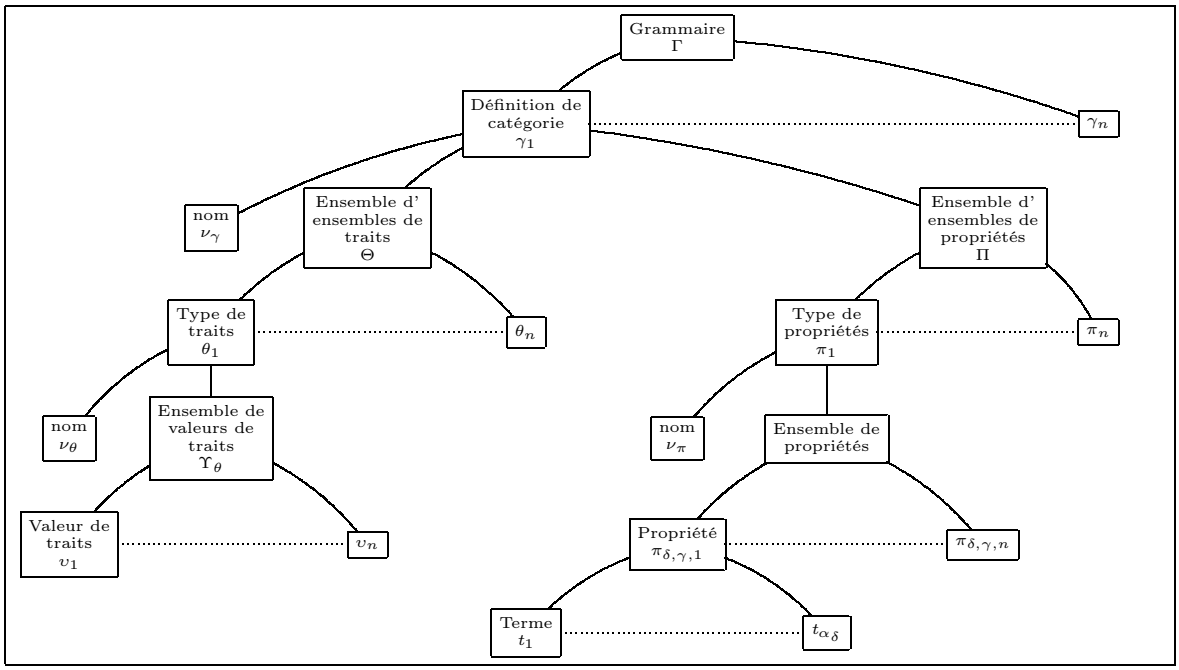


FIG. 6 – Modèle des Grammaires de Propriétés

La figure 6 indique la structure générale d'une grammaire de propriétés. Nous décrivons précisément chacun des éléments de cette structure ci-dessous.

Definition 5 (Une grammaire).

$\Gamma \quad = \quad \{\gamma_i \| i \in [1; Card(\Gamma)]\}$	*(Ensemble des définitions de catégorie $Card(\Gamma)$ est le nombre de catégories.)*

La grammaire est un ensemble de définitions de catégories.

Dans la version DTD qui est donnée en exemple (figure 7), la spécification de l'élément *grammaire* s'accompagne de plusieurs informations utiles : le nom de la grammaire, des commentaires et la référence au type de traits employés (ceci afin de préserver la correspondance entre l'étiquetage de l'énoncé et le jeu de traits de la grammaire.

forme BNF	`grammaire ::= {categorie}`

forme DTD	`<!ELEMENT grammaire (categorie)*>`
	`<!ATTLIST grammaire`
	`label CDATA #REQUIRED`
	`comment CDATA #IMPLIED`
	`type (LPLTRAITS | MLG) "LPLTRAITS">`

FIG. 7 – Grammaire BNF et extrait de DTD donnant la spécification des Grammaires de Propriétés

Definition 6 (Une définition de catégorie).

$$\gamma = (\nu_\gamma, \Pi_\gamma, \Theta_\gamma)$$

(Le nom de la catégorie, Ensemble d'ensembles de propriétés, Ensemble d'ensembles de traits)

Une catégorie γ, définie par le triplet $(\nu_\gamma, \Pi_\gamma, \Phi_\gamma)$ peut être lexicale (comme le nom, le verbe, etc.) aussi bien que syntagmatique (SN, SV, etc.).
L'ensemble Π_γ est constitué d'ensembles de propriétés regroupées par type.
L'ensemble Θ_γ est constitué d'ensembles de traits regroupés par type.

forme BNF	`categorie ::= nomCategorie, traits, proprietes`

forme DTD	`<!ELEMENT categorie (traits, proprietes)>`
	`<!ATTLIST categorie`
	`label ID #REQUIRED`
	`type (CONTINUE | DISCONTINUE) "CONTINUE"`
	`comment CDATA #IMPLIED>`

FIG. 8 – Grammaire BNF et extrait de DTD donnant la spécification des catégories

La figure 8 indique les correspondances sous forme de spécifiacation Backus Naur Form et de DTD pour la notion de catégorie. On remarquera que la version DTD s'est enrichie afin de permettre le repérage de catégories ayant la possibilité d'être discontinues. Ce choix sera explicité plus loin et permet de représenter avec la même grammaire aussi bien des catégories au sens habituel, que des dépendances.

Definition 7 (Un ensemble d'ensembles de traits). $\Theta_\gamma = \{\theta_i | i \in \mathbb{N}\}$

Un ensemble d'ensembles de traits pour une catégorie γ donnée est un ensemble de types de traits θ_i.

forme BNF	`traits ::= \{typeTrait\}`

forme DTD	`<!ELEMENT traits (trait|propagation)*>`
	`<!ATTLIST traits`
	`label CDATA #REQUIRED`
	`comment CDATA #IMPLIED>`

FIG. 9 – Grammaire BNF et DTD pour les ensembles d'ensembles de traits

La figure 9 donne les versions BNF et DTD pour la notion d'ensembles d'ensembles de traits. La version DTD est à nouveau plus riche que ce que nous avons défini, ceci afin de mettre en place un mécanisme de propagation des traits. Ce point sera discuté plus loin.

Definition 8 (Un type de traits). $\theta = (\nu_\theta, \Upsilon_\theta)$

Un type de traits θ est un couple (nom du type de traits, ensemble de valeurs de traits).

forme BNF	`typeTrait ::= nomTypeTrait, {valeurTrait}`

forme DTD	`<!ELEMENT trait (valeur*)>`
	`<!ATTLIST trait`
	`  nomTypeTrait  CDATA #REQUIRED`
	`  comment  CDATA #IMPLIED`
	`  type  (CARACTERE | ENTIER) "CARACTERE">`

FIG. 10 – Grammaire BNF et DTD pour les ensembles de traits

La figure 10 n'apporte pas de modifications à la formulation mathématique des types de traits. Simpelment, la version DTD indique pour les besoins informatiques quel type (caractère, entier, etc.) sera employé pour décrire les valeurs de traits.

Definition 9 (Un ensemble de valeurs de traits). $\Upsilon_\theta = \{\upsilon_{\theta,i} | i \in \mathbb{N}\}$

Les valeurs de traits $\upsilon_{\theta,i}$ peuvent être des entiers, des caractères, des mots etc.

forme BNF	`valeurTrait ::= valeur`

forme DTD	`<!ELEMENT valeur EMPTY>`
	`<!ATTLIST valeur`
	`  label CDATA #REQUIRED`
	`  comment CDATA #IMPLIED>`

FIG. 11 – Grammaire BNF et DTD pour les valeurs de traits

La figure 11 n'apporte aucune modification à la formulation mathématique des valeurs de traits. Le type de ces valeurs est sélectionné au niveau de l'ensemble de traits qui les contient.

Definition 10 (Un ensemble d'ensembles de propriétés).

$$\Pi_\gamma = \{(\nu_\delta, \{\pi_{\delta,\gamma,k} | k \in \mathbb{N}\}) | \delta \in \Delta\}$$

L'ensemble Π_γ est constitué de couples (nom de type de propriétés, ensemble de propriétés de ce type). Les noms de type de propriétés ν_δ utilisés dans ces couples font référence aux définitions de types de propriétés δ appartenant à la sémantique.
Une propriété donnée $\pi_{\delta,\gamma}$, pour un type de propriétés δ dans la définition d'une catégorie γ, peut être définie elle-même comme un ensemble ordonné Λ de termes λ, dont la cardinalité est égale à l'arité du type de propriété δ défini dans la sémantique Δ :

forme BNF	`proprietes ::= {typePropriete}` `typePropriete ::= nomTypePropriete , {propriete}`

forme DTD	`<!ELEMENT proprietes (typePropriete)*>`
	`<!ATTLIST proprietes`
	`  label CDATA #REQUIRED`
	`  comment CDATA #IMPLIED>`
	`<!ELEMENT typePropriete (propriete)*>`
	`<!ATTLIST typePropriete`
	`  label CDATA #REQUIRED`
	`  comment CDATA #IMPLIED>`

FIG. 12 – Grammaire BNF et DTD pour les ensembles d'ensembles de propriétés

Definition 11 (Une propriété).

$\pi_{\delta,\gamma} = \Lambda = \{\lambda_l \| l \in [1; \alpha_\delta]\}$	*Un ensemble ordonné de α_δ termes α_δ étant l'arité du type de propriété δ)*

Un terme λ est une référence ρ à une catégorie $\gamma\prime$ de la grammaire ou bien une expression logique o portant sur des termes :

forme BNF	`propriete ::= {terme}`

forme DTD	`<!ELEMENT propriete (terme)*>`
	`<!ATTLIST propriete`
	`comment CDATA #IMPLIED`
	`weight CDATA "1">`

FIG. 13 – Grammaire BNF et DTD pour les propriétés

Definition 12 (Un terme). $\lambda \quad = \quad \rho_{\gamma\prime}|\gamma\prime \in \Gamma \; ou \; (o, \{\lambda\prime_i|i \in [1; Arite(o)]\})$

L'opération logique *o* peut être un OU, un ET ou un NON, portant sur des termes. Toute combinaison d'opérations logiques peut être écrite. Le sens que nous pouvons donner à une propriété s'interprète en fonction de la disponibilité des références aux catégories et des termes. Nous préciserons plus tard comment évaluer cette disponibilité. Nous pouvons cependant préciser que l'évaluation d'une opération logique OU ou ET suppose la présence des opérandes dans l'énoncé analysé. La forme négative n'est quant à elle évaluable positivement qu'en l'absence de l'opérande. Ceci implique que pour évaluer totalement des contraintes exprimées par la négative, il faut avoir choisi à l'avance dans quelle partie de l'énoncé sera évaluée cette négation. Ceci pose d'énormes problèmes computationels. Concrètement, puisqu'il est possible avec notre formalisme d'écrire des propriétés telles que l'interdiction, ce qui déplace le problème de l'évaluabilité d'une négation à une position plus facilement calculable, nous avons renoncé à employer l'opérateur de négation au niveau des termes. Ainsi, la version DTD ci-dessous ne correspond pas exactement avec la forme BNF de l'opération logique.

forme BNF	`terme ::= {refCateg | operationLogique}`
	`operationLogique ::=`
	`([’NON’] (refCateg | operationLogique)) |`
	`(((refCateg | operationLogique)) (’OU’|’ET’) (refCateg | operationLogique)))`

forme DTD	`<!ELEMENT membre (refCateg | operationLogique)*>`
	`<!ATTLIST membre`
	`comment CDATA #IMPLIED>`
	`<!ELEMENT operationLogique (refCateg | operationLogique)+>`
	`<!ATTLIST operationLogique`
	`label (OU|ET) "OU"`
	`comment CDATA #IMPLIED>`

FIG. 14 – Grammaire BNF et DTD pour les termes de propriétés

Definition 13 (Une référence).

$\rho_{\gamma\prime}$	$= \quad (\nu_{\gamma\prime}\|\gamma\prime \in \Gamma,$ $\zeta_{\nu_{\gamma\prime}})$	*(un nom de catégorie,* *ensemble de spécifications de traits)*

Une référence à une catégorie est le couple (nom de la catégorie, ensemble de spécifications de traits sur cette catégorie).
Un ensemble de spécifications de traits est un ensemble de spécifications atomiques de traits. Une spécification atomique de traits est soit un triplet (nom de type de trait, opérateur équatif, valeur de trait), soit le singleton (nom de type de traits). L'interprétation de cette spécification dépend de la présence ou non de la précision d'une valeur.
Lorsque la valeur n'est pas précisée, cela signifie que la référence doit correspondre avec les autres références de la même propriété pour le type de traits donné dans la spécification. Si une valeur est précisée, cela signifie que la référence devra correspondre avec les traits de l'élément de l'énoncé. Nous verrons plus loin comment fonctionnent ces deux interprétations.

Definition 14 (Un ensemble de spécifications de traits).

$$\zeta_{\nu_{\gamma\prime}} \quad = \quad \{(\nu_{\theta,i}[, e, \upsilon_\theta])|\theta \in \Theta_{\gamma\prime}, i \in \mathbb{N}, e \in \{'=', '\neq'\}, \upsilon_\theta \in \Upsilon\theta\}$$

forme BNF

```
refCateg ::=
    nom,
    {refValeur | refTrait}
refValeur ::=
        nomCategorie,
        operateur,
        valeur
refTrait ::=
        nomTypeTrait
```

forme DTD

```
<!ELEMENT refCateg (refValeur | refTrait)*>
<!ATTLIST refCateg
    label IDREF #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT refValeur EMPTY>
<!ATTLIST refValeur
    label CDATA #REQUIRED
    operateur  CDATA #REQUIRED
    valeur CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT refTrait EMPTY>
<!ATTLIST refTrait
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
```

FIG. 15 – Grammaire BNF et DTD pour les références avec ou sans restriction de traits

L'énoncé et la grammaire sont à présent définis. Avant d'illustrer ce formalisme, nous devons encore considérer la spécification sémantique de la grammaire, qui permet de faire fonctionner l'ensemble de façon cohérente en cours d'analyse.

3.2.6 Formalisation de la sémantique des grammaires

Les contraintes qui permettent d'effectuer des analyses dans le formalisme des Grammaires de Propriétés, sont appelées *propriétés*. Derrière cette appellation, il faut comprendre que les contraintes ne sont pas du même genre que celles qu'on emploie habituellement en parlant de satisfaction de contraintes. Dans ce dernier cas, en effet, il s'agit de calculer une interprétation sur une formule exprimée à l'aide de littéraux pouvant être vrais ou faux. La satisfaisabilité est binaire et calculée sur des termes binaires. Dans le cas des Grammaires de Propriétés, le calcul de satisfaisabilité suppose non plus des littéraux, mais des termes complexes issus de l'énoncé à analyser. Les contraintes à satisfaire sont d'un ordre bien plus élevé et nécessitent un outillage logico-mathématique conséquent. Pour commencer, il faut remarquer que le formalisme des Grammaires de Propriétés propose un ensemble assez petit de types de propriétés. L'obligation, la facultativité, l'unicité, la linéarité, la dépendance, l'exigence et l'exclusion sont les principales. Il s'avère cependant qu'on peut introduire de nouveaux types si nous nous dotons de suffisamment de moyens pour calculer la satisfaction de nouvelles sortes de contraintes. Nous ferons souvent l'amalgame entre le terme *type de propriété* et le mot *propriété* : lorsque nous parlerons par exemple de la *propriété de Linéarité*, nous voudrons désigner le *type de propriété* appelé linéarité. Cet amalgame n'est pas très contraignant, mais il faudra en être averti, car nous aurons aussi à appeler *propriétés* les contraintes écrites dans la grammaire et portant sur des termes présents dans l'énoncé. On pourra ainsi dire *la propriété* de linéarité portant sur le déterminant et le nom dans le groupe nominal. Ou encore, nous pourrons dire *une propriété* de linéarité, pour désigner dans ce cas une contrainte quelconque, dont le type de propriété est *la propriété* de linéarité. Définir le comportement de chaque type de propriété est le rôle de la *spécification sémantique des Grammaires de Propriétés*. Nous devons repérer autant que possible chaque phénomène particulier agissant dans le processus de calcul de satisfaction des propriétés. Comme nous n'avons pas complètement présenté le processus d'analyse, il est trop tôt pour décrire la démarche qui a conduit à décider quels phénomènes discriminants seraient conservés pour définir la base de notre spécification sémantique. Nous allons présenter ici les éléments que nous avons choisi. Le choix sera éclairé plus tard au regard d'exemples précis d'analyse.

La figure 16 montre une vue d'ensemble du modèle de représentation des graphes de spécification sémantique. Chaque type de propriétés y est défini par un ensemble d'éléments que nous définissons ci-après.

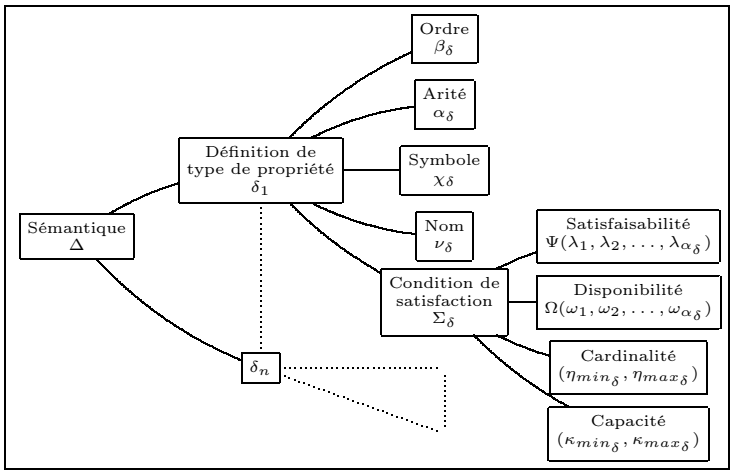


FIG. 16 – Modèle de la sémantique des Grammaires de Propriétés

Une sémantique de la grammaire est simplement un ensemble de définitions de types de propriétés.

Definition 15 (Une sémantique de la grammaire).

Δ	$=$	$\{\delta_i \mid i \in \mathbb{N}\}$	*(Ensemble des Définitions de types de propriétés)*

La sémantique exprime comment fonctionnent les propriétés, type par type. Les représentations BNF et DTD ci-dessous se basent sur ce modèle et permettent à terme de représenter en XML des sémantiques pour les Grammaires de Propriétés.

forme BNF	`Semantique ::= {definitionPropriete}`

forme DTD	`<!ELEMENT semantique (definitionPropriete)*>`
	`<!ATTLIST semantique`
	`label CDATA #REQUIRED`
	`comment CDATA #IMPLIED>`

FIG. 17 – Grammaire BNF et DTD pour la spécification sémantique des Grammaires de Propriétés

Definition 16 (Une définition de type de propriété).

δ	$=$	$(\nu_\delta,$	*(Nom du type de propriétés,*
		$\chi_\delta,$	*son symbole,*
		$\alpha_\delta,$	*son arité,*
		$\Sigma_\delta)$	*ses conditions de satisfaction).*

La définition d'un type de propriétés est simplement le nom, le symbole, l'arité et la liste des conditions de satisfaction propres à ce type de propriétés. Nous définissons ci-dessous ces conditions en ayant à l'esprit que les catégories seront définies plus tard en fonction de ces types de propriétés. En devançant les explications qui seront données plus loin, nous devons préciser deux niveaux de satisfaisabilité :

- Au niveau d'une propriété : au moment du processus d'analyse, nous dirons qu'une propriété est satisfaite si elle satisfait sa fonction de disponibilité et sa fonction de satisfaisabilité.
- Au niveau du type de propriété : nous dirons que, pour une catégorie donnée, dans la mesure où un certain nombre des propriétés qui la définissent sont disponibles et satisfaites, l'ensemble des propriétés du même type de cette catégorie peut à son tour être considéré comme satisfait ou non, en fonction du nombre des propriétés de ce groupe (on fixe pour chaque type de propriétés une cardinalité minimale et maximale acceptables pour qu'un groupe de propriétés du même type soit satisfait), et en fonction du nombre de fois où la même propriété du même type est disponible (on fixe pour chaque type de propriétés une capacité minimale et maximale acceptables pour qu'un groupe de propriétés du même type soit satisfait).

Le premier niveau porte sur la satisfaction des contraintes, indépendamment les unes des autres. Le second permet le regroupement des propriétés en définissant des conditions rendant possible ou interdisant ce regroupement.
Pour bien saisir l'importance de ces deux niveaux dans notre approche, on pourra se référer aux exemples donnés à la fin de ce chapitre.

Definition 17 (Conditions de satisfaction d'une propriété).

Σ_δ	=	$(\kappa_{min_\delta},$	*(Capacité minimale,*
		$\kappa_{max_\delta},$	*Capacité maximale,*
		$\eta_{min_\delta},$	*Cardinalité minimale,*
		$\eta_{max_\delta},$	*Cardinalité maximale,*
		$\Omega(\omega_1, \omega_2, \ldots, \omega_{\alpha_\delta})$	*Disponiblilité,*
		$\Psi(\lambda_1, \lambda_2, \ldots, \lambda_{\alpha_\delta}))$	*Satisfaisabilité)*

Dans cette dernière définition, les fonctions Ω et Ψ sont des fonctions de vérité : elles travaillent respectivement sur les disponibilités et sur les valeurs des éléments de l'énoncé. Ces fonctions doivent être explicitées pour chaque type de propriété. La première exprime le fait qu'une propriété peut être ou non évaluée en fonction des disponibilités des éléments de l'énoncé. La seconde est calculée si la première est satisfaite : elle dira si la propriété est satisfaite, cette fois en fonction des valeurs des éléments de l'énoncé. L'explication de cette formalisation sera donnée plus loin, car elle nécessite de préciser comment une grammaire est rédigée, ce qu'est un énoncé et enfin comment fonctionne le processus d'analyse.

Les autres conditions de statisfaisabilité sont plus simples :

- La capacité indique le nombre de fois qu'une même propriété peut être satisfaite pour une même construction. Par exemple, l'unicité est un type de propriété qui se définit avec une capacité maximale de 1. En effet, dans une même construction Groupe Nominal, si deux noms se trouvent présents et que la grammaire spécifie une contrainte d'unicité sur le nom dans le groupe nominal, alors la contrainte sera satisfaite deux fois, et sa capacité sera enfreinte. Ainsi, on pourra considérer que l'unicité est enfreinte dans cette contruction.
- La cardinalité indique le nombre de propriétés du même type pouvant être satisfaites pour une construction donnée. Par exemple, l'obligation se définit usuellement avec une cardinalité minimale et maximale de 1 (on estime alors qu'une et une seule des

propriétés d'obligation peut être satisfaite pour la même construction). Par exemple, si nous écrivons une grammaire du groupe nominal dans laquelle se trouvent une contrainte d'obligation portant sur un nom et une autre portant sur un adjectif qualificatif, et que dans l'énoncé analysé se présentent à la fois un nom et un adjectif qualificatif, nous aurons deux contraintes d'obligation satisfaites pour un même groupe nominal. Dans ce cas, la condition de cardinalité sera enfreinte et chacune des deux propriétés sera considérée comme violée.

L'ensemble des réflexions portant sur les propriétés, leur usage, leurs limites et leurs intérêts est étudié à la fin de cette première partie. Il faut en effet constater que toutes les propriétés ne fonctionnent pas de la même façon et que certaines d'entre elles imposent un calcul *après construction*, tandis que d'autres peuvent être calculées *avant construction*. Les définitions données ici seront aussi éclairées à la lecture du prochain chapitre portant sur la formalisation de la caractérisation.

forme BNF

```
definitionPropriete ::=
        nom,
        symbole,
        arite,
        capaciteMin,
        capaciteMax,
        cardinaliteMin,
        cardinaliteMax,
        ordonnee,
        ConditionSatisfaisabilite,
        ConditionDisponibilite.
```

forme DTD

```
<!ELEMENT definitionPropriete (conditionSatisfaisabilite, conditionDisponibilite)>
<!ATTLIST definitionPropriete
    nom CDATA #REQUIRED
    symbole CDATA #REQUIRED
    comment CDATA #IMPLIED
    arite CDATA #REQUIRED
    capacite CDATA #REQUIRED
    cardinalite CDATA #REQUIRED
    ordonnee (VRAI | FAUX) "FAUX">
```

Fig. 18 – Grammaire BNF et DTD pour la définition des types de propriétés

La version BNF et DTD de ce modèle (figure 18) est consistante avec lui. Il faut par contre ajouter dans ces représentations tout ce que nous n'avons pas représenté mathématiquement : quelles opérations sont possibles sur quels objets, comment écrire correctement une expression logique, mathématique etc. Les figures 19 et 20 donnent des grammaires typiques de calcul sur des expressions arithmétiques et logiques. Les littéraux désignent ici des éléments de l'énoncé analysé, quelle que soit leur catégorie, puisque la sémantique n'est pas directement reliée à l'énoncé, contrairement à la grammaire. Pour désigner ces éléments de l'énoncé, la sémantique désigne en fait les indices des termes des propriétés de la grammaire. Par exemple, dans une propriété de linéarité portant sur un déterminant et un nom (le déterminant précède le nom), le déterminant est le premier terme, tandis que le nom sera le second. Il est possible de définir une contrainte disant que le rang de début du second terme doit être plus grand que le rang de fin du premier terme. Ceci, grâce à des opérations simples, des comparaisons entre valeurs entières etc.

3.2.7 Formalisation de la caractérisation

Nous n'allons pas décrire ici l'algorithme d'analyse syntaxique qui sera employé dans la prochaine partie, mais nous allons poser quelques définitions qui permettront cette analyse.
Lorsque l'énoncé est analysé, pour chaque *token*, un ensemble de propriétés de la grammaire est susceptible d'etre satisfait ou non-satisfait. D'autres propriétés par contre ne seront pas

```
ConditionSatisfaisabilite ::= expressionBooleenne |
                              expressionLogique
ConditionDisponibilite ::= expressionBooleenne |
                           expressionLogique
expressionLogique ::= facteurLogique |
                      termeLogique |
                      elementLogique
facteurLogique ::= (facteurLogique |
                    termeLogique |
                    elementLogique |
                    expressionBooleenne ),
                    'ET',
                    (facteurLogique |
                     termeLogique |
                     elementLogique |
                     expressionBooleenne)
termeLogique ::= (facteurLogique |
                  termeLogique |
                  elementLogique |
                  expressionBooleenne),
                  'OU',
                 (facteurLogique |
                  termeLogique |
                  elementLogique |
                  expressionBooleenne)
elementLogique ::= litteral |litteralNegative
litteralNegative ::=  'NON', litteral
litteral ::=  numeroTerme,
              ('DISPONIBLE' | 'SATISFAIT')
expressionBooleenne ::= expressionBooleenneBinaire |
                        expressionBooleenneUnaire |
                        constanteBooleenne
expressionBooleenneBinaire ::= expressionArithmetique ,
        ('INF' | 'SUP' | 'EQ' | 'DIF' | 'INFEQ' | 'SUPEQ'),
        expressionArithmetique
expressionBooleenneUnaire ::= ('FONCTIONNEUTRE' |
                               'VERIFDOMAINES'),
                               expressionArithmetique
constanteBooleenne ::= ('VRAI' | 'FAUX')
expressionArithmetique ::= (facteurArithmetique |
                            termeArithmetique |
                            elementArithmetique)
facteurArithmetique ::= (facteurArithmetique |
                         termeArithmetique |
                         elementArithmetique),
                        ('MUL' | 'DIV'),
                        (facteurArithmetique |
                         termeArithmetique |
                         elementArithmetique)
termeArithmetique ::= (facteurArithmetique |
                       termeArithmetique |
                       elementArithmetique),
                      ('PLUS' | 'MOINS'),
                      (facteurArithmetique |
                       termeArithmetique |
                       elementArithmetique)
elementArithmetique ::= (terminal |
                         terminalMoins |
                         constanteArithmetique)
terminalMoins ::= 'MOINS', terminal
terminal ::= numeroMembre,
          ('RANGDEBUT' | 'RANGFIN' | 'VALEUR' | 'DOMAINE')
constanteArithmetique ::= valeur
```

FIG. 19 – grammaire BNF des expressions logico-mathématiques dans la spécification sémantique des GPs

```
<!ELEMENT expressionLogique
    (facteurLogique|termeLogique|elementLogique)>
<!ELEMENT facteurLogique
    ((facteurLogique|termeLogique|
      elementLogique|expressionBooleenne),
    (facteurLogique|termeLogique|
     elementLogique|expressionBooleenne))>
<!ATTLIST facteurLogique
    operateur (ET) "ET">
<!ELEMENT termeLogique
    ((facteurLogique| termeLogique|
      elementLogique|expressionBooleenne),
    (facteurLogique| termeLogique|
     elementLogique|expressionBooleenne))>
<!ATTLIST termeLogique
    operateur (OU) "OU">
<!ELEMENT elementLogique (litteral| litteralNegative)>
<!ELEMENT litteralNegative (litteral)>
<!ATTLIST litteralNegative
    operateur (NON) "NON">
<!ELEMENT litteral EMPTY>
<!ATTLIST litteral
    numeroMembre CDATA #REQUIRED
    accesseur (DISPONIBLE| SATISFAIT) "DISPONIBLE">
<!ELEMENT expressionBooleenne
    (expressionBooleenneBinaire |
     expressionBooleenneUnaire |
     constanteBooleenne)>
<!ELEMENT expressionBooleenneBinaire
    (expressionArithmetique,expressionArithmetique)>
<!ATTLIST expressionBooleenneBinaire
    operateur (INF | SUP | EQ | DIF |INFEQ | SUPEQ) "EQ">
<!ELEMENT expressionBooleenneUnaire (expressionArithmetique)>
<!ATTLIST expressionBooleenneUnaire
    operateur (FONCTIONNEUTRE | VERIFDOMAINES) "FONCTIONNEUTRE ">
<!ELEMENT constanteBooleenne EMPTY>
<!ATTLIST constanteBooleenne
    valeur (VRAI | FAUX) "VRAI">
<!ELEMENT expressionArithmetique
    (facteurArithmetique|termeArithmetique|elementArithmetique)>
<!ELEMENT facteurArithmetique
    ((facteurArithmetique|termeArithmetique|elementArithmetique),
     (facteurArithmetique|termeArithmetique|elementArithmetique))>
<!ATTLIST facteurArithmetique
    operateur (MUL | DIV) "MUL">
<!ELEMENT termeArithmetique
    ((facteurArithmetique|termeArithmetique|elementArithmetique),
     (facteurArithmetique|termeArithmetique|elementArithmetique))>
<!ATTLIST termeArithmetique
    operateur (PLUS | MOINS) "PLUS">
<!ELEMENT elementArithmetique
   (terminal | terminalMoins | constanteArithmetique)>
<!ELEMENT terminalMoins (terminal)>
<!ATTLIST terminalMoins
    operateur (MOINS) "MOINS">
<!ELEMENT terminal EMPTY>
<!ATTLIST terminal
    numeroMembre CDATA #REQUIRED
    accesseur (RANGDEBUT | RANGFIN | VALEUR | DOMAINE) "VALEUR">
<!ELEMENT constanteArithmetique EMPTY>
<!ATTLIST constanteArithmetique
    valeur CDATA #REQUIRED>
```

FIG. 20 – grammaire DTD des expressions logico-mathématiques dans la spécification sémantique des GPs

concernées par l'énoncé. Nous dirons que lorsqu'un élément de l'énoncé est observé, ses étiquettes sont *disponibles*. La disponibilité d'une étiquette rend disponibles les références à ces étiquettes, qui rendent à leur tour évaluables les propriétés et enfin permettent de réaliser la caractérisation.
La sémantique que nous avons définie repose sur la notion de disponibilité. Nous allons définir ci-dessous comment calculer cette disponibilité à tous les stades de l'analyse. Nous donnerons ensuite les définitions des fonctions de satisfaction.

Definition 18 (Disponibilité d'un item).
Un item $\mathcal{I}$ de l'énoncé est toujours disponible.

Lorsque l'analyse débute, nous considérons que l'ensemble de l'énoncé (ses items) est disponible. Analyser revient alors à constituer les ensembles de propriétés satisfaites et non satisfaites, en commençant par tester si elles sont évaluables. Une propriété est évaluable si ses termes (les références et les expressions portant sur des termes) le sont.

Definition 19 (Disponibilité d'une référence). *Une référence ρ_γ doit être interprétée comme une déclaration de variable : elle a un nom ν_γ et un type γ, qui est une catégorie de la grammaire. Une référence peut éventuellement être sous-définie par un ensemble de traits ζ_γ faisant référence aux traits potentiels de la catégorie γ.*

Une référence ρ_γ est disponible pour un item $\mathcal{I}$ si la catégorie et les traits de cet item correspondent à ceux de la référence.
Cette correspondance de traits doit tenir compte de l'opérateur ($=$ ou $\neq$) associé à chaque spécification de traits de ρ_γ.

Definition 20 (Disponibilité d'une expression logique). *Les expressions logiques employées pour définir les termes des propriétés sont disponibles en fonction de leur opérateur et de la disponibilité de leurs termes :*

Une expression OU *est disponible si un de ses opérandes est disponible.*
Une expression ET *est disponible si ses deux opérandes sont disponibles.*
Une expression NON *est disponible si son opérande n'est pas disponible.*

Definition 21 (Disponibilité d'un terme). *Soient $\lambda_{\delta,1}$, $\lambda_{\delta,2}$, ..., $\lambda_{\delta,\alpha_\delta}$ les termes d'une propriété de type δ (ces termes pouvant être des références ou des expressions logiques), avec α_δ l'arité de ce type de propriétés.*
Un terme est disponible dans les conditions exprimées ci-dessus pour la référence et l'expression logique. Nous définissons alors une fonction de vérité $\omega(\lambda_i)$:

Une fonction de vérité $\omega(\lambda)$, associe à un terme λ associe une valeur parmi $\{Vrai, Faux\}$ de la façon suivante : si λ est disponible alors $\omega(\lambda)$ vaut $Vrai$, et $Faux$ sinon.
Nous notons $\omega_{\lambda,1}$, $\omega_{\lambda,2}$, ..., $\omega_{\lambda,\alpha_\delta}$ les disponibilités respectives des termes $\lambda_{\delta,1}$, $\lambda_{\delta,2}$, ..., $\lambda_{\delta,\alpha_\delta}$.

Grâce à cette définition, il est maintenant possible de préciser la disponibilité d'une propriété.

Definition 22 (Disponibilité d'une propriété). *Pour chaque type de propriété δ, la sémantique doit définir ses conditions de disponibilité :*

Soit π_δ une propriété et $\lambda_{\delta,1}$, $\lambda_{\delta,2}$, ..., $\lambda_{\delta,\alpha_\delta}$ ses termes.
Nous appelons fonction de disponibilité d'une propriété π_δ *la fonction de vérité*
$\Omega_\delta(\omega_{\lambda,1}, \omega_{\lambda,2}, \ldots, \omega_{\lambda,\alpha_\delta})$,
qui associe aux disponibilités ω_{λ_i} des termes λ_i
de la propriété δ une valeur parmi $\{Vrai, Faux\}$:
Cette fonction doit être fournie avec la définition de chaque type de propriété en fonction du comportement attendu pour l'analyse tel que :
$\Omega_\delta(\omega_{\lambda,1}, \omega_{\lambda,2}, \ldots, \omega_{\lambda,\alpha_\delta}) \quad \supset$ (π_δ *est disponible*)

Lorsqu'une propriété est disponible, pour un énoncé donné, sa satisfaisabilité peut être évaluée.

Definition 23 (Satisfaisabilité d'une propriété).

Soit π_δ une propriété et $\lambda_{\delta,1}$, $\lambda_{\delta,2}$, ..., $\lambda_{\delta,\alpha_\delta}$ ses termes.
Nous appelons fonction de satisfaisabilité d'une propriété π_δ *la fonction de vérité*
$\Psi_\delta(\lambda_1, \lambda_2, \ldots, \lambda_{\alpha_\delta})$,
qui associe aux termes λ_i de la propriété δ une valeur parmi $\{Vrai, Faux\}$:
Cette fonction doit être fournie avec la définition de chaque type de propriété en fonction du comportement attendu pour l'analyse tel que :
$\Psi_\delta(\lambda_1, \lambda_2, \ldots, \lambda_{\alpha_\delta}) \quad \supset (\pi_\delta$ *est satisfaite*)
Comme cette fonction Ψ est définie sur les termes disponibles, elle peut être exprimée en fonction des valeurs de traits et du rang des items disponibles, accessibles via ces termes.

Les définitions données dans ce chapitre formalisent les Grammaires de Propriétés et rendent possible la création d'applications modulaires dans lesquelles la sémantique sera séparée de la grammaire, et dans lesquelles le processus d'analyse pourra être indépendant de la grammaire choisie.
Les grammaires ainsi formulées seront modifiables sans que les outils aient à être réécrits. Il faudra pour atteindre cet objectif écrire un modèle informatique capable d'intégrer ce formalisme, et rédiger un algorithme générique adapté à ce modèle.
Avant de présenter ce travail, nous allons aborder la question des modèles de représentation pour la sémantique et la grammaire.

3.3 Représentation des Grammaires de Propriétés

3.3.1 Représentations à but computationnel

Avec les définitions introduites ci-dessus, nous allons définir un formalisme à deux niveaux (sémantique de la grammaire et grammaire), guidé par deux grammaires qui expliciteront la manière d'écrire les données de ces deux niveaux.
La figure 21 montre la nécessité d'un formalisme suffisamment générique pour permettre différentes représentations de la même grammaire tout en garantissant des résultats équivalents à celui qui devrait être attendu.

Le gain attendu, du fait de séparer la spécification sémantique de la grammaire et du programme, est la généricité des écritures. La grammaire est ainsi conçue comme un programme dont l'interprétation dépend d'un schéma de sens isolé. C'est grâce à une telle séparation que nous obtiendrons plus tard un analyseur indépendant des modifications apportées à la grammaire. Nous déterminons tout d'abord une grammaire d'écriture des sémantiques

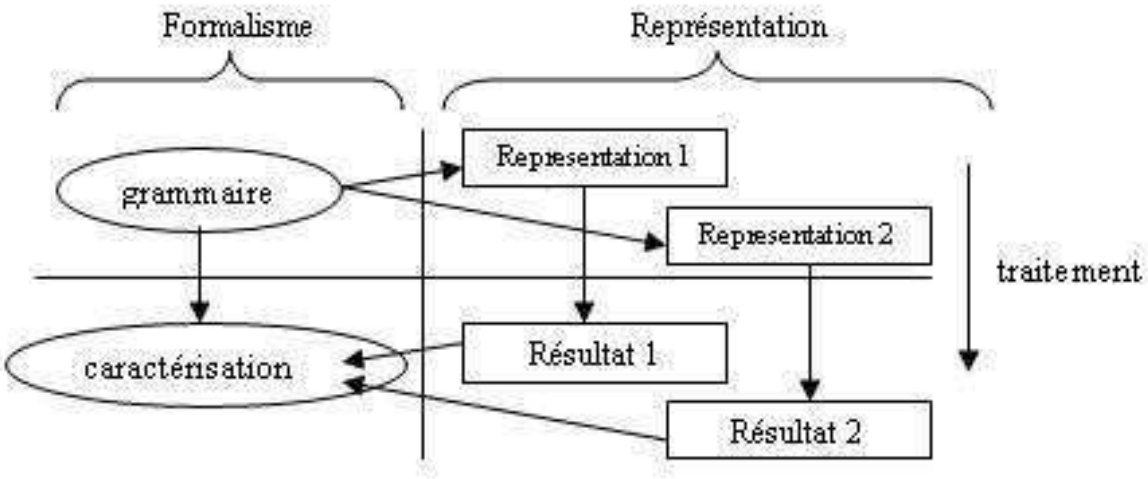


FIG. 21 – Nécessité d'une représentation générique

des Grammaires de Propriétés, sur la base de la formalisation réalisée précédemment, ce qui conduit à un fichier DTD complet. On trouvera en annexe C le contenu de ce fichier DTD ainsi que la grammaire BNF qui lui correspond. La rédaction en XML de fichiers de spécification sémantique des grammaires peut ainsi être contrôlée logiciellement grâce au DTD.

A partir de cette grammaire de la spécification sémantique, nous pouvons représenter en XML les définitions de propriétés qui nous intéressent. Nous allons observer chaque type de propriété parallèlement à sa spécification. A partir de l'étude réalisée précédemment pour définir les critères particuliers nécessaires à une spécification correcte, nous élaborons une description schématique de chaque type de propriété et nous rédigeons la définition XML correspondante. L'ensemble de ce fichier peut être consulté à l'annexe D. L'élément XML *sémantique* englobe la totalité des définitions de propriété exprimées à partir des éléments XML *definitionPropriete*. La définition de la propriété de linéarité, par exemple, est dotée des attributs nom, symbole, arité, capacité et cardinalité. La fonction d'évaluabilité autrement appelée condition de disponibilité est définie comme un sous-élément *conditionDisponibilité*. Ce dernier sous-élément est constitué à son tour d'éléments permettant de définir l'expression logico-mathématique qui spécifie dans quel cadre une propriété est évaluable. De la même façon, le sous-élément *condistionSatisfaisabilité* permet de spécifier dans quel cadre une propriété peut être satisfaite.

3.3.2 Spécification de la linéarité

La linéarité est sans doute la propriété qui se comprend le mieux. Elle fait appel simplement à la notion de rang. Une contrainte de linéarité est évaluable si ses deux termes sont disponibles dans l'énoncé. Elle est satisfaite si le second terme se trouve après le premier.

3.3.3 Spécification de l'obligation

L'obligation, quant à elle, soulève de nombreuses questions. La définition stricte de l'obligation est qu'il s'agit d'une contrainte unaire qui est évaluable et statisfaite si le terme qu'elle désigne est présent et unique dans la construction. Par exemple, dans la grammaire du groupe

Définition de Propriété **linearité** : := [nom= 'linearité', symbole= '«', arité= 2, Ordonnée=Faux, Satisfaisabilité = [$\Omega = \omega_1 \quad \wedge \quad \omega_2$, Ψ = (t1.rangFin < t2.rangDeb), CardinalitéMin = 0, CardinalitéMax = ∞, CapaciteMin = 0, CapaciteMax = ∞].	<definitionPropriete nom="linearite" comment="linéarité: précédence entre items définie par une relation d'ordre entre les rangs des termes d'une clause" symbole="((" arite="2" capacite="[0;infini]" cardinalite="[0;infini]"> <conditionDisponibilite comment="les deux termes doivent être disponibles"> <expressionLogique> <facteurLogique> <elementLogique> <litteral numeroMembre="1" accesseur="DISPONIBLE"/> </elementLogique> <elementLogique> <litteral numeroMembre="2" accesseur="DISPONIBLE"/> </elementLogique> </facteurLogique> </expressionLogique> </conditionDisponibilite> <conditionSatisfaisabilite comment="le rang du premier terme doit être strictement inférieur au rang du second terme"> <expressionBooleenne> <expressionBooleenneBinaire operateur="INF"> <expressionArithmetique> <elementArithmetique> <terminal numeroMembre="1" accesseur="RANGFIN"/> </elementArithmetique> </expressionArithmetique> <expressionArithmetique> <elementArithmetique> <terminal numeroMembre="2" accesseur="RANGDEBUT"/> </elementArithmetique> </expressionArithmetique> </expressionBooleenneBinaire> </expressionBooleenne> </conditionSatisfaisabilite > </definitionPropriete>

TAB. 2 – Spécification de la linéarité

nominal, si on trouve deux contraintes d'obligation, la première portant sur le nom et la seconde sur l'adjectif, et que l'énoncé analysé conduit envisager la construction d'un groupe nominal constitué d'un nom et d'un adjectif, alors la configuration suivante se produira : pour le nom comme pour l'adjectif, la propriété sera satisfaite. Après considération des conditions de capacité et de cardinalité, on aura bien satisfait une et une seule fois chaque contrainte (capacité =1). On aura aussi satisfait le fait qu'au moins une contrainte d'obligation a été satisfaite (cardinalité=2). Le choix d'un noyau (tête de syntagme) pour la propagation des traits, dépendra de l'ordre dans lequel les contraintes ont été écrites (c'est ce que stipule le fait que la propriété d'obligation est ordonnée). Il faut avoir à l'esprit que la spécification que nous donnons pour les propriétés des grammaires doit rester compatible avec des techniques d'analyse dotées d'heuristiques, même si le formalisme des Grammaires de Propriétés ne prévoit rien à ce sujet, la question de la propagation des traits à l'aide d'une tête sélectionnée parmi les termes satisfaisant les propriétés a été sérieusement envisagée. Des versions différentes de l'obligation peuvent être données en modifiant la spécification. C'est ce qui a été fait pour la campagne d'évaluation EASY, dont nous parlerons dans la quatrième partie.

Définition de Propriété	
obligation : := [nom= 'obligation', symbole= '+', arité= 1, Ordonnée=Vrai, Satisfaisabilité = [$\Omega = \omega_1$, Ψ= Vrai, CardinalitéMin = 1, CardinalitéMax = ∞, CapaciteMin = 0, CapaciteMax = 1].	<definitionPropriete nom="obligation" comment="têtes potentielles de la catégorie" symbole="+" arite="1" capacite="[0;1]" cardinalite="[1;infini]" ordonnee="VRAI"> <conditionDisponibilite comment="le terme doit être disponible pour que la clause le soit"> <expressionLogique> <elementLogique> <litteral numeroMembre="1" accesseur="DISPONIBLE"/> </elementLogique> </expressionLogique> </conditionDisponibilite> <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles"> <expressionBooleenne> <constanteBooleenne valeur="VRAI"/> </expressionBooleenne> </conditionSatisfaisabilite> </definitionPropriete>

TAB. 3 – Spécification de l'obligation

3.3.4 Spécification de l'exigence

L'exigence est une propriété plus simple que l'obligation. Elle est évaluable dès que le premier terme est disponible. Elle est alors satisfaite si le second terme est disponible. Notons aussi que l'exigence n'est pas symétrique, c'est à dire que si A exige B, B n'exige pas forcément A.

3.3.5 Spécification de l'exclusion

L'exclusion est évaluable dès que le premier terme est disponible. Elle est alors satisfaite si le second terme n'est pas disponible dans la construction. Contrairement à l'exigence, il faut attendre d'avoir réalisé une construction pour savoir si l'exclusion est satisfaite. Le fait de devoir réaliser un calcul post-construction est un problème épineux qui sera discuté à la fin de cette partie. Remarquons que l'exclusion n'est pas non plus symétrique (A exclut B est

Définition de Propriété
exigence : :=
[nom= 'obligation',
symbole= '=>',
arité= 2,
Ordonnée=Faux,
Satisfaisabilité =
[Ω= ω_1,
Ψ= (t1.estDisponible ∧ t2.estDisponible),
CardinalitéMin = 0,
CardinalitéMax = ∞,
CapaciteMin = 0,
CapaciteMax = ∞].

```
<definitionPropriete
        nom="exigence"
        comment="le terme 1 exige le terme 2"
        symbole="=)"
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
        <conditionDisponibilite comment="les deux termes doivent
                                        être disponibles">
            <expressionLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
            </expressionLogique>
        </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans
                                    condition si elles sont disponibles">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
```

TAB. 4 – Spécification de l'exigence

une contrainte satisfaite en présence de A et en absence de B. Si B est présent et pas A, cette contrainte n'est même pas évaluée.)

3.3.6 Spécification de la dépendance

La dépendance joue un rôle essentiel dans l'analyse avec les Grammaires de Propriétés : pour vérifier si deux catégories s'accordent pour un ou plusieurs de leurs types de traits, on écrira une contrainte de dépendance dans la grammaire. La fonction associée à ce contrôle des types de traits s'appuie sur les éléments présents lors de l'analyse. Si la grammaire contient une propriété de dépendance en genre entre un nom et un déterminant, la fonction VERIF-DOMAINES aura le rôle de contrôler le trait de genre du nom et celui du déterminant. Les domaines à contrôler sont choisis au moment de la rédaction de la grammaire. Pour cela, on utilisera les éléments XML RefCateg, RefValeur et RefTrait. Des exemples seront fournis plus loin, mais on pourra aussi se référer à l'annexe E pour consulter une grammaire complète.

3.3.7 Spécification de l'unicité

Pour l'unicité, comme pour l'obligation, nous considérons que les propriétés sont satisfaites d'emblée dès que le terme qu'elles désignent est disponible. Après construction, on vérifie la cardinalité et la capacité de l'ensemble des contraintes d'unicité, ce qui conduit à les satisfaire ou à les enfreindre toutes. Dans le cas de l'unicité, c'est la condition de capacité (maximum=1) qui joue le rôle crucial permettant de sanctionner ou de valider la propriété.

3.3.8 Spécification de nouvelles propriétés

l'annexe D fournit une version XML de la spécification sémantique que nous avons initiée ici. De nouvelles propriétés y ont introduites. Par exemple, l'*interdiction*, la *contiguïté*, la

Définition de Propriété **exclusion** : := [nom= 'exclusion', symbole= ' ($\neq$), arité= 2, Ordonnée=Faux, Satisfaisabilité = [$\Omega = \omega_1$, Ψ= (t1.estDisponible $\wedge$ $\neg$ t2.estDisponible), CardinalitéMin = 0, CardinalitéMax = ∞, CapaciteMin = 0, CapaciteMax = ∞].	<definitionPropriete nom="exclusion" comment="le terme 1 exclut le terme 2" symbole=")(" arite="2" capacite="[0;infini]" cardinalite="[0;infini]"> <conditionDisponibilite comment="Disponible si les deux membres le sont"> <expressionLogique> <elementLogique> <litteral numeroMembre="1" accesseur="DISPONIBLE"/> </elementLogique> </expressionLogique> </conditionDisponibilite> <conditionSatisfaisabilite comment="le premier terme doit être disponible mais pas le second"> <expressionLogique> <facteurLogique> <elementLogique> <litteral numeroMembre="1" accesseur="DISPONIBLE"/> </elementLogique> <elementLogique> <litteralNegative> <litteral numeroMembre="2" accesseur="DISPONIBLE"/> </litteralNegative> </elementLogique> </facteurLogique> </expressionLogique> </conditionSatisfaisabilite> </definitionPropriete>

TAB. 5 – Spécification de l'exclusion

facultativité et la *constituance*. La première est seulement utilisée à des fins d'expérimentation, car son calcul est très couteux. La seconde et la dernière ont été utilisées lors de la campagne EASY. La troisième fait partie de l'ensemble historique des types de propriété appartenant aux Grammaires de Propriétés. Il apparaît à présent plus clair que la difficulté du modèle que nous avons introduit réside dans l'impossibilité de définir certaines nouvelles propriétés qu'on pourrait juger utile d'introduire pour une analyse spécifique. Toutefois, avec la capacité, la cardinalité, l'arité, la satisfaisabilité et la disponibilité définissables avec la logique du premier ordre et l'arithmétique sur les entiers, nous disposons de suffisamment de degrés de liberté pour travailler fidèlement dans le cadre des Grammaires de Propriétés. Nous regrettons de ne pas pouvoir cerner clairement le domaine formel que définit implicitement l'ensemble des facettes qui constituent notre modèle.

3.3.9 Grammaire des GPs

Sur les bases du formalisme développé jusqu'ici, nous pouvons à présent développer une Grammaire de Propriétés.

A partir des extraits du DTD présenté plus haut (et consultable dans son intégralité à l'annexe B), nous avons pu définir plusieurs grammaires. Avant de les présenter, nous allons tout d'abord étudier simplement un exemple réduit de grammaire qui met en œuvre ce format de document. Dans la grammaire détaillée ci-dessous dans la figure 22 (consultable dans son intégralité à l'annexe l'annexe E), nous trouverons pour l'élément *grammaire* les attributs *label*, *type* et *comment*. Le premier attribut constitue le nom de la grammaire, le second spécifie le type de traits morphosyntaxiques (LPLTRAITS fait ici référence au type de traits du LPL usuellement employés dans le lexique ressource que nous avons présenté plus tôt).

Les sous-éléments directs de l'élément *grammaire* sont les éléments *catégorie* dotés de l'at-

Définition de Propriété
dépendance : :=
[nom= 'dépendance',
symbole= '∼∼',
arité= 2,
Ordonnée=Faux,
Satisfaisabilité =
[Ω= $\omega_1 \wedge \omega_2$,
Ψ = traitsCorrespondent(t1.traits,t2.traits),
CardinalitéMin = 0,
CardinalitéMax = ∞,
CapaciteMin = 0,
CapaciteMax = ∞].

```
<definitionPropriete
        nom="dependance"
        comment="accord (dépendance) entre deux termes"
        symbole="~~"
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
    <conditionDisponibilite comment="les deux termes doivent
                                          être disponibles">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1"
                              accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteral numeroMembre="2"
                              accesseur="DISPONIBLE"/>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites
                    si les domaines de leurs termes sont contrôlés">
        <expressionLogique>
            <facteurLogique>
                <expressionBooleenne>
                    <expressionBooleenneUnaire operateur="VERIFDOMAINE">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="1" accesseur="DOMAINE"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneUnaire>
                </expressionBooleenne>
                <expressionBooleenne>
                    <expressionBooleenneUnaire operateur="VERIFDOMAINE">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="2" accesseur="DOMAINE"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneUnaire>
                </expressionBooleenne>
            </facteurLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
```

TAB. 6 – Spécification de la dépendance

Définition de Propriété
unicité : :=
[nom= 'unicité',
symbole= '1',
arité= 1,
Ordonnée=Faux,
Satisfaisabilité =
[Ω= ω_1,
Ψ= Vrai,
CardinalitéMin = 0,
CardinalitéMax = ∞,
CapaciteMin = 0,
CapaciteMax = 1].

```
<definitionPropriete
        nom="unicite"
        comment="catégories uniques dans un syntagme"
        symbole="1"
        arite="1"
        capacite="[0;1]"
        cardinalite="[0;infini]"
        ordonnee="FAUX">
    <conditionDisponibilite comment="le terme unique doit être disponible
                                          pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites
                        sans condition si elles sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
```

TAB. 7 – Spécification de l'unicité

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE grammaire SYSTEM "grammaire.dtd">
<?xml-stylesheet type="text/xsl" href="grammaire.xsl"?>
<grammaire label="grammaire" type="LPLTRAITS" comment="grammaire exemple 1" >
```

FIG. 22 – Extrait de grammaire XML

tribut *label* qui permet d'identifier le nom de la catégorie à définir. Dans l'exemple choisi, seules les catégories D (déterminant), N (nom) et SN (syntagme nominal) sont définies. Notons que chaque élément *catégorie* contient les sous-éléments *traits* et *propriétés*, qui peuvent éventuellement être vides. Dans l'exemple fourni (figures 23 et 24), on observe que les déterminants sont uniquement définis par des traits et non par des propriétés. La catégorie SN, par contre est définie à la fois par des traits et par des propriétés. Cette grammaire possède d'une certaine manière des éléments terminaux (au sens de la théorie des langages), qui sont le déterminant et le nom (ceux-ci ne pouvant pas être définis à partir d'autres éléments de la grammaire). Le SN par contre est une catégorie récursive non terminale, puisque ce dernier est spécifié par des propriétés mettant en jeu des noms et des déterminants. Cette grammaire exemple n'a pas beaucoup d'intérêt linguistique, puisqu'elle a été délibérément limitée à ces trois catégories. Afin de simplifier la lisibilité de cette grammaire, nous avons aussi choisi de définir le SN à partir de propriétés faisant uniquement référence à des déterminants et des noms. Nous verrons dans d'autres exemples que les catégories terminales peuvent aussi être décrites de façon non terminale dans le formalisme des Grammaires de Propriétés.

```
<categorie label="D" comment="determinant">
        <traits label="D_traits" comment="traits du déterminant">
            <trait label="soucat" comment="pos2">
                <valeur label="a" comment="article?"/>
                <valeur label="d" comment="démonstratif"/>
                <valeur label="i" comment="indéfini"/>
                <valeur label="s" comment="possessif"/>
                <valeur label="t" comment="interrogatif"/>
            </trait>
            <trait label="ordre" comment="pos3">
                <valeur label="1" comment="première personne"/>
                <valeur label="2" comment="deuxième personne"/>
                <valeur label="3" comment="troisième personne"/>
            </trait>
            <trait label="genre" comment="pos4">
                <valeur label="m" comment="masculin"/>
                <valeur label="f" comment="féminin"/>
            </trait>
            <trait label="nombre" comment="pos5">
                <valeur label="s" comment="singulier"/>
                <valeur label="p" comment="pluriel"/>
            </trait>
            <trait label="possesseur" comment="pos7'">
                <valeur label="s" comment="singulier"/>
                <valeur label="p" comment="pluriel"/>
            </trait>
            <trait label="resultatFusion" comment="pos8">
                <valeur label="d" comment="défini"/>
                <valeur label="i" comment="indéfini"/>
            </trait>
            <trait label="fusionAvecQuoi" comment="pos9">
                <valeur label="a" comment="à + le"/>
                <valeur label="d" comment="de + le"/>
            </trait>
        </traits>
        <proprietes label="_props" comment="pas de propriétés">
        </proprietes>
    </categorie>
```

FIG. 23 – Extrait de grammaire XML (suite)

La définition des sous-éléments de l'élément *traits* consiste en une liste de plusieurs ensembles de traits possibles (pour le déterminant, les ensembles de traits sont soucat, ordre, genre, nombre, possesseur, etc.). Pour chacun de ces ensembles, on définit les valeurs possibles

grâce à l'élément *valeur*.

Dans la définition du SN, un autre sous-élément définit comment les traits peuvent être attribués au SN (l'élément *propagation*). Actuellement, le nom des ensembles de traits à propager sert à définir comment un SN recevra les traits hérités de ses constituants. Nous sommes conscients de l'insuffisance de cette information. La mise en œuvre de techniques de propagation de traits sera abordée plus loin, lorsque nous envisagerons les limites et les perspectives des techniques développées.

Le second élément qui définit une catégorie est l'élément *propriétés*. Cet élément est constitué de plusieurs sous-éléments dénommés *propriété* et associés à l'attribut *label* qui spécifie le nom de la propriété utilisée. Pour chacune de ces propriétés, un ensemble d'éléments appelés *clause* permet de préciser un ensemble de contraintes sur des catégories. Chaque *clause* est constituée d'autant d'éléments *membre* que nécessaire (c'est l'arité de la propriété qui spécifie ce nombre). Ainsi, l'obligation, l'unicité, la facultativité nécessitent un seul membre par clause, tandis que la linéarité, l'exigence, l'exclusion et la dépendance en nécessitent deux. Chaque membre peut être défini directement comme une référence à une catégorie de la grammaire (élément *refcateg*), soit comme une alternative entre plusieurs catégories de la grammaire (voir l'élément *opérationLogique*). Dans la grammaire ci-dessous, on observe que seul un nom (N) est nécessaire à la construction d'un SN (propriété d'obligation). Les propriétés de facultativité et d'unicité du SN stipulent que si un déterminant est présent, celui-ci doit être unique. La propriété de linéarité stipule que le déterminant doit précéder le nom. La propriété d'exclusion précise qu'un nom propre exclut la présence d'un déterminant (notons l'emploi du sous-élément *refValeur* pour préciser une contrainte sur des valeurs de traits). La propriété d'exigence pose la nécessité d'un déterminant lorsqu'un nom commun est présent. Enfin, la propriété de dépendance régit l'accord en genre et en nombre entre le déterminant et le nom.

3.4 Caractéristiques importantes

La liste des possibilités qu'offrent les Grammaires de Propriétés n'est pas exhaustive étant donné que de nouvelles propriétés peuvent être introduites. Nous pouvons cependant, vu l'espace de représentation de la sémantique des GPs, se donner des éléments de calcul pour la complexité maximale de l'analyse dans le cadre des GPs. Pour cela, il faut étudier de près les possibilités calculatoires de chaque type de propriété. Il faut aussi envisager le processus d'analyse aussi bien que les possibilités offertes par le formalisme.

Lorsque l'analyse débute, l'entrée et la grammaire sont confrontées d'une manière qui est caractéristique de chaque analyseur. Quelle que soit l'heuristique, quel que soit l'algorithme employé, on retrouve avec les GP un certain nombre de principes qui gouvernent usuellement le processus d'analyse dans d'autres approches ou formalismes. Les principes que nous énumérons ci-dessous ne sont pas obligatoirement appliqués dans l'analyse syntaxique avec les GP et leur emploi dépend du type de la grammaire qui est utilisée autant que de l'algorithme : le *principe de non-répétition* aura tout lieu d'être employé si la grammaire autorise la projection par récursion directe ou indirecte. Le *principe de la souche la plus large* intéresse les analyseurs que nous voulons rendre déterministes, les *principes de densité de satisfaction* (minimale, maximale, locale et propagée) ont intérêt à être utilisés dans l'approche dite "à granularité variable" ainsi que dans les analyseurs pour lesquels le taux de grammaticalité est à prendre en compte. Nous abordons dans ce chapitre les principes qui gouvernent le formalisme des Grammaires de Propriétés ainsi que ceux qui peuvent diriger ou guider le processus d'analyse.

```
<categorie label="SN">
    <traits label="SN_traits" comment="traits du syntagme nominal">
        <trait label="soucat">
            <valeur label="c" comment="commun"/>
            <valeur label="d" comment="propre avec déterminant"/>
            <valeur label="p" comment="propre sans déterminant"/>
            <valeur label="l" comment="latin"/>
        </trait>
        <trait label="genre">
            <valeur label="m" comment="masculin"/>
            <valeur label="f" comment="féminin"/>
        </trait>
        <trait label="nombre">
            <valeur label="s" comment="singulier"/>
            <valeur label="p" comment="pluriel"/>
        </trait>
        <trait label="sigle">
            <valeur label="s" comment="sigle (abréviation, etc.)"/>
        </trait>
        <trait label="typeNomPropre">
            <valeur label="c" comment="pays"/>
            <valeur label="h" comment="habitants"/>
            <valeur label="s" comment="société"/>
        </trait>
        <propagation label="soucat"/>
        <propagation label="genre"/>
        <propagation label="nombre"/>
        <propagation label="sigle"/>
        <propagation label="typeNomPropre"/>
    </traits>
    <proprietes label="_props" comment="propriétés">
        <propriete label="obligation">
            <clause>
                <membre>
                    <refCateg label="N" />
                </membre>
            </clause>
        </propriete>
        <propriete label="facultativite">
            <clause>
                <membre>
                    <refCateg label="D"/>
                </membre>
            </clause>
        </propriete>
        <propriete label="unicite">
            <clause>
                <membre>
                    <refCateg label="D"/>
                </membre>
            </clause>
        </propriete>
        <propriete label="exclusion">
            <clause>
                <membre>
                    <refCateg label="N">
                        <refValeur label="soucat" operateur="=" valeur="p"/>
                    </refCateg>
                </membre>
                <membre>
                    <refCateg label="D"/>
                </membre>
            </clause>
        </propriete>
        <propriete label="linearite">
            <clause>
                <membre>
                    <refCateg label="D"/>
                </membre>
                <membre>
                    <refCateg label="N" />
                </membre>
            </clause>
        </propriete>
        <propriete label="exigence">
            <clause>
                <membre>
                    <refCateg label="N">
                        <refValeur label="soucat" operateur="=" valeur="c"/>
                    </refCateg>
                </membre>
                <membre>
                    <refCateg label="D"/>
                </membre>
            </clause>
        </propriete>
        <propriete label="dependance">
            <clause>
                <membre>
                    <refCateg label="D">
                        <refTrait label="genre"/>
                        <refTrait label="nombre"/>
                    </refCateg>
                </membre>
                <membre>
                    <refCateg label="N">
                        <refTrait label="genre" />
                        <refTrait label="nombre" />
                    </refCateg>
                </membre>
            </clause>
        </propriete>
    </proprietes>
</categorie>
```


FIG. 24 – Extrait de grammaire XML (suite)

3.4.1 Définitions

Definition 24 (Relâchement des contraintes).

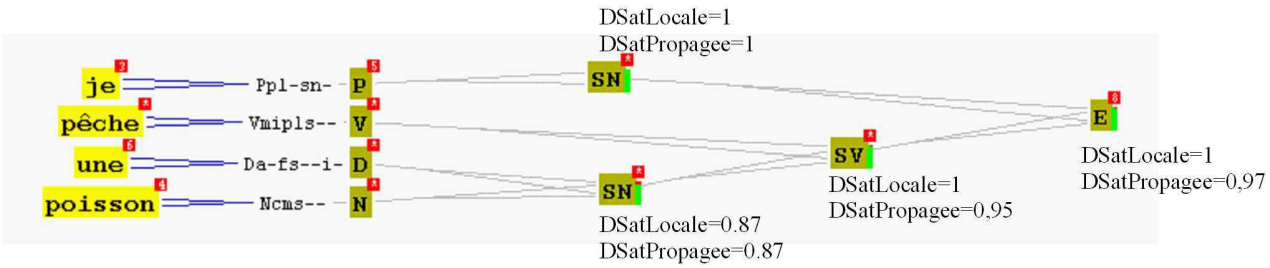
Toutes les contraintes posées dans une Grammaire de Propriétés peuvent être relâchées. Cela signifie, à l'extrême, que l'ensemble des contraintes qui permettent de caractériser un syntagme S peuvent être violées et que le syntagme S n'en est pas moins caractérisé. Le principe de relâchement des contraintes apporte une gradation dans l'idée de **grammaticalité**. De fait, il devient possible de mesurer et de comparer des interprétations, des caractérisations, en fonction du nombre de contraintes satisfaites ou enfreintes. Il devient aussi possible, grâce à cette mesure, de filtrer pendant l'analyse, en fonction du contexte, les caractérisations ayant peu d'intérêt. Ceci nous conduit directement à la *mesure de densité de satisfaction*.

Definition 25 (Satisfaisabilité et densité de satisfaction).

Nous définissons, pour les besoins de l'analyse à proprement parler, une mesure qui permettra de comparer des caractérisations entre elles. Chaque catégorie caractérisée au cours d'une analyse porte deux valeurs qui mesurent sa **densité de satisfaction locale** et sa **densité de satisfaction propagée**.

- au cours de l'analyse, pour chaque catégorie caractérisée, la quantité de propriétés satisfaites et la quantité de propriétés enfreintes est comptabilisée, ce qui définit une mesure de la **densité de satisfaction locale** d'une catégorie.
- lorsqu'une catégorie peut à son tour être caractérisée par un groupe de catégories déjà caractérisées, elle hérite de la moyenne des densités de satisfaction de ses constituants immédiats, ce qui définit la notion de **densité de satisfaction propagée**.

L'exemple donné ci-dessous illustre ce procédé. Dans cet exemple, une propriété de dépendance



en genre est enfreinte dans le second *SN*. Cette violation n'interdit pas de considérer un SN satisfait à 87%. L'information est propagée au SV et à la phrase E, qui du point de vue de ses constituants immédiats est satisfaite à 100%, mais seulement à 97% du point de vue de l'ensemble de ses sous-constituants. En autorisant ou en interdisant des constructions au dessous d'un certain seuil de satisfaction, cette analyse recouvre donc les quatre possibilités de réglage de granularité présentées dans notre réflexion initiale :

- réglage de la hiérarchie des structures caractérisées, en jouant sur le seuil de propagation de la densité de satisfaction.
- réunion des modalités,
- tolérance à l'agrammaticalité en étant permissif sur les seuils de densité propagée et locale,
- modulation du déterminisme des analyses [3].

[3] une heuristique générale portant sur ce point est présentée dans la troisième partie.

Des perspectives en relation avec l'importance relative des seuils à attribuer aux différents types de propriétés sont offertes par les domaines psycho-linguistiques et les sciences cognitives, qui mettent en avant l'importance de certains types de contraintes par rapport à d'autres dans le processus de compréhension des textes. On se référera par exemple à [Winograd, 1983].

La mesure de densité de satisfaction que nous définissons ici est cruciale pour la compréhension de la suite des descriptions et des algorithmes associés à l'analyse syntaxique. Nous donnons donc ci-dessous quelques précisions et définitions importantes pour la suite.

Definition 26 (Mesure de densité de satisfaction).

La densité de satisfaction est un réel compris entre 0 et 1 inclus.

Mesure locale (DSL)	La *densité de satisfaction locale* mesure la qualité immédiate d'une caractérisation. Toute propriété a pour densité de satisfaction locale : 1 si la propriété est satisfaite, 0 si la propriété est enfreinte. Une catégorie caractérisée donnée a pour densité de satisfaction locale la valeur du quotient : $\frac{Somme\ des\ densites\ de\ satisfaction\ locales\ de\ ses\ proprietes}{Nombre\ total\ de\ ses\ proprietes\ evaluees\ (satisfaites\ +\ enfreintes)}$
Mesure propagée (DSP),	La *densité de satisfaction propagée* mesure l'historique d'une caractérisation. Toute propriété de niveau 1 a pour densité de satisfaction propagée la valeur de sa densité de satisfaction locale Toute propriété de niveau plus grand que 1 a pour densité de satisfaction propagée la valeur de sa densité de satisfaction locale multipliée par la valeur du quotient : $\frac{Somme\ des\ densites\ de\ satisfaction\ propagees\ de\ ses\ termes}{Nombre\ total\ de\ ses\ termes}$ Une catégorie caractérisée donnée a pour densité de satisfaction propagée la valeur du quotient : $\frac{Somme\ des\ densites\ de\ satisfaction\ propagees\ de\ ses\ proprietes}{Nombre\ total\ de\ ses\ proprietes\ evaluees\ (satisfaites\ +\ enfreintes)}$

La figure 25 montre comment se propage la mesure de densité de satisfaction : pour l'énoncé "le peinture sèchent", deux contraintes de dépendance sont enfreintes. Lors de l'analyse du premier niveau, le syntagme nominal est construit avec une DSL valant 0,83. Le syntagme verbal est quant à lui parfaitement satisfait. Lors de l'analyse du second niveau, une catégorie phrase est construite, et les traits de nombre du SN et du SV sont alors en conflit (le SN est singulier et le SV est conjugué au pluriel). La DSL au second niveau vaut 0,80 tandis que la DSP à ce niveau vaut 0,718 du fait de l'héritage de constituants malformés. Au fil de l'analyse, grâce à ces deux mesures, il est possible de contrôler directement la construction de nouvelles structures en établissant des seuils de tolérance à l'agrammaticalité ou encore de déclencher des analyses approfondies si aucune construction ne dispose d'une mesure de densité de satisfaction suffisante. Les mécanismes permettant de réaliser ce contrôle sont implantés dans l'analyseur syntaxique SeedParser présenté dans la troisième partie.

Definition 27 (Déterminisme optionnel, grammaticalité et optimalité).

Grâce à la mesure de densité de satisfaction introduite ci-dessus, il devient possible de régler le nombre d'interprétations ambiguës durant l'analyse, simplement en jouant sur un seuil de

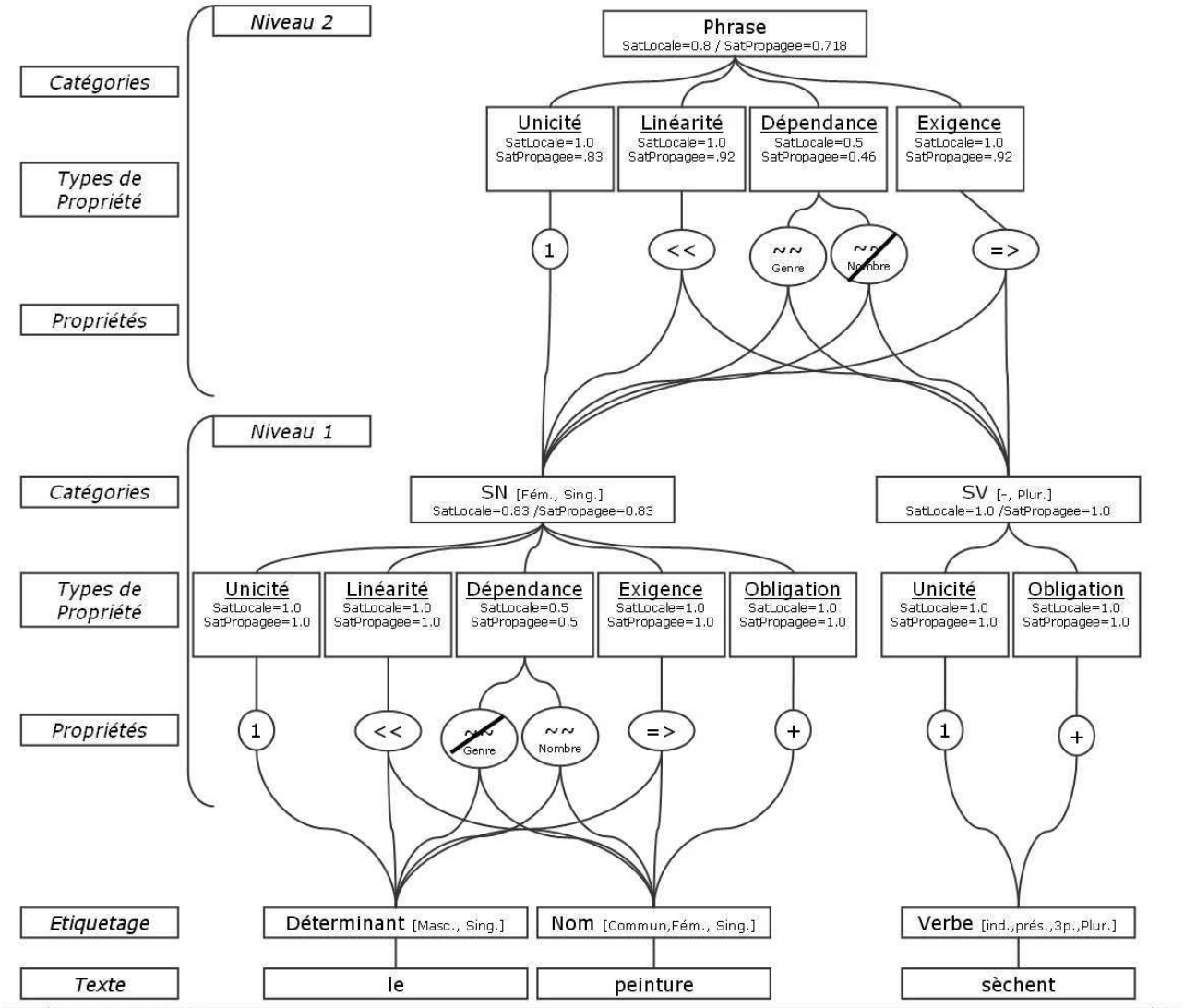


FIG. 25 – Un exemple de mesure de densité de satisfaction locale et propagée

densité de satisfaction : En dessous d'un certain seuil, aucune projection de syntagme n'est tolérée, ce qui contribue aussi à réduire la complexité de l'espace d'analyse. La mesure de densité de satisfaction s'assimile de fait à un certain degré de grammaticalité, ce qui permet ensuite un classement des caractérisations, dans le but de trouver les analyses optimales et éventuellement de passer à une analyse purement déterministe. L'importance de cette mesure ne suffit cependant pas à rendre l'analyse totalement déterministe. D'autre part, l'usage d'un seuil risque -dans des contextes d'analyse où l'entrée est très mal formée- de ne mener à aucune interprétation. C'est pourquoi, d'une part, il faudra se doter d'heuristiques qui complèteront la déterminisation si on souhaite absolument disposer d'une seule interprétation pour l'entrée [4], et d'autre part il faudra se doter de stratégies locales permettant de réanalyser l'entrée en deçà du seuil de densité (lorsqu'aucune caractérisation n'est produite au dessus de ce seuil). Cette dernière technique permettrait de passer d'une *analyse à granularité variable* à une *analyse à granularité automatiquement variable*. Nous n'avons pas développé plus loin cette idée, en constatant que d'une part les entrées conduisaient toujours à une interprétation (ce qui est dû au caractère toujours suffisamment bien formé des entrées et que d'autre part, une réelle sélection de granularité automatique supposerait de prendre en charge d'autres critères interprétatifs que la seule agrammaticalité (en effet, il peut être aussi intéressant de pousser plus avant une analyse syntaxique pour des raisons d'analyse sémantique ou pragmatique).

Definition 28 (Multimodalité).

Les Grammaires de Propriétés permettent de combiner plusieurs cadres interprétatifs, tels que la sémantique et la pragmatique dont nous venons de parler, la syntaxe, la prosodie, le domaine mimo-gestuel etc. Ceci suppose de disposer d'une entrée annotée selon plusieurs critères et d'analyser cette entrée multimodale à l'aide d'une grammaire contenant des descriptions de syntagmes ou de constructions appartenant à chacun de ces cadres interprétatifs, éventuellement en permettant de croiser des interprétations. L'analyse multimodale est donc possible dans le formalisme, mais pour la rendre opérationnelle, il faut d'une part se donner une entrée multimodale (ce qui est actuellement en cours de développement au Laboratoire Parole et Langage) et d'autre part disposer d'un analyseur capable de supporter plusieurs cadres interprétatifs. Nous allons donner ci-dessous plusieurs éléments en faveur d'un tel analyseur et observer quels obstacles techniques s'opposent à son développement.

Definition 29 (Multigrammaticalité).

De même que la multimodalité offre plusieurs cadres interprétatifs pour la même entrée, on pourrait dire qu'au sein d'un même cadre, il est concevable d'envisager plusieurs modèles d'interprétation ; plusieurs grammaires pour la syntaxe, plusieurs grammaires pour la sémantique etc. Là encore, le formalisme des Grammaires de Propriétés est tout à fait à même de représenter plusieurs grammaires pour le même cadre. C'est au programme d'analyse que revient le rôle de résoudre le problème de l'accroissement de complexité qui en résulte. Pour bien saisir cet accroissement de complexité, nous devons effectuer un détour qui permettra d'apporter une solution.

Definition 30 (Principe de projectivité).

Hérité des grammaires de dépendance, le principe de projectivité décrète qu'une catégorie X ayant pour empan les bornes $[x1, x2]$ avec $x1 \leq x2$, peut être constituée des catégories

[4] deux techniques de déterminisation seront comparées en quatrième partie.

Y et Z ayant pour empans respectifs $[y1, y2]$ et $[z1, z2]$ avec $y1 \leq y2, y1 \geq x1, y2 \leq x2$ et $z1 \leq z2, z1 \geq x1, z2 \leq x2$, mais en interdisant le croisement des bornes de Y et Z. C'est à dire, il faut vérifier que : ($y1 \leq z1$ et $y2 \leq z1$) ou ($y1 \geq z2$ et $y2 \geq z2$). On peut aussi dire que ce principe n'autorise que les constructions ayant la forme d'arbres, ou qu'il interdit les graphes acycliques orientés (Directed Acyclic Graphs en anglais). Dans les Grammaires de Propriétés, ce principe n'est pas forcément à vérifier. Les analyseurs peuvent s'en inspirer, mais ce n'est pas une obligation. Ce principe, qui s'applique à la plupart des théories courantes de la syntaxe, a l'avantage de réduire considérablement l'espace de calcul. Dès lors qu'une analyse multimodale ou multigrammaticale (au sens de ce qui a été exposé ci-dessus) devra être accomplie, il sera nécessaire d'enfreindre ce principe puisque plusieurs constructions devront partager plusieurs constituants. dès lors, les processus d'analyse deviendraient extrêmement coûteux en calculs.

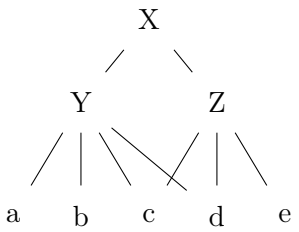


FIG. 26 – Construction interdite par le principe de projectivité

Definition 31 (Grammaires de propriétés colorées).

En réponse à ce problème, il reste possible de respecter le principe de projectivité au sein d'une même grammaire, et d'accepter de l'enfreindre entre grammaires différentes. On doit donc quitter l'idée qu'une seule grammaire puisse contenir l'ensemble des informations appartenant à plusieurs cadres interprétatifs sans distinguer ces cadres par une marque spécifique. D'où l'introduction de la notion de couleur pour les grammaires (ce qui a par exemple été réalisé par Claire Gardent pour distinguer des éléments au sein du formalisme du Lambda Calcul typé). On peut ainsi redéfinir la contrainte de projectivité en disant que celle-ci doit être respectée dans les analyses au sein d'une même couleur grammaticale. De cette façon, l'espace calculatoire n'est plus factoriel, mais au pire exponentiel.

Definition 32 (Principe de contiguité, catégories et relations).

Un autre problème de complexité est posé par les Grammaires de Propriétés : lorsqu'une analyse est réalisée, on cherche à construire un syntagme à partir de constituants contigus. C'est une contrainte forte qui, si elle est enfreinte, suppose l'existence de syntagmes discontinus. Quoique les théories linguistiques soient majoritairement en accord avec l'existence de syntagmes continus, nous interrogeons la notion d'analyse syntaxique en confrontant la notion de syntagme et la notion de relation. Les syntagmes tels que le SN (syntagme nominal) ont des constituants contigus. Les relations, telles que la relation sujet-verbe, ont des termes qui ne sont pas forcément contigus. Les Grammaires de Propriétés permettent d'envisager aussi bien les syntagmes que les relations en termes de caractérisation et d'ensembles de propriétés satisfaites. C'est l'étape de projection, coûteuse en calculs, qui nécessite une grande rigueur dans l'observation du principe de contiguité. En effet, la complexité de l'analyse se trouve à

nouveau démultipliée si on admet d'enfreindre ce principe. Il reste possible, et c'est le choix que nous avons fait, d'utiliser les Grammaires de Propriétés aussi bien pour les syntagmes que les relations, en établissant quelques règles d'écriture des grammaires et quelques principes durant l'analyse. En effet, les relations ont une particularité qui réduit considérablement l'espace de calcul : elles ont un nombre limité de constituants (deux pour la relation sujet-verbe, trois pour la relation de coordination par exemple). Lorsqu'une grammaire représente une relation, comme sujet-verbe, une marque indiquera la possibilité d'être non contigue dans la description de cet élément grammatical. Cette marque sera associée à une valeur entière qui permettra d'indiquer le nombre de constituants de cette relation. Ainsi, l'espace de calcul reste acceptable. C'est cette technique que nous avons employée pour la campagne d'évaluation EASY (voir à ce propos la quatrième partie de ce document).

Definition 33 (Principe de non-répétition).

Les GPs ne construisent pas de preuve. Le processus d'analyse peut cependant se comparer au processus de preuve au moins du fait qu'une trace des propriétés satisfaites ou non est conservée (la caractérisation). On peut imaginer que l'analyse d'une phrase peut conduire à l'explosion de cette caractérisation pour certaines contraintes faisant référence à des catégories dont elles sont elles-mêmes des propriétés. Ex : l'expression <SN : Obl(N,SN),...> indique qu'un SN peut être constitué soit d'un N, soit d'un SN. L'explosion combinatoire qui en découle rend l'analyse indéfinie si nous ne la privons pas -par un procédé ou un principe- de répéter indéfiniment la projection d'un SN au dessus d'un SN. Nous appelons ce principe le principe de non-répétition qui se définit comme une restriction à l'opération de projection : *lorsque l'analyse d'un input conduit à projeter un syntagme X au dessus d'un syntagme X ayant les mêmes bornes, ne pas effectuer cette projection.* L'idée sous-entendue par "au-dessus" est à prendre au sens large : pas forcément immédiatement au dessus, mais pourquoi pas à deux niveaux de projection ou plus. Ce principe est donc valable pour les récursions croisées. Par exemple, [<SN : Obl(N,SA)>, <SA : Obl(SN)>] est un cas de récursion indirecte où il faut aussi éviter la répétition infinie de projection. Ce principe, s'il est employé, réduit l'espace d'analyse assez considérablement, puisqu'on passe d'une complexité théorique infinie à une complexité -au pire- exponentielle.

Definition 34 (Principes de non-projection sur propriété lacunaire).

Les propriétés que nous avons définies plus haut ont des particularités qui peuvent rendre l'analyse très coûteuse en calculs. Nous avons dit que la projection des catégories se faisait selon la densité de satisfaction des propriétés entrant en jeu dans la définition de ces catégories. Il se trouve que certaines propriétés ne peuvent être évaluées que dans la limite d'un certain syntagme. Par exemple, en supposant que le syntagme nominal soit défini entre autres par l'interdiction d'un verbe, il faudra attendre d'avoir construit un SN pour savoir si l'interdiction est satisfaite dans l'envergure de ce SN. Ceci implique un calcul de satisfaisabilité avant projection et un autre après projection. La distinction entre ces deux calculs peut se faire en comparant la sémantique des propriétés qui peuvent être calculées avant projection et celles qui ne peuvent être calculées qu'après. Nous définissons **une propriété lacunaire** de la façon suivante : **une propriété est lacunaire si sa fonction de satisfaisabilité est définie à partir de la négation de la disponibilité d'au moins un de ses termes**. rappelons qu'une propriété est définie sémantiquement par un ensemble de constantes et de fonctions (constantes d'arité, de capacité, de cardinalité ; fonctions de disponibilité et de satisfaisabilié). La fonction de disponibilité indique à partir de quels termes présents dans l'entrée

il est possible de décider que la satisfaisabilité est calculable pour cette propriété. La fonction de satisfaisabilité se calcule sur des propriétés disponibles, en fonction à nouveau de la disponibilité des termes qui la constituent ainsi que des valeurs de leurs traits et de leurs positions respectives dans l'entrée. Ce qui permet d'évaluer directement une propriété de dépendance, par exemple, découle du fait que deux termes doivent être présents pour que la dépendance puisse être disponible. Si une dépendance est disponible, sa satisfaisabilité est calculable immédiatement en fonction des traits des éléments. La linéarité suit le même principe, puisqu'il suffit que deux termes soient présents pour qu'elle soit disponible et on peut alors calculer la satisfaction de cette linéarité en fonction des positions respectives des termes dans l'entrée. L'exclusion pose plus de problèmes : supposons que dans la grammaire du syntagme nominal (SN), nous ayons posé une propriété d'exclusion entre un déterminant et un nom propre. Si un déterminant est présent dans l'entrée, la propriété d'exclusion devient immédiatement disponible. On pourra ensuite juger qu'elle est enfreinte avec tous les noms propres de l'entrée. Par contre, on ne pourra pas dire qu'elle est satisfaite tant qu'on n'aura pas défini dans quel syntagme nominal (entre quelles bornes) elle doit être évaluée. L'exclusion est donc partiellement évaluable avant la projection et partiellement après. Il s'agit d'une propriété lacunaire au sens que nous définissons. Les conséquences sur l'analyse sont assez douloureuses. Si nous voulons rester cohérents avec l'idée qu'une projection peut se faire dès qu'une propriété est satisfaite, il faudrait accepter de construire tous les syntagmes possibles pour mesurer la satisfaction de toutes les propriétés (y compris les lacunaires) et en fonction de cette information, choisir si on conserve ou non la construction. Ce processus implique une surproduction des catégories durant l'analyse. Nous pouvons faire un choix arbitraire, qui dès lors nous éloigne du formalisme des GPs : ignorer les propriétés lacunaires pour le processus de projection, qui de toute façon se trouve à la frontière du formalisme. Ce choix a été utilisé dans l'analyseur que nous avons développé (voir la troisième partie), mais il reste paramétrable au moment de l'activation du processus.

Definition 35 (Générativité).

En fonction de la grammaire, dès lors que certaines catégories peuvent être définies récursivement (directement ou indirectement) à partir d'elles mêmes, la générativité des Grammaires de Propriétés devient un processus théoriquement interminable. C'est le cas des grammaires de Chomsky ou de Greibach, pour tous les axiomes définis récursivement. En dehors de cette remarque, il est tout à fait possible d'envisager la générativité dans le formalisme des GPs. Cependant, à la différence des grammaires génératives, le fait que la linéarité soit posée comme une contrainte "comme les autres" implique un processus de génération plus complexe. Nous ne nous attarderons pas sur ce point, qui ne concerne pas la complexité de l'analyse dans le cadre des GPs.

3.4.2 Conclusion

Pour évaluer la complexité des Grammaires de propriétés, si le principe de non projection sur propriété lacunaire satisfaite est appliqué, les propriétés telles que l'interdiction ou l'exclusion ne sont pas employées pour projeter des catégories lors de l'analyse, elles servent juste d'indice supplémentaire confirmant ou infirmant les choix qui seront faits avec d'autres propriétés satisfaites. Si on ajoute à cela le principe de non-répétition, la complexité de l'analyse reste acceptable. Comme nous le verrons dans la quatrième partie, lorsque l'énoncé est

assez correctement formé, cette complexité est polynomiale. Si par contre l'entrée est absolument mal formée (ce qui est rarissime dans le cadre d'une analyse de corpus, même oraux), la complexité devient exponentielle.

Nous avons posé ici quelques limites et surtout les intérêts de l'analyse dans le formalisme des Grammaires de Propriétés. Avant de considérer la mise en œuvre des analyseurs dans ce cadre, nous avons besoin de définir un ensemble d'outils et de ressources qui permettront l'analyse : un lexique et un programme permettant d'y accéder, un étiqueteur morpho-syntaxique capable de réaliser une désambiguisation. Ces outils ont été réunis dans une *suite logicielle*, afin de contribuer à l'ensemble des travaux nécessaires à l'analyse des langues naturelles. Cette suite se positionne comme une plateforme de développement de ressources, et fonctionne de façon modulaire. Sans ces outils, la réalisation des analyseurs que nous proposons ensuite aurait été impossible ou serait restée purement théorique. C'est grâce à ces outils que nous avons pu effectuer des évaluations assez strictes et participer à la campagne d'évaluation EASY.

Deuxième partie

Ressources et outils

Chapitre 4

La plateforme LPLSuite

Du point de vue informatique, intégrer une grammaire à un système de TALN nécessite la vérification de plusieurs contraintes, provenant autant du type de problème à traiter que des futurs utilisateurs du système. Tout d'abord le linguiste – qui développe la grammaire – et l'informaticien – qui l'intègre à un programme de TALN – manipulent le même formalisme ; il est donc nécessaire de convenir d'un modèle de représentation des objets de la grammaire répondant simultanément aux besoins des deux domaines.

Il est essentiel, de plus, pour la validité des outils ainsi que par économie de moyens, que les modifications faites à une grammaire soient immédiatement répercutées dans le programme qui s'en sert, sans qu'il soit nécessaire de modifier pour autant le programme[5] . A cette fin la représentation de des informations linguistiques en général et de la grammaire en particulier doit être parfaitement indépendante des programmes qui en font usage, ce qui implique d'exclure toute procédure ad hoc.

Enfin, l'outil informatique est lui-même soumis à des contraintes incontournables à plusieurs niveaux, allant du choix d'un format standardisé (tel que le xml), à celui de données et d'algorithmes évitant autant que possible l'explosion combinatoire des traitements. En conséquence de quoi les informations linguistiques seront représentées en mémoire sous forme de structures de données suffisamment souples pour permettre à diverses applications de les manipuler et d'en obtenir rapidement les résultats attendus.

Ces contraintes interviennent simultanément dans les choix de représentation. La problématique qui les gouverne consiste à conserver un regard suffisamment en recul par rapport aux relations qu'entretiennent modèles et théories linguistiques, et à choisir un modèle en démontrant qu'il est à la fois représentatif de la totalité des objets du formalisme, et générique – canonique – dans les représentations qu'il permet.

Partant des considérations et de la problématique exposées ci-dessus, le choix d'une représentation donnée pour les GP découle à la fois des travaux déjà réalisés sur ces mêmes grammaires et des perspectives de développement qu'elles offrent, tant dans le traitement de l'analyse syntaxique à granularité variable (*cf.* par exemple [Balfourier *et al.*, 2002]) que dans les traitements particuliers de la synthèse vocale à l'aide de représentations sémantiques et prosodiques au sein même de la grammaire (*cf.* par exemple [Blache and Hirst, 2001], ou [Blache, 2003]).

Le format de fichier XML, qui permet à la fois une lecture humaine et informatique des données de la grammaire, une correction manuelle comme un chargement aisé et précontraint

[5] On peut se reporter à ce propos à [Kermes and Evert, 2003].

grâce à une spécification de règles syntaxiques à vérifier au sein même de ce fichier (telle que le format DTD), est notre point de départ pour représenter les objets de la grammaire (voir à ce propos [Simov *et al.*, 2001] et [Simov *et al.*, 2002] par exemple).

Le modèle informatique qui s'apparente directement aux données des GP est celui des graphes, dans lesquels les nœuds sont associés à une structure " objet sémantique " permettant de les qualifier, et les arcs représentent les relations sémantiques entre ces objets. Ce modèle sera développé dans la prochaine partie.

Nous commençons dans cette partie par présenter les outils de plus bas niveau et les ressources qui les accompagnent. Ces ressources et outils répondent eux aussi aux critères d'implantation énumérés ci-dessus. Ainsi, le lexique peut être développé manuellement ou/et à l'aide aux outils. De même, les formats d'entrée sortie du segmenteur et de l'étiqueteur, qui obéissent à un modèle DTD + XML permettent la constitution et éventuellement la correction de fichiers étiquetés acceptables pour l'analyse syntaxique.

Les outils que nous développons doivent répondre non seulement à ce critère d'implantation qui pourrait se résumer par la dénomination d'"indépendance ressource/outils", mais aussi à un ensemble de critères classiques en informatique :

- Évolutivité : il doit être possible d'améliorer, d'enrichir les ressources et les programmes indépendamment les uns des autres. Nous démontrons ces possibilités au fil de cette partie.
- Réutilisabilité : les outils et ressources doivent être suffisamment généraux dans leurs fonctionnalités pour pouvoir servir à plusieurs usages sans nécessité de transformation. Avec la programmation orientée objet, la réutilisabilité bénéficie de l'apport de la notion d'héritage. De fait, lorsque l'on souhaite redéfinir une partie d'un traitement associé à un outil, il est possible de programmer un nouvel outil "héritant" des fonctionnalités de son modèle, sans toutefois avoir besoin de tout réécrire. Cette caractéristique a été employée pour définir un modèle de segmenteur et un modèle d'étiqueteur, dont ont ensuite hérité plusieurs segmenteurs et plusieurs étiqueteurs associés à des tâches spécifiques, telles que la participation à la campagne EASY, la programmation d'un outil de calcul de score, etc.
- Robustesse : Les outils doivent résister aux erreurs dues à des entrées mal formées, en déclenchant le cas échéant un processus de correction automatique, ou en restant silencieux face à une information intraitable. Le processus ne doit pas cesser durant un traitement. Il faut autant que possible donner une sortie pour une entrée, quitte à générer une sortie qui indiquera la présence d'une erreur. Tous les outils que nous avons développés obéissent à cette contrainte, même si dans certaines situations la seule réaction valable reste le silence.
- Modularité : Les outils doivent pouvoir être utilisés indépendamment les uns des autres ou bien pouvoir s'appeler (s'exécuter) mutuellement. Chaque outil représente à lui seul l'exécution d'une tâche atomique. L'ensemble des outils, vu à son tour comme un outil, réalise une tâche plus grande, constituée des sous-tâches de chaque élément. Ce principe est appliqué à l'ensemble des outils présentés ci-dessous sous le nom de LPLSuite (figure 27).

Les programmes présentés ici répondent autant que possible aux critères énoncés précédemment. Ils satisfont d'autre part les contraintes formelles élaborées dans la première partie. Pour atteindre ce double objectif, l'implantation des programmes peut s'envisager comme une suite de modules emboîtables. Chaque module procède à un traitement spécifique sur son entrée et sa sortie devient l'entrée du module suivant. Chaque module est capable de faire appel

directement (de façon automatique) au module précédent. Il est aussi possible de contrôler les processus de l'extérieur en lançant séparément les traitements. Chaque module fait appel à une ressource précise qui peut être modifiée manuellement, sans que soit nécessaire la recompilation du programme. Enfin, les entrées, sorties et ressources de chaque module sont formatés en fonction d'une spécification standard de document XML, dont la description est fournie sous forme de grammaire DTD (voir en annexe les formats en question).

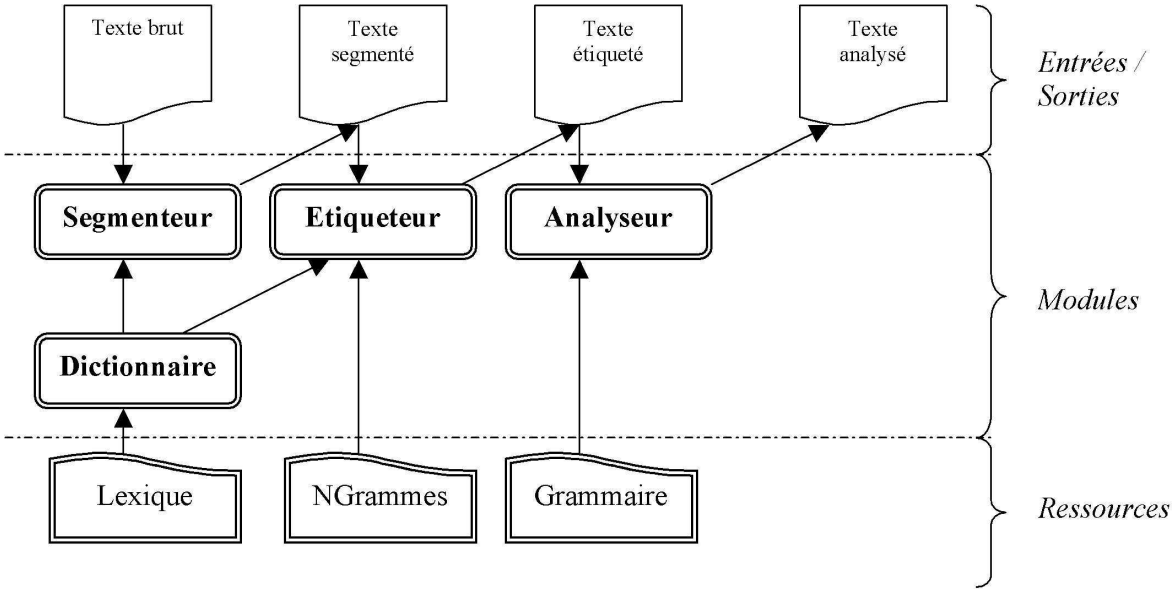


FIG. 27 – La suite de modules LPLSuite

Chapitre 5

Dictionnaire

5.1 Le lexique DicoLPL

La ressource particulière que constitue le dictionnaire a fait l'objet de beaucoup d'études et de travaux d'amélioration. Nous aboutissons actuellement à un lexique défactorisé de 423895 lignes correspondant à 320899 formes orthographiques différents.

Le dictionnaire est constitué d'un lexique et d'un programme permettant la maintenance, la sécurisation et l'interrogation du lexique. Ce projet est la base nécessaire aux outils de plus haut niveau qui auront besoin d'une ressource fiable, c'est pourquoi l'accent a été mis sur la maintenabilité de la ressource.

Le cahier des charges qui a permis l'élaboration de cette ressource a été longuement débattu. D'une part, à cause de l'existence de ressources différentes pour le Français, même à l'époque de cette élaboration (Le lexique Morphalou dont nous parlons plus loin, n'avait pas encore vu le jour). La démarche que nous avons adoptée est commandée par des nécessités parfois contradictoires : besoin de pouvoir embarquer le lexique dans de petits ordinateurs, nécessité de pouvoir modifier à la main le fichier lexique sans outil autre qu'un éditeur de texte, besoin de rapidité d'accès aux données, besoin d'un lexique défactorisé, disponibilité de certains champs pour des évolutions ultérieures, etc. L'efficacité n'est pas vraiment un problème, mais sa question était présente dès l'élaboration du cahier des charges. C'est pour ces raisons, liées au contexte de l'élaboration de la ressource, que certains choix opérationnels peuvent paraître assez simplistes (format de stockage ASCII compressé, par exemple). Ils correspondent paradoxalement à des choix mûrement réfléchis, au carrefour de plusieurs contraintes.

D'un point de vue externe, celui du stockage, étant donné le grand volume de données enregistrées confronté aux deux besoins antagonistes que sont d'une part celui d'accéder rapidement à l'ensemble des informations, qui implique leur compression et leur formatage binaire, et d'autre part celui de maintenir la base à l'aide d'outils de manipulation de texte impliquant la conservation d'un format textuel humainement lisible, nous avons opté pour un formatage hybride :
le lexique est un fichier texte encapsulé dans une archive compressée(format ZIP). Le langage JAVA fournit les bibliothèques de fonctions permettant l'accès rapide à ses 20 méga octets de données. L'archive peut être sécurisée par un mot de passe et le texte peut éventuellement être manipulé à l'aide de logiciels d'édition standards.

Pour répondre aux différents besoins des outils de Traitement Automatique, nous avons déployé une ressource susceptible d'évoluer. Les algorithmes et les structures de données in-

formatiques correspondant à cette ressource (décrits plus bas) ont été pensés pour que le chargement et l'interrogation de cette base soient extrêmement rapides. C'est pourquoi le texte du dictionnaire répond à des contraintes liées directement au programme qui les manipule. Le lexique est pour cette raison une table dont le nombre de colonnes a été limité à douze. Cette contrainte n'empêchera pas des développements futurs et correspond suffisamment aux besoins actuels. Du point de vue interne, le lexique est donc un fichier texte brut (ASCII) dont le nombre de lignes n'est pas fixé, mais dont le nombre de colonnes est limité à 12. Le contenu de deux colonnes d'une même ligne est séparé par un caractère non orthographique (la tabulation). Chaque colonne correspond à un champ particulier, dont le sens est décrit par la table 8 ci-dessous.

N° Colonne	Nom du champ	Format	Signification
1	Mot	Chaine de caracteres	Mot orthographié. Les espaces sont repérés par un tiret-bas "_".
2	MAJ	1 caractère	Information sur la mise en majuscules (voir la table 2)
3	Phon	Chaine de caracteres	Forme phonétisée du mot (voir le codage dans la table 3)
4	Freq	Entier	Fréquence du mot (voir plus bas pour son calcul)
5	Categ	Chaine de caracteres	Catégorie et traits morpho-syntaxiques du mot courant (voir la table 1)
6	Lemme	Chaine de caracteres	Forme lemmatisée du mot courant
7	Val	Chaine de caracteres	traits de valence
8	Sem	Chaine de caracteres	traits sémantiques
9	User	Chaine de caracteres	non attribué
10	User	Chaine de caracteres	non attribué
11	User	Chaine de caracteres	non attribué
12	User	Chaine de caracteres	non attribué

TAB. 8 – Champs du lexique

Comme ce lexique est destiné à une utilisation aussi rapide que possible, les programmes qui le parcourent et qui y effectuent des recherches supposent que cette ressource est "bien formée" en termes de formatage et de tri :

- Le format du tableau lexical respecte la contrainte de défactorisation suivante : une seule ligne correspond au quadruplet (Mot, Phon, Categ, Lemme).
- L'ordre des lignes de la table dépend des clés de tri suivantes (par ordre de priorité) : Mot, Phon, Categ et Lemme. Ce tri est de type alphabétique (assoupli pour les majuscules et les accents) pour la première colonne et lexicographique (ordre du code ASCII) pour

les trois dernières colonnes.
Ce type de dictionnaire est dit défactorisé par opposition à d'autres lexiques pour lesquels une seule ligne est réservée pour une forme orthographiée. Le volume du fichier s'en trouve accru, mais proportionellement à la facilité de retrouver des informations. Les contraintes ci-dessous, inhérentes à la constitution et à l'usage de ce lexique sont établies initialement :

- Certaines colonnes sont réservées pour un usage ultérieur.
- Les mots acceptés dans ce lexique ne doivent pas être des affixes, mais toujours des mots (simples ou composés) du langage courant. Ainsi, les préfixes et suffixes tels que "anti", "hecto", "isme", "able" en sont rejetés.

L'extrait de lexique donné dans la table 9 met en évidence les caractéristiques de son format. On y observe l'ordre de tri et la défactorisation du mot résidant.

Mot	MAJ	Phon	Freq	Categ	Lemme	...
Allemagne	1	al@manj@	38830	Ndfs-c	Allemagne	...
allemand	0	al@ma~	16708	Afpms-	allemand	...
Allemand	1	al@ma~	1762	Ndms-h	Allemand	...
...						
aller	0	ale	13132	Vmn——	aller	...
allèrent	0	alER@	107	Vmis3p–	aller	...
...						
allergie	0	alERZi	173	Ncfs–	allergie	...
allergies	0	alERZi	65	Ncfp–	allergie	...
...						
résidant	0	Rezida	302	Afpms-	résidant	...
résidant	0	Rezida	181	Ncms–	résidant	...
résidant	0	Rezida	520	Vmpp—-	résider	...

TAB. 9 – Extraits du lexique

Nous présentons en annexe F les tables de format et d'encodage pour les entrées du lexique. La table 1 de cette annexe présente la liste des traits morpho-syntaxiques possibles pour chaque catégorie que contient le lexique. Ces valeurs de traits sont inspirées du lexique Multext, avec quelques petites modifications ou ajouts tenant compte du format de traits de la campagne d'évaluation Grace. Certains traits de cas, par exemple, n'étaient pas interprétables de la même façon. Le codage des numéraux cardinaux, des mots d'origine étrangère, etc. a été revu pour définir une ressource homogène.

La table 2 de cette annexe décrit pour le champ MAJ (seconde colonne) l'information permettant de savoir si le mot peut/doit être en majuscule/minuscule pour son initiale/sa totalité.

Le codage phonétique du lexique avec l'alphabet SAMPA (table 3) répond aux critères énoncés plus haut : la saisie manuelle à l'aide d'un éditeur de texte simple est possible et un logiciel de phonétisation peut produire ce type de sortie.

Les autres champs prévus dans le dictionnaire ne sont pas tous renseignés. Actuellement, les codages pour la valence et les informations sémantiques ne sont pas définies. Plusieurs personnes travaillent au développement et à l'enrichissement de la ressource, dans des directions assez diverses. La partie du lexique la plus développée et pratiquement aboutie est celle qui permet aux autres outils de la plateforme de fonctionner : les informations syntaxiques

(catégories et traits morphosyntaxiques), les fréquences, formes phonétisées et lemmatisées sont intégralement renseignés.

Un lien effectif est entretenu entre le lexique et les grammaires que nous développons. Ce lien, dans la perspective de généricisation des ressources que nous avons adoptée, suppose que le codage des catégories et des traits soit partagé entre grammaire et lexique. Un fichier simple, reflétant exactement le contenu de la table 1 de l'annexe F permet aux modules d'analyse syntaxique de connaître et d'interpréter ces informations. Des travaux sont actuellement menés pour permettre certains traitements et expériences avec des traits très différents. L'idée de travailler sur des variantes d'un même lexique, avec des codages différents pour les traits syntaxiques, devrait ainsi permettre de comparer et d'évaluer des représentations diverses de la langue, avec des grammaires complètement différentes. Des travaux de linguistique formelle sont actuellement menés dans ce sens par Marie-Laure Guénot.

La ressource volumineuse que constitue ce lexique a été construite en plusieurs étapes semi-automatisées. La première étape a été la collecte de dictionnaires libres, la phonétisation, la standardisation des traits morpho-syntaxiques, la lemmatisation et la défactorisation des données. Ce processus est illustré par la figure 28 :

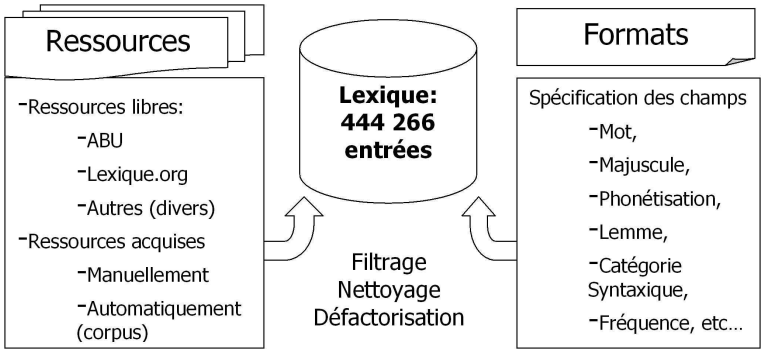


FIG. 28 – Conception du dictionnaire

Une seconde étape a permis de compléter la ressource : en parcourant de vastes corpus, nous avons enrichi le vocabulaire et vérifié les erreurs orthographiques. Troisièmement, nous avons filtré les redondances du dictionnaire en observant la règle d'unicité du quadruplet (Mot, Phon, Categ, Lemme). Enfin, le calcul des fréquences a été effectué grâce à des programmes de segmentation, d'étiquetage et de calcul de fréquences sur des corpus, présentés plus loin. Le dictionnaire ainsi achevé est trié automatiquement puis sauvegardé dans une archive zippée et protégée.

La ressource que constitue notre lexique a subi de nombreuses transformations et nécessité plusieurs années de travail. Son utilisation dans les outils que nous allons décrire dans la suite de cette partie est primordiale, mais nous verrons aussi plus loin que cette ressource permet d'en constituer d'autres (les lexiques noyaux). Enfin, nous avons choisi de rejoindre le projet Morphalou[6] et d'intégrer notre lexique à celui qui devrait rapidement devenir un des lexiques les plus riches de la langue Française. L'intérêt du lexique Morphalou réside dans sa structuration XML compatible avec la norme ISO/TC 37. Notre apport permettra d'enrichir

[6]voir [Romary *et al.*, 2004] http ://loreley.loria.fr/morphalou/

Morphalou avec une grande quantité d'informations phonétiques et fréquencielles, ainsi que de repérer nos propres erreurs. Les nombreuses différences de format entre Morphalou et DicoLPL ralentissent le processus. Les jeux de traits de l'un ne sont pas totalement représentés dans l'autre. La représentation hiérarchisée et factorisée des informations dans Morphalou correspond mal avec notre représentation redondante et défactorisée. Cependant, un grand nombre de phénomènes pourront être observés et étudiés au moment de la fusion des deux lexiques. La disjonction des deux lexiques mettra en évidence les erreurs et les phénomènes rares par exemple.

5.2 Le module Dictionnaire

Les programmes qui gravitent autour du lexique DicoLPL permettent d'une part sa maintenance et d'autre part son interrogation. Le *module dictionnaire* contient ces deux types d'outils : L'outil graphique dédié à cette tâche, le fréquenceur (chargé d'évaluer les fréquences des mots par analyse de corpus), les méthodes JAVA d'accès direct au dictionnaire.

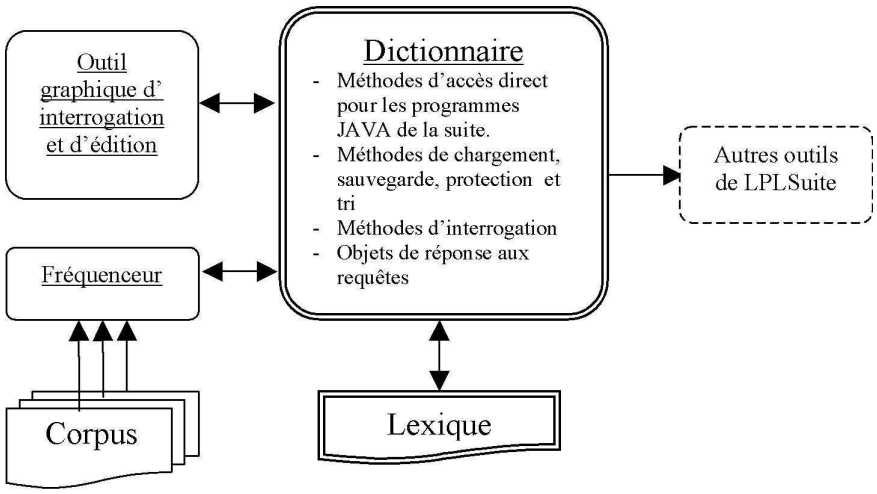


FIG. 29 – Vue d'ensemble du module Dictionnaire

Le programme *Dictionnaire* fonctionne essentiellement autour de la classe JAVA *Dictionnaire*, accessible en ligne de commande aussi bien que par une méthode publique de chargement. L'usage est prévu en accès direct, pour des manipulations visant à construire ou à préparer la ressource lexicale, ou en accès indirect pour des programmes de plus haut niveau nécessitant l'accès aux fonctionnalités de ce module par l'intermédiaire de méthodes publiques. L'objet Dictionnaire décrit ici fait appel à des bibliothèques utilitaires que nous avons développé séparément afin de permettre leur usage dans d'autres modules. Notamment la bibliothèque ByteTools (outils généraux dédiés au travail sur les caractères, leur forme octale et les chaînes de caractères) : méthodes de comparaison, de transtypage, d'accentuation, de désaccentuation, de mise en majuscule ou en minuscule. Enfin, l'objet Dictionnaire est associé à d'autres objets utilitaires décrits ci-dessous et permettant son interrogation. Les points d'accès au dictionnaire se résument à la liste ci-dessous.

– **Mode graphique en ligne de commande :**
la commande `java Dictionnaire -gui` déclenche le traitement par défaut de formatage

et de tri du lexique à partir d'un fichier préparé manuellement, pour le rendre utilisable par le module.

– **Mode console en ligne de commande :**
la commande `java Dictionnaire -tri -v(erbose) -i(nput) dico.zip -o(utput) newdico.zip` Dictionnaire charge dico.zip, le trie optionnellement (-tri) et sauve le lexique obtenu newdico.zip. L'option -v ou -verbose (mode bavard) fait afficher des informations relatives aux traitement effectués.

– **Mode indirect par accès au constructeur :**
la construction `Dictionnaire dico= new Dictionnaire ( "chemin_dico.zip")` déclenche la création d'une instance de l'objet Dictionnaire avec chargement d'un lexique valide. Les méthodes publiques données dans la table 10 sont alors accessibles. Certaines font directement partie de l'objet Dictionnaire tandis que les méthodes de recherche dans le lexiquefont appel à deux objets utilitaires associés au module : les classes ReponseDeRecherche et ReponseMasquée. Ces derniers objets offrent la possibilité de fournir très rapidement les indices des lignes du lexique correspondant à un critère précis, avec la possibilité d'assouplir éventuellement la rigueur autour des majuscules et des accents, ce que résument les tables 11 et 12.

Méthode	Description
public Dictionnaire (String chemin_fichier_dico_zip)	Constructeur sur fichier Lexique préformaté
int taille ( )	nombre de lignes du lexique
void charge (String fichier_in)	Effectue le chargement en mémoire des tables du lexique. Ce traitement est accéléré par le choix de structures de données destinées à l'optimisation pour la vitesse (indexation et tri)
void sauve (String fichier_out)	Effectue la sauvegarde du lexique (ce dernier est alors supposé trié)
byte[] getDico (int ligne, byte champ)	Les données correspondant au champ demandé pour la ligne demandée du lexique sont retournées dans un tableau d'octets
byte[] getGraphie (int ligne)	Le champ orthographique de l'entrée lexicale d'une ligne donnée
byte[] getMaj (int ligne)	Le champ Majuscule de l'entrée lexicale d'une ligne donnée
byte[] getPhonie (int ligne)	Le champ phonétique de l'entrée lexicale d'une ligne donnée
int getFrequence (int ligne)	La fréquence de l'entrée lexicale d'une ligne donnée
byte[] getCategorie (int ligne)	La categorie morpho-syntaxique de l'entrée lexicale d'une ligne donnée
byte[] getLemme(int ligne)	Le champ Lemme de l'entrée lexicale d'une ligne donnée
byte[] getSeme (int ligne)	Le champ Sème de l'entrée lexicale d'une ligne donnée
byte[] getValence (int ligne)	La Valence de l'entrée lexicale d'une ligne donnée
int[] tri ( )	Effectue le tri du dictionnaire et retourne le tableau d'indices correspondant aux numéros des lignes triées. Le critère de tri est précisé plus loin.
int CompareMots (int m1, int m2)	Critère de tri (voir plus loin pour l'algorithme de comparaison).
ReponseDeRecherche ChercheMot(byte[]s)	fournit la fourchette d'indices correspondant au mot cherché dans le lexique
ReponseDeRecherche ChercheMot(byte[]s, ReponseDeRecherche r)	fournit la fourchette d'indices correspondant au mot cherché dans le lexique en affinant une réponse déjà connue
ReponseDeRecherche ChercheMotsCandidats(byte[] n)	fournit la fourchette d'indices des lignes du lexique dont la graphie commence par le mot cherché
ReponseDeRecherche ChercheMotsCandidats(byte[] n, ReponseDeRecherche r)	fournit la fourchette d'indices des lignes du lexique dont la graphie commence par le mot cherché, en affinant une réponse déjà connue

Tab. 10 – Méthodes publiques du module Dictionnaire

Méthode	Description
public ReponseDeRecherche()	Constructeur vierge pour une recherche effectuée par le Dictionnaire.
public ReponseDeRecherche(ReponseDeRecherche r)	Constructeur par recopie d'un objet du même type(permet d'affiner une réponse sans perdre l'ancienne)
public ReponseDeRecherche(int ideb, int ifin, int longueur, byte[] b)	Constructeur renseigné (on crée la recherche d'un mot "b" dans le dictionnaire entre les indices ideb et ifin)

Tab. 11 – Méthodes publiques de l'objet ReponseDeRecherche du module Dictionnaire

– **Mode Visionneuse en ligne de commande :**
la commande `java DicoViewer [dico.zip]` déclenche l'affichage de la visionneuse de dictionnaire. Nous explicitons ci-dessous le fonctionnement et les particularités de cet outil.

Méthode	Description
ReponseMasquee (Dictionnaire dico, ReponseDeRecherche r, byte[] mot, boolean desaccentue, boolean sans_casse, boolean premier_mot)	Constructeur principal de l'objet RéponseMasquée. Cet objet suppose l'existence d'une RéponseDeRecherche valide pour un Mot donné dans un Dictionnaire donné. L'objet fournit un itérateur masquant les parties de réponse ne correspondant pas aux filtres fournis sur l'accentuation, la casse et la particularité des mots débutant une phrase.
public void init()	Réinitialise l'itérateur au début de la réponse de recherche.
boolean estPasFini()	Informe sur l'état de l'itérateur (reste-t-il des éléments valides non consommés pour les filtres donnés ?)
int taille()	Nombre total d'éléments valides présents dans la réponse de recherche pour les filtres donnés.
int suivant()	Indice du prochain élément de l'itérateur répondant aux critères de filtrage.

TAB. 12 – Méthodes publiques de l'objet ReponseMasquee du module Dictionnaire

Nous donnons ci-dessous quelques éléments algorithmiques accessibles à tous les outils de plus haut niveau (Segmenteur et Etiqueteur), qui permettent une recherche rapide, uniquement dans le champ contenant les formes orthographiées des mots. En effet, la rapidité étant un des critères les plus importants pour les outils de plus haut niveau, le module Dictionnaire doit autant que possible satisfaire cette contrainte. les recherches par mot othographié peuvent être strictes ou souples -selon le besoin- quant à la casse et à l'accentuation des mots. La rapidité de ces recherches présuppose qu'un tri a préalablement été réalisé sur l'ensemble du lexique. La clé primaire de ce lexique est assez grande, ce qui impose une fonction de comparaison assez grande. Nous en donnons ci-dessous quelques principes.
Nous fournissons ensuite les principes algorithmiques employés pour la visionneuse, qui diffèrent complètement des précédents. Pour cela, nous introduisons des outils mathématiques basés sur les intervalles discontinus d'entiers positifs. Grâce à ces outils, le hachage du lexique permet d'accélérer la recherche dans le dictionnaire par rapport à un parcours exhaustif classique. Nous passons en effet de 10 heures de calcul pour une recherche d'homophones hétérographes, à une durée plus acceptable de cinq minutes. L'algorithme de recherche par réductions d'intervalles est de complexité polynômiale, nous en fournissons plus bas les grandes lignes.

Pour commencer, nous nous intéressons aux algorithmes accessibles aux outils de plus haut niveau. Le segmenteur et l'étiqueteur ont besoin de connaitre rapidement l'appartenance d'un mot au dictionnaire, son indice permettant ensuite de récupérer les informations telles que la catégorie morphosyntaxique. La figure 30 illustre ces algorithmes, présente les structures de données associées et leurs différences à l'aide d'un exemple basé sur la recherche du mot *état*.

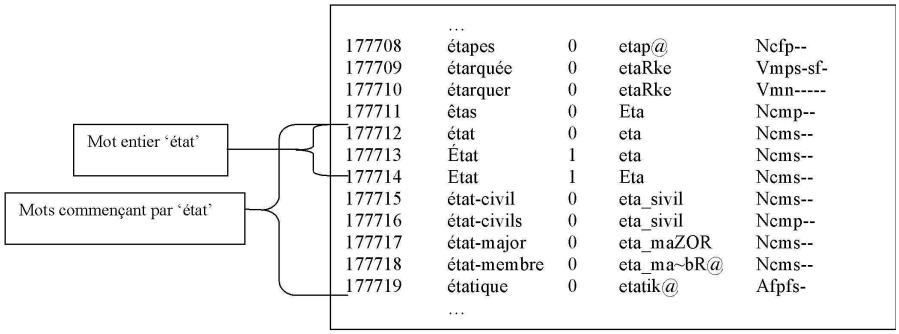
L'organisation du lexique et son interrogation supposent qu'un ordre de tri soit respecté. Cet ordre porte sur plusieurs champs qui forment la "clé" du lexique :

(graphie,MAJ,categorie,phonie,lemme).

L'ordre entre deux mots est dirigé par la comparaison désaccentuée et démajusculée des chaines de caractères ou bien l'ordre sur les entiers pour le champ MAJ. Comparer deux mots *m1* et *m2* se fait de la façon suivante :

Le tri du lexique consiste alors en un simple *quicksort* ayant la fonction donnée par l'algorithme 1 comme fonction de comparaison.

En se basant sur l'ordre de tri donné ci-dessus, la recherche dans le lexique s'effectue par dichotomie.
Il y a quatre situations de recherche envisageables pour une orthographe donnée. Ces recherches sont rendues nécessaires par les programmes de plus haut niveau -comme le segmenteur ou le



- Pour une recherche par mot complet

```
Structure RéponseDeRecherche
    IndexDebut      ∈[1;taille(Lexique)]        =177712
    IndexFin        ∈[1;taille(Lexique)]        =177714
    MotCherche      : chaîne                    =''état''
```

- Pour une recherche par le début du mot

```
Structure RéponseDeRecherche
    IndexDebut      ∈[1;taille(Lexique)]        =177712
    IndexFin        ∈[1;taille(Lexique)]        =177719
    MotCherche      : chaîne                    =''état''
```

- Pour une recherche avec filtres.

```
On initialise l'objet ci-dessous
Objet  RéponseMasquée
  Réponse           :RéponseDeRecherche         =(177712,177714,''état'')
  Majuscules        ∈{VRAI;FAUX}                =VRAI
  Accents           ∈{VRAI;FAUX}                =VRAI
  DebutPhrase       ∈{VRAI;FAUX}                =FAUX
  Procédure         Init (){…}
  Fonction          Suivant(){…}                ∈[1;taille(Lexique)]
  Fonction          Fini(){…}                   ∈{VRAI;FAUX}

Les appels successifs ci-dessous retournent les valeurs suivantes :
      Fini()        →    FAUX
      Suivant()     →    177712
      Fini()        →    VRAI

Avec une autre initialisation de l'objet RéponseMasquée
  Réponse           :RéponseDeRecherche         =(177712,177714,''état'')
  Majuscules        ∈{VRAI;FAUX}                =FAUX
  Accents           ∈{VRAI;FAUX}                =VRAI
  DebutPhrase       ∈{VRAI;FAUX}                =FAUX
Les appels successifs ci-dessous retournent alors les valeurs :
      Fini()        →    FAUX
      Suivant()     →    177712
      Fini()        →    FAUX
      Suivant()     →    177713
      Fini()        →    VRAI
```

FIG. 30 – Recherches du mot etat dans le lexique

Algorithme 1 Algorithme de comparaison dans le lexique

Entrées: $m1, m2 \subset [1; taille(Lexique)]$
Sorties: la comparaison de $m1$ et $m2 \in \mathbb{Z}$
{retourne une valeur négative si $m1 < m2$, nulle si $m1 = m2$, positive sinon.}

soit $cmp \subset \mathbb{Z}$
$cmp \leftarrow CompareBytesSansAccentNiCasse(getGraphie(m1), getGraphie(m2))$
{Si les graphies sont égales}
si $cmp = 0$ **alors**
  $cmp \leftarrow getMaj(m1) - getMaj(m2)$
  {Si les casses sont identiques}
  si $cmp = 0$ **alors**
    $cmp \leftarrow CompareBytesSansAccentNiCasse(getCategorie(m1), getCategorie(m2))$
    {Si les catégories sont identiques}
    si $cmp = 0$ **alors**
      $cmp \leftarrow CompareBytesSansAccentNiCasse(getPhonie(m1), getPhonie(m2))$
      {Si les formes phonétisées sont identiques}
      si $cmp = 0$ **alors**
        $cmp \leftarrow CompareBytesSansAccentNiCasse(getLemme(m1), getLemme(m2))$
        {Si les lemmes sont identiques}
        si $cmp = 0$ **alors**
          {les clés sont totalement identiques : erreur dans le lexique ou identité de $m1$ et $m2$. On effectue alors une comparaison des graphies sensible à la casse et aux accents}
          Retourner $CompareBytes(getGraphie(m1), getGraphie(m2))$
        finsi
      finsi
    finsi
  finsi
finsi
{Sinon, quel que soit le cas, on retourne la valeur de comparaison}
Retourner *cmp*.

fréquenceur) qui parcourent des textes (de gauche à droite), pour retrouver les informations lexicales portant sur ces mots. Deux de ces recherches permettent d'envisager une optimisation de la dichotomie :

- On recherche un mot exact M en n'ayant jamais effectué cette recherche auparavant
- On recherche un mot exact M en ayant déjà effectué une recherche sur le début de ce mot
- On recherche tous les mots commençant par un mot M en n'ayant jamais effectué cette recherche auparavant
- On recherche tous les mots commençant par un mot M en ayant déjà effectué une recherche sur le début de M

Dans les second et quatrième cas, on dispose d'une réponse de recherche R contenant la solution recherchée, car l'ajout d'une lettre dans un mot diminue la taille de l'ensemble des solutions. Dans ces deux cas, on peut donc réutiliser les connaissances déjà obtenues sur le début du mot. La notion de réponse de recherche suppose une structure très simple contenant toutes les informations permettant de retrouver des éléments du lexique ainsi que le "mot" ayant déclenché cette recherche.

Algorithme 2 Algorithme de recherche dans le lexique

notons ReponseDeRecherche **le type construit** | indexDebut $\subset [1; taille(Lexique)]$
| indexFin $\subset [1; taille(Lexique)]$
| MotCherche $\subset$ chaîne

Entrées: | $Mot \subset$ chaîne
| $AncienneReponse \subset$ ReponseDeRecherche
| $Lexique \subset$ Dictionnaire
| $MotExact \subset [VRAI, FAUX]$

Sorties: retourne une réponse basée sur le mot cherché, un lexique donné et éventuellement une ancienne réponse, en fonction du type de recherche (exacte ou sur début de mot commun). la valeur VIDE est renvoyée si le mot est introuvable.

soient $Reponse \subset$ ReponseDeRecherche
$gauche, droite, milieu \subset \mathbb{N}$
$cmp \subset \mathbb{Z}$
{étape 1 : chercher une occurrence du mot dans le dico, par dichotomie}
{étape 1.1 : cherchons à réutiliser une ancienne recherche}
$Reponse.MotCherche \leftarrow Mot$
si $(AncienneReponse = VIDE ou AncienneReponse.MotCherche = VIDE)$ **alors**
  $Reponse.indexDebut \leftarrow 1$
  $Reponse.indexFin \leftarrow taille(Lexique)$
sinon
  $Reponse.indexDebut \leftarrow AncienneReponse.indexDebut$
  $Reponse.indexFin \leftarrow AncienneReponse.indexFin$
finsi
$gauche \leftarrow Reponse.indexDebut$;
$droite \leftarrow Reponse.indexFin$
{étape 1.2 : cherchons une occurrence du mot dans le lexique}
répéter
  $milieu \leftarrow (gauche + droite)/2$

```
    si ((milieu < 1)ou(milieu >= taille(Lexique))) alors
      retourner VIDE
    finsi
    cmp ← CompareBytesSansAccentNiCasse(Mot, getGraphie(milieu))
    si cmp > 0 alors
      si gauche = milieu alors
        gauche ← gauche + 1 ;
      sinon
        gauche ← milieu ;
      finsi
    sinon
      si cmp<0 alors
        si droite = milieu alors
          droite ← droite − 1 ;
        sinon
          droite ← milieu ;
        finsi
      finsi
    finsi
    si gauche > droite alors
      retourner VIDE
    finsi
  jusqu'à cmp = 0
  {étape 2 : à partir de l'occurrence trouvée, prolongeons notre recherche :}
  {étape 2.1 : à gauche de l'occurrence trouvée}
  pour gauche = milieu avec gauche >= 1 par gauche ← gauche − 1 faire
    si CompareBytesSansAccentNiCasse(Mot, getGraphie(gauche)) <> 0 alors
      Sortir de cette boucle
    finsi
  fin pour
  {étape 2.2 : repositionner l'index au dernier bon element}
  si gauche < milieu alors
    gauche ← gauche + 1
  finsi
  {étape 2.3 : à droite de l'occurrence trouvée}
  pour droite = milieu avec droite <= taille(Lexique) par droite ← droite + 1 faire
    si CompareBytesSansAccentNiCasse(Mot, getGraphie(droite)) <> 0 alors
      Sortir de cette boucle
    finsi
  fin pour
  {étape 2.4 : repositionner l'index au dernier bon element}
  si droite > milieu alors
    droite ← droite − 1
  finsi
  {étape 3 : retourner le résultat}
  Reponse.indexDebut ← gauche
  Reponse.indexFin ← droite
```

retourner Reponse

Lorsque les interrogations du lexique portent sur des critères plus fins (sensibilité à la casse, aux accents et à la majuscule en début de phrase), on utilise une structure itérative basée sur la réponse de recherche définie dans l'algorithme 2. Cette structure dynamique *ReponseMasquee* fournit une méthode d'accès aux éléments d'une *ReponseDeRecherche* correspondant à des filtres. Elle occupe peu de mémoire et fonctionne rapidement par vérification des filtres sur les éléments d'une réponse. L'exemple de la figure 30 montre les différences entre les appels à la fonction Suivant() en fonction de la sensibilité du filtre aux majuscules.

A présent, nous allons nous intéresser aux algorithmes spécifiques de recherche d'information dans le lexique, qui sont accessibles depuis la visionneuse de dictionnaires.

Avant de présenter cette visionneuse, nous souhaitons montrer la première interface qui a été développée dans le but de tester le lexique et les fonctions d'accès, de tri et de recherche qui lui sont associées via le module Dictionnaire. Cette première interface fonctionnait sur console/shell en accès par ligne de commande.

Ce petit programme permettait d'effectuer des recherches classiques par mot dans un lexique.

Il permettait en outre d'utiliser et de tester la bibliothèque de fonctions de comparaison de mots (en tenant ou non compte de la casse par exemple).

La première capture de ce programme, ci-dessous, montre le menu principal après chargement.

```
D:\dossiers\developpement\LPLSuite\Binaires>java JTAL.Dictionnaire.DicoTest DicoLPL.zip
*** Chargement du dictionnaire: ... .................Nom: DicoLPL-12oct04.txt
...Uncompressed Size: 23773300
.....Compressed Size: 3328905
.................CRC: 1347334471
.............Comment: null

> Nombre de lignes detectees = 444266 lignes
> Estimation du nombre d'index champs= 4886926 champs
> Estimation de la taille de l'index lignes= 1735 Ko
> Estimation de la taille de l'index champs= 4772Ko
OK ***
------- D I C T I O N N A I R E ---------------page: 0 / 37022
 MOT    M       PHON    FREQ    CATEG   LEMME   SEME    V       VAR
--------------------------------------------------------------------------------

'en     ‘       &a~     124     Pp3n-d- en      ?       ...     ?
-ce     ‘       _s@     17663   Pp3n-j- ce      ?       ...     ?
-ci     ‘       _si     3972    Rgp     ci      ?       ...     ?
-elle   ‘       _El@    15217   Pp3fsni il      ?       ...     ?
-elle   ‘       _El@    1113    Pp3fso- lui     ?       ...     ?
-elles  ‘       _El@    5541    Pp3fpni il      ?       ...     ?
-elles  ‘       _El@    305     Pp3fpo- lui     ?       ...     ?
-en     ‘       _a~     6170    Pp3n-d- en      ?       ...     ?
-il     ‘       _il     61755   Pp3msni il      ?       ...     ?
-ils    ‘       _il     20648   Pp3mpni il      ?       ...     ?
-je     ‘       _Z@     2196    Pp1-sni il      ?       ...     ?
-la     ‘       _la     5167    Pp3fsj- le      ?       ...     ?
--------------------------------------------------------------------------------

   page (P)recedente
   page (+) avancer de 10 pages
   page (S)uivante
   page (-) reculer de 10 pages
   (I)nfos
   chercher (M)ot
   chercher (C)andidats commencant par...
   (T)ester deux mots
   (Q)uitter
   VOTRE CHOIX ?
choix>
```

Ce menu affiche des pages de douze entrées lexicales présentées en tableau tabulé et permet de changer de page, ainsi que d'effectuer quelques requêtes.

La capture ci-dessous montre le résultat d'une recherche stricte pour le mot *maison*

```
choix>
m
quel mot?>
maison
maison  ‘       mEzo~   31553   Ncfs--  maison  ?       ...     ?
Réponse:maison:264946,264946
```

La capture ci-dessous montre le résultat de la recherche de tous les mots commençant par *maison*

```
choix>
c
quel mot?>
maison
maison  ‘       mEzo~   31553   Ncfs--  maison  ?       ...     ?
maisonnée       ‘       mEzOne  64      Ncfs--  maisonn?e       ?       ... ?
maisonnées      ‘       mEzOne  6       Ncfp--  maisonn?e       ?       ... ?
maisonnette     ‘       mEzOnEt@        57      Ncfs--  maisonnette     ?   ...     ?
maisonnettes    ‘       mEzOnEt@        81      Ncfp--  maisonnette     ?   ...     ?
maisons ‘       mEzo~   8435    Ncfp--  maison  ?       ...     ?
Maisons-Neuves  a       mEzo~_n9v@      1       Np-s--  Maisons-Neuves  ?   ...     ?
Réponse:maison:264946,264952
```

La capture ci-dessous montre enfin la comparaison de deux mots (ou groupes de mots) selon une batterie de tests permettant de vérifier la fiabilité de la bibliothèque de fonctions de comparaison.

```
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse]:
--------------------------------------------------------------------------------
byte [] S1=Être ou n’être làn̈d;
byte [] S2=etre ou n’etre land;
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      -1      0
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       1       0
byte [] S1=Être ou n’être làn̈d;
byte [] S2=Etre ou n’etre land;
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1       -1     0       0
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       0       0
byte [] S1=Être ou n’être làn̈d;
byte [] S2=être ou n’être land;
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      -1      0
ByteTools.CompareBytes[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       1       0
```

```
--------------------------------------------------------------------------------
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse]:
--------------------------------------------------------------------------------
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=l'etre et le nean;
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      -1      1
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       1       0
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=L'ETRE ET LE NEAN;
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      1       1
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       0       0
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=l'être et le néan;
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      1       -1      1
ByteTools.EstAuDebut[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       0       1        0


--------------------------------------------------------------------------------
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse]:
--------------------------------------------------------------------------------
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=l'etre et le nean;
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      -1      0
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       1       0
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=L'ETRE ET LE NEAN;
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      -1      0       0
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       1       0       0
byte [] S1=L'ÊTRE ET LE NÉANT;
byte [] S2=l'être et le néan;
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S1,S2);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    -1      0       -1      0
ByteTools.DebutMotsEgaux[SansAccent/SansCasse/SansAccentNiCasse](S2,S1);
    normal  Minusc  Dessac  Min+Des
    ------  ------  ------  -------
    1       0       1       0
```

Cette batterie de tests met en évidence les valeurs retournées par nos fonctions de comparaison de mots. Ces fonctions ont été définies pour comparer les caractères composant des mots représentés sous formes de tableaux d'octets (tableaux de Byte en JAVA).

Un premier jeu de quatre fonctions (CompareBytes, CompareBytesSansAccent, CompareBytesSansCasse et CompareBytesSansAccentNiCasse) indique si deux mots sont égaux à la casse ou à l'accentuation près.

Un second jeu de quatre fonctions (EstAuDebut, EstAuDebutSansAccent, EstAuDebutSansCasse et EstAuDebutSansAccentNiCasse) indique si le premier mot est au début du second à la casse ou à l'accentuation près.

Un troisième jeu de quatre fonctions (DebutMotsEgaux, DebutMotsEgauxSansAccent, DebutMotsEgauxSansCasse et DebutMotsEgauxSansAccentNiCasse) indique si le premier mot commence le second (ou réciproquement), à la casse ou à l'accentuation près.

Ces trois ensembles de fonctions servent les outils de plus haut niveau qui accèdent au lexique.

La version actuelle de la visionneuse de dictionnaire est vraiment destinée à effectuer la maintenance de lexique. Elle offre toutes les fonctionnalités d'un tableur associées à celles d'un moteur d'interrogation basé sur les expressions régulières. Enfin, un système de requêtes élaborées permet de rechercher des sous-ensembles de lexique correspondant à plusieurs critères.

Les deux captures 31 et 32 montrent respectivement l'écran de chargement de la visionneuse (où l'on peut suivre la progression du hachage d'un lexique) et l'écran principal de navigation dans le lexique.

Fig. 31 – DicoViewer : fenêtre de chargement

Les figures 33, 34 et 35 donnent un aperçu de l'outil de visualisation de dictionnaires, de ses menus et fonctionnalités. Celui-ci permet de parcourir un lexique et d'effectuer des recherches sur l'ensemble de ses champs à l'aide d'expressions régulières. Les résultats de la recherche peuvent être exportés dans un fichier au format texte.
Son but est de permettre un enrichissement manuel de la ressource, de croiser des requêtes complexes et de mettre en évidence des particularités.
Dans ce but, des algorithmes spécifiques ont été développés, notamment pour permettre des recherches complexes et rapides. Ces algorithmes ne sont pas accessibles aux modules de plus haut niveau (Segmenteur et Etiqueteur), car un hachage préalable (onéreux en temps et en mémoire) empêche leur usage au cours d'analyses syntaxiques. Le chapitre suivant, qui présente des éléments algorithmiques, propose donc deux techniques de recherche, complètement différentes, en fonction de l'usage et de la rapidité attendus.

L'idée est cette fois de permettre un filtrage pertinent des données, grâce à des requêtes basées sur des expressions régulières. Comme dans une base de données, il est possible d'interroger le lexique selon des critères assez fins, puis d'affiner une requête, ou encore de croiser deux requêtes.

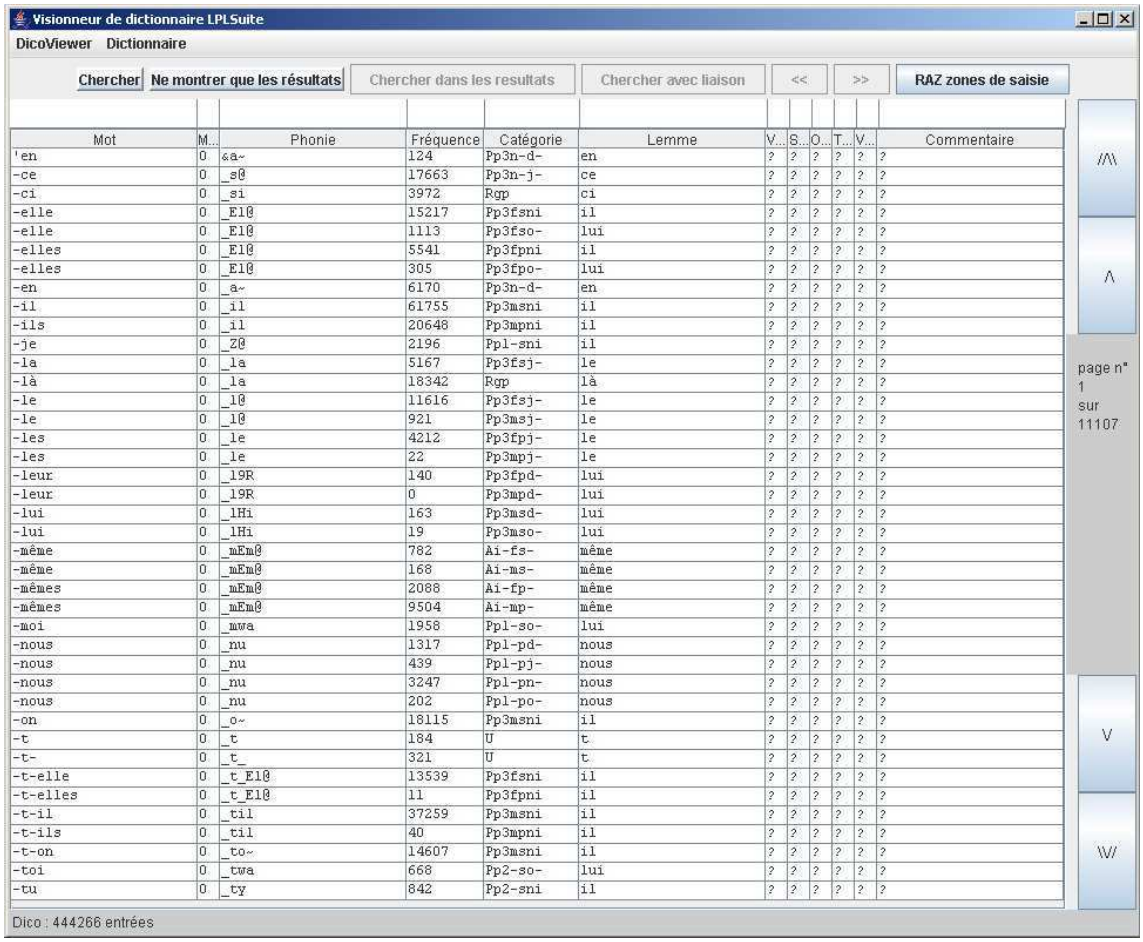


FIG. 32 – DicoViewer : page principale de l'outil

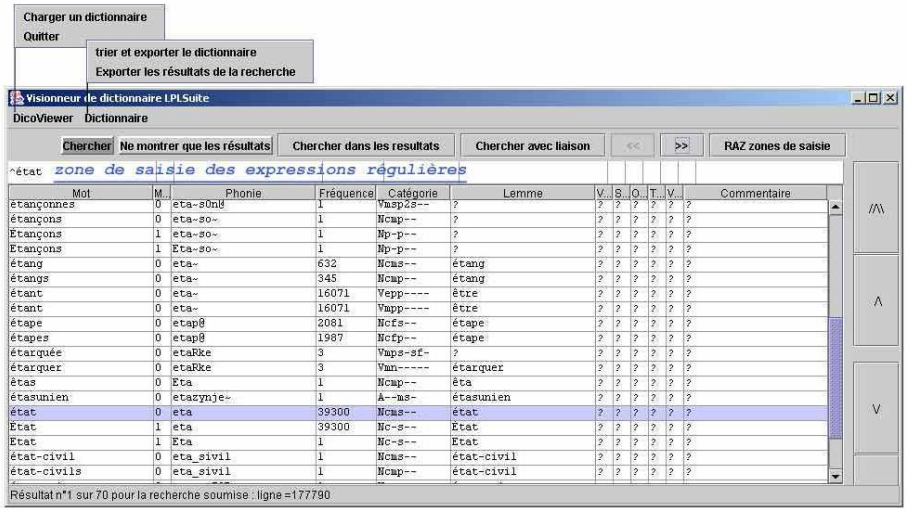


FIG. 33 – DicoViewer : l'outil de visualisation et d'interrogation du Dictionnaire

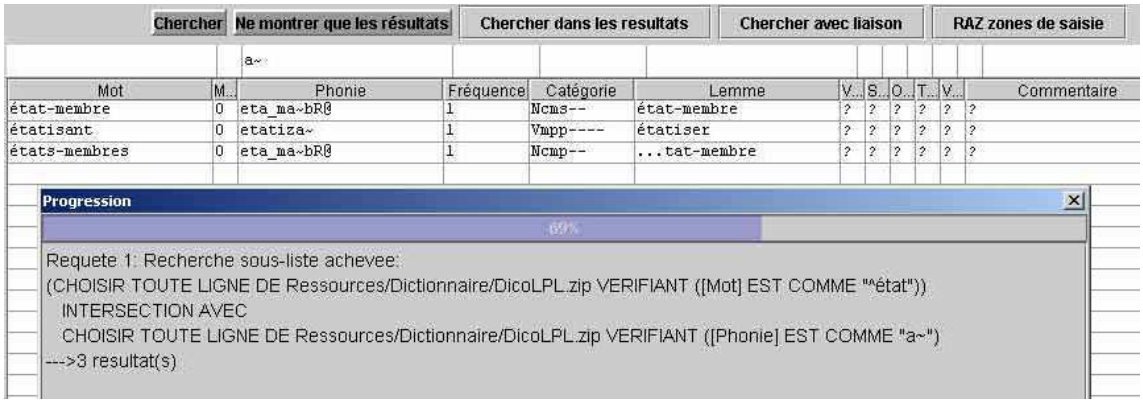


FIG. 34 – DicoViewer : recherche parmi les résultats d'une recherche précédente

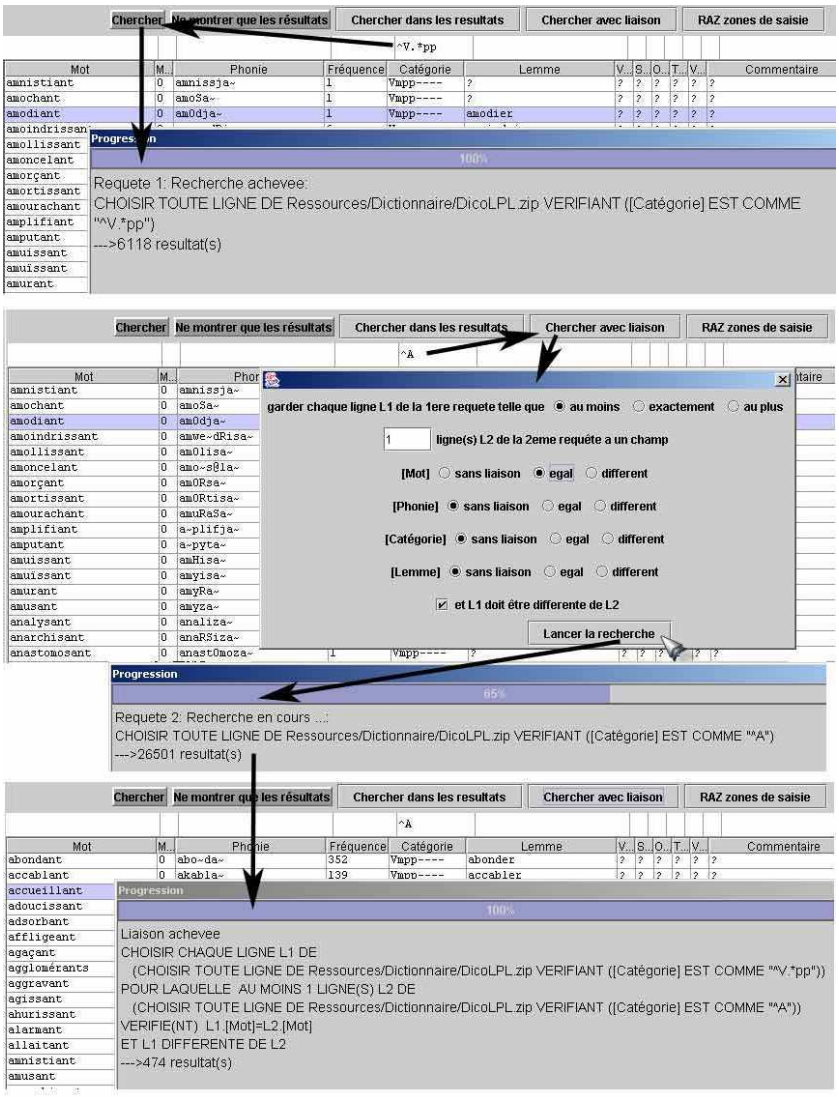


FIG. 35 – DicoViewer : recherche avec liaison

Exécuter une requête dans le lexique s'effectue depuis la visionneuse en saisissant d'abord des expressions régulières dans la zone de saisie. On déclenche la recherche en appuyant sur le bouton chercher. Les résultats de la recherche sont mémorisés tant qu'une nouvelle requête n'est pas déclenchée. Il est alors possible d'exporter les résultats de la recherche, ou encore de n'afficher qu'eux dans la table de visualisation. Pour fonctionner rapidement, l'algorithme de recherche ne peut pas s'appuyer sur un ordre de tri précis puisque cette fois les recherches peuvent s'effectuer sur n'importe quels champs du lexique. Nous devons donc parcourir le lexique itérativement, de haut en bas, sans connaissance a-priori sur son contenu. La recherche dans des chaînes de caractères est directement accessible en JAVA grâce à la bibliothèques d'objets REGEX, disponible à partir de la version 1.4 de JAVA. Nous utilisons les fonctions associées au package REGEX pour vérifier qu'un élément du lexique correspond avec une expression régulière. En ce qui concerne la mise en forme des résultats d'une recherche, nous avons mis en place un moteur de recherche basé sur les réductions d'intervalles d'entiers. Contrairement à la technique implantée directement au sein du module Dictionnaire, où un seul intervalle (que nous appelions "fourchette de recherche") correspondait avec un critère, nous devons à présent gérer la possibilité de construire des intervalles discontinus, dont la taille peut être assez proche de celle du lexique. Plutôt que de stocker dans un tableau dynamique la totalité des numéros de ligne des entrées du lexique répondant à un critère précis, nous ne mémorisons dans notre résultat qu'un ensemble de couples *(début,fin)* correspondant à l'ensemble des intervalles répondant à notre recherche. D'autre part, cette technique est rapide et du fait de sa mathématisation, elle supporte parfaitement des opérations telles que l'union, l'intersection et la disjonction d'intervalles. Avec la technique de réduction d'intervalles discontinus, l'action de *chercher dans le lexique* se révèle identique à l'action de *construire un ensemble d'intervalles des parties de lexiques correspondant avec les critères de recherche.*

La table 13 ci-dessous illustre le principe de réduction d'intervalles discontinus : l'intersection et l'union sont les opérations de base applicables aux objets IntervalleComposite. Nous proposons dans cet exemple deux intervalles I1 et I2, et nous montrons le résultat des opérations d'intersection et d'union effectuées sur ces deux intervalles. Les valeurs des variables taille et longueur associées à ces objets évoluent en fonction des opérations qui leur sont appliquées.

intervalle			taille	longueur
$I1$	=	$[1] \cup [3;4] \cup [7] \cup [10;11]$	4	6
$I2$	=	$[0;2] \cup [4;8]$	2	8
$I1 \cap I2$	=	$[1] \cup [4] \cup [7]$	3	3
$I1 \cup I2$	=	$[0;8] \cup [10;11]$	2	11

TAB. 13 – Intersection et union d'intervalles composites

L'exemple de la table 14 donné ci-dessous montre sur un lexique imaginaire doté d'un seul champ, l'intervalle construit à partir de la recherche de tous les mots contenant un ou plusieurs "a" consécutifs (dont l'expression régulière se note "a*").

Les objets développés pour représenter les intervalles continus obéissent aux mêmes impératifs de rapidité et d'occupation de la mémoire que les autres outils. Nous définissons dans la table 15 ci-dessous les deux objets Java Intervalle et IntervalleComposite qui permettent de représenter respectivement "un intervalle entre deux entiers positifs" et "un ensemble d'intervalles simples non contigus et n'ayant aucune intersection".

Avec ces objets, nous avons mis en œuvre quatre types de recherche accessibles depuis la visionneuse :

numéro de ligne	mot	correspondance avec "a*"	évolution du résultat
1	ABCD	non	Resultat=[]
2	abcd	oui	Resultat=[2]
3	dcbabcd	oui	Resultat=[2, 3]
4	bab	oui	Resultat=[2, 4]
5	bcde	non	Resultat=[2, 4]
6	zaaz	oui	Resultat=[2, 4] ∪ [6]
7	zza	oui	Resultat=[2, 4] ∪ [6, 7]
8	zzz	non	Resultat=[2, 4] ∪ [6, 7]
etc			

TAB. 14 – Exemple de recherche par réduction d'intervalles

Méthode	Description
Intervalle(int a, int b)	construit un intervalle entre a et b
Intervalle()	construit un intervalle vide
int getMin()	fournit le minimum de l'intervalle
void setMin(int newMin)	change le minimum de l'intervalle
int getMax()	fournit le maximum de l'intervalle
void setMax(int newMax)	change le maximum de l'intervalle
boolean contient(int x)	teste l'appartenance de x à l'intervalle
boolean contientIntervalle(Intervalle intervalle)	teste l'appartenance d'un intervalle à l'intervalle
boolean coupeIntervalle(Intervalle intervalle)	teste l'intersection de deux intervalles
boolean egale(Intervalle intervalle)	teste l'égalite à l'intervalle
boolean precede(Intervalle intervalle)	teste la précédence à l'intervalle
boolean suit(Intervalle intervalle)	teste la succession à l'intervalle
IntervalleComposite()	construit un intervalle composite vide
boolean contient(int x)	teste l'appartenance de x à l'intervalle
void ajoute(Intervalle i)	ajoute un intervalle à l'intervalle (en réduisant éventuellement les intersections)
void ajoute(int a)	ajoute un singleton à l'intervalle (en réduisant éventuellement les intersections)
void ajoute(int a,int b)	ajoute un intervalle borné à l'intervalle (en réduisant éventuellement les intersections)
long getLongueur()	donne le nombre de singletons de l'ensemble des intervalles
int getTaille()	donne le nombre d'intervalles distincts
int get(int i)	donne le singleton numéro i
void clear()	vide l'intervalle
IntervalleComposite intersectionAvec(IntervalleComposite ic)	fournit l'intersection de deux intervalles composites
IntervalleComposite unionAvec(IntervalleComposite ic)	fournit l'union de deux intervalles composites

TAB. 15 – Méthodes publiques des objets Intervalle et IntervalleComposite

- la recherche simple : à partir des critères (expressions régulières) saisis par l'utilisateur, nous générons un intervalle composite $I1$ initialement vide, auquel nous ajoutons tous les numéros de ligne du lexique ayant une correspondance réussie avec la requête. L'exécution de cette requête suppose le parcours intégral du lexique.
- la recherche affinée : à partir d'une requête déjà effectuée dont les résultats sont stockés dans un intervalle $I0$, il est possible de rechercher un sous-intervalle correspondant à de nouveaux critères. Dans ce cas, on procède comme pour la recherche simple, à la différence près que l'ensemble des index du lexique à parcourir se limite à ceux appartenant à l'intervalle $I0$. On obtient ainsi un intervalle $I1$ qui répond aux critères de raffinement.
- la recherche avec liaison : l'utilisateur prépare une première requête dont les résultats sont stockés dans $I1$ et une seconde requête dont les résultats sont stockés dans $I2$. Ensuite, l'utilisateur choisit l'opération de liaison, qui permet de générer une troisième requête vérifiant des critères de correspondance champ par champ. La réponse $I3$ ainsi obtenue pourra contenir des éléments appartenant à $I1$ tels qu'il existe au moins/au plus/exactement n éléments de $I2$ ayant un champ donné égal/différent. Ce type de requête complexe permet d'isoler assez aisément les homophones hétérographes, les homographes hétérophones, les verbes qui ont aussi une forme othographiée identique adjectivale etc. Ce type de requête peut être assimilé à une requête de base de données dans

laquelle on effectue une jointure entre deux tables sur plusieurs critères simultanément. L'idée est d'offrir à l'utilisateur les moyens de répondre à des questions très complexes.
- la recherche avec union : l'utilisateur prépare une première requête dont les résultats sont stockés dans $I1$ et une seconde requête dont les résultats sont stockés dans $I2$. Ensuite, l'utilisateur choisit l'opération d'union et une requête est aussitôt générée de sorte que l'union des intervalles $I1$ et $I2$ soit stockée dans $I3$.

Avec ces type de requêtes, il a été possible d'enrichir rapidement et manuellement le lexique, en isolant des phénomènes particuliers.

L'enrichissement de ce lexique passe aussi par l'utilisation d'outils de traitement automatique. Leur présentation est faite ci-dessous, ce qui nous permettra de montrer ensuite comment ils sont employés à l'amélioration et au calcul des données du lexique (notamment en ce qui concerne la fréquence et la phonétisation). Nous verrons aussi à la fin de cette partie comment cette ressource peut être employée avec les outils de fréquençage pour en extraire des sous-lexiques particuliers.

Les objets que nous avons présentés ici servent aux outils de plus haut niveau et permettent la conceptions d'applications rapides malgré le volume des données chargées. Le format interne du lexique est une table binaire rapidement accessible et triable, mais il est difficile d'y insérer une ligne ou d'en supprimer une. La maintenance du dictionnaire peut être effectuée directement dans la version textuelle du lexique, ou enfin à l'aide d'une base de données permettant des requêtes particulières sur des champs quelconques. Les améliorations futures porteront sur la qualité et la précision des catégories morpho-syntaxiques, sur l'ajout des valences verbales, la constitution de jeux de traits sémantiques, la modulation des fréquences en fonction de l'étiquette plutôt qu'en fonction de la graphie. Un dictionnaire n'est jamais achevé. Celui-ci continue d'être enrichi avec les travaux sur corpus et à l'aide de ressources libres.

Chapitre 6

Segmenteur

Le premier niveau de découpage d'un texte est automatisé par le programme Segmenteur qui repère les ponctuations, les nombres, les mots simples et les mots composés, qu'ils appartiennent ou non au dictionnaire. Ce module est interrogeable par la ligne de commande ou depuis les modules de plus haut niveau de LPLSuite. La figure 36 illustre son positionnement parmi les autres outils de la suite.
Il est important de poser la rapidité de segmentation comme critère principal pour la conception de ce programme. C'est pourquoi l'automate d'analyse que nous présentons ci-dessous est aussi complexe : il faut lire chaque caractère du fichier traité une et une seule fois, tout en construisant les objets résultant de l'analyse. Le programme *Segmenteur* s'appuie sur la classe accessoire JAVA *Segment* et sur la classe *Dictionnaire* accessible selon les modalités énumérées précédemment. La classe *Segment* définit les constantes de type et les méthodes d'accès à un mot (mot simple, composé, nombre, ponctuation etc.). Le texte source est lu, analysé et transformé en une liste dynamique d'objets de type *Segment*. Pour effectuer cette analyse, le dictionnaire est régulièrement consulté afin de vérifier l'existence d'un mot, d'un mot composé etc. Tous les mots, y compris ceux qui n'appartiennent pas au lexique, subissent une analyse basée sur un automate de découpage. L'accès au lexique est donc un complément d'information utile mais facultatif dans le processus de découpage : celui-ci permet de marquer les mots connus, de repérer les fautes éventuelles (majuscules et accents par exemple).

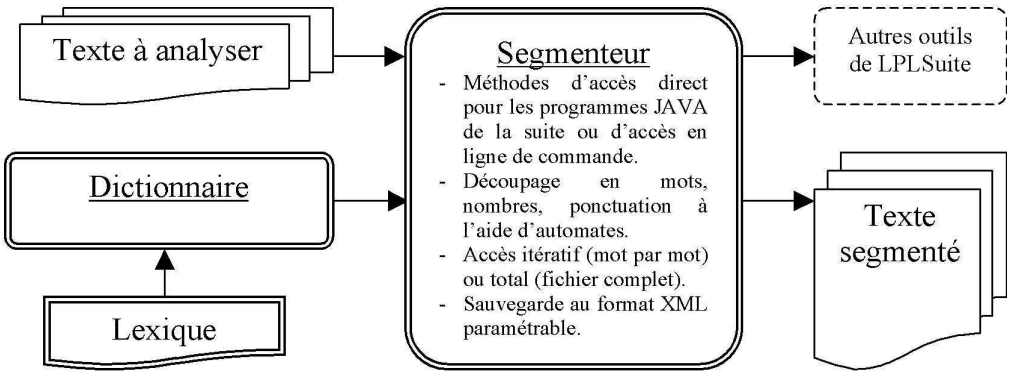


Fig. 36 – Vue d'ensemble du module Segmenteur

Les points d'accès au segmenteur se résument à la liste ci-dessous.
– **Mode console en ligne de commande :**

la commande
`java Segmenteur -D(ictionnaire) FichierDico.zip -F(ichier source) src.txt -S(ortie) sortie.segt.txt [-O(ptions)]`
provoque la segmentation du texte source avec la ressource dictionnaire donnée en paramètre. Les options permettent de sélectionner l'organisation des informations produites : quels champs du lexique doivent apparaître dans la sortie, quel formatage pour la sortie (un mot par ligne ou bien un espace entre chaque token) etc.

– **Mode indirect séquentiel par accès au constructeur :**
la construction
`Segmenteur seg= new Segmenteur ( Dictionnaire dico, BufferedReader org, boolean silencieux, boolean ligneParLigne)`
déclenche l'analyse d'un texte déjà ouvert via un objet BufferedReader, avec un dictionnaire déjà ouvert comme ressource d'appui. Ce type de construction est utilisé par des programmes de plus haut niveau (l'étiqueteur par exemple), pour une interrogation séquentielle de la liste des segments produits. Dans cette constructions, aucun fichier sortie n'est produit. Les segments produits doivent ensuite être demandés au Segmenteur via la méthode ProchainSegment(...) décrite ci-dessous.

– **Mode indirect complet par accès au constructeur :**
la construction
`Segmenteur seg= new Segmenteur ( String dicopath, String org, String dst, boolean silencieux, boolean ligneParLigne)`
déclenche l'analyse d'un texte contenu dans le fichier source "org", en ouvrant préalablement un dictionnaire nommé "dicopath" comme ressource d'appui. Ce type de construction est utilisé par des programmes de plus haut niveau pour une segmentation complète du texte source vers un fichier destination "dst".

Méthode	Description
public Segmenteur(Dictionnaire dico, BufferedReader org,boolean silencieux,boolean ligneParLigne)	Constructeur sur fichier source et dictionnaire déjà ouverts.
public Segmenteur(String dicopath, String org, String dst,boolean silencieux,boolean ligneParLigne)	Constructeur sur fichier source et dictionnaire pas encore ouverts, production d'un fichier sortie.
public Segment prochainMetaSegment()	Retourne le prochain segment disponible dans le texte source (voir ci-dessous pour plus de détails sur les Segments).

Tab. 16 – Méthodes publiques du module Segmenteur

Constante	Description
public static final int INDEFINI=0	type de segment indéfini
public static final int NOMBRE=1	type de segment numérique
public static final int MOT=2	segment reconnu comme mot du lexique
public static final int PONCTUATION=3	ponctuation
public static final int EOF=10	segment spécial de fin de fichier
Méthode	**Description**
public Segment (String mot,ReponseDeRecherche r,int type)	Constructeur avec initialisation.
public Segment(Segment s)	Constructeur par duplication.
public String getMot()	Accesseur pour la chaine de caractères constituant le segment.
public int getType()	Accesseur pour le type de segment.
public int getIndexDeb()	Accesseurs pour les numéros d'index dans le lexique lorsque le type de segment est "MOT"
public int getIndexFin()	
public boolean estPonctuationDure()	Répond vrai si le segment est une ponctuation de fin de phrase.

Tab. 17 – Méthodes et constantes publiques de l'objet Segment du module Segmenteur

Les objets fournis par le segmenteur à la suite de l'analyse d'un texte sont des *segments*.

Ceux-ci sont produits grâce à la méthode *ProchainMetaSegment()* qui déclenche l'automate d'analyse caractère par caractère. L'analyseur syntaxique qui produit les segments est un automate à pile classique, dans lequel la pile sert à éviter de perdre du temps à revenir en arrière dans le fichier source lorsqu'un mot supposé composé s'avère non composé : dans ce cas, en effet, seule la partie valide (dans le dictionnaire) d'un mot est considérée comme un segment valide. Le reste des caractères analysés mais non inclus dans le segment produit sont empilés afin de resservir pour l'analyse du segment suivant.

La table 18 ci-dessous montre le fonctionnement du segmenteur sur une phrase particulière, afin de mettre en évidence certaines de ses caractéristiques face à des mots composés connus ou inconnus, des nombres et des ponctuations. Ce module obéit à un ensemble de règles assez complexes pour la détection d'éléments composés.

Nous pouvons encore l'améliorer pour la reconnaissance de d'éléments tels que les acronymes, les noms propres composés etc. Sans une connaissance lexicale suffisamment large, le segmenteur se trouve mis à mal par des mots composés inconnus. Dans certains cas, il faudrait pratiquement une connaissance syntaxique de la phrase avant de trancher entre deux interprétations possibles. C'est le cas pour le token "en fait" qui peut tantôt être interprété comme un seul mot de catégorie adverbiale, tantôt comme la succession du pronom "en" suivi du verbe conjugué "fait". Ce seul exemple pourrait remettre en cause le principe d'une analyse morphosyntaxique ascendante, partant d'une désambiguisation lexicale, puis morphosyntaxique, puis syntaxique. En effet, beaucoup de modèles plaident pour une collaboration entre les niveaux d'interprétation, ne rendant possible la désambiguisation qu'au prix d'une confrontation entre les données de chaque niveau, sans hiérarchie préétablie. Nous pensons cependant que, si actuellement les modules de segmentation et d'étiquetage morphosyntaxique se trouvent isolés et contraints à une préséance des tâches, ceci n'est dû qu'à un choix opérationnel visant avant tout à implanter au seulement au niveau syntaxique le processus d'analyse guidé par les Grammaires de Propriétés. Ce choix a guidé notre travail et se situe comme d'autres travaux ayant eux aussi pour objet de mettre en œuvre une analyse syntaxique fondée sur des éléments pré désambiguisés. Nous verrons par la suite (dans la partie consacrée aux analyseurs syntaxiques) comment il serait possible de déployer le formalisme et les outils sur des éléments non-désambiguisés. C'est alors que la possibilité pour l'instant exclue d'effectuer une analyse sur des éléments non désambiguisése sera démontrée, sans travail supplémentaire, et ce du fait même du fonctionnement de l'analyseur basé sur les Grammaires de Propriétés.

Grâce à la S.N.C.F. j'irai dès que possible à Sylans-Châtillon-de-Michaille en dépit du qu' en-dira-t-on à l'égard des chemins de fer qui ne vaut pas grand-chose, ou à Buenos-Aires par l'avion aujourd' hui-même pour 4 572.54Fr. (avec 25.17% de réduction).

type	token	catégories possibles	commentaires
Mot	Grâce_à	Sp-	mot composé avec espace
Mot	la	Da-fs–d-, Ncm—, Pp3fsj-, Rgp	
Mot	S.N.C.F.	Npfss-	mot composé avec points
Mot	j'	Pp1-sn-	
Mot	irai	Vmif1s–	
Mot	dès_que	Cs	mot composé avec espace
Mot	possible	Afpfs-, Afpms-, Ncms–	
Mot	à	Ncfp–, Ncfs–, Ncmp–, Ncms–, Spa, Vaip3s–, Vmip3s–	
Inconnu	Sylans-Châtillon-de-Michaille		mot inconnu composé avec tirets
Mot	en_dépit_du	Sp-+Da-ms–dd	mot composé avec espaces
Mot	qu'en-dira-t-on	Ncmp–, Ncms–	mot composé avec apostrophe et tirets
Mot	à_l'égard_des	Sp-+Da-mp–dd	mot composé avec espace et apostrophe
Mot	chemins_de_fer	Ncmp–	mot composé avec espace
Mot	qui	Pr——, Pr-fp–, Pr-fs–, Pr-mp–, Pr-ms–, Pt——	
Mot	ne	Af-ms-, Rpn, Vmps-sm-	
Mot	vaut	Vmip3s–	
Mot	pas_grand-chose	Ncfp–, Ncfs–, Ncmp–, Ncms–, Pi-ms–	mot composé avec espace et tiret
Ponct	,		
Mot	ou	Cc, Pr——, Pr-fp–, Pr-fs–, Pr-mp–, Pr-ms–, Pt——, Rgp	
Mot	à	Ncfp–, Ncfs–, Ncmp–, Ncms–, Spa, Vaip3s–, Vmip3s–	
Mot	Buenos-Aires	Np-s–	mot composé avec tiret
Mot	par	Sp-	
Mot	l'	Da–s–d-, Pp3-sj-	
Mot	avion	Ncms–	
Mot	aujourd'hui	Rgp	mot composé avec espace
Mot	-même	Ai-fs-, Ai-ms-	
Mot	pour	Sp-	
Nombre	4572.54		nombre composé avec espace séparateur de milliers et point pour séparer les décimaux
Mot	Fr.	Ncfps-, Ncfss-, Ncmps-, Ncmss-	
Ponct	(		
Mot	avec	Sp-	
Nombre	25.17		nombre composé avec point pour séparer les décimaux
Ponct	%		
Mot	de	Ncms–, Spd	
Mot	réduction	Ncfs–	
Ponct	)		
Ponct	.		

TAB. 18 – Exemple de segmentation d'une phrase

Chapitre 7

Étiqueteur

Le second niveau de découpage d'un texte est automatisé par le programme Etiqueteur qui reçoit en entrée les segments fournis par le Segmenteur et qui sélectionne pour chacun d'eux le sous-ensemble de catégories morpho-syntaxiques qui les décrivent le mieux en fonction du contexte déjà analysé.

Phrase par phrase, un moteur de désambiguisation basé sur une resssource d'apprentissage et couplé à un moteur de concordance syntaxique, produit l'ensemble de catégories désambiguisées pour chaque segment. Ce module est interrogeable par la ligne de commande ou depuis les modules de plus haut niveau de LPLSuite. La figure 37 illustre son positionnement parmi les autres outils de la suite.

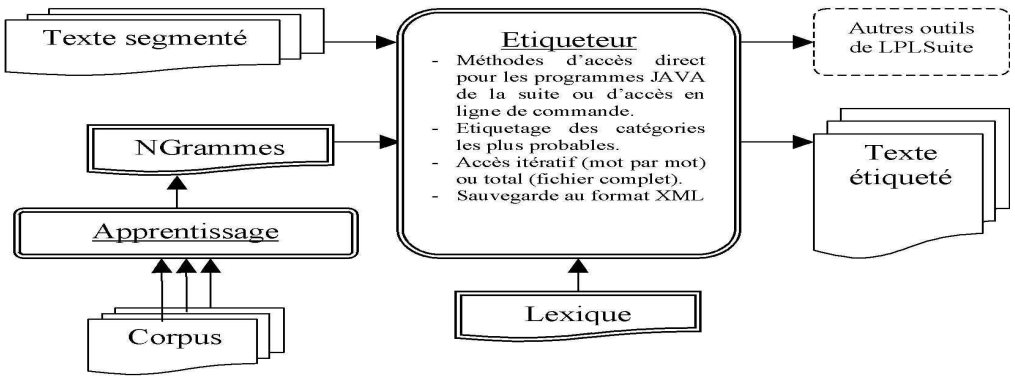


FIG. 37 – Vue d'ensemble du module Etiqueteur

Il est important de poser la rapidité de l'étiquetage comme critère principal pour la conception de ce programme. Notons aussi la possibilité offerte aux modules de plus haut niveau de ne conserver que les désambiguisations qui les intéressent ou de ne pas faire appel à la désambigüisation, pour éventuellement produire eux-mêmes cette désambiguisation *on line* durant l'analyse. Nous pourrons observer l'intérêt d'un tel choix lors de l'étude des analyseurs.

Le programme *Etiqueteur* fonctionne essentiellement autour des classes JAVA *Dictionnaire* (et son lexique) et *Segmenteur* , accessibles selon les modalités énumérées précédemment. Les précisions algorithmiques données plus loin éclaireront l'usage des objets et méthodes décrits ici : une phrase doit être intégralement désambiguïsée par l'étiqueteur pour pouvoir fournir à la demande les renseignements concernant chaque segment étiqueté. Les points d'accès à

l'étiqueteur se résument à la liste ci-dessous.

- **Mode console en ligne de commande :**
  La commande
  `java JTAL.Etiqueteur.Etiqueteur -d dico.zip -s syntaxe.zip -o suffixe [-souple|-strict(par defaut)] [-v (verbose)] Repertoire | Corpus.txt`
  déclenche l'étiquetage d'un texte ou d'un répertoire complet avec pour ressources un dictionnaire, un fichier "syntaxe" contenant les informations statistiques permettant la désambiguïsation. La commande permet de choisir le suffixe ajouté au nom des fichiers étiquetés. Enfin, il est possible de sélectionner un mode souple, plus tolérant aux fautes d'accent et de casse.
- **Mode indirect par accès au constructeur :**
  La construction
  `java etiqueteur=new Etiqueteur(String dicoPath, String syntaxePath, String srcePath, String suffixe, boolean strict, boolean séquentiel, boolean silencieux)`
  déclenche les mêmes calculs, depuis un programme de plus haut niveau (un analyseur par exemple). La différence par rapport à la ligne de commande réside dans la possibilité de réaliser séquentiellement l'étiquetage : si le mode séquentiel est choisi, une désambiguïsation phrase par phrase ou étiquette par étiquette est déclenchée au besoin du programme appelant, grâce aux méthode publiques prochainePhrase() et prochaineEtiquette().
  Il est encore possible d'appeler le constructeur suivant :
  `java etiqueteur=new Etiqueteur(Dictionnaire dico,String syntaxePath, String srcePath, String suffixe, boolean strict, boolean séquentiel, boolean silencieux)`
  qui permet le même traitement, mais en évitant de rouvrir un objet Dictionnaire lorsqu'un tel objet a déjà été chargé en mémoire.

Méthode	Description
Etiqueteur(Dictionnaire dico,String syntaxePath, String srcePath, String suffixe, boolean strict, boolean séquentiel, boolean silencieux)	Constructeur dictionnaire déjà ouvert
Etiqueteur(String dicoPath,String syntaxePath, String srcePath, String suffixe, boolean strict, boolean séquentiel, boolean silencieux)	Constructeur sur dictionnaire pas encore ouvert
public Phrase prochainePhrase()	Lorsque le mode séquentiel est enclenché, il faut appeler cette méthode pour provoquer l'étiquetage d'une phrase
public Etiquette prochaineEtiquette()	Lorsque le mode séquentiel est enclenché, il faut appeler cette méthode pour demander l'étiquetage d'un segment

TAB. 19 – Méthodes publiques du module Etiqueteur

Les méthodes publiques énumérées ci-dessus s'appuient sur des objets structurés qui intègrent l'objet Segment présenté plus haut : L'objet "Phrase" (correspondant à un énoncé), est un tableau d'objets "Etiquette". La notion de phrase ou d'énoncé reste ambiguë, car elle dépend de l'approche, de la présentation du corpus en entrée, ainsi que de critères arbitraires. C'est pourquoi l'objet Etiqueteur reste avant tout modifiable par les techniques habituelles d'héritage. La méthode ProchainePhrase pourra ainsi être réécrite ou surchargée en fonction des besoins de l'utilisateur. Un objet "Etiquette" est composé d'un objet "Segment", associé à une liste des catégories candidates à la désambiguïsation et un tableau des catégories choisies par l'étiqueteur. Le principe de l'étiquetage se ramène à la détermination pour chaque Etiquette d'une phrase, de ses catégories les plus probables en fonction de son contexte d'appa-

rition. Pour ce faire, les objets Phrase et Etiquette donnent accès aux constantes et méthodes suivantes :

Méthode	Description
public Phrase()	Le constructeur d'une phrase vide (vecteur d'Etiquette vide)
public void ajouteMot (Etiquette e)	Méthode d'ajout
public Vector lesEtiquettes()	méthode de récupération de la totalité du vecteur d'Etiquette
public int nbEtiquettes()	donne la quantité de mots étiquetés pour l'objet Phrase
public Etiquette getEtiquette(int index)	retourne l'objet Etiquette situé à la position index dans la Phrase

TAB. 20 – Méthodes et constantes publiques de l'objet Phrase du module Etiqueteur

L'objet phrase est vide initialement : c'est le processus de désambiguisation qui insérera les objets Etiquette dans un objet Phrase, au fil du parcours de texte.

Méthode	Description
public Etiquette(Segment s)	Constructeur qui associe à un Segment donné un vecteur des catégories candidates et un tableau des catégories choisies par l'étiqueteur (vides à la création)
public setListeCategoriesPotentielles(int[] liste)	méthode principale permettant au programme appelant de transmettre les catégories possibles pour cet objet
public int[] getListeCategoriesPotentielles()	méthode principale permettant au programme appelant de récupérer les catégories possibles pour cet objet
public int[] getListeCategoriesDesambiguisees()	méthode principale permettant au programme appelant de récupérer les catégories choisies par l'étiqueteur pour cet objet
public Segment getSegment()	cette méthode fournit un accès au Segment qui constitue l'Etiquette. On peut ainsi récupérer toutes les informations correspondant au mot analysé

TAB. 21 – Méthodes et constantes publiques de l'objet Etiquette du module Etiqueteur

Le tableau 22 donne un exemple d'étiquetage obtenu avec le module Etiqueteur. La désambiguisation se base sur les éléments algorithmiques exposés plus loin. Cet exemple fait apparaitre la désambiguisation statistique et celle basée sur la résolution d'accords. Nous constatons à l'usage que ce module a des qualités certaines, mais que des phénomènes mériteraient comme pour la segmentation de bénéficier d'une analyse de plus haut niveau, avant de procéder à la désambiguisation. Comme nous le disions pour le segmenteur, cette désambiguisation à postériori sera rendue possible par le programme d'analyse syntaxique. C'est pourquoi le module Etiqueteur est capable au besoin de proposer une désambiguisation mais de fournir quand même la liste complète des possibilités à l'analyseur syntaxique qui aura fait appel à lui.

La technique de désambiguïsation que nous utilisons emploie une ressource apprise : un corpus d'apprentissage aura été préalablement converti en cette ressource, sous forme d'un fichier contenant les probabilités d'occurrence des catégories en fonction de leur contexte. Ceci constitue en fin de compte une base de n-grammes (probabilité d'une catégorie en connaissant son contexte d'apparition, où un 0-gramme correspond à un contexte vide, un 1-gramme correspond à un contexte d'un mot, un 2-gramme ou bigramme correspond à un contexte de 2 mots, etc.). Nous mémorisons les n-grammes pour n variant de 0 à 5, en ne retenant que ceux qui se présentent plus de 500 fois dans un corpus de référence de 1 200 000 mots. La technique d'apprentissage que résume la figure 38 est assez classique et nous ne la développons pas ici.

Lorsqu'un étiquetage est déclenché, un Segmenteur est créé et s'appuie sur un Dictionnaire pour produire séquentiellement un objet Segment par element du texte à analyser. Chaque Segment donne lieu à la création d'un objet Etiquette. L'objet "Etiquette" est constitué d'un objet Segment, qui permet de retrouver toutes les informations associées à un mot (le type du Segment et si celui-ci est un mot les index du dictionnaire qui lui correspondent). A la création de l'objet Etiquette, celle-ci contient seulement le segment qui lui est transmis. Deux autres variables composent cet objet : une liste des catégories potentielles du Segment, et une

Au début de la dernière guerre, mes fonctions m' ont successivement appelé à différents postes, d'abord à la tête du 20e corps, et ce sont alors les opérations de Lorraine jusqu'à la fin d'août 1914.

token	type	catégorie choisie	catégories possibles	commentaires
Au	Mot	Sp-+Da-ms–da	Sp-+Da-ms–da	
début	Mot	Ncms–	Ncms–	
de	Mot	Spd	Spd	
la	Mot	Da-fs–d-	Da-fs–d-, Ncm—, Pp3fsj-	préférence statistique pour Spd+D face à Spd+Pp
dernière	Mot	Afpfs-	Afpfs-, Ncfs–	préférence pour Spd+D+A+N plutot que Spd+D+N+N
guerre	Mot	Ncfs–	Ncfs–	
,	Ponct	Wm		
mes	Mot	Ds1fp-s–	Ds1fp-s–, Ds1mp-s–	accord en genre avec "fonctions"
fonctions	Mot	Ncfp–	Ncfp–	
m'	Mot	Pp1-sj-	Pp1-sj-, Px1-s–	
ont	Mot	Vaip3p–	Vaip3p–, Vmip3p–	bonne détection de l'auxilliaire avoir Va+...+Vmps
successivement	Mot	Rgp	Rgp	
appelé	Mot	Vmps-sm-	Af-ms-, Vmps-sm-	préférence du verbe en présence de l'auxilliaire avoir
à	Mot	Spa	Spa	
différents	Mot	Di-mp—-	Afpmp-, Ai-mp-, Di-mp—-	préférence de Sp+D+N plutot que Sp+A+N ou Sp+...+V
postes	Mot	Ncmp–	Ncfp–, Ncmp–, Vmip2s–, Vmsp2s–	idem
,	Ponct	Wm		
d'abord	Mot	Rgp	Rgp	
à	Mot	Spa	Spa	
la	Mot	Da-fs–d-	Da-fs–d-, Ncm—, Pp3fsj-	
tête	Mot	Ncfs–	Ncfs–	
du	Mot	Sp-+Da-ms–dd	Sp-+Da-ms–dd	
20	Nombre	Dk-ms—-	ListeInterne	la "liste interne" est un ensemble de catégories désambiguisables pour les numéraux connues par l'Etiqueteur
e	Mot	Ncms–	Ncfp–, Ncfs–, Ncmp–, Ncms–	ici, le lexique fournit des catégories erronées ou il faudrait interpréter 20e comme un ordinal
corps	Mot	Ncms–	Ncmp–, Ncms–	
,	Ponct	Wm		
et	Mot	Cc	Cc	
ce	Mot	Pd-mp–	Dd-ms—-, Pd-mp–, Pd-n—	
sont	Mot	Veip3p–	Veip3p–, Vmip3p–	ici, erreur de détection : on a interprété "sont" comme auxilliaire au lieu de simple verbe conjugué
alors	Mot	Rgp	Rgp	
les	Mot	Da-fp–d-	Da-fp–d-, Da-mp–d-, Pp3fpj-, Pp3mpj-	préférence de D+Nc+Sp plutôt que Pp+Nc+Sp
opérations	Mot	Ncfp–	Ncfp–	
de	Mot	Spd	Spd	
Lorraine	Mot	Nc-s–	Nc-s–, Np-s–	ici, en absence de statistiques discriminantes, la première des propositions (Nc) est choisie
jusqu'à	Mot	Sp-	Sp-	
la	Mot	Da-fs–d-	Da-fs–d-, Ncm—, Pp3fsj-	préférence pour Sp+D plutôt que Sp+N ou Sp+Pp
fin	Mot	Ncfs–	Afpms-, Ncfs–, Rgp	préférence pour D+N+Spd plutôt que D+A+Spd ou D+R+Spd
d'	Mot	Spd	Spd	
août	Mot	Ncms–	Ncms–	
1914	Nombre	Ak-ms-	ListeInterne	
.	Ponct	Wd		

TAB. 22 – Exemple d'étiquetage d'une phrase extraite d'un corpus

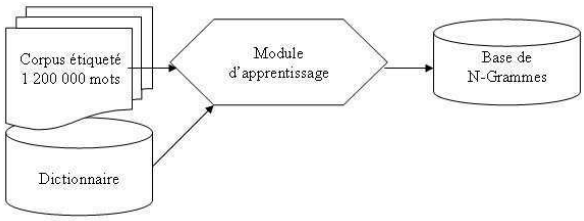


FIG. 38 – Principe de l'apprentissage des N-Grammes

liste des catégories désambiguisées pour ce Segment. Ces deux listes sont initialement vides. Lorsque l'étiquette est créée, le programme appelant effectue la recherche des catégories possibles pour le Segment concerné : la fonction "RechercherCategoriesPotentielles" utilisée dans l'algorithme décrit plus bas effectue les tâches suivantes : si le segment qui lui est transmis est un mot, alors les index contenus dans l'objet Segment permettent de retrouver la liste de ses catégories potentielles. Cette liste est recopiée dans le tableau "CategoriesPotentielles" de l'objet Etiquette. Si par contre l'objet est un nombre, il est possible d'assigner automatiquement une liste de catégories acceptables pour les nombres (numéraux cardinaux, etc.) au tableau "CategoriesPotentielles" de l'étiquette. Si le segment est de type ponctuatif, il faut transmettre des catégories potentielles particulières appartenant au jeu de traits utilisé par le désambiguiseur. Si enfin le segment est un mot inconnu, le programme qui crée l'étiquette peut proposer une liste assez large des catégories possibles pour un mot inconnu. Cette liste aura été préparée au préalable de la façon suivante : la liste des catégories de mots inconnus ne contient aucune catégorie dont la totalité des représentants est supposée connue dans le dictionnaire. Il n'y aura donc aucune conjonction, aucune catégorie des flexions du verbe avoir ou être etc., puisque la totalité de ces formes est répertoriée dans le dictionnaire. Cette recherche de la liste des catégories Potentielles constitue l'étape de prédésambiguisation, qui permettra au processus de désambiguisation de choisir dans cette liste une (ou éventuellement plusieurs) catégorie la plus probable. Lorsque tous les segments d'une phrase sont chargés et stockés dans des Etiquettes, un accès au fichier de NGrammes permet de charger les informations statistiques nécessaires à la désambiguïsation. L'algorithme 3 présenté ci-dessous donne les grandes lignes du processus de désambiguïsation.

La maximisation de la probabilité de chaque catégorie potentielle fait appel à un calcul assez coûteux en mémoire et en temps, puisqu'il s'agit de construire la liste de toutes les successions possibles de catégories pour l'ensemble de la phrase. Ainsi, une phrase de n mots ayant potentiellement chacun deux catégories, provoque la construction de 2^n listes différentes. Une première heuristique nous fait travailler sur des tronçons de phrase séparés par des ponctuations, ce qui réduit significativement l'espace de recherche. Une seconde optimisation consiste à ne considérer que des "fenêtres" de calcul dont la largeur ne dépasse pas celle du n-Gramme le plus grand disponible à partir du point de calcul. Enfin, nos évaluations ont montré qu'un découpage automatique en tronçons correspondant à des chunks délimités par des mots outils non ambigus et de petite taille ne faisait pas perdre de qualité à l'étiqueteur (voir la partie Evaluation).

Une partie du processus de désambiguisation est effectuée selon le jeu de traits des éléments parcourus, cette fois non pas à l'aide de n-grammes, mais à partir de données morpho-syntaxiques : l'accord en genre et en nombre permet en général de retrouver la bonne catégorie des pronoms, adjectifs, verbes et déterminants pour lesquels un doute aurait pu subsister après

Algorithme 3 Algorithme de désambiguisation (Etiqueteur)

Fonction ProchainePhrase

Entrées: *segmenteur* : *Segmenteur (un segmenteur déja ouvert sur le texte analysé)*
NGrammes : *FichierdeNGrammes*

Sorties: Retourne un objet *Phrase* (la liste mise à jour des catégories les plus probables pour chacun des éléments de la prochaine phrase du texte).

```
soient phrase : Phrase
       etiquette : Etiquette
       segment : Segment
phrase ← nouvellePhrase()
segment = segmenteur.ProchainMetaSegment()
tantque segment.getType() <> Segment.EOF faire
  etiquette ← nouvelleEtiquette(segment)
  RechercherCategoriesPotentielles(etiquette)
  phrase.ajouteMot(etiquette)
  si segment.estPonctuationDure() alors
    DesambiguisePhrase(phrase, NGrammes)
    retourner phrase
  finsi
  segment = segmenteur.ProchainMetaSegment()
fin tantque
```

Procedure DesambiguisePhrase

{cette procedure désambiguise une phrase. Les détails de l'algorithme sont trop volumineux pour être présentés ici; nous en proposons donc une forme résumée}

Entrées: *phrase* : *Phrase*
NGrammes : *FichierdeNGrammes*

Sorties: Calcule et met à jour les catégories désambiguisées de l'objet *phrase*.

```
soient phrase : Phrase
       etiquette : Etiquette
       i ⊂ Entiers
phrase ← nouvellePhrase()
pour i = 0 avec i < phrase.nbEtiquettes() par i ← i + 1 faire
  etiquette ← phrase.getEtiquette(i)
  {Ici, on calcule la probabilité d'occurence de chaque catégorie potentielle de l'étiquette, en
  fonction du contexte et en se servant du fichier NGrammes. Les catégories de probabilité
  maximale sont stockées dans le tableau etiquette.CategoriesDesambiguisees.}
fin pour
```

la désambiguisation statistique. Pour réaliser cette opération, on mémorise sous forme de pile les traits de genre et nombre rencontrés en contexte. Cette pile s'accroît et diminue en fonction des mots rencontrés. Lorsque deux mots sont confrontés en sommet de pile, ils doivent correspondre, ce qui fournit la plupart du temps une indication de désambiguisation. On peut alors les supprimer de la pile.

L'étiqueteur fonctionne sur la base d'un apprentissage, grâce au segmenteur et au lexique. Ces trois points d'appuis ont leurs propres faiblesses. L'étiqueteur fera d'autant moins d'erreurs que les modules sur lesquels il s'appuie seront améliorés, c'est pourquoi il faut envisager les liens qu'entretiennent tous ces programmes et leurs ressources sous l'angle de l'enrichissement.

Chapitre 8

Enrichissement des ressources

8.1 Fréquenceur

Afin d'enrichir le lexique et de calculer au besoin les fréquences lexicales spécifiques à un ensemble de corpus, nous avons développé un outil de fréquençage. Comme l'indique la figure 39, cet outil fait appel aux résultats de l'étiquetage pour en déduire les fréquences des entrées du lexique, par mot et par catégorie. A partir d'un lexique initial, étant donné un ensemble de textes, nous obtenons en sortie du fréquenceur un ensemble de résultats composés de la façon suivante :

- l'étiquetage (XML) correspondant aux textes donnés en entrée
- l'ensemble des fréquences de chaque couple *(mot, catégorie)*
- un lexique des mots inconnus, un lexique des noms-propres et une nouvelle version du lexique initial. Pour chacun de ces lexiques, la colonne *fréquence* est mise à jour.

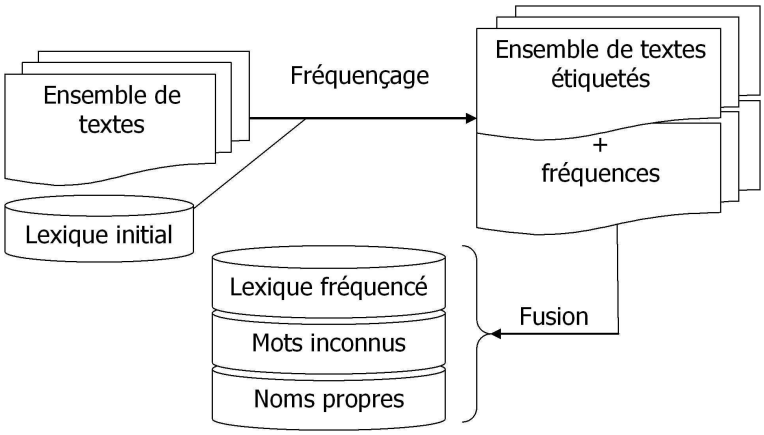


FIG. 39 – Modules de calcul des fréquences lexicales

Grâce à cet outil, il nous est possible de générer un lexique dont les fréquences seront spécifiques à un type de texte (journalistique, oral, médical, etc.). La version actuelle de *DicoLPL* dispose des fréquences acquises sur 153 millions de mots tirés du journal Le Monde, de ressources littéraires gratuites, de transcriptions de corpus oraux et des textes spécifiques

(domaine médical, corpus de mails etc.).

Lorsque les trois lexiques fréquencés sont obtenus à partir d'un lexique initial, nous disposons en outre d'une liste des mots inconnus et des noms propres, que nous pouvons trier par ordre de fréquence. Ceci autorise l'amélioration des entrées du lexique, mais nécessite un travail manuel assez fastidieux. A titre d'exemple, le fréquençage du corpus de la campagne EASY, doté d'un million de mots, a fait apparaitre environ 7000 formes différentes inconnues. Une grande part d'entre elles correspondaient à des *fautes* d'orthographe volontaires ou non situées dans la partie du corpus correspondant aux "mails". Dans cette partie là se retrouvent aussi beaucoup de mots abrégés, de néologismes, de transcriptions orthographiques d'onomatopées, d'adresses internet etc.. La plupart des autres mots inconnus étaient finalement très rarement des mots bien orthographiés, à l'exception de ceux tirés de la section "Medicale" du corpus. Un enrichissement manuel du lexique (voir figure 40) peut être largement soutenu par l'opération de fréquençage, qui prépare et ordonne les mots inconnus, et même propose (via l'étiqueteur) une catégorie potentielle assez probable pour ces mots. Pour l'évaluation de l'étiqueteur en fonction du type de mot (connu/inconnu), on se reportera à la partie traitant de l'évaluation des outils.

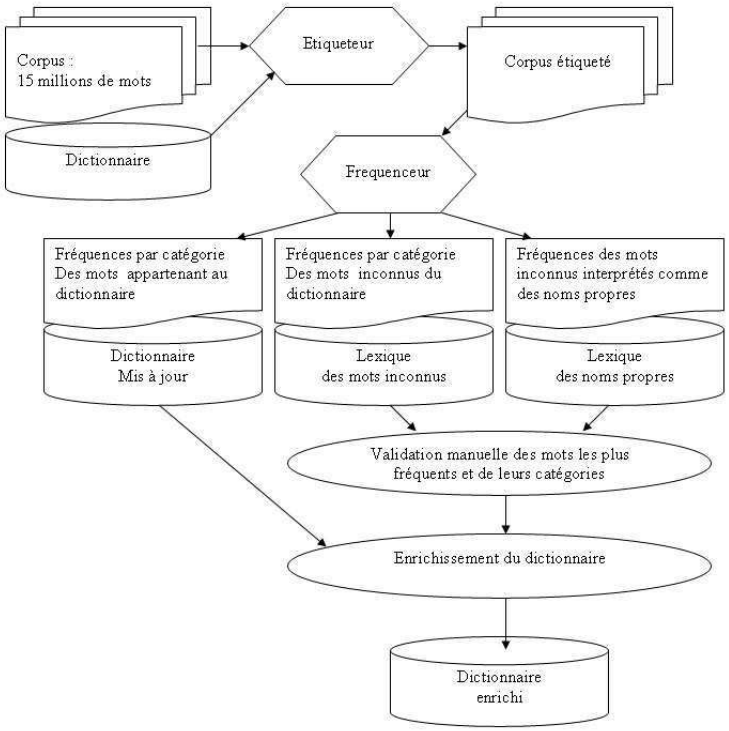


FIG. 40 – Principe de frequençage et d'enrichissement du dictionnaire

8.2 Mesure de couverture

Les outils et le lexique sont liés dans leur fonctionnement comme dans la qualité de leurs résultats. Ainsi, pour le même étiqueteur, deux versions du lexique donneront des résultats différents. Pour le même lexique, deux versions de l'étiqueteur donneront aussi des résultats

différents.

8.2.1 Mesure de couverture

Nous pouvons cependant mesurer la couverture du lexique pour un corpus donné. Il s'agit alors de calculer le quotient :

nombre de mots reconnus / nombre total de mots

La couverture actuelle du lexique représente 96% des corpus analysés (153 millions de mots).

8.2.2 Mesure de score

Lorsque nous souhaitons une information plus fine concernant l'étiquetage à l'aide de nos outils, il faut alors disposer d'un corpus de référence, pour lequel chaque mot est associé à une catégorie morphosyntaxique certifiée. Il est alors possible de mesurer pour chaque catégorie les scores de rappel et précision. C'est dans ce cadre que nous avons pu calculer un score de 95% sur le corpus Multitag. Cette question est développée dans la partie consacrée à l'évaluation.

Grâce à la mesure de couverture, il est possible de considérer la qualité d'un lexique pour un corpus donné. C'est ainsi que nous pourrons développer (de la façon exposée plus bas) des sous-lexiques spécifiques du Français, dédiés à des types de corpus particuliers.

8.3 Autres techniques d'enrichissement

L'amélioration du lexique ne dépend pas seulement des outils présentés ici. La forme phonétisée des entrées est obtenue grâce à un phonétiseur inspiré du projet *Syntaix* (cf. [Di Cristo, 1998]) pour la conception d'un système de synthèse vocale. D'autres outils spécifiques sont en cours de développement, pour la lemmatisation, l'extraction des traits de valence ainsi que le codage de traits sémantiques. Renseigner et corriger ces derniers champs nécessite encore un travail manuel très important. Il faut noter que cette ressource fondamentale pour le TALN est à la fois la base de calcul et et la cible des outils qui l'utilisent, dans la perspective de son enrichissement. Un *feedback* assez répétitif est associé à un filtrage manuel. Dans ce sens, nous ne voulons pas conclure que la ressource est achevée. Elle reste encore assez faible pour certains types de mots, comme les acronymes, les noms propres, etc.

Chapitre 9

Un lexique noyau du français contemporain

Le lexique une fois constitué, il est nécessaire de le valider, c'est à dire de vérifier sa couverture en analysant les résultats obtenus par le traitement de corpus décrit plus haut. Chaque entrée, en fonction de sa fréquence et de sa catégorie, peut ainsi être validée. Par ailleurs, l'analyse de ces mêmes corpus permet de fournir des indications pour la constitution d'un dictionnaire minimal (ou *lexique noyau*) du français ayant une couverture maximale. Un des avantages des lexiques noyaux se trouve dans leur légèreté, nécessaire pour un embarquement de ressources dans de petites plateformes dotées de peu de mémoire. La situation s'est présentée avec l'application PCA évoquée en cinquième partie. Une telle ressource est aussi d'une grande importance pour le futur. Il n'est en effet pas possible d'enrichir un très grand lexique manuellement. Or, nombre d'informations ne peuvent aujourd'hui être acquises totalement automatiquement, notamment les informations sémantiques. Un lexique noyau permet d'identifier un nombre limité d'entrées lexicales qu'il est possible d'enrichir y compris manuellement. L'objectif est à terme de disposer d'une ressource lexicale très complète, comportant des informations syntaxiques, sémantiques, voire pragmatiques. Un tel développement pourrait être réalisé dans un cadre coopératif, plusieurs équipes travaillant sur un même ensemble d'entrées. Nous présentons dans ce chapitre les bases de l'identification de ce lexique noyau.

9.1 10.000 formes couvrent 90% de l'usage écrit

La figure 41 illustre quelles sont les propriétés de couverture du français associées à notre lexique. Sur l'axe des abscisses, les formes ont été classés par fréquences d'occurrence décroissantes. La forme 1 correspond ainsi à la forme *de* (associée aux traits [Spd]), la forme apparaissant le plus fréquemment en français, suivie de la deuxième forme la plus fréquente *et*, etc. Sur l'axe des ordonnées est noté le pourcentage de couverture du français des n premières formes les plus fréquentes. Ainsi, les 10 formes les plus fréquentes composent en moyenne 21% des énoncés du français, les 10.000 formes les plus fréquentes composent en moyenne 90% des énoncés, etc.

Il est intéressant de noter que les 54.000 formes les plus fréquentes couvrent 99% du français et que seul 181.000 des 444.000 formes composant notre lexique ont été observées dans le corpus de 153 millions de mots analysés lors du fréquençage. Les formes peu usitées, au nombre de 263.000, soit 60% de notre lexique environ, sont en majorité composées de verbes

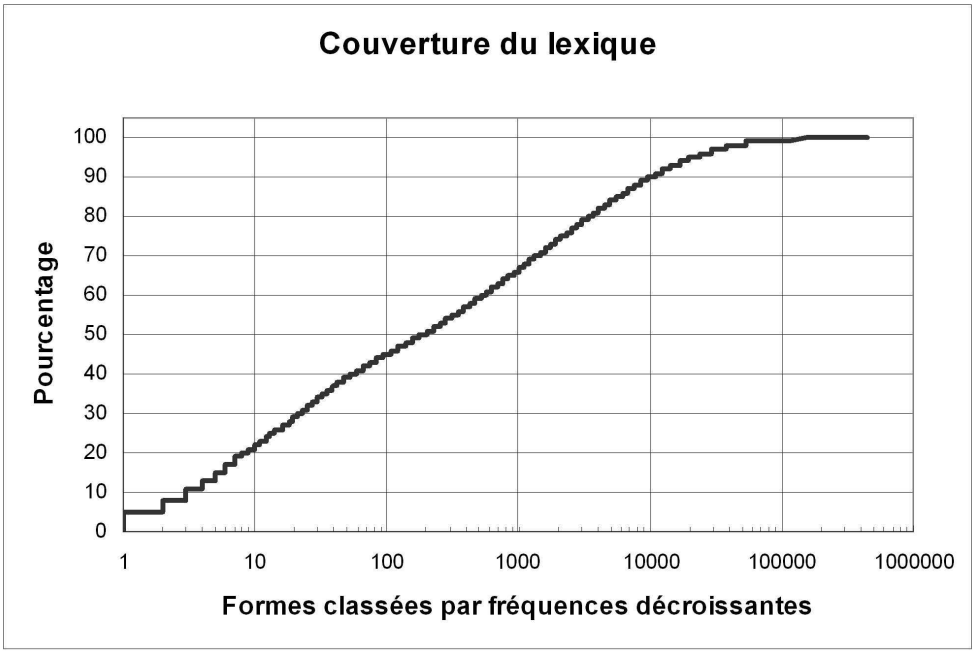


FIG. 41 – Couverture d'un corpus en fonction du nombre de formes du lexique (Echelle logarithmique).

(à 80%), de noms communs (à 15%) et d'adjectifs (à 5%). De plus, un lexique limité aux 10.000 formes les plus fréquentes couvrirait en moyenne 90% du français, ou autrement dit, pour des énoncés de 100 mots, 10 mots en moyenne seraient absents de ce lexique de 10.000 formes. Cette constatation nous conduit à proposer un lexique noyau du Français contemporain.

9.2 Techniques de sélection du lexique noyau

La qualité de l'information concernant la fréquence en français contemporain de chacune des entrées du lexique complet permet de concevoir très aisément et rapidement un lexique noyau (dorénavant LN) des mots les plus fréquents. Outre l'outil très maniable que représente un tel dictionnaire pour le TAL, c'est aussi l'occasion d'évaluer diachroniquement l'évolution du lexique de base du français depuis "Le Français Fondamental" (cf. [Gougenheim 1964]). Une telle étude est présentée dans [Blache, 2005].

Afin de sélectionner les formes pertinentes du LN, nous avons essayé deux méthodes différentes que nous avons évaluées selon la même procédure pour choisir la plus efficace. Puis nous avons construit, avec cette dernière, différentes versions de taille croissante du lexique noyau : de 15.000, 20.000 et 30.000 formes pour comparer leurs résultats de couverture et choisir le meilleur "rendement" (taille/couverture).

Les méthodes de sélection des formes pertinentes pour LN que nous avons évaluées sont les suivantes :

1. *Fréquence par catégorie (LN10cat)* : Cette méthode consiste à conserver les mêmes proportions entre grandes catégories (lexicales) que celles que l'on trouve dans le lexique

général à savoir : Verbes (V) 65%, Noms (N) 20,7%, Adjectifs (A) 13,64%, Adverbes (R) 0,5% ; en conservant l'intégralité des catégories grammaticales, beaucoup moins étendues : pronoms (P), prépositions (S), déterminants (D), conjonctions (C) occupant les O,16% du lexique restant avec quelques mots "inconnus" (U).

Les résultats concernant cette version du dictionnaire noyau correspondent à l'abréviation LN10cat (voir la figure 42).

2. *Fréquence seuil globale (LN10f)* : La seconde méthode, beaucoup plus élémentaire, consiste à sélectionner les 10.000 formes les plus fréquentes sans tenir compte de la répartition par catégories. Ne conserver que les formes dont la fréquence est supérieure à 1091 permet de sélectionner exactement 10000 entrées.

   Cette version du dictionnaire porte le nom suivant : LN10f.

Enfin, pour élaborer les lexiques de taille croissante, nous avons suivi la seconde méthode. Elle permet d'obtenir les lexiques LN15f (fréquence>613, 15.017 formes), LN20f (fréquence>389, 19.990 formes), LN30f (fréquence>193, 30.018 formes).

9.3 Comparaison des différentes versions du lexique noyau

Afin de comparer les différentes versions du LN présentées ci-dessus nous les avons soumises à un test de couverture (voir le chapitre 8.2.1) sur deux types différents de corpus : un corpus écrit et un corpus oral.

- Le corpus écrit compte 580.000 mots et regroupe un grand nombre d'articles de presse publiés dans le journal Le Monde.
- Le corpus oral comporte quant à lui 435.000 mots et regroupe le Bristol Corpus, un ensemble de 95 entretiens enregistrés et transcrits par Kate Beeching (1988-1990), ainsi que des corpus de parole recueillis au LPL.

La figure 42 donne les résultats de couverture pour ces différentes versions.

DicoNoyau	Corpus écrit (580.000 mots)	Corpus oral (435.000 mots)
LN10cat	88,16%	91,15%
LN10f (f>1091)	88,63%	91,56%
LN15f (f>613)	91,07%	93,60%
LN20f (f>389)	92,50%	94,60%
LN30f (f>193)	94,08%	96,46%
DicoLPL	96,21%	99,02%

FIG. 42 – Tableau des couvertures des différents dictionnaires noyaux testés sur deux corpus (oral et écrit)

Ceux-ci appellent plusieurs commentaires.

- Le premier concerne la dernière ligne du tableau qui évalue la couverture du lexique général DicoLPL : nous constatons que cette couverture n'est pas totale et qu'elle est meilleure pour le corpus oral que pour le corpus écrit, remarque qui vaut aussi pour les autres dictionnaires. Ceci s'explique selon nous par le fait que l'écrit utilise un vocabulaire beaucoup plus étendu et varié que l'oral. Les résultats de DicoLPL permettent d'évaluer les performances de couverture des autres versions de LN, moins étendues, par comparaison.

– On remarque aussi que la couverture de LN10f est meilleure que celle de LN10cat sur le corpus écrit comme sur le corpus oral. La méthode 2 qui consiste à sélectionner les formes pertinentes pour le dictionnaire noyau à partir d'une fréquence seuil globale est donc meilleure que la méthode 1 qui cherche à préserver les proportions relatives entre les catégories grammaticales. C'est pourquoi nous avons choisi la méthode 2 pour établir les dictionnaires LN15, 20 et 30f.
– On constate enfin que les performances de couverture s'améliorent régulièrement au fur et à mesure que le LN contient plus de formes, ce qui est bien sûr attendu. Il faut néanmoins noter qu'il existe un saut qualitatif plus important entre LN10f et LN15f qu'entre LN15f et LN20f ou LN20f et LN30f alors même que l'écart de taille entre ces deux derniers est plus important. Le dictionnaire noyau de 15000 formes apparaît donc comme la version optimale pour obtenir la plus grande couverture avec un nombre réduit de formes.

9.4 Conclusion

L'ensemble d'outils et de ressources présentés dans cette partie constitue une véritable plateforme de développement de lexique. Celle-ci répond à un certain nombre de besoins à la fois en termes de richesse d'informations, mais également de développements futurs. Les outils présentent l'avantage d'être modulaires et adaptables. le lexique peut évoluer et être régulièrement mis à jour. Nous pouvons également développer des lexiques spécialisés pour des applications particulières. Il peut être en effet nécessaire d'adapter une ressource à un corpus ou un type de texte donné, par exemple en produisant des fréquences spécifiques. Mais il est également nécessaire de continuer à enrichir le lexique en ajoutant de nouvelles informations. Notre approche permet de rationaliser le choix des entrées sur lesquelles travailler en proposant la construction d'un lexique noyau élaboré sur la base d'une véritable analyse de la langue. Les informations sémantiques, pragmatiques etc. sont en effet les plus difficiles à acquérir automatiquement. C'est pourquoi nous défendons la démarche qui consiste à concentrer les efforts sur un sous-lexique dont la couverture a été vérifiée sur corpus. Un lexique noyau qui offre l'avantage d'être contrôlable. D'autre part, un lexique de petite taille offre de nombreuses possibilités d'études sur l'usage.

Le fait de disposer d'un grand lexique de formes n'en reste pas moins un atout, puisque c'est à partir d'une telle ressource que peuvent être extraits des sous-lexiques *ad hoc* couvrant des types de texte de domaines divers : l'écrit littéraire, scientifique, journalistique, médical, l'oral transcrit, le *web* etc. ont des spécificités que le *fréquençage* permet d'isoler. A partir d'un lexique noyau, un enrichissement manuel particulier peut alors être envisagé au besoin.

Les outils présentés ici offrent de plus la possibilité d'être pilotés par des modules d'analyse syntaxique, afin d'extraire une information plus riche des texte analysés. C'est ce traitement qu'aborde la prochaine partie.

Troisième partie

L'analyse syntaxique avec les Grammaires de Propriétés

Cette partie prolonge la précédente, dans le sens où nous présentons ici encore des outils qui s'emboitent complètement dans l'ensemble LPLSuite. Nous avons cependant préféré accorder un traitement particulier aux questions qui entourent la conception d'un analyseur syntaxique. En effet, dans le cadre pratique développé jusqu'à présent, les choix opérationnels et théoriques avaient pour objectif de permettre une analyse de haut niveau, basée sur une reformulation de l'entrée textuelle sous forme de texte étiqueté, éventuellement désambiguisé. Les choix que nous allons faire à présent portent quant à eux sur l'analyse syntaxique elle-même. Il s'avère nécessaire de comparer différentes approches, de mettre en place le formalisme développé dans la première partie, puis de montrer comment ce formalisme peut donner lieu à la programmation d'un programme complet d'analyse syntaxique ayant à son tour les mêmes caractéristiques que celles requises pour les outils conçus précédemment, à savoir la modularité, la dissociation programme/ressource, la maintenabilité, la capacité d'être redéfini (surchargé au sens de la programmation orientée objet) etc. Dans un premier temps, nous allons mettre en place plusieurs spécifications d'analyseurs syntaxiques dont les caractéristiques devront répondre aux critères formels élaborés dans la première partie : souplesse, robustesse, mais autant que possible richesse et informativité. Nous montrerons ainsi comment concevoir un programme d'analyse susceptible de répondre aux questions posées autour de la notion de granularité variable. Après avoir défini ces spécifications, nous développerons la conception des programmes qui leur correspondent. Un analyseur superficiel (chunker), un analyseur superficiel plus riche (shallow parser), un analyseur profond basé sur les Grammaires de propriétés et enfin un analyseur à granularité variable basé lui aussi sur les Grammaires de Propriétés. Nous verrons en quoi ces programmes donnent des résultats différents et correspondent de plus en plus avec les critères énumérés précédemment (tant techniques que formels). Ceci permettra alors de continuer notre étude dans la partie suivante par une évaluation systématique des outils.

Voyons tout d'abord comment, à partir des notions développées dans la première partie, vont se dégager plusieurs approches qui se positionnent dans une même perspective de souplesse et de robustesse, avec cependant des caractéristiques de plus en plus riches.

Afin de permettre les évaluations et les comparaisons systématiques nécessaires à la validation des outils, nous avons conçu plusieurs programmes dédiés à la tâche d'analyse syntaxique.

Le premier est un analyseur très superficiel de parenthésage syntaxique (chunker) basé sur un algorithme d'automates à transitions d'état, construisant des structures plates (sans emboitement) et sans information catégorielle (pas d'information sur la catégorie des blocs parenthésés).

Le second un analyseur, plus riche, basé sur les Grammaires de Propriétés, mais faisant appel à une forme réduite et compilée de ces grammaires, dont la routine d'analyse procède elle aussi par parenthésage de blocs -éventuellement emboités- grâce à un algorithme basé sur la reconnaissance de coins gauches et de coins droits des syntagmes. Cet analyseur reste encore dans la famille des analyseurs à automates, puisque aucune contrainte n'est réellement vérifiée au cours de l'analyse.

Le troisième analyseur a été développé par Jean Marie Balfourier (voir [Balfourier *et al.*, 2002]. Il s'agit d'un analyseur profond utilisant les Grammaires de Propriétés. Ses différences avec le dernier analyseur que nous avons développé sont nombreuses. Cependant, il constitue l'outil le plus proche de ce dernier, ce qui le rend très important pour les comparaisons à effectuer en termes de complexité et de résultats.

Le dernier analyseur obéit à tous les critères énumérés dans la première partie. Il réalise en cours d'analyse la satisfaction des contraintes posées dans une Grammaire des Propriétés.

Le mécanisme de résolution est beaucoup plus complexe que pour les analyseurs précédents, plus souple et plus robuste, plus paramétrable et plus indépendant de la grammaire. Sa mise en œuvre est le fruit de développements logico-mathématiques autour de la satisfaction de contraintes dans le cadre des Grammaires de Propriétés. Nous en avons ébauché les détails dans la première partie. L'implémentation du modèle nécessite une très grande abstraction des niveaux d'analyse. Une différenciation des descriptions et des spécifications, ce qui mènera à un programme très volumineux, mais très paramétrable.

Les algorithmes que nous avons développés sont décrits au fil de cette partie en termes techniques et un exemple illustrera à chaque fois leur fonctionnement.

L'évaluation de la complexité de ces approches est traitée dans la prochaine partie.

Chapitre 10

Analyseurs développés pour les comparaisons

10.1 Un parenthéseur simple

Dans le but de comparer nos techniques, nous avons conçu un chunker robuste et simple. Ce programme rapide donnera une idée de la complexité minimale pour les techniques basées sur des Grammaires de Propriétés. Cet algorithme se fonde sur la technique du Chink/Chunk proposée par Liberman et Church (voir [Liberman and Church, 1992]) et sur le chunker de Di Cristo (voir [Di Cristo, 1998]). Son mécanisme consiste en une segmentation de l'entrée en chunks, à l'aide d'un automate à états finis qui utilise des *mots fonction* comme frontières de bloc. Une amélioration de la notion de chunk est mise en application, en utilisant des conjonctions comme éléments neutres pour chunks en cours de construction. Cet algorithme donne déjà des résultats intéressants pour calculer des groupes prosodiques dans le synthétiseur vocal SYNTAIX.

La technique d'analyse par chunks est assez ancienne. Abney, dans son article de 1991 (Parsing by chunks)[7], reprend cette notion : "[I begin] [with an intuition] : [when I read] [a sentence], [I read it] [a chunk] [at a time]".

Liberman et Church dans leur article de 1991 (Text Analysis and Word Pronunciation in Text to Speech Synthesis) reprennent cette idée et la simplifient en utilisant seulement deux sortes de groupes : function word* et content word *, correspondant à des groupes de mots (function words) appelés aussi f-groups. Une amélioration de cet analyseur de f-groups est proposée pour un faible coût en rangeant les verbes tensés dans la première catégorie et les pronoms démonstratifs dans la seconde. Ces groupes, selon la terminologie employée en 1975 par Ross et Tukey pour un algorithme d'indexation, sont à présent appelés chink* et chunk*

En se basant sur un étiquetage morpho-syntaxique, l'analyseur parcourt la totalité de l'entrée en un seul passage. L'automate obéit à des règles simples d'ouverture et de fermeture de blocs selon la grammaire suivante :

Le découpage en blocs utilise donc une grammaire non ambigüe visant à construire une phrase composée de plusieurs groupes appelés ici XCHINKCHUNK éventuellement entourés de ponctuations. Un groupe XCHINKCHUNK est lui-même constitué de zéro, un ou plusieurs groupes CHINK ou NEUTRE suivis d'au moins un groupe CHUNK ou NEUTRE. Un groupe CHUNK est constitué par des mots ayant une catégorie particulière : verbe conjugué, nom,

[7] [Abney, 1991]

	règle		commentaire
FINALPONCT	: :=	'.'|'...'|' ?'|' !'|EndOfFile	Ponctuation forte
PUNCT	: :=	','|' ;'|'-'|'('|')'|'{'|'}	Ponctuation faible
NEUTRE	: :=	CC|CS|Ak|Nk|Pk|Dk	Conjonctions et numéraux
CHINK	: :=	V[ae]|D|P|Rpn|S|X)	Un Chink est tout sauf un Chunk
CHUNK	: :=	Vm|N|A|Rgp|Pp	Un chunk
CHINKCHUNK	: :=	(CHINK|NEUTRE)* (CHUNK|NEUTRE)+	Alternance Chink, Chunk
AUTRE	: :=	(PUNCT)+	Série de ponctuations
XCHINKCHUNK	: :=	AUTRE*|CHINKCHUNK|AUTRE*	Chink Chunk entouré d'autre chose
Phrase	: :=	(XCHINKCHUNK)*(FINALPUNCT)	Règle principale

Tab. 23 – Grammaire Chink/Chunk

adverbe, pronom personnel ou adjectif. Un groupe NEUTRE est constitué de conjonctions ou de numéraux. Enfin, un groupe CHINK est constitué des catégories ne définissant ni un CHUNK, ni un NEUTRE. L'idée est de borner ainsi des groupes n'ayant pas de signification syntaxique au sens linguistique couramment employé, mais à la façon développée par Church et Liberman, dans la ligne prolongée par Philippe Di Cristo, de définir des ensembles de mots dont la métrique peut correspondre avec des groupes intonatifs.

Le premier exemple montre une représentation non hiérarchique de la phrase découpée en chunks. Aucune information linguistique n'est fournie.

```
[(PHRASE)
[(CHINKCHUNK)Le compositeur et son librettiste ont su créer]
[(CHINKCHUNK)un équilibre dramatique astucieux]
[(CHINKCHUNK)en mariant]
[(CHINKCHUNK)la comédie espiègle]
[(CHINKCHUNK)voire égrillarde]
[(CHINKCHUNK)et le drame]
[(CHINKCHUNK)le plus profond]
[(CHINKCHUNK)au coeur des mêmes personnages]]
```

Fig. 43 – Exemple d'analyse avec l'algorithme Chink/Chunk

10.2 Un analyseur superficiel et déterministe basé sur les Grammaires de Propriétés

Le second analyseur a été développé dans le but d'obtenir une information linguistiquement plus riche que celle offerte par le premier analyseur (avec la détection des syntagmes au lieu des simples blocs parenthésés et avec l'introduction de structures emboitées), ainsi que pour observer la progression de la complexité algorithmique entre le simple chunker et les derniers analyseurs que nous présentons plus loin.

La technique utilisée reste symbolique, se base sur les Grammaires de Propriétés dans le sens où celles-ci permettent d'exprimer l'organisation syntaxique d'une langue, et utilise des algorithmes basés sur la reconnaissance de coins gauches et coins droits potentiels de chaque syntagme.

Par cette technique, nous obtenons une information hiérarchique et grammaticale tout en préservant la robustesse et l'efficacité du traitement. Nous nous servons pour cela d'une grammaire représentée dans le formalisme de grammaire de propriété décrit ci-dessus. Un des intérêts principaux de ce formalisme est qu'il ne se sert pas réellement de la notion de grammaticalité, la remplaçant par un concept plus général de caractérisation. Il devient alors possible de proposer une description en termes de propriétés syntaxiques pour n'importe quel genre d'entrée (grammaticale ou non). L'ouverture et la fermeture des chunks se fonde ici sur une information compilée à partir de la grammaire. Cette information consiste en un ensemble de coins gauches et coins droits potentiels, ainsi que les constituants potentiels pour les chunks. Elle est obtenue en compilant les propriétés de linéarité, d'obligation, d'exigence et d'exclusion décrites dans les chapitres précédentes ainsi que, indirectement, une propriété de constituance. A partir de cette grammaire compilée, l'analyseur fonctionne de la manière suivante :

Deux piles, une des catégories ouvertes et une seconde des catégories fermées, sont construites après l'analyse de chaque nouveau mot observé dans l'entrée : nous pouvons ouvrir de nouvelles catégories ou fermer des catégories déjà ouvertes en appliquant quelques règles. Cet algorithme étant récursif, les actions d'ouverture, de continuation et de fermeture sont elles aussi récursives. C'est la raison pour laquelle les règles doivent avoir une définition stricte afin d'être certain que l'algorithme sera déterministe et se terminera toujours. Cette technique superficielle d'analyse peut être vue comme ensemble de règles de production/réduction/coupure.

Règle 1 : Ouvrez une expression p pour la catégorie courante c si c peut être le coin gauche de p. On notera les coins gauches sous l'abréviation **CG** à partir d'ici.

Règle 2 : N'ouvrez pas une catégorie déjà ouverte si la catégorie appartient au syntagme courant ou est son coin droit. Autrement, nous pouvons la rouvrir si le mot courant peut seulement être son coin gauche. On notera les coins droits sous l'abréviation **CD**. Les catégories pouvant continuer un syntagme (constituants potentiels) seront notés **CP** à partir d'ici.

Règle 3 : Fermez les syntagmes ouverts si le syntagme le plus récemment ouvert ne peut ni continuer l'un d'entre eux, ni être l'un de leurs coins droits.

Règle 4 : Lors de la fermeture d'une expression, appliquez les règles 1, 2 et 3. Ceci peut fermer ou ouvrir de nouvelles expressions prenant en compte toutes les catégories y compris celles des syntagmes.

Une liste des coins gauches, constituants potentiels et coins droits est générée par un programme qui *compile* littéralement une grammaire exprimée dans le formalisme des Grammaires de Propriétés. Pour cela, le programme charge une grammaire exprimée en XML (le modèle de représentation a été exposé dans le chapitre 3.3.9 de la partie 1). Une fois chargée, cette grammaire est exprimée sous forme de graphe en mémoire (à nouveau, nous renvoyons le lecteur la fin de cette partie, chapitre 11.1 pour la représentation des Grammaires de Propriétés sous forme de graphe). La phase de compilation consiste à analyser le graphe de la grammaire en faisant apparaitre récursivement les constituants potentiels de chaque syntagme, leurs coins gauches et leurs coins droits. Ceci est rendu possible par l'interprétation des contraintes de linéarité pour la détection des CGs et CDs. Pour la détection des CPs, on repère dans le graphe toutes les catégories apparaissant dans les contraintes d'obligation, de facultativité et de dépendance.

On obtient ainsi une liste aussi exhaustive que possible des CGs, CDs et CPs de chaque syntagme. La figure 44 montre un fichier obtenu après compilation d'une grammaire très simplifiée pour les besoins de l'exemple. On indique quels syntagmes peuvent être commencés, achevés ou constitués par chaque catégorie. Eventuellement, des restrictions de traits sont apportées si la grammaire le spécifie.

```
CATEGORIE= adj.
     GAUCHE:    SN.
     GAUCHE:    SA.
     DROITE:    SA.
     CONSTITUE: SN/type=i.
     CONSTITUE: SA/type<>i.
CATEGORIE= det.
     GAUCHE:    SN.
     CONSTITUE: SN.
CATEGORIE= N.
     DROITE:    SN.
     DROITE:    SP.
     DROITE:    SV.
     CONSTITUE: SN.
     CONSTITUE: SP.
     CONSTITUE: SV/type=c
CATEGORIE= prep.
     GAUCHE:    SP.
     CONSTITUE: SP.
CATEGORIE= V.
     GAUCHE:    SV.
     DROITE:    SV
     CONSTITUE: SV/type=m.
CATEGORIE= SN.
     DROITE:    SP.
     DROITE:    SV.
     CONSTITUE: SP.
     CONSTITUE: SV/cate<>p.
     CONSTITUE: P.
CATEGORIE= SA.
     DROITE:    SN.
     DROITE:    SV.
     CONSTITUE: SN.
     CONSTITUE: SV.
CATEGORIE= SP.
     DROITE:    SN.
     DROITE:    SA.
     DROITE:    SV.
     CONSTITUE: SN.
     CONSTITUE: SA.
     CONSTITUE: SV.
CATEGORIE= SV.
     DROITE:    SP.
     CONSTITUE: SP/forme=p,temps=p,forme=n.
```

FIG. 44 – Exemple de grammaire compilée en constituants gauches, internes et droits

```
[(phrase)
[(NP)Le compositeur
[(AP) et]
son librettiste]
[(VP)ont su créer]
[(NP) un équilibre
[(AP)dramatique astucieux]]
[(Compl)en
[(VP)mariant]]
[(NP)la comédie
[(AP)espiègle voire égrillarde et]
le drame
[(Sup)le plus profond]]
[(PP)au cœur de
[(NP)les
[(AP)mêmes]
personnages]]]
```

FIG. 45 – Exemple d'analyse superficielle

Ce deuxième exemple donne une représentation hiérarchique de la phrase, divisée en chunks portant une étiquette. Comme nous avons employé une version précompilée de la grammaire (raccourcie) et parce que nous avons forcé quelques choix syntaxiques afin de préserver le déterminisme et la terminaison de l'analyse, il s'avère que des erreurs ont été faites par l'analyseur superficiel : Les conjonctions sont distinguées. Malgré ces lacunes, la coupure est améliorée et la plupart des catégories sont détectées correctement.

10.3 Un analyseur profond et non déterministe basé sur les Grammaires de Propriétés

Pour des comparaisons plus importantes, nous avons aussi utilisé un analyseur basé sur les Grammaires de Propriétés conçu par Jean-Marie Balfourier. L'analyse profonde est directement basée sur les Grammaires de Propriété. Elle consiste, pour une phrase donnée, à établir tous les sous-ensembles possibles d'éléments juxtaposés qui puissent décrire une catégorie syntaxique. Un sous-ensemble est clairement caractérisé s'il satisfait les contraintes de la grammaire. Ces sous-ensembles s'appellent des *arcs*, ils décrivent un segment de la phrase entre deux positions.

À la première étape, chaque catégorie lexicale est considérée comme arc du niveau 0. La phase suivante revient à produire tous les sous-ensembles possibles d'arcs au niveau 0. Le résultat est un ensemble d'arcs de niveau 1. Les étapes suivantes fonctionnent de la même manière et produisent tous les sous-ensembles possibles d'arcs, chaque étape correspondant à un niveau. L'algorithme finit quand aucun nouvel arc ne peut être créé.

Un arc est caractérisé par :

– Une position initiale et finale dans la phrase,
– Une catégorie syntaxique,

– Un ensemble de propriétés syntaxiques décrivant la catégorie,
– Un ensemble de constituants : un unique constituant lexical au niveau 0, et un ou plusieurs arcs aux autres niveaux.

Le dernier exemple (figure 46) présente deux des phrases de couverture maximale produites par l'analyseur profond. Cette figure, qui illustre l'ambiguïté d'attachement de pp, montre une structure hiérarchique. Reste que chaque étiquette représente l'état du système de contraintes après évaluation.

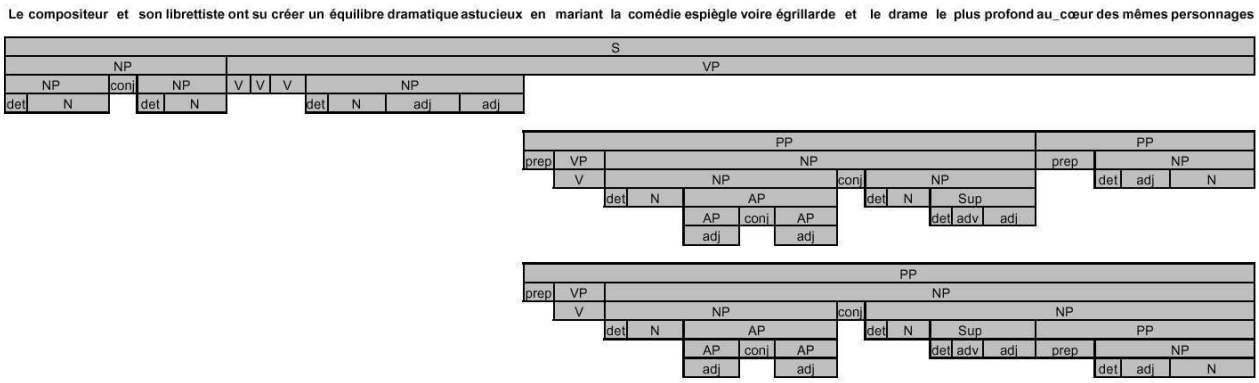


Fig. 46 – Exemple d'analyse profonde non déterministe

Chapitre 11

Un analyseur à granularité variable

Dans la direction exposée dans les deux parties précédentes, nous avons programmé un analyseur syntaxique beaucoup plus complexe que les précédents, afin de mettre en œuvre le formalisme des Grammaires de Propriétés, ainsi que les notions développées plus haut, telles que la caractérisation s'appuyant sur la mesure de *densité de satisfaction* et la séparation du programme et de la grammaire ainsi que de ses spécifications. Un temps de calcul raisonnable a été préservé et peut encore être amélioré, comme nous le verrons plus loin. L'idée maitresse pour le développement de cet analyseur repose sur les contraintes fortes suivantes :

- Le programme d'analyse est doté d'un algorithme général qui s'appuie sur une représentation générique des Grammaires de Propriétés. Cette représentation s'effectue grâce à des graphes, dont la spécification est précisée ci-dessous, et dont la structure permet une analyse syntaxique ascendante. On trouvera plus de précisions à ce propos dans la suite de ce chapitre.
- L'analyse syntaxique grâce à ce programme dépend uniquement d'un système de mesure de la densité de satisfaction. Analyser, selon cet algorithme, consiste à produire un graphe (le *graphe de caractérisation*), à partir d'une entrée (le texte étiqueté), et en s'appuyant sur le *graphe de la grammaire* (qui décrit quelles propriétés peuvent contribuer à la caractérisation de quelles catégories). Ce dernier graphe fait référence à des propriétés dont la spécification est fournie dans un dernier graphe : le *graphe sémantique*. Grâce au graphe sémantique, la satisfaction de chaque type de propriété est explicitée en dehors du programme lui-même, en termes logico-mathématiques, afin de permettre une redéfinition du comportement de la grammaire indépendamment de toute redéfinition de la grammaire elle-même ou du programme lui-même. Le programme d'analyse est donc simplement capable d'interpréter des graphes de sémantique, de grammaire, et à partir d'une entrée étiquetée, de fournir un graphe de caractérisation.
- L'algorithme de caractérisation, qui s'appuie sur la structure générique de graphes (sémantiques, grammaticaux et de caractérisation) est un système de satisfaction de contraintes qui répond aux spécificités du formalisme des Grammaires de Propriétés. La technique employée permet de construire la totalité d'une caractérisation en temps limité. En attendant de l'exposer complètement, nous pouvons à présent dire qu'il repose sur un processus répétitif qui s'arrête lorsqu'aucune évolution du graphe de caractérisation n'est possible. L'initialisation du processus passe par une transformation de l'entrée étiquetée en un graphe de caractérisation minimal (la souche de la caractérisation). Chaque étape de ce processus peut alors être divisée en deux parties importantes : la

première consiste à satisfaire toutes les propriétés de la grammaire qui font référence à des éléments du graphe de caractérisation. C'est l'étape dite *étape de dissémination*. Ce nom provient du mécanisme qui consiste à produire dans le graphe de caractérisation un ensemble de réplicats des nœuds et arcs de la grammaire pouvant être identifiés comme parents de *référents* des éléments déjà présents dans la caractérisation. La seconde consiste à considérer quels ensembles de propriétés satisfaites peuvent donner lieu à la construction de catégories dans le graphe. Ceci en fonction de seuils de satisfaction (paramétrables par l'utilisateur du programme). Chaque catégorie produite devient un nœud du graphe de caractérisation, connecté à toutes les propriétés qui auront contribué à sa création. Cette étape est dite l'*étape de projection*. Ce nom résume le mécanisme qui consiste à récolter tous les nouveaux nœuds du graphe de caractérisation et à les répliquer autant de fois qu'il est possible de construire une catégorie de la grammaire ayant ces nœuds comme descendants potentiels.

L'algorithme, quoique très complexe à mettre en place, permet de rendre générique le processus de caractérisation, quelle que soit la grammaire et sa spécification sémantique. Nous en verrons les détails plus loin.

– les structures de graphes doivent permettre autant un calcul rapide et une occupation mémoire minimale qu'une faculté à être représentés graphiquement, si le programme qui les utilise en a besoin. D'autre part, si le programme qui les utilise n'en a pas besoin, cette part graphique doit pouvoir être totalement négligée (aucune occupation mémoire). Pour cela, le modèle objet des graphes doit permettre une différenciation de la forme graphique et de la partie informative de l'objet associé aux nœuds et aux arcs. C'est un module de contrôle qui sera chargé de faire l'interface entre ces facettes du même objet.

Dans les développements que nous allons effectuer ci-dessous, nous employons la notion de graphe, qui recouvre à la fois la sémantique, la grammaire et la caractérisation. Cette notion est utilisée plutôt que celle d'arbre, puisque le modèle des données, les algorithmes utilisés et les structures construites durant l'analyse, appartiennent réellement au domaine des graphes, dont l'algorithmique est plus complexe que celle qu'on pourrait développer autour de graphes. Les graphes construits sont très proches d'arbres, puis qu'il s'agit de graphes orientés acycliques (DAG en anglais). La caractérisation (le produit de l'analyse), est aussi un graphe DAG, même si chaque construction prise isolément est un arbre. Durant l'analyse, les trois niveaux (sémantique, grammaire et caractérisation) sont interconnectés pour permettre les traitements. L'ensemble, vu comme une seule structure, forme alors un graphe complexe, qui n'est plus un DAG. La généricité des structures implique donc d'utiliser la notion générale de graphes. Le modèle a été développé en envisageant des heuristiques particulières permettant d'éviter l'explosion combinatoire attendue avec une analyse Bottom-Up par contraintes.

11.1 Modèle de graphes pour la représentation des Grammaires de Propriétés

Comme l'indiquent les caractéristiques des structures exposées ci-dessus, nous avons mis en place un modèle de graphes que résume la figure 47 :

– Un *graphe* est constitué de *nœuds* et d'*arcs*.
– Un *nœud* est doté d'une partie informative (*objet sémantique*) et d'une partie graphique (*figure de nœud*). De plus, un *nœud* mémorise quels arcs arrivent sur lui ou partent de lui (*arcs entrants* et *arcs sortants*).

- Un *arc* est doté d'une partie informative (*objet sémantique*) et d'une partie graphique (*figure d'arc*). De plus, un *arc* mémorise les nœuds qui constituent ses extrémités (*nœud origine* et *nœud destination*). Avec cet objet, le graphe devient instantanément orienté. Cette caractéristique est nécessaire pour représenter les graphes des Grammaires de Propriétés. Nous verrons plus loin que cela facilite le parcours et rend possibles certains traitements en temps raisonnable.
- Une *figure* de nœud ou d'arc dépend du programme qui les mettra en pratique. Certaines spécifications sont données dans le modèle, mais peuvent être surchargées si besoin. Le programme Accolade, présenté dans la dernière partie, fait usage de ces objets et permet la représentation des graphes de Grammaires de Propriétés au fil de l'analyse syntaxique.
- Un *objet sémantique* associé aux nœuds et aux arcs constitue la part la plus importante de ces objets : ils sont chargés de stocker toutes les informations utiles pour les traitements à effectuer. Tout d'abord, ces objets sont typés (selon une nomenclature que nous verrons plus loin) en fonction de leur position dans le graphe. A titre d'exemple, un nœud dont l'objet sémantique est de type *propriété* ne peut être parent -antécédant d'un arc n'ayant pour descendant- que des nœuds dont l'objet sémantique est de type *référent à une catégorie* La nomenclature des types oblige la construction de graphes bien formés (selon le formalisme déployé dans la partie 1, lors de la spécification formelle des Grammaires de Propriétés).
  Un *objet sémantique* est associé à un second type, chargé de préciser si l'objet qui le contient est un nœud ou un arc.
  Enfin, un *objet sémantique* contient aussi un ensemble (un vecteur) d'objets de type *information*.
- Un objet *information* est simplement une structure à trois variables : un type, un nom et une valeur. Ce type d'objet apporte une abstraction de la notion de variable informatique. Toute variable est en effet une association type/nom/valeur.
  Un objet *information* est donc un conteneur, mémorisant toute sorte de chose qui peut être associée à un nœud ou à un arc du graphe. Par exemple, la valeur textuelle associée à nœud, comme le mot "Det" qui représente la catégorie des déterminants, ou le mot "Linéarité" qui représente la propriété de linéarité. L'information fonctionne comme une clé. Avec le type et le nom de la variable, on accède à sa valeur. Dans certains cas, le modèle ainsi précisé permet d'ajouter une infinité d'informations à un nœud du graphe, sans avoir à redéfinir la structure de l'objet nœud ou de l'objet arc qui devrait contenir ces informations. Dans d'autres cas, il a été préférable de modifier un peu le modèle afin de gagner du temps de traitement et de simplifier ainsi la recherche systématique d'une information dans tout le graphe.

Grâce à ce modèle, il a été possible de construire des modules de chargement et de sauvegarde pour les graphes. ainsi, un graphe de grammaire, stocké dans un fichier XML, conforme à sa DTD (voir la partie 1), peut être transformé en un graphe en mémoire, reflétant exactement la totalité des informations de la grammaire. Les graphes de spécification sémantique sont chargés et sauvegardés selon le même principe. Enfin, tout fichier étiqueté est préchargé puis converti en graphe de caractérisation, afin d'initialiser le processus d'analyse.
Le fonctionnement de ces graphes, comme l'ajout d'un nœud, ou d'un arc, ou la suppression, la modification, ainsi que la gestion de la représentation graphique de ses éléments (qui suppose

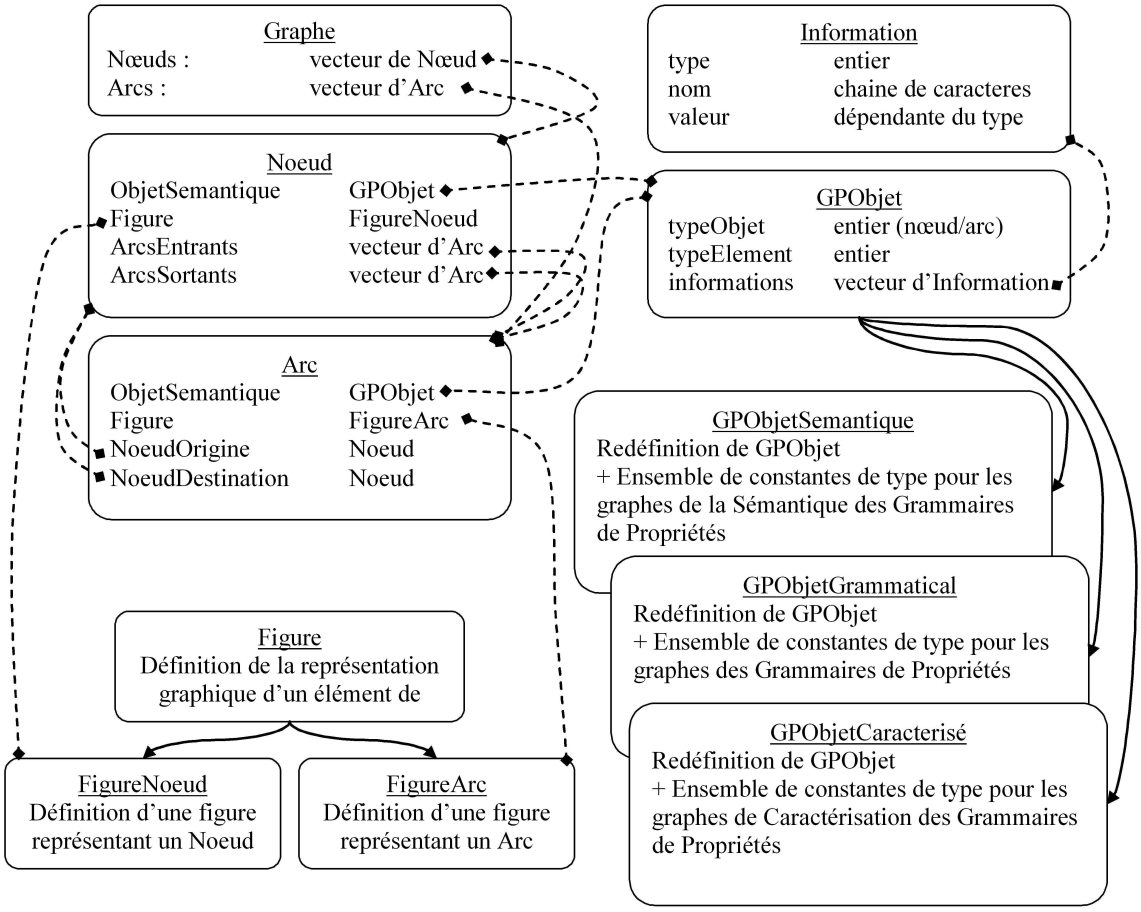


FIG. 47 – Modèle de graphe pour les Grammaires de Propriétés

de faire l'interface entre données et figures dès qu'une modification a lieu, soit graphiquement, soit dans les informations contenues par les éléments du graphe), est réalisée par un module de contrôle. Le contrôleur détecte les événements sur un graphe et effectue les opérations en conséquence. C'est ce qu'indique la figure 48.

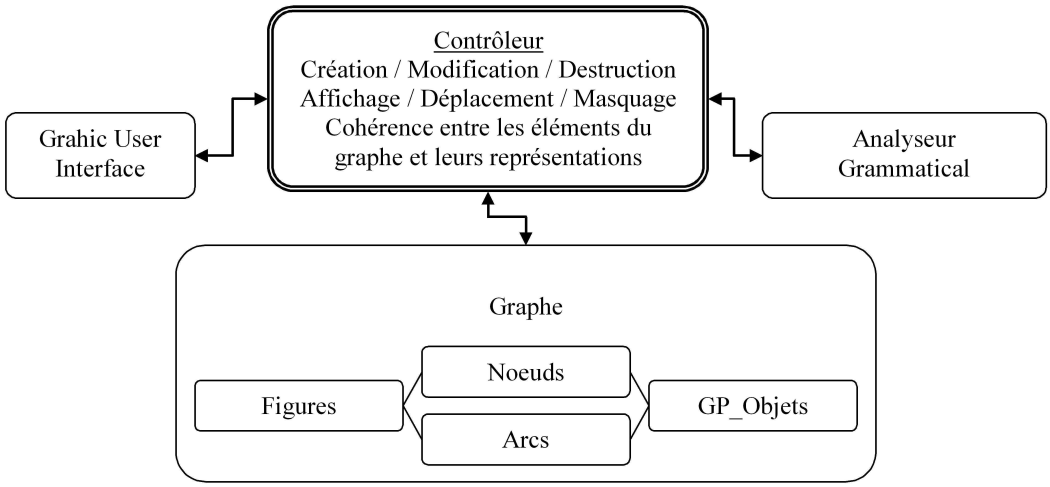


FIG. 48 – Contrôleur pour les graphes

11.2 Implantation du modèle de graphes pour la spécification sémantique des Grammaires de Propriétés

Nous présentons ci-dessous, à titre d'exemple une figure (figure 49) représentant le graphe obtenu après chargement d'un fichier XML contenant une spécification sémantique des Grammaires de Propriétés. Cette sémantique est conforme au formalisme exposé en partie 1.

A chaque élément et chaque attribut de la DTD est associé un nœud ou un arc, doté d'un objet sémantique spécifique contenant les informations recueillies dans le fichier XML. Ce graphe sémantique constitue une ressource accessible depuis l'analyseur à chaque fois qu'une contrainte doit être évaluée. Ce graphe décrit en effet le fonctionnement de chaque propriété. En le consultant, on peut connaître les conditions d'évaluation et de satisfaction pour chaque propriété de la grammaire. La figure 49 schématise par exemple un graphe sémantique dans lequel plusieurs *définitions de propriété* sont associées à un ensemble de nœuds qui décrivent en particulier chaque information nécessaire à la définition d'une propriété : un symbole, l'arité, l'ordre, la capacité et la cardinalité, l'évaluabilité et la satisfaisabilité.

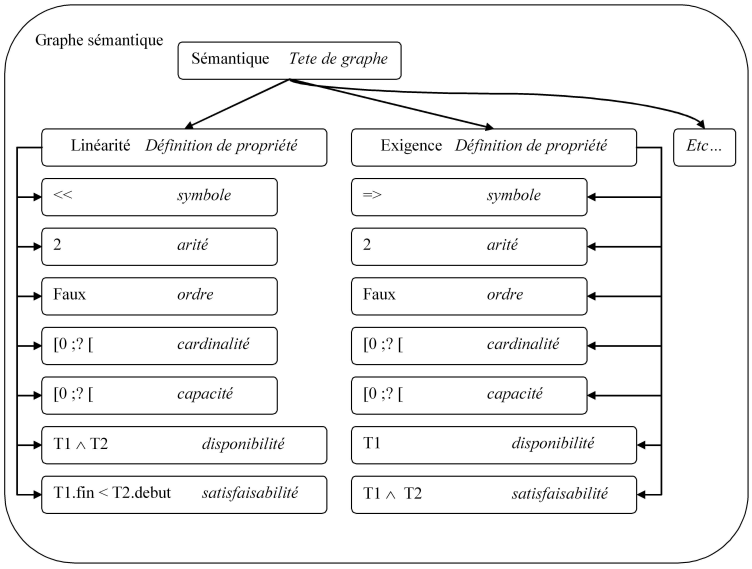


Fig. 49 – Graphe de la sémantique des Grammaires

11.3 Implantation du modèle de graphes pour les Grammaires de Propriétés

De la même façon que pour le graphe sémantique, un graphe grammatical est construit automatiquement avant l'analyse à proprement parler. Un fichier XML répondant aux spécifications de la DTD est chargé et chaque information est insérée dans le graphe grammatical.

Ce graphe constitue la seconde ressource nécessaire à l'analyse syntaxique. Lorsqu'un élément du texte est analysé, la grammaire indique quels sont les propriétés qui en font usage. Un exemple d'analyse sera donné plus loin.

Dans le graphe grammatical donné ici en exemple (figure 50, un nœud *grammaire* a pour fils l'ensemble des catégories de la grammaire. Pour chacune de ces catégories, il est possible de développer un sous-graphe contenant les spécifications de traits et un autre sous-graphe contenant les spécifications de propriétés. Ces derniers sous-graphes se divisent à leur tour en plusieurs sous-graphes, conformément à la spécification donnée en première partie. La *spécification de traits* se subdivise en ensembles de *types de traits* qui se divisent à leur tour en ensembles de *valeurs de traits*. La *spécification de propriétés* se divise en ensembles de *propriétés* (une propriété de linéarité, d'exigence, de dépendance etc.) qui se divisent en ensembles de *contraintes* (relations appartenant à la même propriété). Dans l'exemple donné, on trouvera deux contraintes pour la linéarité dans le syntagme nominal. La première indique la précédence entre le déterminant et l'adjectif. La seconde une précédence entre le déterminant et le nom. Dans ce graphe, les nœuds intitulés nom, adjectif, déterminant etc. qui font partie d'une spécification de contrainte, sont appelés des termes. Chaque terme doit avoir un nom qui correspond avec une catégorie de la grammaire (afin d'assurer la cohérence de l'analyse). Ces termes font explicitement référence aux catégories spécifiées par la grammaire. A cause de cette référence et dans le but de faciliter l'analyse (comme nous pourrons le voir plus loin),

chacun de ces termes est relié par un arc à la catégorie à laquelle il fait référence.

Un graphe grammatical est donc entièrement descendant (et se comporte comme un arbre) de la racine jusqu'aux termes, et un arc spécial (de référence) relie chaque terme à la catégorie qui porte son nom, ce qui en fait un graphe cyclique (si on se contente d'observer les connexions indépendamment de leur sens). Les algorithmes de parcours que nous allons employer sur ce graphe (comme sur le graphe sémantique qui est un arbre), seront donc les mêmes que ceux employés pour parcourir des arbres.

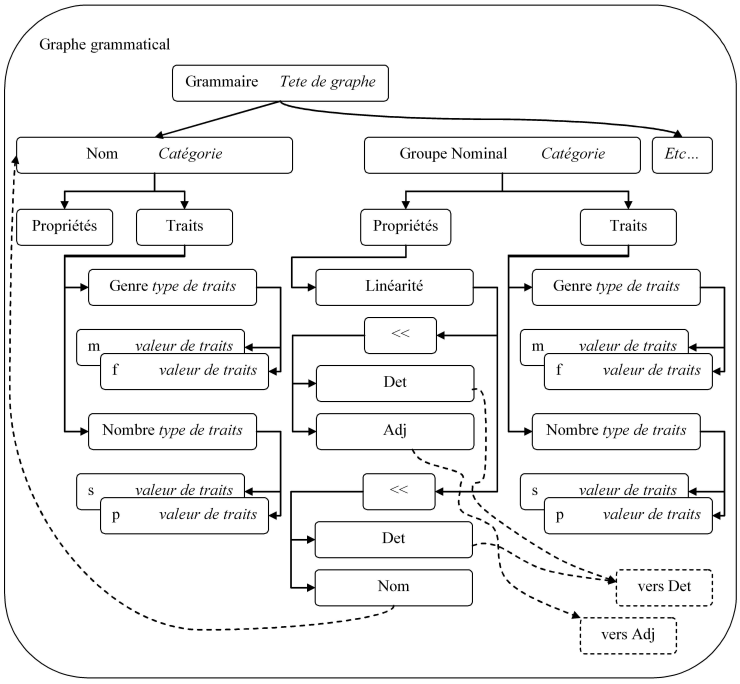


FIG. 50 – Graphe des Grammaires

11.4 Implantation du modèle de graphes pour la caractérisation d'une entrée avec les Grammaires de Propriétés

Lorsque les graphes grammatical et sémantique sont mémorisés, l'analyse du texte peut commencer. Celle-ci consiste à produire un graphe de caractérisation reflétant les catégories induites par l'analyse.

Nous supposerons à présent que le texte analysé a déjà subi un traitement d'étiquetage (association de catégories aux entrées, sans nécessairement désambiguiser ces catégories). L'étiquetage fournit la série des étiquettes de catégories qui vont permettre de démarrer l'analyse. Chaque étiquette doit être reconnue par la grammaire.

Lors du chargement de cet étiquetage, un début de graphe est construit en associant les mots et leurs étiquettes dans l'ordre d'apparition des entrées. Une numérotation est simplement

associée à chacun des nœuds nouvellement créés afin de conserver l'indice de chaque élement. Dans l'exemple donné à la figure 51, la partie basse du graphe constitue ce graphe initial.

Le processus de caractérisation (l'analyse) va permettre de construire les niveaux supérieurs. Par exemple en partant du premier déterminant et du premier nom, une contrainte de linéarité va être satisfaite et un syntagme nominal va pouvoir être caractérisé.

Le propos de ces premiers exemples n'est pas de donner l'algorithme d'analyse, mais seulement de montrer la modélisation du graphe d'analyse. C'est pourquoi nous n'insistons pas sur les phénomènes particuliers qui pourraient détourner de cette modélisation. Notamment, le calcul de densité de satisfaction n'est pas précisé dans cette figure pour laquelle les valeurs de densité de satisfaction sont toutes égales à 100%. Le point de départ de l'analyse, comme nous venons de le voir est un premier graphe de caractérisation ne contenant pas encore de caractérisation.

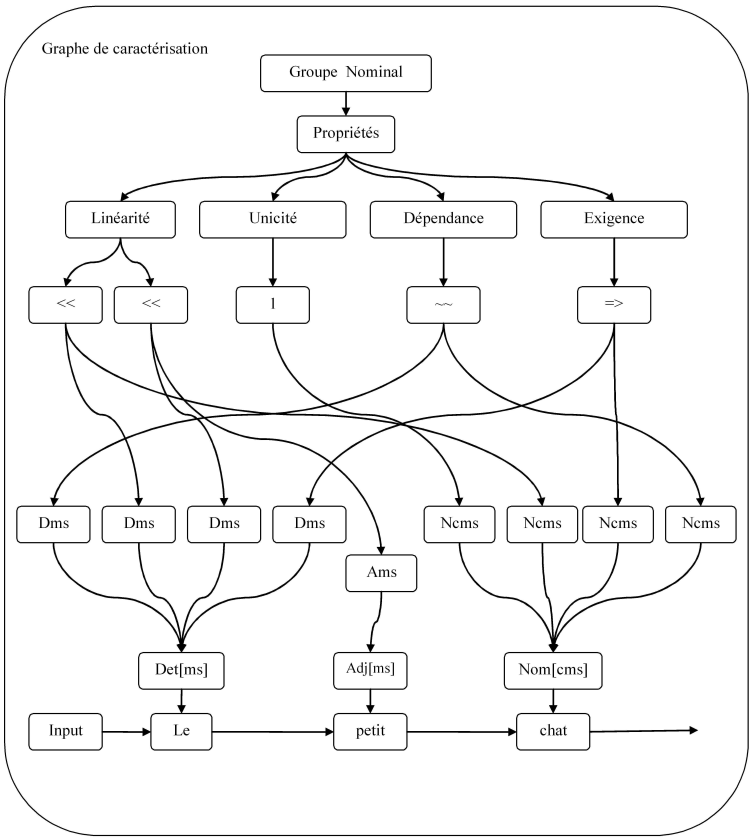


FIG. 51 – Graphe de caractérisation

11.5 Interconnections entre les trois graphes

Ce qui va réellement permettre aux algorithmes d'analyse de fonctionner, ce qui va permettre de construire une caractérisation de l'entrée, ce qui permet de connaitre, pour chaque

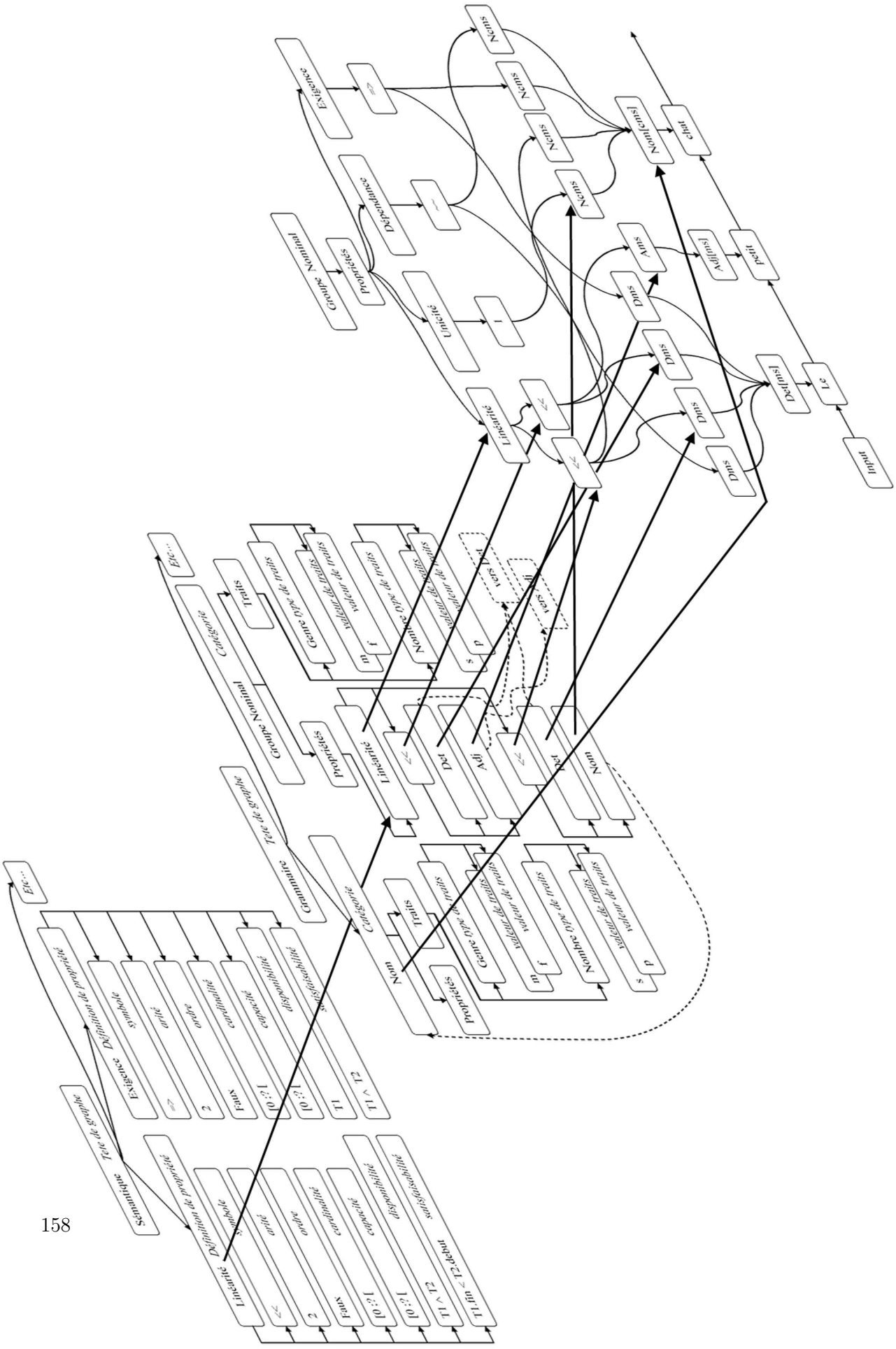
élément de l'entrée l'ensemble des nœuds de la grammaire qui lui sont associés et pour chaque élement de la grammaire l'ensemble des nœuds du graphe sémantique qui décrivent son fonctionnement, réside dans un niveau d'interconnexion qui englobe les trois graphes.

La figure 52 met en évidence ce dernier niveau de connexion qui définit le *graphe d'analyse* constitué des trois graphes décrits ci-dessus. Rappelons qu'un arc, dans les structures programmées dans notre analyseur, est connu (accessible) de ses nœuds d'origine et de destination. Autrement dit, depuis un nœud, il est possible de retrouver tous ses antécédents et tous ses descendants directs. C'est grâce à cette information que vont fonctionner les algorithmes d'analyse.

Le nœud du graphe sémantique décrivant le fonctionnement de la linéarité, par exemple, est relié par un arc à chaque nœud du graphe grammatical ayant le nom de linéarité. Il en est ainsi pour chaque propriété de la grammaire. Lorsqu'un nœud de la grammaire concerne la linéarité, il est alors très facile de reconstituer le fonctionnement de cette propriété.

De la même façon, les nœuds de la grammaire sont connectés aux nœuds du graphe de caractérisation. Lors du chargement de l'entrée, le graphe de caractérisation préliminaire est connecté au graphe grammatical par des arcs reliant les catégories aux étiquettes morphosyntaxiques (dans notre exemple, un arc relie la catégorie "nom" à l'étiquette "nom"). Grâce à ce lien, il sera possible de *confronter la grammaire avec l'entrée*, ce qui constitue le fondement des algorithmes mis en place. Au fur et à mesure que l'analyse progresse, de nouveaux arcs et de nouveaux nœuds seront ajoutés au graphe de caractérisation. Ces derniers seront reliés à la grammaire pour permettre à l'analyse de progresser. Ainsi (dans notre exemple), à partir du nœud donnant une étiquette "Nom" au mot "chat", une recherche dans le graphe grammatical permet de repérer tous les *nœuds termes* ayant la catégorie "Nom" pour référent. Chacun de ces nœuds est *dupliqué* dans le graphe de caractérisation, est relié à la catégorie qui lui est référente et est relié au graphe grammatical afin de propager l'analyse. A partir de ces nœuds termes, il est possible de rechercher à nouveau dans la grammaire quels seront les nœuds *de contrainte* qui peuvent être confrontés avec l'entrée. Nous n'allons pas encore entrer dans les détails de ce processus.

L'idée à retenir est qu'à chaque étape de l'analyse, la connexion des trois graphes est utilisée pour confronter l'entrée à la grammaire et pour calculer la satisfaction des contraintes en fonction de la sémantique qui leur est associée.



11.6 Elements d'analyse

L'analyseur que nous avons développé est très complexe. Nous résumons ici les principes de base de son fonctionnement. L'algorithme utilise les trois graphes décrits précédemment et suppose l'existence d'un ensemble de méthodes permettant de parcourir ces graphes, d'y rechercher une information etc. Rappelons que le graphe de caractérisation contient initialement les étiquettes morphosyntaxiques associées aux entrées à analyser.

L'algorithme 4 indique les grandes lignes de l'analyse, en utilisant deux grandes fonctions : *DisseminationSouche* et *RecolteSouche*.
L'analyse syntaxique consiste en deux grandes étapes : la dissémination des contraintes et la récolte des contraintes permettant de projeter des catégories.

La fonction *DisseminationSouche*, détaillée par l'algorithme 5, permet de développer l'ensemble des contraintes de la grammaire qui pourraient réagir avec l'entrée (c'est l'étape qui correspond à la notion de *caractérisation* du formalistme des GPs).
La fonction *RecolteSouche*, décrite par l'algorithme 6, effectue le recensement des contraintes participant à la caractérisation de chaque catégorie de la grammaire et projette à chaque fois que c'est nécessaire un nouveau nœud dans le graphe de caractérisation ayant les caractéristiques de cette catégorie (c'est l'opération de *projection* des catégories).
Ces deux fonctions font en réalité appel à de nombreuses sous-fonctions et leur version JAVA implantée dans SeedParser fait plus de 2000 lignes de code pour chacune d'elles. Il s'avère trop difficile de simplifier ne serait-ce qu'un peu le contenu de ces fonctions pour en donner un algorithme en pseudo-code. Nous avons préféré présenter succinctement leur fonctionnement, en utilisant des descriptions explicites.

On réalise grâce à la fonction DisseminationSouche une sorte de réaction entre la grammaire et une partie du graphe de caractérisation (soucheDissemination). De cette réaction sont issus de nouveaux nœuds et arcs qui propagent les informations grammaticales au sein de la caractérisation. Nous ne détaillons pas le processus de satisfaction de contraintes, qui fait appel à l'ensemble des graphes disponibles et qui parcourt simplement le graphe sémantique pour valider ou invalider les propriétés en cours d'analyse. Lorsque la dissémination ne produit plus de nœuds, le moteur d'analyse procède à l'appel de la fonction RecolteSouche. Cette dernière consiste en une projection (qui est un processus ascendant), mais nécessite une analyse descendante partielle (repérer dans la caractérisation les nœuds susceptibles d'être des descendants de chaque catégorie de la grammaire). Ce processus requiert une tentative de construction, souvent avortée, pour chaque catégorie et pour chaque sous-ensemble de nœuds présents dans la caractérisation. Pour éviter l'explosion combinatoire attendue, des heuristiques sont mises en œuvre, en cherchant cependant à préserver autant que possible la généricité des traitements. L'algorithme 6 fait appel à la notion de clique, pour désigner des ensembles de nœuds ayant la particularité d'être des descendants possibles d'une catégorie donnée et pour autre caractéristique la particularité de ne pas entrer en conflit les uns avec les autres pour la conservation du principe de projectivité. L'ensemble des nœuds d'une clique, une fois connecté à une catégorie, forme donc un arbre, sans croisements. Une heuristique de coupure intervient dans cette fonction, afin d'éviter la récursion infinie. C'est ce que nous avons appelé *répétition verticale* dans la première partie de ce document.

Algorithme 4 Principes d'analyse pour l'analyseur profond à granularité variable

Entrées: $semantique, grammaire, caracterisation \subset graphe$
Sorties: caractérisation d'une entrée en fonction d'une grammaire et de sa sémantique.
{Le graphe de caractérisation contient toutes les informations à l'issue du traitement}

soit $soucheDissemination \subset tableaudenoeuds$
soit $soucheCaracterisation \subset tableaudenoeuds$
soit $noeudsCrees \subset tableaudenoeuds$
soient $dissemination, recolte \subset \{VRAI, FAUX\}$
{récupérer tous les noeuds termes de la caractérisation initiale dans les tableaux souches}
$soucheDissemination \leftarrow noeudsTermes(caracterisation)$
$soucheRecolte \leftarrow noeudsTermes(caracterisation)$
{initialisation pour entrer dans la boucle}
$recolte \leftarrow VRAI$
{tant qu'une caractérisation a lieu}
tantque *recolte* **faire**
  $recolte \leftarrow FAUX$
  $dissemination \leftarrow VRAI$
  $noeudsCrees \leftarrow \oslash$
  {tant qu'une dissémination des contraintes a lieu}
  tantque *dissemination* **faire**
    {procéder à la dissémination des contraintes sur la souche par étapes successives}
    {le graphe caractérisation reçoit chaque noeuds nouvellement créé}
    $dissemination \leftarrow DisseminationSouche(soucheDissemination, caracterisation, noeudsCrees, semantique, grammaire)$
    {le tableau noeudsCrees contient la liste des noeuds générés lors de cette dissémination}
    {ajoutons ce tableau à la souche de dissémination}
    $soucheDissemination \leftarrow soucheDissemination \bigcup noeudsCrees$
  fin tantque
  {à la suite de la dissemination, tous les noeuds créés ne sont pas forcément reliés à des contraintes ; il faut donc nettoyer la caractérisation de ces noeuds inutiles}
  $NettoyerCaracterisation(caracterisation, noeudsCrees)$
  {ensuite, il est possible de commencer à caractériser des catégories}
  $noeudsCrees \leftarrow \oslash$
  $reaction \leftarrow RecolteSouche(soucheRecolte, caracterisation, noeudsCrees semantique, grammaire)$
  {le tableau noeudsCrees contient la liste des noeuds catégoriels générés lors de cette caractérisation}
  {affectons ce tableau à la souche pour recommencer }
  $soucheRecolte \leftarrow soucheRecolte \bigcup noeudsCrees$
  $soucheDissemination \leftarrow noeudsCrees$
fin tantque

Algorithme 5 Fonction de dissémination des contraintes pendant l'analyse

Fonction DisseminationSouche
Entrées:
in $caracterisation, semantique, grammaire$: *graphe*
in $soucheDissemination, noeudsCrees$: *tableau de noeuds*
{Les trois graphes reçus sont transmis par la procédure principale d'analyse}
{A l'issue de cette fonction, le graphe caractrisation est enrichi des nouveaux noeuds créés.}
{Ces nouveaux noeuds créés sont aussi stockés dans le tableau noeudsCrees.}
{Le tableau soucheDissemination sert de support pour savoir à partir de quels noeuds s'effectue la dissémination.}
Sorties: Une valeur booleenne est retournée : $VRAI$ si une dissémination a eu lieu, $FAUX$ sinon.

◦ Pour chaque noeud NC de la souche de dissémination, chercher son noeud référent NG dans la grammaire.
◦ Chercher dans la grammaire à quelle propriété PG est relié NG (remonter un arc du graphe).
◦ Pour cette propriété, chercher si il existe d'autres termes et les mémoriser dans une liste ListeTermes (redescendre les arcs partant de PG).
◦ Pour chaque noeud de ListeTermes, chercher les occurences existantes dans la souche de dissémination et les stocker dans une liste ListeTermesSouche.
◦ Construire un noeud PC dans la caractérisation, qui sera une réplique de PG, et qui aura pour termes (noeuds descendants) NC et les autres noeuds de ListeTermesSouche. ◦ Evaluer la satisfaisabilité de PC grâce à la sémantique. (ce qui demande de récupérer le sous-graphe de spécification sémantique associé au type de propriété dont PG est membre).
◦ Si PC est valide (évaluable), le conserver ;
◦ Sinon l'éliminer.
◦ Continuer la dissémination avec un autre noeud de la souche.

Fin Fonction DisseminationSouche

Algorithme 6 Fonction de récolte des contraintes pendant l'analyse

Fonction RecolteSouche
Entrées:
in $caracterisation, semantique, grammaire$: *graphe*
in $soucheRecolte, noeudsCrees$: *tableau de noeuds*
{Les trois graphes reçus sont transmis par la procédure principale d'analyse}
{A l'issue de cette fonction, le graphe caractrisation est enrichi des nouveaux noeuds créés.}
{Ces des nouveaux noeuds créés sont aussi stockés dans le tableau noeudsCrees.}
{Le tableau soucheRecolte sert de support pour savoir à partir de quels noeuds s'effectue la récolte.}
Sorties: Une valeur booleenne est retournée : $VRAI$ si une projection a eu lieu, $FAUX$ sinon.

◦ Pour chaque noeud catégoriel NG de la grammaire,
◦ Construire un arbre AG ayant pour racine NG et pour sous-arbre une réplique du sous-arbre de NG allant jusqu'aux noeuds de type *propriété*.
◦ Récolter dans SoucheRecolte l'ensemble ListeProjetable des noeuds ayant pour référent grammatical un descendant possible d'un noeud de l'arbre AG.
◦ Répartir ListeProjetable en sous-listes non croisées (chaque sous-liste est une clique du graphe de caractérisation respectant le principe de projectivité).
◦ Pour chacune de ces sous-listes, créer une réplique AC de l'arbre A et raccorder les éléments de cette sous-liste à l'arbre AG.
◦ Evaluer la satisfaisabilité de AG. Pour cela, il faut éventuellement créer et évaluer un ensemble de nouvelles contraintes (les contraintes que nous avons nommées *lacunaires*)
◦ Si AG est valide et qu'aucune répétition verticale n'est constatée dans AG et ses sous-arbres, alors on conserve AG.
◦ Sinon on détruit AG.
◦ Continuer la récolte.

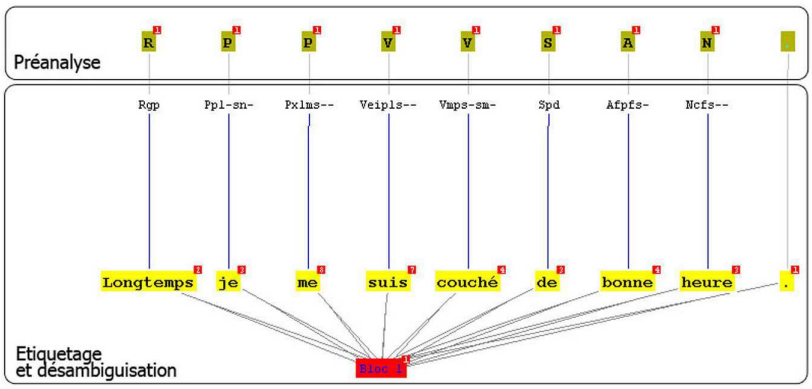
Fin Fonction RecolteSouche

11.7 Exemples commentés

Afin de mettre en évidence le fonctionnement de l'analyseur, nous avons choisi de commenter un premier exemple simple. La phrase "Longtemps je me suis couché de bonne heure" est analysée avec la grammaire choisie pour la campagne d'évaluation EASY. Pour faciliter la compréhension, nous ne montrerons pas à pas que certains phénomènes correspondant à l'analyse d'une partie de la phrase. En particulier, nous voulons suivre en détail l'analyse des mots "de bonne heure" qui doivent former un syntagme prépositionnel, c'est pourquoi nous donnons aussi un extrait de la grammaire qui montre l'agencement des propriétés décrivant le syntagme prépositionnel (GP). La figure 53 montre la grammaire des GP, ainsi que l'état initial du graphe d'analyse. Lorsque la phrase étiquetée est chargée par le programme, un graphe de caractérisation minimal est généré. Chaque mot est représenté par un noeud. Pour chacun de ces nœuds, un nœud *étiquette* lui est rattaché. Il est possible dès cette étape de ne pas choisir une seule étiquette par mot et de conserver les ambiguités morphosyntaxiques, mais cela aurait alourdi notre exemple. A chaque étiquette, une préanalyse (étape préparatoire) va permettre d'associer un nœud spécifique qui correspond exactement à un double d'un des nœuds de la grammaire. Du point de vue algorithmique, nous emploierons l'idée du dédoublement (de duplication) de nœuds du graphe, mais du point de vue pratique, lors de l'exécution du programme, cette duplication sera obtenue par instanciation d'une copie (clone) de l'objet considéré. Par exemple, le premier mot [Longtemps] est rattaché à l'étiquette [Rgp], qui est rattachée à son tour à un nœud [R] directement créé à partir du nœud catégoriel [R] provenant de la grammaire. Ceci suppose que chaque catégorie (même considérée comme terminale) pouvant être utilisée durant l'analyse, doit être spécifiée dans la grammaire. Grâce à cette préanalyse, on dispose d'un premier niveau de caractérisation qui permet de connecter le graphe de caractérisation avec le graphe grammatical. Ceci permettra à l'algorithme d'analyse d'effectuer pas à pas une remontée de la grammaire (au fil des liens de référence entre catégories et entre contraintes) et de construire peu à peu le graphe d'analyse qui reflétera ce parcours.

Lorsque la préanalyse est achevée, l'algorithme de dissémination/caractérisation peut entrer en jeu. Les nœuds créés lors de la préanalyse constituent ce que nous appellerons la *souche de dissémination initiale* et la *souche de récolte initiale* La figure 54 montre la première étape de dissémination (nous nous intéressons en particulier aux trois nœuds [S], [A] et [N] correspondant aux mots "de bonne heure" : chaque élément du graphe de caractérisation a "*le souvenir*" du nœud de la grammaire dont il est le double. Concrètement, chaque nœud du graphe de caractérisation est lié par un pointeur au nœud du graphe grammatical dont il est le double. Inversement, chaque nœud de la grammaire qui est dédoublé (pour que son double soit intégré au graphe de caractérisation) "se souvient" de tous ses doubles. Concrètement, un tableau de pointeurs lui permet de retrouver tous ses doubles. Toute duplication se fait en mémorisant le lien de duplication grâce à un double pointage du réplicat vers le nœud grammatical et du nœud grammatical vers l'ensemble de ses réplicats connus. Pour faire fonctionner la dissémination, il suffit de partir de chaque nœud de la souche de dissémination, de rechercher dans la grammaire l'original de ce nœud et de remonter les liens qui lient ce nœud à d'autres nœuds de la grammaire. Dès qu'un lien remontant est parcouru, on se retrouve face à un nouveau nœud prêt à être ajouté à la caractérisation. Cette routine s'exécute pas à pas, en remontant

Graphe de caractérisation au début de l'analyse



Grammaire de Propriétés du Groupe Prépositionnel

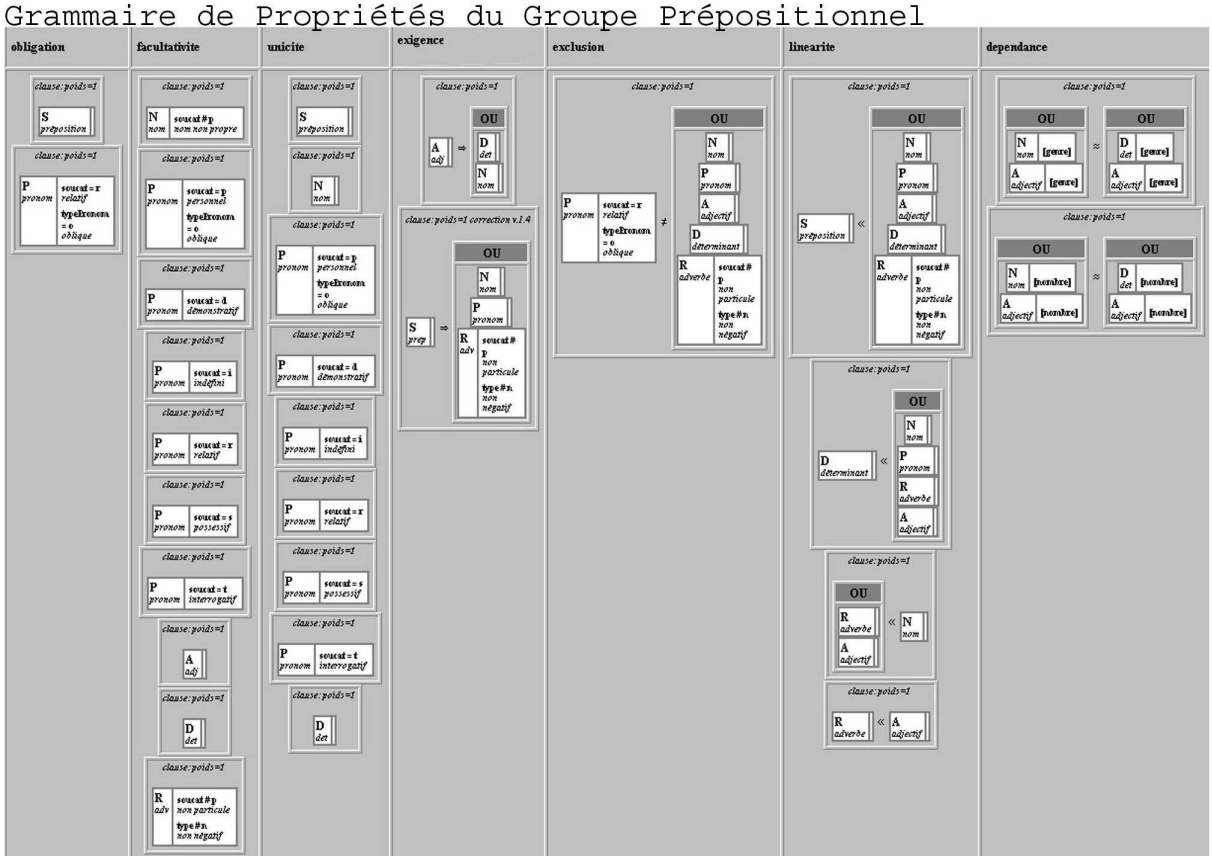


FIG. 53 – Exemple d'analyse avec SeedParser : préanalyse

d'un niveau à la fois. Pour faciliter la lecture des liens remontants dans la grammaire, nous avons préféré conserver la mise en forme tabulaire plutôt que la représentation sous forme de graphe de la grammaire. A partir de la souche de dissémination initiale, chaque nœud de cette souche étant de type catégoriel, les seuls liens remontants dans la grammaire sont les liens de référence que l'on a pu observer dans les schémas de modèle pour les grammaires de propriété. Un lien de référence permet simplement de retrouver la définition de catégorie [S] à partir de tous les endroits de la grammaire qui font référence à la catégorie [S]. Ces endroits peuvent être aussi nombreux qu'on le souhaite. La définition de catégorie doit par contre être unique. La première étape de dissémination part donc de la souche de dissémination. Pour chaque nœud de cette souche, on va rechercher son original dans la grammaire. De cet original, qui est forcément de type catégoriel (lors de la première étape de dissémination), on remonte les liens qui sont forcément des liens de référence intercatégoriels. On arrive ainsi aux nœuds référents (ceux qui interviennent directement au sein des propriétés). Dans la figure 54, ces nœuds référents (catégories qui font référence à des catégories) sont entourés d'un cercle noir. Pour chacun de ces nœuds, on va créer un double dans le graphe de caractérisation et le mettre en relation avec le nœud de la souche qui lui correspond. Cela donne cette dissémination en étoile autour des nœuds de la souche.

Lorsque la première étape de dissémination s'achève, si des nœuds ont été créés, on remplace la souche de dissémination par l'ensemble des nœuds nouvellement créés et on recommence à remonter les antécédants de ces nœuds dans la grammaire, et ainsi de suite, jusqu'à rencontrer des nœuds qui ne peuvent pas participer à la dissémination (nœuds décrivant les types de propriétés). Dans la figure (55), on met en évidence le résultat des étapes de dissémination qui suivent la première étape : Le nœud référençant un [N] dans la propriété d'obligation, par exemple, va se retrouver associé à un nœud de type relationnel (une contrainte d'obligation). Le nœud référençant un [S] dans la propriété de linéarité, par exemple, va se retrouver associé à un nœud de type relationnel (une contrainte de linéarité), qui sera elle même parente d'un autre nœud de type relationnel (un opérateur booléen binaire "OU") qui sera placé au dessus des nœuds [N] et [A] dans la même contrainte. L'étape de dissémination ne se contente pas de construire des nœuds les uns au dessus des autres en les reliant par des arcs, mais elle réalise l'analyse des contraintes de la grammaire, ceci en accord avec la sémantique des propriétés de la grammaire. La dissémination d'une contrainte d'obligation au dessus d'un [N] suppose que l'on va vérifier l'évaluabilité et la satisfaisabilité de cette contrainte pour ce nœud précisément. Il en va de même avec les autres propriétés unaires (facultativité, unicité) ou binaires (linéarité, exclusion, exigence, dépendance). La dissémination réalise l'évaluation des contraintes évaluables et concrétise cette évaluation par une création de nœuds dans le graphe de caractérisation. Les nœuds de type relationnel ont la particularité de pouvoir être associés à une mesure de satisfaction (variant de 0 à 100). Si la contrainte est évaluable, un nœud est créé et lorsque le nœud est créé, on calcule son taux de satisfaction, toujours en accord avec la sémantique des propriétés. Dans la figure (55), les nœuds non évaluables de la grammaire sont représentés par une croix noire. Dans le cas d'une non-évaluabilité, la dissémination s'arrête de fait et continue jusqu'à ce qu'il n'y ait plus aucune création de nœuds. Par exemple, pour être évaluée, une contrainte de linéarité entre un adverbe et un adjectif a besoin d'un adverbe et d'un adjectif présents dans l'input). On ne s'intéresse à l'adverbe situé au début de la phrase dans l'exemple fourni. Dans la véritable analyse, beaucoup plus de phénomènes entrent en jeu et rendent les choses moins lisibles.

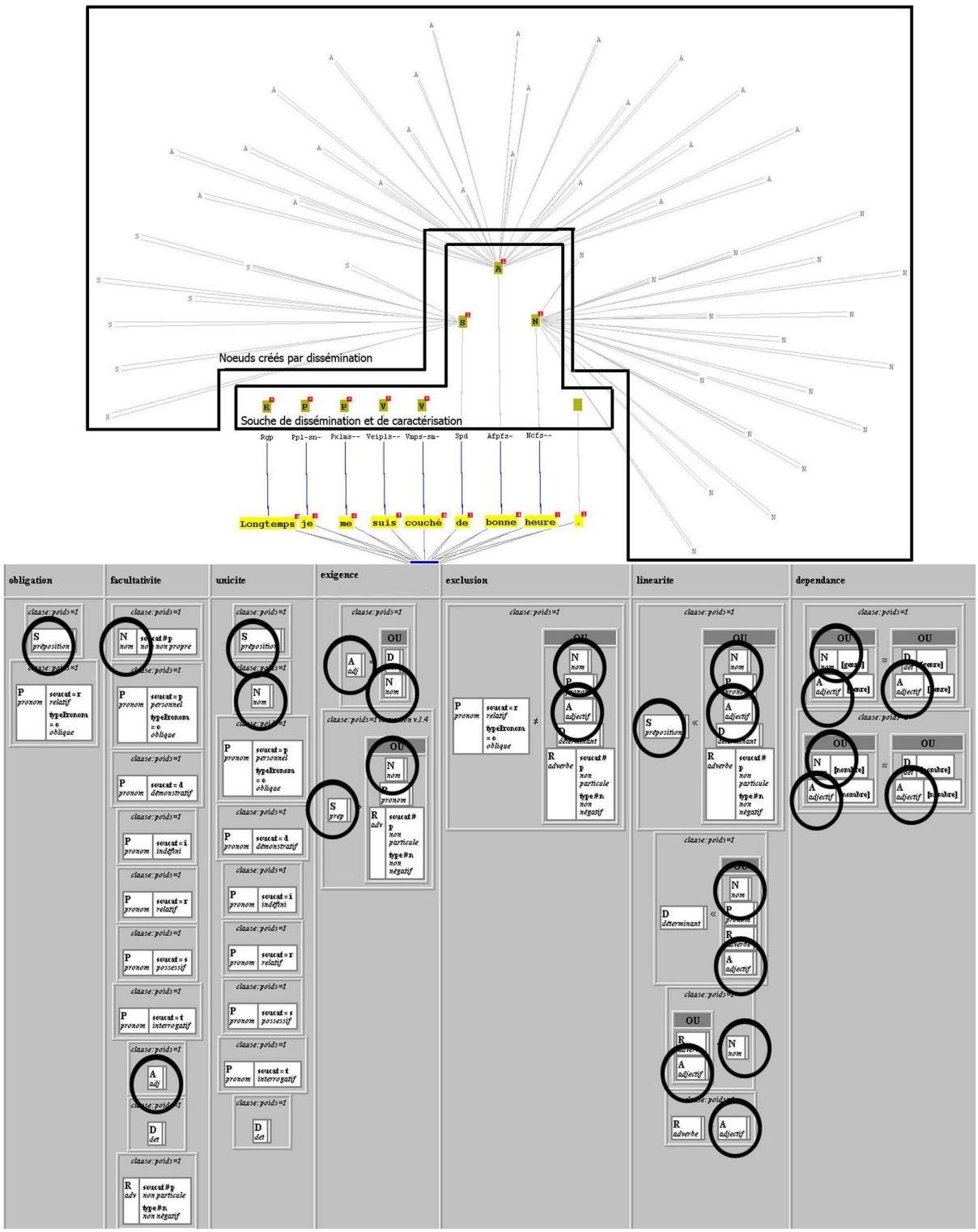


FIG. 54 – Exemple d'analyse avec SeedParser : première étape de dissémination

A la fin de la dissémination, toutes les contraintes évaluables ont donné lieu à la création de nœuds associés à une mesure de leur taux de satisfaction. Cette "soupe de contraintes" est le lieu où la phase de récolte va puiser des éléments et les regrouper pour former des constructions valides.

11.8 Déterminisation

Le module d'analyse SeedParser est doté d'une routine supplémentaire qui fait appel à une classe d'objets facilement redéfinissable (pour être modifiée à volonté dans un logiciel qui utiliserait le module) servant à rendre déterministe le résultat de l'analyse. Jusqu'à présent, nous avions laissé de côté la question du déterminisme en disant que d'une part l'ambiguïté syntaxique était préservée par le parseur, et que d'autre part, un filtrage en cours d'analyse était permis simplement par le paramétrage des seuils de densité de satisfaction. Mais même en obligeant l'analyseur à ne conserver que les constructions dont la satisfaction serait égale à 100%, une ambiguïté peut subsister à la fin de l'analyse. Dans certaines situations, nous voulons pouvoir ne sélectionner qu'une seule proposition, alors jugée comme *la plus pertinente*. Cette situation s'est présentée lors de l'évaluation de l'analyseur pour la campagne EASY (voir la prochaine partie). Ceci nous éloigne beaucoup des objectifs initiaux qui consistaient à rendre compte de l'ambiguité. Cependant, certains outils de plus haut niveau requérant moins de finesse peuvent demander une déterminisation de la sortie de l'analyseur. SeedParser offre donc une classe d'objets destinée à sélectionner une analyse et une seule parmi celles qu'il propose normalement. Cet objet est construit et appelé par le programme SeedParser à la fin de chaque phrase. Si le programme appelant le souhaite, il peut récupérer le résultat déterminisé issu de cet objet, ou encore créer sa propre classe dérivée de la première et écrire son propre mécanisme de déterminisation. Ce type d'analyse déterministe a été employé lors de la campagne EASY pour produire des résultats non ambigus, comme le requerrait le protocole d'évaluation. Par défaut, SeedParser propose une sélection de la sortie la plus pertinente avec une technique assez coûteuse en nombre d'opérations, mais optimale dans le sens où elle choisira de conserver les structures qui maximisent localement et globalement la mesure de densité de satisfaction. Pour fonctionner, cette technique nécessite le repérage préalable des catégories ambiguës au sein d'une analyse. L'idée d'ambiguité rejoint ici encore le principe de projectivité. Si deux structures ont une partie de graphe de caractérisation en commun, elles sont dites ambigues. Des catégories entrant en conflit pour un ou plusieurs de leurs constituants forment ce que nous appelons une clique d'ambiguïté. Une phrase peut contenir plusieurs cliques d'ambiguïté. L'idée maîtresse de l'algorithme réside dans la maximisation clique par clique des fonctions de mesure de densité de satisfaction. La fonction à maximiser tient compte non seulement de la densité de satisfaction de chaque groupe de constituants, mais aussi de son nombre total de propriétés évaluées et satisfaites, ceci afin de mettre en avant des structures un peu moins satisfaites, mais plus complexes, face à des structures très satisfaites, mais dotées de peu de propriétés. On trouve ainsi la meilleure proposition pour chaque clique. C'est le repérage des cliques qui pose un problème de complexité, car il nécessite un contrôle des constituants tel que la combinatoire associée est exponentielle. Les cliques étant assez souvent petites, nous avons conservé cette technique, en lui adjoignant

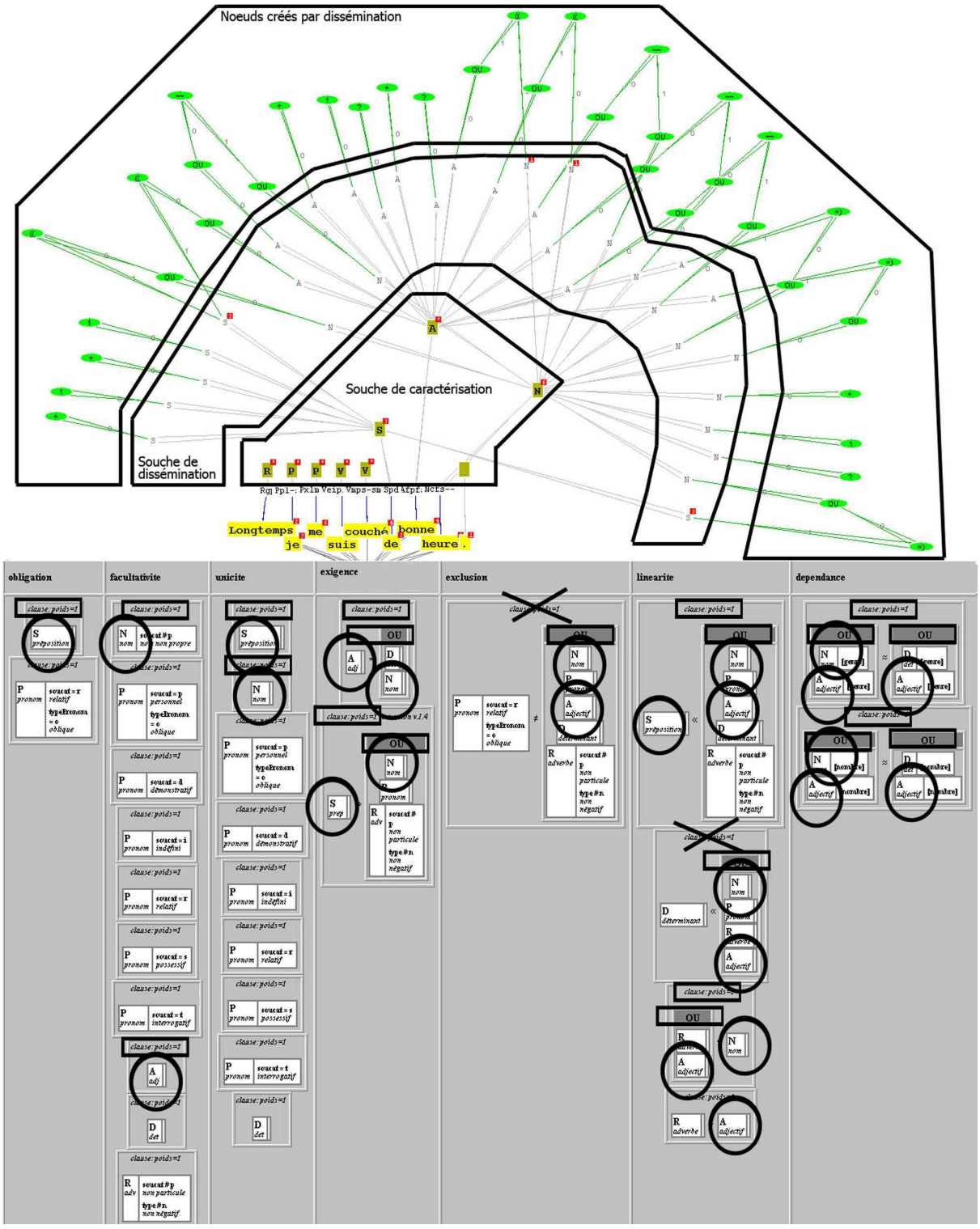


FIG. 55 – Exemple d'analyse avec SeedParser : autres étapes de dissémination

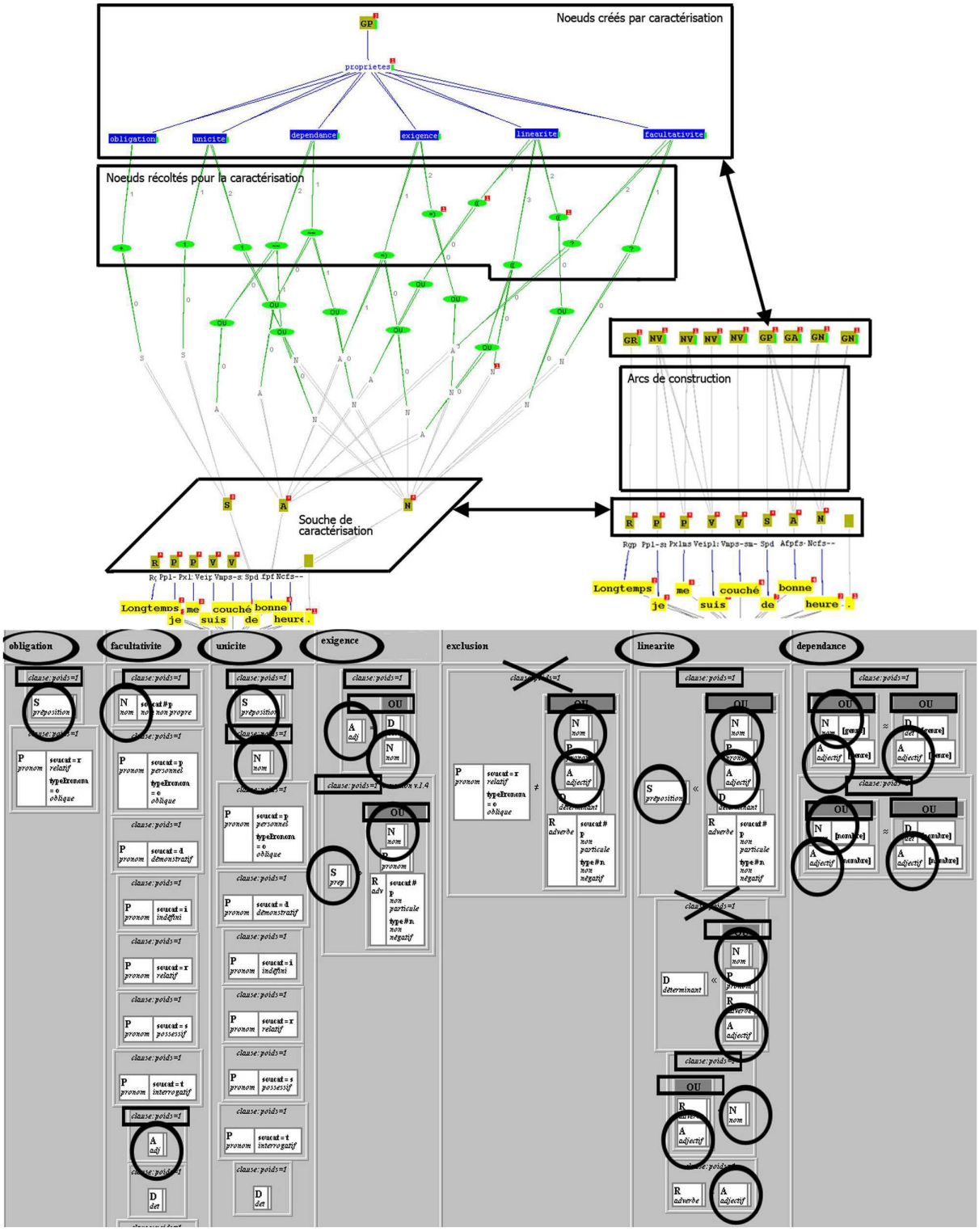


FIG. 56 – Exemple d'analyse avec SeedParser : étape de récolte / caractérisation

des heuristiques de coupure dans les ensembles trop grands. Il en résulte une déterminisation optimale autant que possible : en effet, certaines constructions optimales pour la fonction de densité de satisfaction restent en conflit avec une mesure égale. Dans ce cas, c'est le hasard qui déterminera la structure choisie. D'autres techniques de déterminisation étaient possibles, notamment celles qui maximisent le nombre de constituants dans les constructions, afin de favoriser les grandes couvertures. Cette dernière technique a été employée avec l'analyseur non déterministe profond présenté plus haut avant SeedParser, pour lui permettre de participer aussi aux tests de la campagne EASY. Il est vrai que la déterminisation nous éloigne des besoins linguistiques et nous ramène près des nécessités informatiques d'économie en temps de calcul. C'est pourquoi nous laissons une porte ouverte à l'éventuel utilisateur afin qu'il choisisse ce qui convient pour le traitement qu'il désire effectuer. Dans les problèmes d'évaluation qui se poseront dans la prochaine partie, la question de la déterminisation est peu étudiée, même si elle mériterait un approfondissement. Ceci est dû au fait qu'il nous manque encore les moyens techniques pour distinguer si des erreurs sont commises par les techniques de déterminisation, ou bien si elles sont dûes aux algorithmes d'analyse, ou encore si elles découlent de l'emploi de grammaires ou de spécifications sémantiques sensiblement différentes.

11.9 Conclusion

Nous avons réuni dans le même outil aussi bien les qualités d'un analyseur superficiel et déterministe que celles d'un analyseur profond et non déterministe. A cela s'ajoute le travail effectué pour approcher le traitement d'analyse à granularité variable. L'ensemble des outils et des ressources doit être évalué. C'est le propos de la prochaine partie. Précisions dès à présent certains problèmes qui apparaitront lors de l'évaluation : La technique d'analyse développée pour SeedParser est assez lente, ce qui ne découle pas des algorithmes, mais des temps de parcours de certaines structures. En effet, pour vérifier une propriété, il faut pour l'instant se référer au graphe de spécification sémantique de cette propriété, puis construire une réelle analyse de cette spécification, vérifier par des techniques de comparaison de graphes, que la propriété est satisfiable, et recommencer pour chaque autre propriété. Une solution à ce problème se trouve dans l'idée de compiler (une fois les graphes chargés) les procédures de calcul de satisfaisabilité sous forme de code rapide à l'exécution. Toutefois, comme la formalisation des Grammaires de Propriétés n'est pas encore achevée, que les modèles de données et les structures correspondantes sont encore appelés à évoluer, nous avons abandonné cette possibilité tant qu'une formalisation stable ne sera pas choisie. Il manque encore beaucoup de choses pour traiter la propagation des traits par exemple. Pour SeedParser, il faudrait pratiquement compiler du code Java autogénéré à partir d'une sémantique et d'une grammaire. Ceci est possible en utilisant la réflexivité de Java et présente sans doute l'évolution la plus probable de SeedParser dans ses développements ultérieurs. Le gain en efficacité serait substanciel à l'endroit le plus critique de l'algorithme d'analyse, ce qui pourrait rendre acceptables les temps d'analyse pour des corpus tels que ceux de la campagne EASY (voir la prochaine partie).

Quatrième partie

Evaluation des outils

Toutes les techniques qui ont été présentées jusqu'à présent doivent être évaluées, qu'il s'agisse de vérifier la fiabilité des algorithmes ou qu'il s'agisse de vérifier la qualité des résultats produits par les outils. Evaluer un ensemble d'outils suppose de s'être au préalable interrogé sur le sens de l'*évaluation*. La qualité des résultats obtenus après un traitement effectué par les outils que nous voulons valider dépend étroitement de la conception des outils. Nous ne pouvons pas nous contenter d'observer ces résultats. Il faut non seulement définir une technique d'évaluation pour les sorties des outils, mais aussi définir des critères permettant de différencier les qualités propres aux algorithmes employés, de repérer leurs défauts, etc. Avant de procéder aux évaluations, nous présentons donc ci-dessous les différents critères qu'il nous a paru important d'étudier, en séparant ceux qui concernent les éléments informatiques des éléments linguistiques.

En ce qui concerne les outils, indépendamment des résultats, nous devons vérifier que leurs algorithmes opèrent suffisamment rapidement. Pour cela, nous pouvons calculer ou mesurer la complexité des traitements en fonction de la taille des données. Dans les outils de bas niveau, tels que le dictionnaire et le segmenteur, la complexité peut aisément se calculer, mais dans le cas de l'étiqueteur morphosyntaxique ou des analyseurs syntaxiques, la complexité algorithmique devient trop dépendante de l'entrée, ce qui rend quasiment impossible un calcul de complexité théorique. Seule l'observation de ces derniers modules permet de mesurer des complexités moyennes et d'interpréter les cas limites afin de détecter les situations d'explosion combinatoire.

La question d'évaluer des analyseurs (même superficiels) est un problème en soi. À la différence de l'étiquetage lexical (POS-tagging), beaucoup d'aspects peuvent changer d'un système à l'autre, y compris les sorties de l'analyseur elles-mêmes.

D'une manière générale, évaluer un système consiste à comparer pour une entrée donnée sa sortie à un résultat de référence, normalisé. Dans le cas de l'analyse syntaxique, la référence est un treebank, la comparaison se fait en comparant les parenthésages respectifs de l'analyseur et du Treebank. Ceci nécessite d'abord la disponibilité d'un treebank (une telle ressource existe seulement pour peu de langues) et signifie également que l'analyseur doit produire le même type d'information que celui du corpus de référence. Ce point peut être problématique. D'abord, parce que le parenthésage n'est pas totalement libre de toute théorie, ensuite, parce qu'une telle ressource indique habituellement une seule solution. Enfin, comme expliqué ci-dessus, le parenthésage n'est pas la seule information que nous voudrions évaluer.

De plus, il nous paraît intéressant de ne pas limiter une évaluation à la seule comparaison des différentes sorties : il est également nécessaire, afin d'interpréter une telle comparaison, de donner quelques indications sur les ressources et les techniques employées par le système. Par exemple, il est important d'avoir des indications sur :

- La couverture lexicale : nombre d'entrées, représentation (propriétés lexicales).
- La couverture syntaxique : nombre de catégories, différents phénomènes syntaxiques.
- La stratégie d'analyse : robustesse, efficacité

Nous proposons ici d'extraire des informations d'évaluation à partir d'une technique de comparaison. En d'autres termes, nous montrons que comparer différents analyseurs, à condition que la méthode soit systématique, permet dans certains cas de donner des éléments d'évaluation.

Pour conduire l'évaluation des outils dans le cadre assez large que nous venons de définir, nous allons tout d'abord considérer les mesures de complexité des algorithmes, puis nous nous intéresserons plus en détail aux techniques d'évaluation avec ou sans données de référence. Nous introduirons alors un outil de multiplexage permettant de réaliser différentes comparai-

sons et en outre de signaler les éléments caractéristiques de différentes techniques.

Chapitre 12

Evaluation de la complexité des algorithmes

12.1 Complexités mesurées

Nous avons comparé les complexités des quatre analyseurs syntaxiques présentés dans la partie précédente. La mesure s'appuie sur un décompte systématique, instruction par instruction, des opérations effectuées par les analyseurs au fil de l'analyse. Les programmes dont nous avons calculé la complexité ont pour entrée des corpus très variables (en taille, en qualité, en contenu) et sont dotés d'algorithmes sensibles au contexte, ce qui rend quasiment impossible un calcul de complexité moyenne. Celle-ci doit donc plutôt être mesurée. Comme l'indiquent les résultats présentés ci-dessous, les complexités progressent de façon croissante, de l'analyseur superficiel aux analyseurs profonds en passant par celui qui utilise une forme compilée de la grammaire.

Dans les statistiques données ici, nous avons dû choisir une échelle permettant une représentation assez facilement comparable des courbes de complexité. Nous sommes à la fois contraints par le besoin de comparer visuellement les courbes, d'où le choix d'une échelle logarithmique même pour l'algorithme le plus simple, et par les questions d'échelle de grandeur. L'instruction pour le plus simple et le million d'instructions pour l'algorithme le plus complexe.

12.2 Complexité de l'analyseur superficiel

L'algorithme Chink/Chunk est une manière simple mais efficace de détecter les frontières syntaxiques. Comme l'indique la figure 57, dans les cas moyens, optimaux et pires, pour M phrases, chaque phrase se composant de Nm mots, sa complexité a un ordre de M*Nm*Constante. C'est-à-dire une complexité linéaire.

12.3 Complexité de l'analyseur intermédiaire

Avec l'algorithme d'analyse superficielle, nous pouvons détecter et annoter plus de données syntaxtiques et hiérarchiques : dans les cas moyens, pires et meilleurs, pour M phrases, chaque phrase se composant de Nm mots ; pour un ensemble de C catégories précompilées, sa

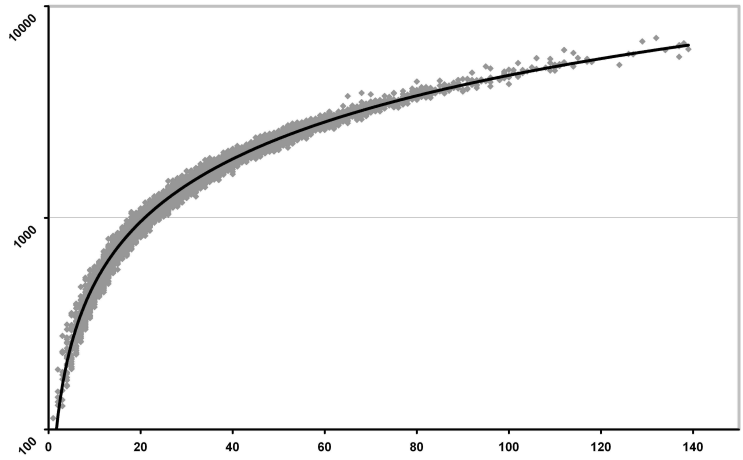


FIG. 57 – Instructions / nombre de mots pour Chink/Chunk (échelle logarithmique)

complexité est de l'ordre de M*C*(Nm^2+Nm)*Constante. C'est-à-dire une complexité polynomiale (voir la figure 58).

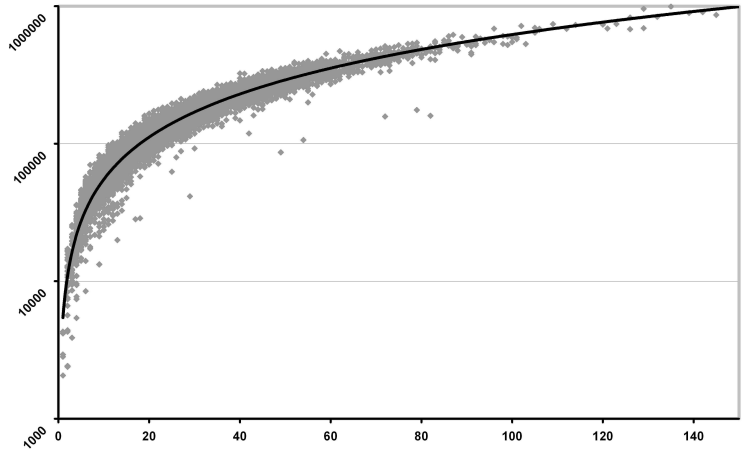


FIG. 58 – Instructions / nombre de mots pour l'analyseur superficiel (échelle logarithmique)

12.4 Complexité de l'analyseur profond

Contrairement aux deux autres algorithmes, nous observons pour l'analyseur profond présenté au chapitre 10, section 10.3, une plus grande dispersion des résultats.

La complexité théorique de cet algorithme est exponentielle, mais son fonctionnement et sa manière de d'être contraint en permanence par la grammaire lui confèrent un fonctionnement moyen réellement polynômial (figure 59), même si l'analyse de deux phrases de taille identique pourra donner des temps d'analyse très différents. Ainsi, si Nm est le nombre de

mots d'une phrase, la meilleure évaluation de sa complexité correspond à un polynôme de degré 2,4 ($Nm^{2,4}$*Constante).

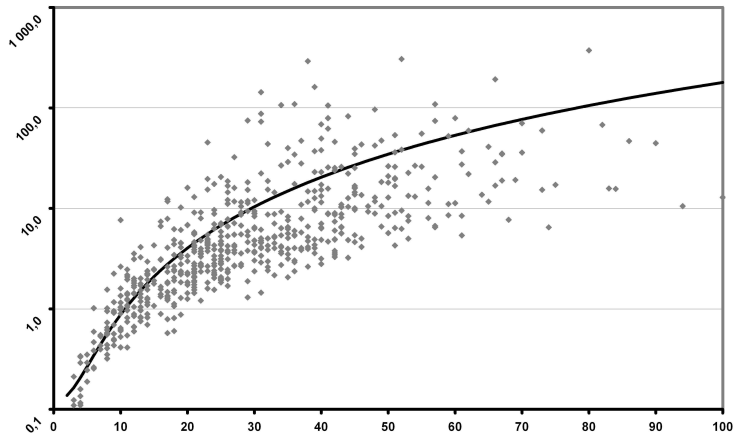


FIG. 59 – Million d'instructions / nombre de mots pour l'analyseur profond (échelle logarithmique)

12.5 Complexité de l'analyseur à granularité variable

De la même façon que l'analyseur précédent, l'analyseur à granularité variable a une complexité moyenne réellement polynômiale, même si l'analyse de deux phrases de taille identique pourra donner des temps d'analyse très différents.

Si Nm est le nombre de mots d'une phrase, la meilleure évaluation de sa complexité correspond à un polynôme d'ordre 2,3 ($Nm^{2,3}$*Constante) (voir la figure 60).

On remarquera que le degré de ce polynôme est légèrement plus petit que celui de l'analyseur précédent, mais le coefficient constant est, lui de l'ordre de 10 fois plus grand. En effet, le coût en instructions pour permettre la modularité et de l'indépendance entre le programme et la grammaire se situe dans un ensemble assez constant d'accès aux structures (graphes) représentant la grammaire et sa sémantique. Ces opérations atomiques sont assez nombreuses même pour un nombre restreint de mots à analyser. Une optimisation est prévue de façon à précompiler le graphe grammatical en un ensemble d'informations plus aisément accessibles. Cette compilation a été évitée jusqu'à présent étant donnée l'évolution constante des structures de données depuis le début du développement de cet analyseur.

12.6 Conclusion

Les complexités mesurées pour les quatre analyseurs mettent en évidence une gradation entre l'analyseur superficiel et les analyseurs profonds. Le fait de dissocier les données (la

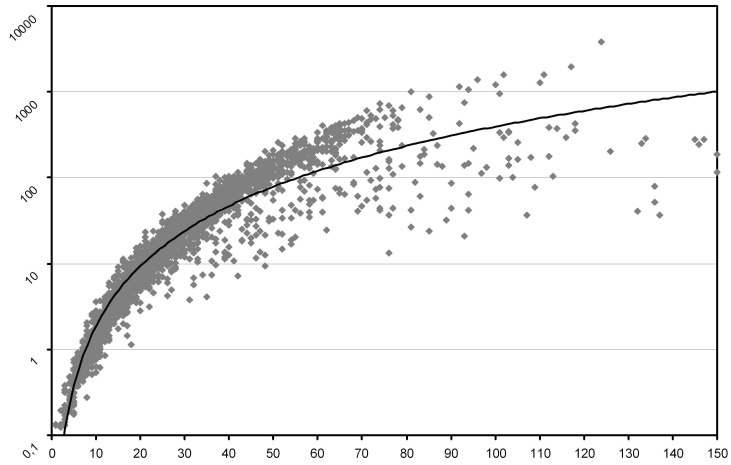


FIG. 60 – Million d'instructions / nombre de mots pour l'analyseur SeedParser (échelle logarithmique)

grammaire et sa spécification sémantique) et le programme (comme dans le dernier analyseur) a une conséquence assez coûteuse en instructions d'accès aux structures de données. En effet, ces structures sont d'autant plus complexes que leur modèle de données est abstrait. Des optimisations pourraient être envisagées afin de réduire (en compilant au maximum les informations critiques) les temps d'accès à ces données.

Une des critiques les plus courantes à l'égard des formalismes symboliques récents basés sur les contraintes réside dans l'idée que ces formalismes donnent systématiquement des algorithmes exponentiels. Nous avons débattu cette question au fil des chapitres précédents, en concluant que la complexité théorique des programmes basés sur les Grammaires de Propriétés devait être exponentielle voire factorielle, mais il manquait une confirmation mesurable.

Les algorithmes donnés pour le programme SeedParser ont de fait une complexité théorique factorielle dans le *pire des cas* (pour une grammaire extrèmement récursive, avec une entrée très complexe et très mal formée). Il s'avère par contre que ce *pire des cas* ne se produit pas : en effet, les phrases analysées sont toujours écrites ou transcrites dans le souci d'être un tant soit peu comprises. Les phrases obéissent donc à un ensemble de règles qui croisent celles des grammaires que nous développons. Un ensemble suffisant de contraintes est donc pratiquement toujours satisfait, ce qui donne lieu à la projection de catégories. Le pire des cas se produirait si l'ensemble des contraintes satisfaites donnait lieu à une projection de catégories nombreuses et ambiguës entre elles. Dès que des portions de phrases analysées sont assez peu ambiguës, l'analyse converge, et réduit d'autant l'espace de calcul autour des constructions non ambiguës.

Le dernier analyseur est de plus doté de stratégies de réanalyse si trop peu de contraintes sont satisfaites au dessus d'un certain seuil de *densité de satisfaction*. De façon incrémentale, l'analyseur va tendre vers une caractérisation *suffisamment informative*, au prix de plusieurs tentatives. Ces tentatives ont un coût qui ne nuit pas à la complexité du calcul. Au contraire, puisque la somme des complexités des analyses reste un polynôme, dont le degré est d'autant plus petit que les seuils de tolérance fixés pour limiter la densité de satisfaction sont élevés. L'évaluation de la complexité fournit de précieux renseignements quant au rendement des

programmes dans des situations d'usage courant. Le dernier analyseur s'avère encore trop lent, ce qui est imputable à son état de développement plus qu'à une impossibilité théorique. L'approche que nous appelons *à granularité variable* conserve donc notre intérêt.

Chapitre 13

Evaluation qualitative sans référence

13.1 Un multiplexeur pour l'évaluation comparative des parseurs

L'idée de retrouver les mêmes frontières ou parenthésages par le biais de différents analyseurs syntaxiques nous a conduits à imaginer un programme susceptible de fusionner, à l'aide de paramètres, les sorties de différents analyseurs. Un des buts sous-tendus par cette technique est l'élimination des parenthésages erronnés et la conservation des meilleurs. La figure **??** présente le schéma du fonctionnement de cette technique.

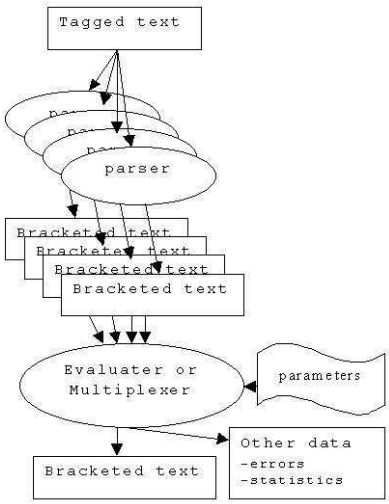


FIG. 61 – Schéma général de fonctionnement du multiplexeur

Le programme que nous appelons multiplexeur (multiplexer en Anglais) est susceptible de jouer le rôle de programme d'évaluation (evaluater en Anglais) si l'une de ses entrées était un corpus de référence. Nous verrons plus bas qu'il permet d'évaluer de façon comparative certaines qualités propres aux analyseurs dont nous disposons.

Cette technique de multiplexage devant traiter des *ensembles* de frontières, les paramètres employés seront de deux sortes :

– Des opérateurs ensemblistes (union, intersection, complément)

– Des poids donnant plus ou moins d'importance à tel analyseur plutôt qu'à tel autre pour une catégorie donnée. Afin d'éviter les erreurs connues d'un analyseur ou au contraire donner la priorité à un analyseur.

En l'absence de corpus de référence, nous ne pouvons que comparer les sorties de différents analyseurs et effectuer une expertise manuelle sur les résultats afin d'affiner les paramètres du multiplexeur. On peut aisément imaginer que ce programme fonctionnerait à merveille si nous disposions d'un tel corpus de référence. Nous pourrions aussitôt injecter ce corpus en entrée du multiplexeur avec un poids maximal pour toutes ses catégories et nous obtiendrions une évaluation des qualités et des défauts de nos analyseurs.

Mais tous les paramètres utilisables par ce programme ne peuvent pas être trouvés sans une bonne évaluation des sorties de chaque analyseur. Ces deux besoins (paramètres et évaluation) sont si proches qu'il est difficile de les distinguer.

C'est seulement par une méthode empirique d'évaluation et de paramétrage, puis de réévaluation et de reparamétrage (de manière rétroactive allant de l'évaluation au paramétrage puis des paramètres affinés à une nouvelle évaluation), que nous pouvons réaliser un programme d'évaluation valide pour un jeu d'analyseurs donnés.

A la lecture de ce qui précède, il parait fastidieux de travailler avec un tel programme. Retenons cependant que seul le paramétrage doit être réalisé manuellement. Le reste du travail, à savoir l'extraction des frontières communes, est automatique.

L'intérêt de ce programme apparaît alors si on se place du point de vue de la problématique initiale : nous ne voulons pas exclure des analyses apparemment contradictoires, mais au contraire les traiter comme des interprétations en concurrence pour un même texte. C'est exactement ce que fait ce programme. Dans cette optique, l'avantage de ce programme est de fournir une sélection de granularité parmi plusieurs techniques. Une réponse directe à notre problématique.

L'évaluation présentée ici repose sur deux expériences menées sur un corpus de 13236 phrases étiquetées extraites du journal Le Monde, Nous les devons au projet CLIF (voir [Abeillé, 2003], [Abeillé *et al.*, 2003] et http ://www.talana.linguist.jussieu.fr).

Deux types d'étiquetage des catégories lexicales ont été employés : l'original (projet CLIF) et celui obtenu en utilisant l'étiqueteur automatique WinBrill (voir [Brill, 1992] et [Lecomte and Paroubek, 1996]).

Ces choix découlent d'une volonté de réaliser nos expériences d'une part sur des phrases courantes et d'autre part sur des textes étiquetés automatiquement.

L'avantage de disposer d'un corpus étiqueté (au niveau lexical) par des experts a fourni l'idée de la première expérience.

Afin de mieux appréhender ce qui peut être déterminé par notre multiplexeur, nous n'utiliserons comme paramètres que l'opérateur d'intersection et le même poids pour chaque sortie d'analyseur. D'autres travaux proposeront des analyses plus fines.

Dans les expériences qui suivent, les algorithmes proposés à l'étude sont :

A1 l'analyseur Chink/Chunk décrit plus haut

A2 l'analyseur superficiel décrit plus haut

A3 un autre analyseur superficiel qui n'est pas détaillé ici. Cet analyseur utilise aussi les Grammaires de Propriétés

A4 l'analyseur profond décrit plus haut.

13.2 Première expérience

La première expérience se propose de comparer les **frontières des chunks**, en fonction de **l'algorithme** et de **l'étiquetage**. Pour ce faire, nous avons simplifié massivement la sortie générée par nos programmes A2, A3 et A4 afin de n'en conserver que les chunks, sacrifiant par là les données linguistiques et les informations hiérarchiques pour ne disposer que d'analyses directement comparables avec celles de A1.

La question à laquelle cette expérience doit répondre est la suivante : L'étiquetage morpho-syntaxique influence-t-il l'analyse syntaxique ?

Le protocole expérimental est décrit par la figure 62. Chaque analyseur est déclenché sur deux variantes du même corpus : l'une étiquetée à la main (jeu d'étiquettes CLIF noté Human CLIF dans les figures et tables ci-après) et l'autre obtenue à avec l'étiqueteur et le jeu d'étiquettes BRILL (voir [Lecomte and Paroubek, 1996])). Les données à évaluer constituent donc 8 fichiers correspondant aux 4 analyseurs. Déclencher le multiplexeur sur ces fichiers nous obligerait à confronter deux à deux ces fichiers, soit 28 confrontations à réaliser.

Comme cette expérience n'a pour but que d'évaluer l'influence de l'étiquetage sur l'analyse, nous n'exposerons ici que les résultats de la comparaison de A1 avec A2.

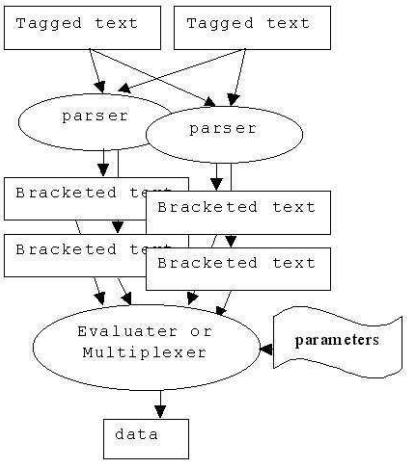


FIG. 62 – Première expérience : évaluation de la sensibilité à la qualité d'étiquetage

La table 24 indique les premières statistiques obtenues : le nombre moyen de mots par chunks varie sensiblement d'un algorithme à l'autre, mais aussi en fonction de la technique d'étiquetage employée.

Algorithme	Mots par chunk
A1 Humain	3.49
A1 avec WinBRILL	3.34
A2 Humain	2.02
A2 avec WinBRILL	1.96

TAB. 24 – Résultats de la première expérience : longueur moyenne des chunks

La table 25 ci-dessous donne le nombre de chunks proposés par un système et trouvés

par un autre système. Chaque multiplexage réalisé (après avoir supprimé toute information hiérarchique des fichiers testés).

1ère évaluation.	A1 Humain	A1 WinBRILL	A2 Humain	A2 WinBrill
A1 Humain	100%	92%	82%	77%
A1 WinBRILL	89%	100%	75%	81%
A2 Humain	50%	47%	100%	82%
A2 WinBRILL	45%	50%	80%	100%

TAB. 25 – Résultats de la première expérience : pourcentages de frontières repérées par une méthode par rapport à celles repérées par une autre.

Cette table peut être lue de la façon suivante :

Les frontières communes à A1 avec un étiquetage humain et A2 avec étiquetage humain, représentent 82% du nombre de frontières fournies par A1

Les frontières communes à A2 avec un étiquetage humain et A1 avec étiquetage humain, représentent 50% du nombre de frontières fournies par A2

Il ressort de cette expérience que les différences de frontières d'un même analyseur pour deux étiquetages sont de 2 à 3 %, ce qui indique que l'étiquetage automatique reste pertinent pour la notion de frontière face à un étiquetage expert.

Ce résultat est éclairé par une autre information : le nombre moyen de mots par chunk (table précédente).

Une seconde conclusion est que les algorithmes sont sensibles à la qualité de l'étiquetage (c.a.d. ils réagissent à la variabilité) : ces résultats indiquent que A1 perd jusqu'à 10% de ses frontières pour un étiquetage non humain et A2 jusqu'à 20%.

Un dernier point est que A1 et A2 ont réellement de 47% à 82% de frontières communes. A la suite de ce qui a été énoncé plus haut, nous pouvons utiliser les frontières communes afin d'harmoniser et de garantir la qualité des différents sorties. C'est ce point que nous discuterons dans la seconde expérience.

13.3 Seconde expérience

La seconde expérience a pour objectif de comparer les trois approches basées sur les Grammaires de Propriétés. Les frontières communes seront comparées catégorie par catégorie. Le schéma de la figure 61 illustre le déroulement de cette expérience. Un même texte étiqueté est donné en entrée des trois analyseurs A2, A3 et A4. Les tables 26 à 31 donnent les résultats obtenus.

Algorithme	A2	A3	A4
Chunks/phrase	15.03	19.04	18.97
Mots/chunk	1.90	1.50	1.50

TAB. 26 – Statistiques de la seconde expérience

Les approches A2 et A3 sont très différentes (48% de frontières communes en moyenne). Ceci est dû en partie aux différences entre les jeux d'étiquettes syntaxiques employés : A3

SN	A2	A3	A4
A2	100%	54%	45%
A3		100%	100%
A4			100%

TAB. 27 – Frontières communes de SN

SV	A2	A3	A4
A2	100%	29%	27%
A3		100%	75%
A4			100%

TAB. 28 – Frontières communes de SV

SA	A2	A3	A4
A2	100%	50%	43%
A3		100%	86%
A4			100%

TAB. 29 – Frontières communes de SA

SP	A2	A3	A4
A2	100%	57%	49%
A3		100%	85%
A4			100%

TAB. 30 – Frontières communes de SP

COORD	A2	A3	A4
A2	-	0%	-
A3		100%	0%
A4			-

TAB. 31 – Frontières communes de COORD

utilise des catégories que A2 ne connaît pas. Plus précisément, les SN, SA, SP et SV ont respectivement 55%, 50%, 57% et 30% de frontières communes.

A3 est plus proche de A4, qui cherche à satisfaire toutes les contraintes (90% de couverture). Les SN, SA, SP et SV ont respectivement 100%, 85%, 86%, 71% de frontières communes.

Un guide de lecture pour ces tables peut se trouver dans le fait que l'algorithme A4 a donné les meilleurs résultats en comparaison à une évaluation experte de 10 phrases. Il vient que la plupart des frontières communes que A4 partage avec A3 et A2 doivent porter un grand poids et doivent être multiplexées avec un opérateur d'intersection.

Un autre information réside dans le fait que A3 connaît des catégories qu'ignorent A2 et A4 (cf. table COORD). Cette connaissance implique que la catégorie COORD de A3 doit être incluse avec un poids de 100% et un opérateur d'union.

Ces résultats de la seconde expérience impliquent deux conclusions :

Les frontières communes nous informent sur l'originalité ou le conformisme d'un analyseur par rapport à un autre.

La simple connaissance de ce que fait un analyseur nous permet de paramétrer les opérateurs ensemblistes et leurs poids.

13.4 Conclusion

L'évaluation sans corpus de référence s'est avérée riche en enseignements pour améliorer nos outils ainsi que nos ressources. La technique de multiplexage nous a paru intéressante et nous souhaitons poursuivre son développement en effectuant une évaluation de ses caractéristiques et de son paramétrage dans le cadre d'un test sur les corpus annotés qui seront disponibles à la fin de la campagne d'évaluation EASY. C'est à partir des corrections effectuées dans nos systèmes grâce à cette première évaluation, que nous avons pu constituer la plateforme existante. L'évaluation de cette dernière sur corpus de référence est abordée par le prochain chapitre.

Chapitre 14

Evaluation qualitative avec corpus de référence

14.1 Evaluation de l'étiqueteur

L'étiqueteur morphosyntaxique de la plateforme LPLSuite a fait l'objet de nombreux remaniements. Son algorithme de désambiguisation est basé sur un apprentissage des probabilités de transition entre catégories, à l'aide d'un corpus de référence. Le premier corpus utilisé était celui du CLIF, annoté selon un jeu de traits proche de Multext. Le problème de l'évaluation s'est très vite posé, car la seule vérification que nous pouvions effectuer consistait à étiqueter ce corpus avec notre outil et à comparer les résultats avec la référence. Comme le corpus de test et le corpus d'apprentissage étaient les mêmes, nous obtenions juste une information sur la pertinence et la finesse de l'apprentissage. En améliorant l'outil, il est rapidement apparu nécessaire de se doter d'autres corpus d'apprentissage et de référence. Le corpus Multitag (issu de la campagne GRACE) s'est avéré utile pour réaliser de nouveaux apprentissages et pour effectuer de nouvelles évaluations. Le jeu de traits GRACE étant plus proche et plus fin de celui de notre désambiguiseur, nous avons beaucoup amélioré les scores et affiné les algorithmes afin de ne pas perdre en robustesse ce que nous avions gagné en qualité. De fait, le désambiguiseur étant basé sur des N-Grammes variables allant de simples bigrammes à des 5-grammes dans certains contextes, de grandes phrases devenaient de moins en moins faciles à analyser. Une première idée a consisté en un simple découpage de l'énoncé en morceaux délimités par des ponctuations. Ainsi, la qualité de la désambiguisation n'en était pas altérée et le parcours de l'espace combinatoire de calcul en était grandement réduit. Une seconde idée basée sur un découpage encore plus petit en chunks délimités par des mots outils non ambigus a permis de réduire définitivement le temps de calcul dans des proportions acceptables. La complexité de l'algorithme d'étiquetetage étant ainsi réglée, nous nous sommes concentrés sur la question de l'évaluation qualitative de ce programme. Les ressources de référence qui ont permis d'évaluer les scores que nous allons fournir sont le corpus CLIF et le corpus MULTITAG. Un programme de calcul de score a été développé afin de comparer nos résultats avec la référence. Le score se calcule en comparant la meilleure proposition de l'étiqueteur (il pourrait avoir de meilleurs scores si nous prenions en compte les autres propositions du désambiguisateur) avec la référence. Le premier problème posé par cette évaluation se situe dans le découpage en tokens. En effet, notre propre lexique ne dispose pas des mêmes tokens que ceux de Multitag ou du CLIF. Les mots composés et certains groupes de mots se sont avérés incomparables.

Une modification de l'étiqueteur était donc nécessaire : nous avons modifié les paramètres de ce programme pour qu'il puisse prendre en compte des énoncés fournis avec des 'tokens imposés'. De ce fait, une réduction de qualité indéniable s'est produite, puisque la variété des entrées de notre lexique n'était plus systématiquement accessible. En présence d'un mot inconnu, l'étiqueteur procède à une désambiguisation à partir d'une liste des catégories les plus probables. Cette liste est très grande et les chances de prédire une bonne catégorie dépendent beaucoup du contexte. Si beaucoup de mots inconnus se trouvent autour d'un mot inconnu, il est improbable que la bonne catégorie sera prédite. Cette situation se produit assez souvent, car les mots inconnus sont soit des noms propres, soit des mots étrangers et dans ces deux cas, ils apparaissent souvent parmi d'autres mots du même genre. Les scores obtenus sont donc à prendre en considération en ayant tous ces éléments à l'esprit.

Les tables ci-dessous donnent les scores généraux obtenus sur le corpus Multitag. La table 32 donne les résultats pour l'ensemble des mots du corpus de référence, qu'ils soient repérés ou non dans notre lexique. La table 33 donne les résultats pour l'ensemble des mots du corpus de référence à l'exception des mots inconnus dans notre lexique.

Catégorie	Rappel	Précision	FScore	Nombre
Adjectifs	81.14	81.17	81.15	49483
Adverbes	79.65	79.70	79.68	39360
Conjonctions	90.82	90.85	90.83	33102
Determinants	93.34	93.36	93.35	99957
Inconnus	11.68	11.68	11.68	796
Interjection	55.4	55.4	55.4	703
Noms	93.4	93.16	93.10	166811
Prepositions	95.94	95.95	95.94	102623
Pronoms	92.17	92.19	92.18	61569
Ponctuations	99.97	99.97	99.97	104527
Verbes	97.60	97.63	97.62	96463
TOTAUX	93.25	93.29	93.27	755394

Tab. 32 – Scores généraux de l'étiqueteur

Nous pouvons conclure à une bonne qualité de l'étiquetage pour cet outil qui en est encore à un stade de développement assez peu avancé. Ses lacunes se situent dans la gestion des mots inconnus, des interjections et surtout des adjectifs. Or le problème de l'étiquetage erroné des adjectifs est lié à la présence dans le lexique d'un grand nombre de verbes au participe passé aussi annotés comme des adjectifs. Il est difficile de savoir si la référence a été correctement annotée pour les *adjectifs*. Comme nous avons pu le voir dans la campagne EASY, ce problème est récurrent dans l'évaluation des outils. D'autre part, du fait que l'usage de Ngrammes variables (jusqu'aux 5-grammes) nécessite de grands corpus d'entrainement, le corpus d'entrainement final était constitué des corpus CLIF et MULTITAG. L'évaluation a été réalisée sur MULTITAG avec tokens imposés. Cette évaluation est donc doublement tronquée. Nous n'avons donc pas pour l'instant la possibilité de connaitre la vraie efficacité de l'étiqueteur sur des corpus inconnus. Le fait d'utiliser une partie du corpus d'entrainement comme corpus d'évaluation aboutit à une mesure assez pertinente de la capacité de l'étiqueteur à être bien entrainé. Le score moyen de 85% correspond dans ce sens à une sensibilité d'apprentissage et de restution de 85%. Il aurait fallu mener une évaluation sur d'autres corpus pour connaitre la

Catégorie	Rappel	Précision	FScore	Occurences
Adjectifs	81.41	81.44	81.42	48906
Adverbes	83.72	83.76	83.74	36271
Conjonctions	92.12	92.16	92.14	32633
Determinants	93.41	93.43	93.42	99349
Inconnus	14.90	14.90	14.90	624
Interjection	60.75	60.75	60.75	637
Noms	94.36	94.49	94.42	150259
Prepositions	96.86	96.87	96.87	100941
Pronoms	92.30	92.33	92.32	61479
Ponctuations	99.99	99.99	99.99	104512
Verbes	97.65	97.68	97.66	96345
TOTAUX	94.5	94.9	94.7	731956

Tab. 33 – Scores généraux de l'étiqueteur hors mots inconnus

qualité réelle de l'étiqueteur. Cependant, nous n'avons pas pu trouver de corpus permettant cette évaluation. Dans les résultats de EASY fournis plus loin, le meilleur de nos analyseurs ne dépasse pas 85% de score, ce qui correspond à la même borne. Le lien n'est sans doute pas fortuit, car nous avons démontré que l'apprentissage actuel ne permettait pas de restituer plus de 85% des données apprises. Ainsi, la borne supérieure du score d'étiquetage est forcément dans cet ordre de grandeur. On ne pourra guère attendre de meilleurs résultats en analyse syntaxique tant que l'étiqueteur ne sera pas amélioré. En se basant sur ces scores, nous choisissons néanmoins d'utiliser cet outil, qui présente de nombreux atouts dans l'ensemble des modules de LPLSuite.

14.2 Evaluation des analyseurs syntaxiques (la campagne EASY)

14.2.1 L'évaluation d'analyseurs syntaxiques

L'évaluation en analyse syntaxique a fait l'objet de programmes de recherche à part entière (voir [Spark-Jones and Galliers, 1996], [Mariani and Veronis, 1999] et [Paroubek *et al.*, 2005]). Outre les corpus étiquetés et arborés, les bases de phrases-tests constituent une ressource précieuse. La plupart des grands projets de développement d'analyseurs syntaxiques ont élaboré de telles bases. C'est le cas par exemple du projet ANLT (Alvey Natural Language Tools) proposant des analyseurs basés sur GPSG. Dans ce domaine, l'opération la plus systématique en termes de ressources développées, de couverture et de nombre de langues reste bien entendu TSNLP (cf. [Estival and Lehmann, 1997]). Ce projet a proposé une méthodologie, des conventions d'annotation ainsi que des outils pour l'élaboration de tels jeux. L'idée est d'identifier un certain nombre de phénomènes linguistiques et pour chacun d'entre eux, de proposer une série de phrases. Une des particularités de TSNLP est de proposer également des phrases mal formées, ce qui permet de tester le comportement de l'analyseur en terme de reconnaissance mais également de robustesse.

L'évaluation automatique des résultats d'analyse conduit souvent, y compris en se limitant aux grammaires de constituants et compte tenu de la diversité des choix d'analyse et des catégories, à ne considérer que le pourcentage de syntagmes bien reconnus et à assimiler le taux

d'échec aux parenthèses qui " croisent " celles du corpus de référence ([Black *et al.*, 1991]), technique utlisée dans Parseval, critiquée par [Lin, 2003] ou [Carroll *et al.*, 2003] (voir aussi à ce propos les actes du workshop " Beyond Parseval ", organisé dans le cadre de LREC 2002). La technique consiste à mesurer dans quelle mesure un analyseur est capable de reproduire les informations contenues dans un corpus arboré en mesurant la précision, le rappel ainsi que le nombre de croisements de parenthèses entre le corpus de référence et la sortie de l'analyseur. On aboutit, en fait, à un compromis entre le taux d'erreur (pourcentage de mots ou de phrases ne recevant pas la bonne analyse) et le taux d'ambiguïté (pourcentage de mots ou de phrases recevant plus d'une analyse). Plusieurs critiques sont faites à ce type d'évaluation. Tout d'abord, les mesures proposées sont valides pour des analyseurs syntagmatiques. De plus, les annotations (cf. par exemple le Penn treebank) relèvent quelquefois de choix théoriques et formels pertinents dans un cadre particulier, mais peu adaptés pour d'autres approches. Par ailleurs le choix des étiquettes associées aux catégories, est d'une granularité élevée, ce qui ne permet pas toujours la description de phénomènes variés. Ce type de mesures pénalise cependant les analyseurs proposant des annotations plus précises (donc un plus grand nombre de parenthèses que dans le corpus de référence). Il est donc difficile, voire impossible, de représenter des structures syntaxiques complètes sans être dépendant d'un formalisme donné et donc pas possible d'utiliser directement un corpus annoté pour évaluer des analyseurs utilisant d'autres formalismes. Une solution consiste à appuyer l'évaluation non pas sur les structures, mais plutôt sur les relations syntaxiques. C'est ce que propose le schéma d'annotation de relations grammaticales (cf. [Carroll *et al.*, 2003] et [Briscoe and Carroll, 2002]). Les auteurs utilisent l'annotation d'un ensemble de relations syntaxiques identifiables indépendamment du formalisme choisi. Ces relations sont hiérarchisées, autorisant ainsi la spécification de niveaux de précision différents dans l'annotation. Pour les analyseurs (probabilistes ou symboliques) basés sur des grammaires de constituants, les premiers résultats publiés faisaient état, pour des corpus anglais de type journalistique, d'un pourcentage maximum de 70 % de phrases totalement (et correctement) analysées, avec, pour les analyseurs à base linguistique, 4 à 5 analyses par phrase en moyenne. Les premières estimations obtenues dans le cadre de la campagne Easy d'évaluation d'analyseurs du français (encore à ce jour en cours de dépouillement), fait apparaître une nette amélioration des résultats, atteignant 85% de bons résultats pour des corpus hétérogènes, composés de textes journalistiques, littéraires ou de langue parlée. Plusieurs facteurs expliquent ces résultats, à commencer par la nette amélioration des ressources (lexiques électroniques, corpus annotés) et des technologies développées. Il faut souligner que pour cette campagne, les participants devaient fournir une sortie unique dans un format unique. Il est intéressant de noter que les approches symboliques semblent donner des résultats de niveau équivalent à celui des approches probabilistes. Ceci est donc extrêmement encourageant dans la perspective d'une véritable interaction entre linguistique et informatique permettant ainsi de dépasser les simples techniques d'ingénierie.

14.2.2 La campagne EASY

La campagne EASY [Paroubek *et al.*, 2005] propose d'analyser syntaxiquement un million de mots du Français courant. Cette campagne s'est déroulée de 2004 à 2005 et les résultats sont attendus pour la fin de l'année 2005. Des résultats préliminaires ont été fournis et nous en donnerons une analyse ci-après. Parmi les systèmes ayant participé à cette campagne on trouve notamment :

- IFSP : propose une segmentation en syntagmes noyaux (chunks), le repérage des fonctions syntaxiques principales et des dépendances, (cf [Aït-Mokhtar and Chanod, 1997])
- FIPS : analyseur développé dans le cadre de GB, avec des applications en traduction automatique et en synthèse de la parole (cf. [Laezlinger and Wehrli, 1991] et [Wehrli, 1997])
- SYNTEX : basé sur une analyse en dépendance, identifiant les principales relations syntaxiques (cf. [Bourigault and Fabre, 2000]). Ce système permet d'extraire d'un corpus des syntagmes.
- Les analyseurs du projet ATOLL : SXLFG (analyseur LFG, cf. [BOULLIER 2005]) et FRMG (analyseur TAG, cf. [Thomasset and Villemonte De La Clergerie, 2005]).
- Les analyseurs du LPL : analyseurs superficiels et profonds, utilisant le formalisme des Grammaires de Propriétés (cf. [Balfourier *et al.*, 2005]).
- L'analyseur du GREYC (cf. [Giguet, 1997]) qui effectue des analyses partielles selon un modèle librement inspiré de L. Tesnière (voir [Tesnière, 1959]).
- LLP2 : analyseur de LTAG, avec un architecture inspirée de XTAG (voir [Paroubek *et al.*, 1992] et [Doran and Srinivas, 2000]).

Les corpus fournis aux participants sont répartis en plusieurs ensembles, comme le montre la figure 63.

type	**répartition**
Général	21%
Littéraire	23%
Mail	15%
Médical	6%
Oral	28%
Questions	7%

FIG. 63 – Répartition des corpus de la campagne EASY

Ces corpus sont fournis dans différents formats :

- Brut,
- Segmenté en énoncés
- Segmenté en énoncés et en tokens,
- Segmenté en énoncés et tokens et étiqueté morphosyntaxiquement sans désambiguïsation.

Nous avons utilisé notre lexique, qui est compatible avec nos outils d'analyse. Ce lexique (DICOLPL) est constitué de 440000 entrées défactorisées dotées de catégories morphosyntaxiques. Nous l'avons complété manuellement de façon à couvrir la totalité des mots du corpus.

Un autre lexique des *formes composées* a été fourni pour les besoins de la campagne. Ce dernier lexique a été réétiqueté afin de correspondre avec nos jeux de traits morphosyntaxiques.

Les participants doivent fournir une analyse syntaxique répondant aux critères définis dans le Guide d'Annotation PEAS qui précise la norme grammaticale à partir de laquelle ont été annotés les corpus de référence. C'est sur la base de cette référence que sont évalués les analyseurs.

Exemple tiré du Guide PEAS 1.6 :

Les groupes nominaux sont constitués d'un nom éventuellement précédé d'un déterminant (...) et/ou d'un adjectif antéposé accompagné de ses modifieurs (qui peuvent comprendre des adverbes), d'un nom propre ou d'un pronom NON clitique.
Quand un adjectif est précédé d'un déterminant, il reste adjectif mais forme un GN (...).
Quand plusieurs noms propres se succèdent sans déterminant ni préposition, ils forment un seul GN (...).

Nous avons donc reformulé l'intégralité du Guide PEAS dans le formalisme des Grammaires de Propriétés en analysant pas à pas les informations susceptibles d'être traduites sous forme de contraintes.

La particularité et l'intérêt de ce protocole d'évaluation basé sur une grammaire sans emboitement, sans hiérarchie, avec des choix grammaticaux parfois discutables, est d'imposer aux participants en plus d'un format de sortie commun, une reformulation complète de la grammaire et éventuellement une modification de leurs outils, afin de mener à bien la campagne.

Nous avons présenté trois analyseurs à cette campagne. Les trois utilisent une entrée identique se présentant sous la forme d'un texte tokenisé et étiqueté.

Chaque token est imposé et accompagné d'un ensemble d'étiquettes possibles fournies de façon automatisée grâce à l'étiqueteur WinBrill. Nous avons réétiqueté ces tokens à l'aide de notre propre étiqueteur (celui de LPLSuite), afin de faire correspondre leurs traits morpho-syntaxiques avec ceux utilisés dans notre grammaire, ainsi que dans le but d'affiner la qualité de l'étiquetage.

La figure 64 schématise la procédure que nous avons mise en place pour finaliser l'étiquetage.

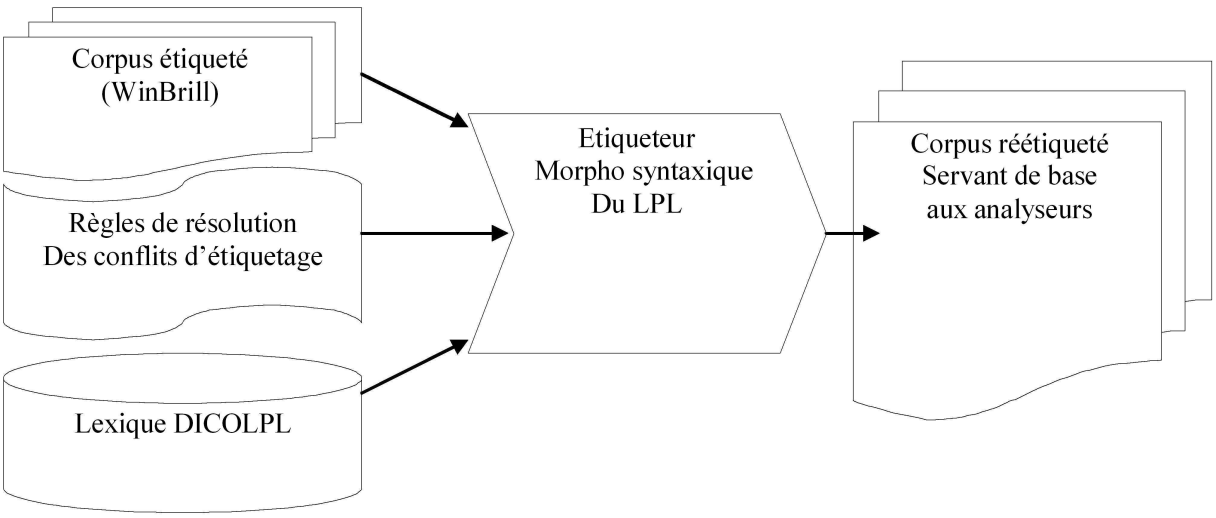


FIG. 64 – Procédure d'étiquetage dans la campagne EASY

Un des intérêts de l'analyse symbolique se retrouve ici dans le fait qu'il est tout à fait possible de constituer une grammaire spécifique plutôt que d'adapter les outils pour les rendre compatibles avec la tâche. Nous ne disposons pas d'une grammaire prédéfinie pour la langue Française, mais nous sommes capables d'en développer en fonction des besoins. Le développement d'une grammaire en amont de l'analyse correspond dans ce sens à une possiblité offerte par notre approche. Dans ce sens, à part l'obligation de fournir une sortie déterminisée et les

questions de formatage de la sortie des analyseurs, nous avons utilisé les possiblités formelles de notre approche. Nous voulons replacer la question de l'adaptation des outils et des ressources à la tâche au coeur de la problématique de la campagne EASY : est-ce l'analyseur qui est évalué ou bien l'ensemble de la chaîne de traitements ou encore la représentation grammaticale qui accompagne forcément l'analyse ? A ce propos, au delà des résultats présentés ci-dessous, nous avons défendu lors des réunions préparatoires de la campagne, l'idée de présenter une fiche d'identité des analyseurs qui tienne compte de tous ces éléments. Nous pourrons ainsi mieux comparer les techniques, les heuristiques, les approches, au sein de leurs chaines de traitement.

Analyseur n ° 1

Le premier analyseur que nous avons présenté est un analyseur superficiel basé sur les Grammaires de Propriétés qui prend en entrée un texte étiqueté et désambiguisé. Il construit dans une première passe l'ensemble des groupes, puis les relations. La construction des groupes repose sur des informations syntaxiques partielles. Plus précisément, seules les propriétés de constituance et de linéarité ont été utilisées.

La stratégie repose sur une technique d'analyse déterministe *coin-gauche*. Il s'agit de repérer pour chaque token, grâce aux deux propriétés citées, sa faculté d'être coin gauche d'un groupe. Dans de nombreux cas, les informations citées sont suffisantes et l'initialisation du groupe correspondant est systématique. En revanche, certaines situations nécessitent la vérification du contexte immédiat. C'est par exemple le cas des groupes PV, qui débutent par une préposition. Cette dernière initialise habituellement un GP, mais dans un contexte verbal à droite, la préposition devient coin gauche du PV. Chaque initialisation de groupe entraîne la fermeture du groupe précédent.

Le mécanisme est donc en une passe unique et consiste à analyser successivement toutes les suites de trois tokens (le token candidat coin gauche, son contexte gauche et son contexte droit). La connaissance du type du groupe en cours permet de compléter la décision d'initialisation.

Les relations sont calculées dans une seconde passe sur la base des groupes construits. Chaque relation correspond à un traitement spécifique. Un premier traitement consiste à construire différentes tables regroupant les groupes et formes susceptibles d'être source ou cible d'une relation. Chaque relation consiste ensuite à parcourir ces tables et vérifier, en fonction des positions des candidats, leur appartenance à une relation. Dans certains cas (par exemple la relation complément-verbe) les candidats sont des ressources uniques. En d'autres termes, un candidat ne peut être complément d'un seul verbe. Cette information, correspondant à une consommation de ressource, est ajoutée dans les tables pour les items concernés. La détection des relations repose donc globalement sur des critères topologiques. Il s'agit d'une approximation qui ne permet pas d'assurer un bon contrôle ce qui entraîne une surgénération.

Les techniques utilisées par l'analyseur superficiel sont donc très simples, ce qui a bien entendu des conséquences sur le résultat obtenu (en particulier pour ce qui concerne les relations). L'avantage majeur de cette technique est sa robustesse et son efficacité : le corpus total est analysé en 4 minutes environ.

Analyseur n ° 2 Le second analyseur est un analyseur profond et non déterministe basé complètement sur les Grammaires de Propriétés. Ce dernier a été présenté au chapitre 10 section 10.3. Un seul passage est réalisé pour une entrée :

- Génération des constituants valides (par vérification des contraintes de la grammaire de propriétés).
- Génération de constituants 'hors grammaire EASY' permettant de situer des consti-

tuants de niveau supérieur et de repérer les relations.

L'analyseur est plus lent que le premier. La complexité moyenne reste polynomiale. L'algorithme d'analyse est non déterministe. La déterminisation se fait à la fin de l'analyse, par une sélection des constituants les plus couvrants.

Analyseur n ° 3 Le troisième analyseur présenté est l'analyseur profond SeedParser présenté au chapitre 11.

Un passage est effectué pour une entrée : génération des constituants valides (par vérification des contraintes de la grammaire de propriétés). Les relations sont considérées comme des constituants discontinus. Leur temps de calcul trop grand a mené à abandonner les relations pour la campagne d'évaluation. L'analyseur est très lent du fait de son manque d'optimisation pour les grands corpus. Sa complexité moyenne est polynomiale et un grand gain dans le temps d'analyse peut être attendu en optimisant l'algorithme de satisfaction de contraintes.

L' analyse est partiellement déterministe en cours d'analyse. Un algorithme de déterminisation est donc employé en fin d'analyse :

- Durant l'analyse (déterminisation *on line*), filtrage autour d'un seuil de densité de satisfaction.

- Si besoin, à la fin de l'analyse, on déterminise le résultat par une sélection des 'meilleurs' constituants parmi ceux qui entrent en conflit (voir chapitre 11).

Cet analyseur présente l'avantage de séparer les algorithmes et les données (le programme est indépendant de la grammaire et de sa sémantique, contrairement au second analyseur qui doit les inclure au sein même du programme). Son inconvénient réside par contre dans sa lenteur. La complexité moyenne reste polynomiale, mais la durée du traitement est de plusieurs jours sur plusieurs machines pour le million de mots que constitue le corpus. Pour cette raison, bien qu'il était possible de calculer aussi bien les constituants que les relations à l'aide de ce parseur, nous n'avons pu effectuer que la première tâche dans les délais impartis. Ceci confronte les contingences de la campagne d'évaluation aux possibilités réelles des analyseurs, lorsque ceux-cis sont programmés dans le cadre de la recherche expérimentale en laboratoire. La lenteur réelle de ce dernier analyseur est un handicap, mais les possibilités d'amélioration et d'optimisation sont encore nombreuses. Notamment, comme indiqué dans la présentation de l'analyseur SeedParser, l'idée de précompiler les traitements les plus critiques devrait permettre un gain de complexité substantiel.

La figure 65 montre le déroulement de la campagne pour les trois analyseurs.

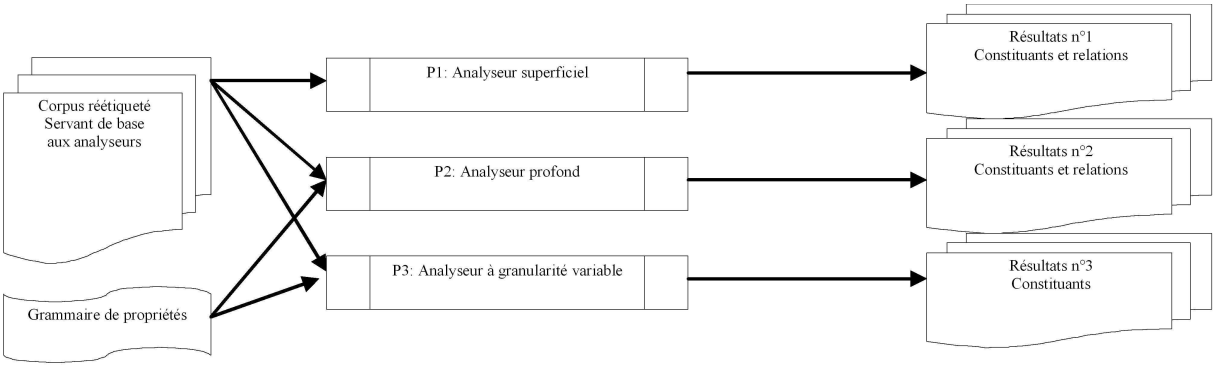


FIG. 65 – Déroulement de la campagne EASY pour trois analyseurs

La campagne se déroule en plusieurs étapes.

La première étape a consisté à réétiqueter, désambiguiser les corpus et affiner la grammaire en fonction de tests approfondis.

Une fois l'étiquetage stabilisé, chaque participant a effectué plusieurs analyses et corrigé éventuellement des algorithmes ou des heuristiques afin d'améliorer les résultats. L'évaluation se faisant alors par sondages empiriques dans les sorties.

Enfin, une analyse finale est réalisée.

Notons que les temps de calcul différents des trois analyseurs ont eu pour effet de permettre beaucoup de tentatives au premier analyseur (quatre minutes pour l'analyse complète), un peu pour le second (un jour d'analyse sur une machine) et une seule pour le dernier (quatre jours sur plusieurs machines avec abandon des relations).

Avant d'aborder les résultats préliminaires, plusieurs remarques peuvent déjà être faites à propos de la campagne et des analyseurs.

14.2.3 A propos des constituants

Intérêts de l'approche symbolique avec les GP : granularité différente selon l'approche, en se basant sur la même grammaire.

Certains conflits entre constituants ne sont pas résolus du fait de l'imprécision de la désambiguisation (par exemple GP et GN introduits par un déterminant amalgamé à une préposition)

14.2.4 Grammaire

Le guide PEAS indique des traitements différents pour certains syntagmes en fonction de l'appartenance ou non de ses constituants à la langue Française. Ceci dépend du lexique et n'est pas repérable de la même façon par les annotateurs de la référence et les étiqueteurs automatiques.

Nous avons fait le choix de nous référer uniquement aux étiquettes fournies aux analyseurs, et aux propriétés de la grammaire, au risque de ne pas correspondre avec la référence.

14.2.4.1 Relations

Comme il le fait pour les groupes, le premier parseur établit des relations en fonction de leur *constituance* et de leur *linéarité,* en utilisant une pile pour repérer les sources et les cibles des relations. Cette technique est surgénérative et globalement moins précise que celle du second parseur qui a intégré les relations comme étant des contraintes de *dépendance* caractéristiques.

14.2.4.2 Protocole

Certains points ne sont pas précisés par le guide PEAS. Il est alors fort possible que la référence ne concorde pas avec nos analyses. Par exemple, pour les répétitions dans les transcriptions de l'oral (nous avons choisi la première option qui consiste à inclure les éléments répétés dans le même syntagme) :
a. <**NV**> **il il se tachait** </**NV**> sa sa <**NV**> **il ne ne buvait** </**NV**> que des Blancs
b. il <**NV**> **il se tachait** </**NV**> sa sa il ne <**NV**> **ne buvait** </**NV**> que des Blancs

Les analyseurs symboliques offrent certains avantages par rapport aux analyseurs stochastiques pour l'annotation de corpus, en particulier si le formalisme qu'elles utilisent permet

une flexibilité de traitement ; c'est le cas des Grammaires de Propriétés dans lesquelles la granularité d'analyse peut être choisie. Ce réglage s'effectue en choisissant le type et le nombre de contraintes à satisfaire. Nous sommes donc en mesure, à partir d'une même grammaire et d'une même stratégie d'analyse, de proposer plusieurs types de traitement offrant des résultats plus ou moins détaillés en fonction des besoins. Là où les approches stochastiques nécessitent un réglage particulier en fonction de chaque tâche d'annotation demandée, une approche symbolique du type de celle décrite ici permet au contraire d'envisager une réutilisation à la fois des ressources exploitées (lexique, grammaire), mais également des moteurs d'analyse.

14.2.5 Résultats préliminaires

Au terme de cette campagne, deux lots de résultats préliminaires nous ont été transmis. Ceux-ci ne concernent que l'analyse en constituants (et pas l'analyse des relations). En attendant les résultats définitifs, nous avons analysé ces résultats afin d'en retirer des informations relatives à la qualité des algorithmes mis en place dans l'ensemble des outils développés (étiqueteur, analyseurs), ainsi que des grammaires et du lexique.

Le premier lot de résultats fournit les détails relatifs aux corpus oraux. Un score a été calculé catégorie par catégorie, corpus par corpus. La formule de calcul de ce score ne nous a pas été transmise, mais nous avons reçu l'indication que le score était calculé de façon stricte (à la frontière près entre les syntagmes du corpus de référence et du corpus analysé). Ces premiers scores reflètent simplement la qualité relative des analyseurs entre eux. Le fait de disposer des détails catégorie par catégorie est intéressant car nous pouvons en déduire des informations précieuses pour améliorer grammaire et algorithmes.

Le second lot de résultats est plus volumineux et cependant moins détaillé, car il s'agit des résultats définitifs en constituants pour la totalité des corpus de référence. Tous les types de corpus s'y retrouvent, du général à l'oral en passant par le médical et le littéraire. Les scores de précision, de rappel et de f-mesure sont donnés corpus par corpus, analyseur par analyseur, avec à chaque fois une *valeur stricte* et une *valeur floue*. Ces dernières correspondent respectivement à des scores calculés *à **la** frontière près* et *à **une** frontière près*.

Le second lot de résultats est intégralement retranscrit dans les trois tables 34, 35 et 36 ci-dessous.

Notes de lecture pour ces statistiques : *Dans les résultats présentés ci-dessous, l'analyseur LPL1 correspond au second analyseur présenté plus haut. L'analyseur LPL2 correspond au premier et LPL3 au troisième. Cette perturbation dans l'ordre de numérotation des analyseurs est due au choix de soumettre les résultats dans l'ordre alphabétique des auteurs.*

Un tableau plus synthétique de ces scores est donné par la table 37.

La table 37, tout comme les trois précédentes, montre tout d'abord le succès systématique de LPL1 (analyseur profond) face à LPL2 (analyseur superficiel) et LPL3 (analyseur à granularité variable). Cette supériorité se vérifie encore lorsqu'on observe les détails. Les différences de f-mesure floue entre LPL1 et LPL3 sont de l'ordre de 3% et entre LPL3 et LPL2 de l'ordre de 5%. Cependant, LPL3 se retrouve derrière LPL2 dans le cas du corpus médical et du corpus de questions. Diverses conclusions découlent directement de la lecture de ces tables :

– Chaque analyseur a un score "flou" meilleur que le score "strict", ce qui était prévisible. LPL1 gagne en moyenne 4%, LPL2 gagne 8% et LPL3 gagne 4%. L'analyseur superficiel est donc moins précis que l'analyseur à granularité variable, qui est à son tour moins précis que l'analyseur profond.

corpus	Type de corpus	precision	écart type	rappel	écart type	f-mesure	écart type	Precision floue	écart type	Rappel flou	écart type	f-mesure floue	ecart type	taux croisement frontieres	ecart type
Total		**78,32**	*5,29*	**78,86**	*5,18*	**80,29**	*4,68*	**83,73**	*3,29*	**84,37**	*3,32*	**84,84**	*2,47*	**3,31**	*2,10*
	général	**85,16**	*13,12*	**85,17**	*12,24*	**85,94**	*11,17*	**87,35**	*11,63*	**87,46**	*10,88*	**88,06**	*9,73*	**0,79**	*1,42*
	littéraire	**84,41**	*12,44*	**86,75**	*10,36*	**85,53**	*11,13*	**86,91**	*10,72*	**89,62**	*9,00*	**88,14**	*9,48*	**0,79**	*1,44*
	mail	**76,93**	*23,93*	**78,14**	*22,57*	**81,44**	*18,10*	**80,57**	*20,98*	**82,47**	*20,83*	**84,86**	*15,51*	**0,83**	*1,56*
	médical	**79,17**	*17,67*	**78,42**	*18,43*	**80,97**	*14,78*	**83,59**	*14,88*	**81,48**	*16,09*	**83,76**	*12,65*	**3,25**	*5,23*
	oral	**73,66**	*22,31*	**74,48**	*21,36*	**76,25**	*18,50*	**81,79**	*17,13*	**83,01**	*16,27*	**83,12**	*14,45*	**5,27**	*7,12*
	questions	**86,62**	*11,33*	**85,19**	*10,38*	**85,96**	*10,24*	**88,23**	*10,15*	**87,05**	*9,83*	**87,40**	*9,44*	**1,40**	*2,47*
general_elda	général	84,86	*14,65*	84,41	*14,78*	87,14	*11,33*	86,57	*14,09*	86,03	*14,05*	88,80	*10,50*	0,90	*1,59*
general_lemonde	général	85,12	*11,43*	84,57	*10,90*	84,81	*10,78*	87,13	*10,55*	86,58	*9,97*	86,65	*9,95*	0,86	*1,55*
general_mlcc	général	84,81	*15,68*	84,87	*14,16*	85,70	*12,70*	87,59	*13,39*	87,66	*12,46*	88,18	*11,02*	0,82	*1,54*
general_senat	général	85,83	*10,74*	86,82	*9,10*	86,14	*9,89*	88,10	*8,47*	89,58	*7,01*	88,59	*7,46*	0,60	*1,00*
litteraire_1	littéraire	85,37	*12,27*	87,31	*10,45*	86,21	*11,27*	87,79	*10,53*	90,20	*8,58*	88,82	*9,37*	0,70	*1,24*
litteraire_2	littéraire	84,65	*10,68*	88,06	*8,29*	86,22	*9,36*	86,79	*9,43*	90,41	*7,48*	88,46	*8,33*	0,60	*1,07*
litteraire_3	littéraire	84,27	*13,62*	85,56	*11,40*	84,70	*12,43*	87,03	*11,29*	88,67	*10,05*	87,59	*10,15*	1,15	*2,12*
litteraire_4	littéraire	83,33	*13,18*	86,06	*11,28*	84,97	*11,47*	86,01	*11,64*	89,19	*9,88*	87,69	*10,08*	0,72	*1,32*
mail_9	mail	77,55	*23,33*	78,18	*21,89*	81,29	*17,75*	81,66	*19,95*	83,08	*19,35*	84,98	*14,94*	0,79	*1,49*
mail_10	mail	76,31	*24,54*	78,09	*23,25*	81,58	*18,44*	79,48	*22,02*	81,85	*22,31*	84,75	*16,09*	0,87	*1,63*
medical_1	médical	78,96	*20,50*	77,50	*19,81*	82,66	*13,77*	80,78	*19,49*	79,34	*19,10*	84,21	*12,99*	1,30	*2,17*
medical_2	médical	85,23	*10,86*	83,34	*12,16*	84,05	*11,44*	85,54	*10,49*	83,71	*12,24*	84,39	*11,09*	1,48	*2,63*
medical_3	médical	76,70	*16,45*	78,11	*16,66*	77,05	*16,47*	82,86	*13,81*	83,21	*12,66*	82,54	*12,54*	4,16	*6,30*
medical_4	médical	82,49	*14,73*	82,10	*14,91*	82,78	*13,95*	87,30	*11,48*	85,75	*12,62*	86,11	*11,44*	3,51	*5,74*
medical_6	médical	72,48	*25,83*	71,04	*28,62*	78,31	*18,25*	81,49	*19,11*	75,37	*23,83*	81,58	*15,21*	5,81	*9,31*
oral_delic_1	oral	73,81	*23,07*	75,85	*21,24*	77,09	*19,25*	81,99	*18,81*	85,09	*16,18*	84,03	*15,50*	3,05	*4,74*
oral_delic_2	oral	68,76	*21,62*	72,09	*19,62*	72,39	*17,90*	77,87	*18,14*	82,05	*17,21*	81,80	*14,12*	3,78	*5,35*
oral_delic_3	oral	75,82	*21,64*	76,18	*21,13*	75,95	*20,70*	83,34	*16,32*	83,98	*16,73*	83,44	*14,79*	3,98	*6,15*
oral_delic_4	oral	74,68	*19,80*	76,82	*20,15*	77,46	*17,67*	83,29	*13,39*	87,61	*13,95*	84,89	*13,15*	3,34	*5,03*
oral_delic_5	oral	72,41	*22,01*	69,88	*21,50*	71,76	*20,23*	84,10	*16,58*	81,09	*14,95*	83,04	*13,37*	8,14	*8,78*
oral_delic_6	oral	61,21	*26,21*	63,53	*25,84*	67,19	*20,24*	67,85	*26,29*	71,02	*25,96*	74,09	*19,11*	6,17	*7,36*
oral_delic_7	oral	66,77	*16,93*	67,13	*14,66*	67,65	*13,73*	80,53	*14,78*	81,02	*12,29*	80,02	*12,45*	7,99	*6,59*
oral_delic_8	oral	68,35	*21,94*	74,58	*19,92*	72,99	*17,90*	72,00	*21,53*	78,66	*18,34*	76,81	*17,00*	3,84	*5,10*
oral_delic_9	oral	77,96	*16,75*	77,13	*15,24*	76,96	*14,85*	85,58	*12,58*	84,65	*11,40*	84,45	*10,14*	5,19	*6,71*
oral_elda_1	oral	74,95	*25,63*	74,81	*25,18*	78,82	*19,62*	84,61	*16,63*	84,97	*17,12*	85,68	*14,30*	5,74	*9,30*
oral_elda_2	oral	73,67	*25,78*	69,58	*27,50*	76,03	*20,62*	85,72	*13,34*	80,95	*16,07*	85,32	*11,08*	8,93	*10,29*
oral_elda_3	oral	77,27	*24,81*	79,22	*22,41*	82,35	*18,22*	84,91	*16,86*	87,69	*13,78*	86,04	*15,17*	5,25	*9,06*
oral_elda_5	oral	78,04	*22,62*	77,30	*23,23*	79,47	*20,31*	83,92	*17,36*	82,76	*18,18*	83,02	*17,42*	6,59	*9,69*
oral_elda_6	oral	82,53	*23,15*	81,62	*22,64*	85,60	*17,50*	88,43	*15,97*	88,01	*16,10*	89,16	*13,51*	4,19	*7,57*
oral_elda_8	oral	78,73	*22,54*	81,44	*20,06*	81,98	*18,72*	82,64	*18,42*	85,64	*15,87*	84,93	*15,70*	2,80	*5,07*
questions_amaryllis	questions	88,37	*8,56*	88,01	*8,38*	88,10	*8,11*	89,39	*8,03*	89,11	*8,27*	89,16	*7,68*	0,79	*1,44*
questions_trec	questions	84,87	*14,10*	82,36	*12,39*	83,81	*12,38*	87,07	*12,27*	84,99	*11,40*	85,65	*11,20*	2,00	*3,50*

TAB. 34 – Résultats en constituants pour l'analyseur LPL-1

corpus	type de corpus	precision	*ecart type*	rappel	*ecart type*	f-mesure	*ecart type*	precision floue	*ecart type*	rappel flou	*ecart type*	f-mesure floue	*ecart type*	taux croisement frontieres	*ecart type*
Total		**69,61**	*5,10*	**73,65**	*5,21*	**73,51**	*4,38*	**76,17**	*3,56*	**80,81**	*2,97*	**79,27**	*2,84*	**5,04**	*2,12*
	général	**73,14**	*17,08*	**78,29**	*15,72*	**77,55**	*13,73*	**76,44**	*15,52*	**82,04**	*13,93*	**80,77**	*12,09*	**2,05**	*3,14*
	littéraire	**68,64**	*16,94*	**75,53**	*14,86*	**72,25**	*15,12*	**74,46**	*14,54*	**82,52**	*11,96*	**77,98**	*12,84*	**3,56**	*4,64*
	mail	**66,51**	*27,21*	**72,65**	*25,27*	**73,80**	*20,42*	**71,86**	*24,54*	**79,06**	*22,32*	**78,24**	*18,40*	**2,45**	*4,13*
	médical	**76,81**	*14,81*	**80,99**	*14,35*	**78,97**	*13,89*	**80,75**	*13,26*	**84,55**	*11,81*	**82,58**	*11,45*	**3,40**	*5,09*
	oral	**66,02**	*24,15*	**68,98**	*22,98*	**70,40**	*19,50*	**75,13**	*19,34*	**78,90**	*17,78*	**78,07**	*16,02*	**7,17**	*8,66*
	questions	**76,44**	*16,60*	**78,18**	*14,96*	**77,29**	*15,12*	**79,72**	*14,15*	**81,69**	*13,04*	**80,66**	*13,11*	**4,72**	*6,18*
general_elda	général	76,16	*16,10*	80,74	*14,75*	80,73	*12,19*	77,86	*16,13*	82,65	*13,75*	82,49	*11,89*	1,46	*2,43*
general_lemonde	général	71,53	*15,43*	76,87	*13,95*	74,64	*13,66*	75,51	*13,75*	81,18	*12,02*	78,07	*12,57*	2,85	*4,21*
general_mlcc	général	73,19	*20,73*	78,81	*19,64*	78,64	*16,51*	76,88	*18,40*	82,87	*17,08*	82,18	*13,79*	2,16	*3,57*
general_senat	général	71,69	*16,07*	76,77	*14,55*	76,20	*12,54*	75,49	*13,78*	81,46	*12,87*	80,36	*10,13*	1,73	*2,37*
litteraire_1	littéraire	70,49	*17,72*	76,90	*15,08*	73,31	*16,51*	75,44	*15,25*	82,81	*12,20*	78,68	*13,73*	2,99	*3,95*
litteraire_2	littéraire	68,66	*13,99*	77,75	*12,63*	73,24	*12,42*	73,98	*12,06*	84,35	*9,70*	78,49	*10,49*	3,54	*4,11*
litteraire_3	littéraire	70,33	*17,88*	75,85	*15,18*	73,13	*15,87*	75,81	*15,75*	82,18	*12,91*	78,44	*13,80*	2,75	*4,32*
litteraire_4	littéraire	65,09	*18,15*	71,62	*16,55*	69,31	*15,70*	72,61	*15,10*	80,73	*13,03*	76,30	*13,36*	4,96	*6,19*
mail_9	mail	66,71	*26,69*	73,00	*24,72*	73,51	*20,45*	72,18	*24,10*	79,70	*21,29*	77,82	*18,51*	2,31	*3,94*
mail_10	mail	66,31	*27,74*	72,31	*25,82*	74,09	*20,39*	71,55	*24,99*	78,43	*23,34*	78,66	*18,29*	2,59	*4,31*
medical_1	médical	76,65	*16,30*	83,43	*13,44*	80,31	*13,71*	78,92	*14,83*	86,12	*11,70*	82,76	*12,10*	1,79	*2,81*
medical_2	médical	80,83	*12,27*	84,26	*11,71*	82,37	*11,86*	81,28	*11,77*	84,76	*11,27*	82,84	*11,40*	1,80	*2,94*
medical_3	médical	70,93	*14,07*	74,60	*15,38*	72,37	*14,30*	77,46	*13,33*	80,39	*11,60*	78,47	*11,51*	5,36	*6,64*
medical_4	médical	75,95	*15,27*	82,86	*13,43*	79,28	*13,81*	80,49	*13,19*	86,95	*11,29*	83,11	*11,34*	2,98	*4,90*
medical_6	médical	79,70	*16,16*	79,77	*17,81*	80,52	*15,77*	85,61	*13,18*	84,54	*13,21*	85,75	*10,89*	5,08	*8,16*
oral_delic_1	oral	71,91	*24,77*	74,62	*23,11*	75,40	*20,69*	79,53	*20,64*	82,43	*18,89*	81,99	*16,80*	4,89	*7,14*
oral_delic_2	oral	65,90	*21,69*	69,87	*21,02*	69,88	*18,53*	74,73	*18,43*	79,55	*19,19*	79,01	*15,24*	5,51	*7,25*
oral_delic_3	oral	67,59	*22,95*	68,98	*21,74*	68,45	*21,79*	77,39	*17,79*	79,37	*18,49*	78,38	*16,68*	8,35	*10,07*
oral_delic_4	oral	68,18	*22,93*	67,82	*21,51*	69,66	*19,99*	76,68	*19,22*	77,19	*16,72*	77,49	*16,61*	9,48	*9,32*
oral_delic_5	oral	63,21	*25,11*	62,03	*24,60*	67,56	*19,11*	76,51	*20,70*	74,74	*19,06*	79,32	*14,20*	10,93	*10,24*
oral_delic_6	oral	51,97	*25,71*	58,11	*26,74*	60,64	*18,87*	59,51	*24,41*	66,98	*24,90*	67,60	*17,45*	7,44	*8,72*
oral_delic_7	oral	54,64	*20,04*	59,84	*20,75*	58,75	*17,71*	65,03	*19,40*	71,19	*18,47*	66,77	*19,03*	9,17	*6,86*
oral_delic_8	oral	62,71	*23,15*	68,84	*23,31*	67,19	*19,77*	68,95	*21,20*	76,06	*19,58*	73,79	*17,41*	7,90	*8,21*
oral_delic_9	oral	61,75	*16,47*	65,99	*15,17*	64,56	*14,06*	73,13	*13,99*	78,46	*11,34*	76,37	*10,30*	8,52	*8,20*
oral_elda_1	oral	67,13	*29,06*	69,87	*26,87*	73,75	*21,09*	76,83	*20,86*	80,85	*18,66*	79,85	*17,37*	6,35	*9,80*
oral_elda_2	oral	66,07	*27,01*	68,41	*24,68*	71,72	*19,80*	77,94	*16,19*	80,48	*14,33*	81,27	*12,32*	7,87	*9,74*
oral_elda_3	oral	68,03	*28,78*	71,25	*25,90*	75,40	*20,23*	77,42	*20,13*	82,83	*16,03*	79,64	*18,03*	6,91	*10,48*
oral_elda_5	oral	73,62	*23,09*	75,86	*22,06*	76,52	*20,55*	81,96	*17,99*	84,67	*15,97*	82,79	*16,51*	5,30	*8,43*
oral_elda_6	oral	79,53	*25,32*	79,86	*24,27*	83,75	*18,96*	85,89	*18,26*	86,25	*18,01*	88,07	*14,20*	4,72	*8,43*
oral_elda_8	oral	68,13	*26,20*	73,41	*22,92*	72,83	*21,30*	75,54	*20,95*	82,41	*17,11*	78,66	*18,17*	4,23	*7,00*
questions_amaryllis	questions	77,77	*13,71*	79,70	*12,83*	78,56	*12,98*	80,65	*11,62*	82,73	*11,37*	81,50	*11,42*	4,09	*4,76*
questions_trec	questions	75,10	*19,50*	76,65	*17,09*	76,01	*17,25*	78,79	*16,68*	80,65	*14,72*	79,81	*14,80*	5,36	*7,60*

TAB. 35 – Résultats en constituants pour l'analyseur LPL-2

corpus	type de corpus	precision	ecart type	rappel	ecart type	f-mesure	ecart type	precision floue	ecart type	rappel flou	ecart type	f-mesure floue	ecart type	taux croisement frontieres	ecart type
Total		**74,32**	*5,50*	**74,44**	*5,95*	**76,21**	*5,10*	**80,21**	*3,25*	**80,20**	*3,85*	**81,03**	*3,20*	**5,28**	*2,99*
	général	**80,80**	*14,15*	**80,71**	*13,60*	**81,54**	*12,56*	**83,50**	*12,61*	**83,47**	*12,17*	**84,14**	*10,97*	**1,86**	*2,86*
	littéraire	**81,95**	*13,31*	**83,94**	*11,65*	**82,89**	*12,16*	**84,72**	*11,62*	**87,07**	*10,24*	**85,78**	*10,52*	**1,64**	*2,58*
	mail	**75,39**	*24,47*	**76,18**	*23,56*	**79,41**	*19,33*	**79,92**	*21,00*	**81,26**	*21,03*	**83,57**	*16,29*	**2,16**	*3,69*
	médical	**76,17**	*17,77*	**75,64**	*18,51*	**78,12**	*14,94*	**81,03**	*15,09*	**79,05**	*16,01*	**81,31**	*12,83*	**4,12**	*6,15*
	oral	**69,12**	*23,32*	**69,10**	*22,98*	**71,53**	*19,76*	**77,62**	*18,51*	**77,57**	*18,19*	**78,43**	*16,21*	**8,24**	*9,32*
	questions	**79,33**	*14,25*	**78,20**	*13,77*	**79,33**	*12,87*	**82,27**	*12,29*	**81,42**	*12,16*	**81,63**	*11,92*	**3,29**	*4,93*
general_elda	général	80,63	*15,79*	79,78	*15,82*	82,70	*12,82*	83,15	*14,99*	82,10	*14,74*	85,11	*11,74*	1,92	*2,96*
general_lemonde	général	80,40	*12,84*	80,55	*12,35*	80,47	*12,30*	82,95	*12,03*	83,08	*11,22*	82,84	*11,41*	2,26	*3,44*
general_mlcc	général	79,56	*17,11*	79,58	*16,17*	80,44	*14,81*	82,52	*14,97*	82,50	*14,47*	83,08	*12,91*	1,74	*2,95*
general_senat	général	82,59	*10,87*	82,93	*10,04*	82,56	*10,29*	85,37	*8,47*	86,20	*8,26*	85,53	*7,81*	1,51	*2,09*
litteraire_1	littéraire	81,20	*13,90*	82,44	*12,89*	81,69	*13,18*	83,94	*12,10*	85,58	*11,06*	84,60	*11,28*	2,50	*3,46*
litteraire_2	littéraire	81,94	*11,59*	84,93	*9,89*	83,32	*10,51*	84,55	*10,37*	87,75	*8,67*	86,02	*9,38*	1,54	*2,46*
litteraire_3	littéraire	81,51	*14,59*	82,57	*12,51*	81,82	*13,50*	84,60	*12,37*	86,01	*11,23*	85,05	*11,26*	1,71	*2,97*
litteraire_4	littéraire	83,14	*13,15*	85,82	*11,30*	84,75	*11,47*	85,80	*11,66*	88,94	*10,00*	87,46	*10,16*	0,79	*1,44*
mail_9	mail	75,42	*24,58*	75,81	*23,72*	79,02	*19,59*	80,50	*20,72*	81,49	*20,06*	83,42	*16,19*	2,51	*4,29*
mail_10	mail	75,35	*24,37*	76,56	*23,41*	79,80	*19,08*	79,34	*21,29*	81,03	*22,01*	83,72	*16,38*	1,80	*3,10*
medical_1	médical	75,25	*20,28*	73,61	*20,22*	78,87	*14,22*	77,38	*19,59*	75,65	*19,36*	80,68	*13,57*	2,71	*3,92*
medical_2	médical	81,05	*12,58*	81,02	*13,19*	80,96	*12,63*	82,68	*10,95*	82,47	*11,73*	82,49	*11,20*	2,78	*4,32*
medical_3	médical	74,12	*15,81*	75,15	*16,48*	74,30	*15,93*	80,46	*13,66*	80,40	*12,93*	79,95	*12,67*	4,45	*6,53*
medical_4	médical	80,26	*14,64*	79,84	*14,52*	80,54	*13,54*	85,33	*11,65*	83,73	*12,45*	84,12	*11,26*	4,19	*6,24*
medical_6	médical	70,19	*25,56*	68,58	*28,16*	75,95	*18,36*	79,31	*19,59*	72,99	*23,61*	79,30	*15,46*	6,47	*9,75*
oral_delic_1	oral	69,41	*26,31*	69,69	*26,07*	72,33	*22,64*	77,42	*21,26*	78,42	*20,31*	78,40	*18,96*	7,38	*9,09*
oral_delic_2	oral	66,14	*20,75*	67,51	*20,59*	68,70	*18,26*	74,83	*19,04*	76,73	*19,14*	77,53	*15,93*	6,06	*7,34*
oral_delic_3	oral	68,50	*22,32*	67,53	*22,95*	67,98	*21,97*	78,66	*17,04*	77,44	*18,88*	77,80	*16,17*	9,36	*10,59*
oral_delic_4	oral	71,42	*21,62*	71,14	*22,30*	71,98	*20,60*	79,65	*15,34*	79,82	*17,34*	79,32	*16,00*	8,22	*9,40*
oral_delic_5	oral	71,73	*22,58*	68,49	*21,89*	71,80	*19,66*	83,19	*16,62*	79,08	*16,03*	81,59	*14,14*	8,94	*9,17*
oral_delic_6	oral	56,11	*24,15*	57,54	*24,35*	61,66	*18,53*	63,96	*24,41*	65,83	*24,17*	69,56	*16,92*	8,64	*8,64*
oral_delic_7	oral	61,91	*19,33*	60,37	*18,60*	61,87	*16,52*	75,03	*18,50*	72,85	*16,78*	73,25	*16,75*	13,31	*9,59*
oral_delic_8	oral	65,47	*22,80*	69,75	*21,89*	69,20	*19,18*	69,48	*20,42*	74,07	*18,45*	73,33	*16,38*	7,45	*7,97*
oral_delic_9	oral	69,98	*17,98*	68,31	*18,12*	69,90	*16,34*	78,27	*15,43*	75,38	*14,72*	76,20	*14,27*	10,35	*9,70*
oral_elda_1	oral	70,37	*27,08*	70,32	*27,31*	74,91	*20,99*	80,35	*19,12*	80,49	*19,64*	81,67	*16,84*	7,81	*11,36*
oral_elda_2	oral	67,34	*24,97*	63,78	*26,12*	70,10	*20,50*	80,11	*14,82*	75,25	*15,99*	79,78	*12,62*	12,19	*10,83*
oral_elda_3	oral	70,79	*25,62*	72,17	*23,46*	76,45	*18,43*	79,38	*17,43*	81,50	*15,93*	80,18	*16,54*	7,34	*11,14*
oral_elda_5	oral	73,32	*25,33*	72,62	*25,06*	74,78	*22,70*	79,20	*20,59*	78,07	*20,30*	79,34	*18,75*	8,29	*10,92*
oral_elda_6	oral	79,53	*25,27*	79,30	*24,77*	83,49	*19,52*	85,62	*18,04*	85,96	*18,14*	87,27	*15,22*	3,88	*6,94*
oral_elda_8	oral	74,84	*23,61*	78,06	*21,22*	77,78	*20,51*	79,23	*19,54*	82,69	*17,04*	81,18	*17,67*	4,31	*7,05*
questions_amaryllis	questions	81,36	*11,04*	80,60	*10,88*	80,88	*10,76*	84,04	*9,75*	83,28	*9,83*	83,56	*9,53*	2,96	*4,08*
questions_trec	questions	77,30	*17,45*	75,80	*16,66*	77,78	*14,99*	80,49	*14,82*	79,55	*14,48*	79,71	*14,30*	3,62	*5,77*

TAB. 36 – Résultats en constituants pour l'analyseur LPL-3

Nom Parseur	type de corpus	precision	rappel	f-mesure	precision floue	rappel flou	f-mesure floue
LPL1	général	85,16	85,17	85,94	87,35	87,46	88,06
	littéraire	84,41	86,75	85,53	86,91	89,62	88,14
	mail	76,93	78,14	81,44	80,57	82,47	84,86
	médical	79,17	78,42	80,97	83,59	81,48	83,76
	oral	73,66	74,48	76,25	81,79	83,01	83,12
	questions	86,62	85,19	85,96	88,23	87,05	87,40

Nom Parseur	type de corpus	precision	rappel	f-mesure	precision floue	rappel flou	f-mesure floue
LPL2	général	73,14	78,29	77,55	76,44	82,04	80,77
	littéraire	68,64	75,53	72,25	74,46	82,52	77,98
	mail	66,51	72,65	73,80	71,86	79,06	78,24
	médical	76,81	80,99	78,97	80,75	84,55	82,58
	oral	66,02	68,98	70,40	75,13	78,90	78,07
	questions	76,44	78,18	77,29	79,72	81,69	80,66

Nom Parseur	type de corpus	precision	rappel	f-mesure	precision floue	rappel flou	f-mesure floue
LPL3	général	80,80	80,71	81,54	83,50	83,47	84,14
	littéraire	81,95	83,94	82,89	84,72	87,07	85,78
	mail	75,39	76,18	79,41	79,92	81,26	83,57
	médical	76,17	75,64	78,12	81,03	79,05	81,31
	oral	69,12	69,10	71,53	77,62	77,57	78,43
	questions	79,33	78,20	79,33	82,27	81,42	81,63

TAB. 37 – Synthèse des résultats pour les trois analyseurs

- L'analyse approfondie est majoritairement meilleure que l'analyse superficielle, même pour le type d'analyse attendu (analyse en syntagmes plats, sans emboîtements). Autrement dit, un analyseur approfondi semble mieux capable d'analyse superficielle qu'un analyseur superficiel.
- L'analyse de textes littéraires ou généraux assez structurés est meilleure que celle de textes transcrits à partir de l'oral, des corpus de mails et médicaux. Pour chaque parseur, cette tendance se vérifie avec toutefois une inversion pour LPL2 pour lequel le score sur corpus médical est de loin meilleur que tous ses autres scores. LPL1 perd environ 6% de qualité entre le meilleur score de f-mesure floue (littéraire) et le plus mauvais (oral). LPL2 perd 7% et LPL3 perd 9%. Ces différences montrent à quel point les approches sont variables autant du point de vue algorithmique que du point de vue des ressources grammaticales et de leur emploi. La stabilité de LPL2 par rapport aux analyseurs profonds s'explique du fait de la légèreté des structures employées, moins sensibles à la bonne formation du contexte analysé. La variabilité dans LPL1 et LPL3 est plus grande, justement du fait de l'emploi de grammaires beaucoup plus structurées et donc beaucoup plus sensibles à la bonne formation de l'entrée. Cependant, ces écarts restent acceptablent et montrent que l'approche analytique dans le formalisme des Grammaires de Propriétés conserve une grande fiabilité même face à des corpus peu normés.
- Les différences nettes entre LPL1 et LPL3 sont en moyenne de l'ordre de 2 à 4%. Ces deux analyseurs suivent par contre exactement la même variabilité en fonction du type de corpus. Les courbes de ces variations sont pratiquement parallèles. La différence de sensibilité entre ces deux analyseurs peut s'expliquer du fait que deux grammaires différentes ont été utilisées (dans ce cas, la mise en évidence des différences permettra d'affiner les grammaires). Une autre explication à cet écart constant provient de la technique de déterminisation employée. Celle-ci n'appartient pas au formalisme des Grammaires de

Propriétés, mais constitue le seul moyen possible de fournir une sortie unique à partir d'une analyse non déterministe. Lorsque la totalité des résultats sera disponible, il sera alors possible de mettre clairement en évidence les différents défauts de ces techniques.

A partir de ce dernier tableau, nous avons représenté l'ensemble des scores avec les trois figures 66, 67 et 68. Ces figures sont des représentations caractéristiques des scores de chaque analyseur. On y retrouve les conclusions données ci-dessus.

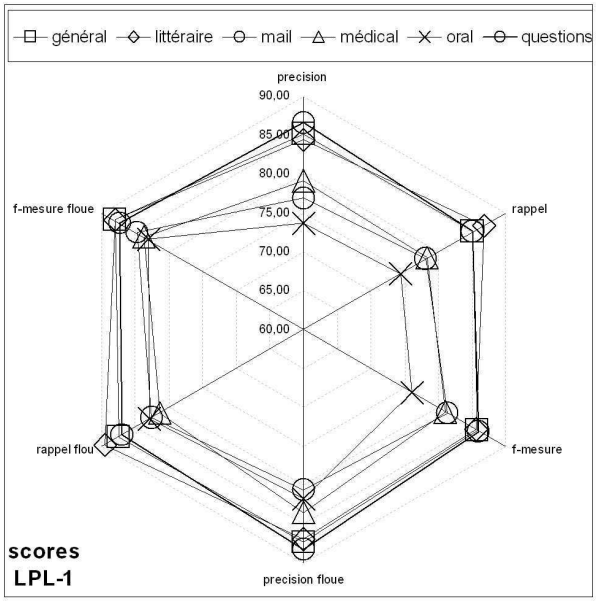


FIG. 66 – Scores préliminaires stricts et flous pour LPL-1

Ces figures permettent de comparer les résultats d'un analyseur donné entre différents corpus. Elles permettent aussi de comparer assez rapidement les analyseurs entre eux. Il apparaît que la qualité de l'analyse est pour les trois analyseurs meilleure sur des corpus écrits normés (littéraires, médicaux) et moins bonne sur les corpus oraux. Les corpus généralistes donnent des scores moyens. Les analyseurs profonds (LPL-1 et LPL-3) sont globalement meilleurs que l'analyseur superficiel (LPL-2). On constate aussi une constance pour chaque analyseur entre ses scores de précision, rappel et f-mesure, ce qui nous incite à analyser plus en détailles résultats en se concentrant sur le score de f-mesure qui permet d'appréhender plus aisément les phénomènes observables.

La table 38 et la figure 69 donnent ainsi un aperçu plus clair des différences entre les analyseurs par type de corpus.

En gardant à l'esprit ce que nous venons d'observer, nous pouvons nous intéresser aux détails de l'analyse sur les corpus oraux. Le premier lot de résultats, que résument les tables 39 et 40, permet d'observer, analyseur par analyseur, corpus par corpus, et catégorie par catégorie, les différences d'analyse. Le score fourni ne correspond à une mesure stricte du pourcentage de constituants trouvés par rapport au nombre de constituants du corpus de référence.

A partir de ces tables, la figure 70 met en évidence certaines différences des analyseurs :

– L'analyseur superficiel (LPL-2) semble mieux analyser les groupes nominaux et les groupes prépositionnels oraux que ses concurrents. Dès que la totalité des résultats

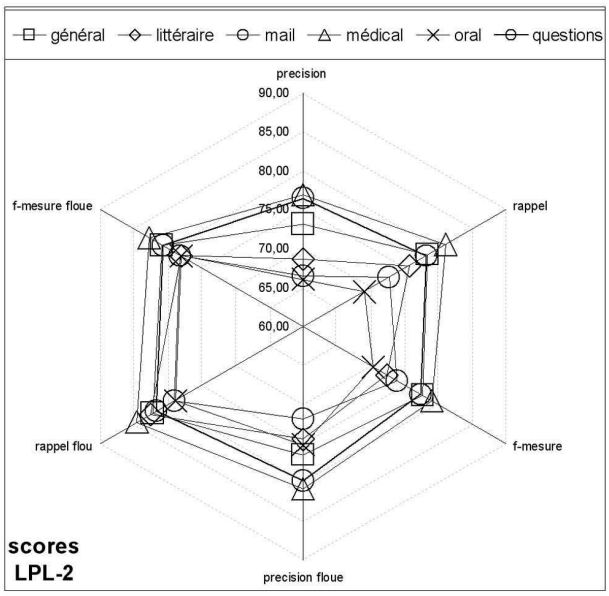


FIG. 67 – Scores préliminaires stricts et flous pour LPL-2

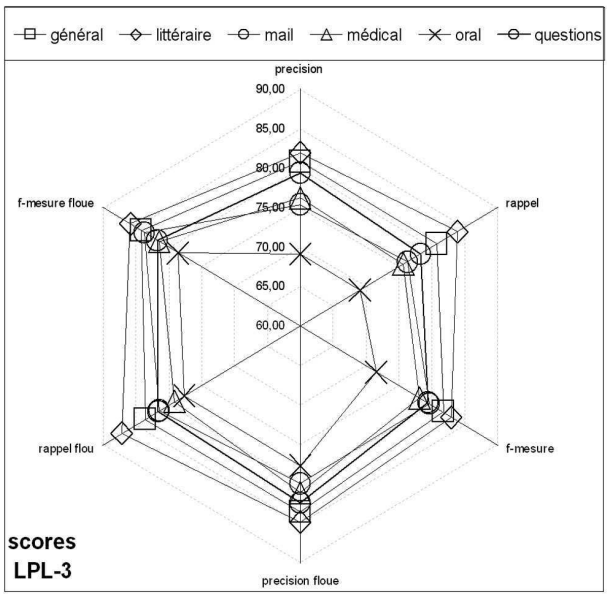


FIG. 68 – Scores préliminaires stricts et flous pour LPL-3

type de corpus	f-mesure LPL-1	f-mesure floue LPL-1	f-mesure LPL-2	f-mesure floue LPL-2	f-mesure LPL-3	f-mesure floue LPL-3
général	85,94	88,06	77,55	80,77	81,54	84,14
littéraire	85,53	88,14	72,25	77,98	82,89	85,78
mail	81,44	84,86	73,80	78,24	79,41	83,57
médical	80,97	83,76	78,97	82,58	78,12	81,31
oral	76,25	83,12	70,40	78,07	71,53	78,43
questions	85,96	87,40	77,29	80,66	79,33	81,63
Moyenne par type	**82,68**	**85,89**	**75,04**	**79,72**	**78,81**	**82,48**
Moyenne pondérée	**80,29**	**84,84**	**73,51**	**79,27**	**76,21**	**81,03**

TAB. 38 – Synthèse des f-mesures pour les trois analyseurs par type de corpus

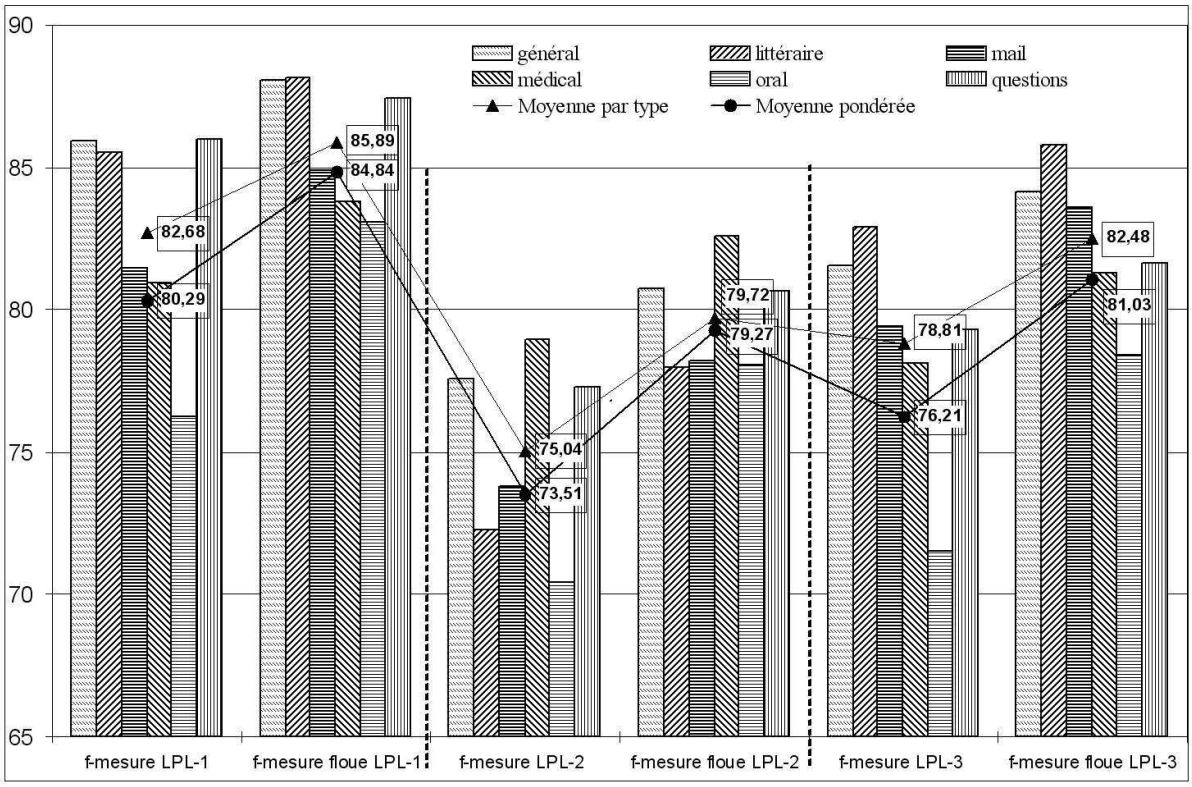


FIG. 69 – Synthèse des f-mesures pour les trois analyseurs par type de corpus

categorie	corpus	LPL-1	LPL-2	LPL-3
GA	C1	80	80	80
GA	C2	86	86	83
GA	C3	93	93	90
GA	C4	87	87	87
GA	C5	100	100	100
GA	C6	93	93	93
GA	C7	81	75	81
GA	C8	93	96	93
GA	C9	100	100	97
GN	C1	79	82	78
GN	C2	91	90	85
GN	C3	92	93	86
GN	C4	88	90	82
GN	C5	87	90	84
GN	C6	81	82	77
GN	C7	88	80	88
GN	C8	83	85	78
GN	C9	96	96	87
GP	C1	80	84	77
GP	C2	78	80	78
GP	C3	82	86	85
GP	C4	83	83	72
GP	C5	86	86	82
GP	C6	80	76	67
GP	C7	76	78	73
GP	C8	94	94	82
GP	C9	85	83	81

categorie	corpus	LPL-1	LPL-2	LPL-3
GR	C1	88	86	83
GR	C2	75	77	71
GR	C3	80	79	72
GR	C4	89	86	87
GR	C5	81	76	81
GR	C6	79	74	77
GR	C7	79	68	77
GR	C8	91	85	91
GR	C9	72	70	64
NV	C1	92	91	90
NV	C2	89	85	87
NV	C3	91	86	89
NV	C4	84	80	85
NV	C5	94	89	93
NV	C6	92	89	92
NV	C7	86	76	76
NV	C8	89	89	90
NV	C9	91	78	88
PV	C1	72	72	72
PV	C2	86	66	86
PV	C3	79	60	79
PV	C4	66	66	86
PV	C5	88	88	88
PV	C6	100	100	100
PV	C7	86	66	100
PV	C8	91	34	91
PV	C9	72	72	72

TAB. 39 – Résultats préliminaires pour les corpus oraux.

catégorie	LPL-1	LPL-2	LPL-3
NV	**90,33**	90,00	89,33
GN	87,22	**87,56**	82,78
GP	82,67	**83,33**	77,44
GA	**81,56**	77,89	78,11
GR	**89,78**	84,78	87,78
PV	**82,22**	69,33	86,00
TOTAL	**85,63**	**82,15**	**83,57**

TAB. 40 – Synthèse des résultats préliminaires pour les corpus oraux.

sera disponible, nous pourrons vérifier si cette tendance est encore vraie sur l'ensemble des corpus. Si tel n'était pas le cas, on pourrait en déduire que les groupes nominaux et prépositionnels à l'oral ont réellement une grammaire différente de celle de l'écrit. Cette question pourrait permettre d'améliorer précisément les grammaires de LPL1 et LPL3.
– L'analyseur profond (LPL-1) conserve toujours les meilleurs scores.
– L'analyseur à granularité variable (LPL-3) suit la même tendance que LPL-1 en restant toujours en dessous de ses scores, sauf pour les groupes PV (groupes verbaux introduits par une préposition). Nous disponsons ici d'une information clé pour comprendre les origines des écarts constatés au sein des grammaires employées dans LPL1 et LPL3. Comme l'ensemble des résultats n'est pas disponible, nous ne poussons pas plus loin nos investigations.

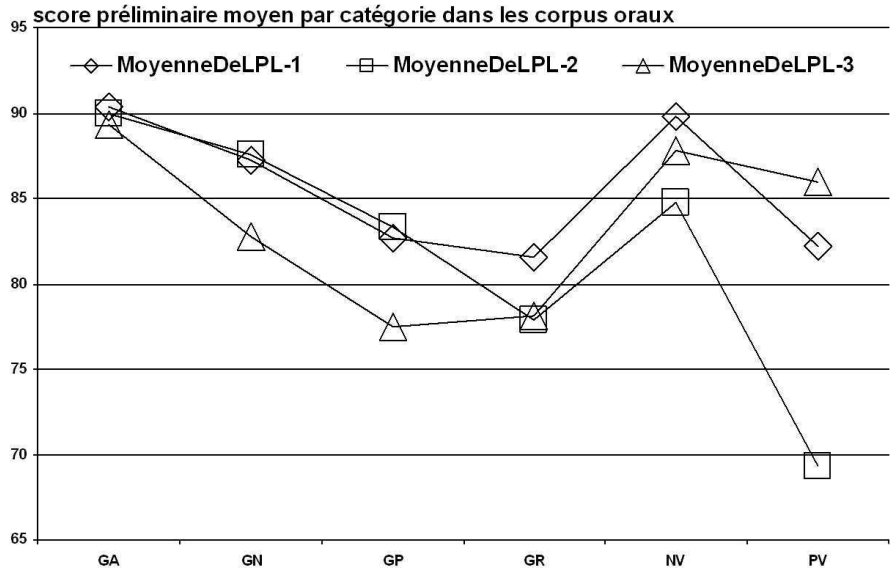


FIG. 70 – Comparaison des trois analyseurs sur les corpus oraux catégorie par catégorie.

Chapitre 15

Conclusion

Dans cette partie, nous avons abordé la question de l'évaluation des analyseurs syntaxiques, ainsi que des outils qui les précèdent dans la chaîne des traitements. Notre première conclusion est qu'il est possible d'évaluer nos outils, même en l'absence de corpus de référence, mais que la participation à des campagnes à grande échelle (telles que la campagne EASY) apporte des informations cruciales pour comparer nos outils et ressources avec d'autres approches. La seconde conclusion porte sur la qualité relative des analyseurs testés. Nous avons pu observer l'influence de l'étiquetage sur la qualité de l'analyse, mais pour un étiquetage identique, les analyseurs produisent des résultats assez différents. Il est possible, en observant de près ces différences, de déterminer lesquelles proviennent d'éléments grammaticaux (ressources grammaticales différentes) et lesquelles proviennent de divergences d'interprétation algorithmique (différentes spécifications sémantiques pour les analyseurs profonds et techniques de construction différentes entre les analyseurs). Enfin, de nombreuses perspectives d'amélioration des outils sont ouvertes. Notre étiqueteur morpho-syntaxique peut encore être amélioré en tenant compte du fait que la plupart des défaillances sont en grande partie liées aux erreurs d'étiquetage dans le dictionnaire, tel le surnombre de formes verbales au participe passé interprétées comme adjectifs. Ces défaillances n'en sont pas pour autant des erreurs. D'autre part, les bases de NGrammes sont sensibles aux traits, ce qui accroît le nombre de variations possibles et diminue d'autant la qualité de la base. Il faudrait au moins multiplier la taille du corpus d'apprentissage par dix (actuellement, quelques millions de mots) pour espérer des informations suffisantes sur tous les types de catégories avec tous les traits possibles. Les grammaires peuvent être améliorées en utilisant les résultats du multiplexage : en effet, les points forts et les points faibles de chaque approche sont mis en évidence, ce qui permet d'isoler les erreurs et les divergences entre les approches. Nous attendons encore beaucoup d'informations à partir des résultats de la campagne EASY. D'ores et déjà, le classement de nos trois analyseurs parmi les six premiers concurrents (troisième pour LPL1, quatrième pour LPL3 et sixième pour LPL2) donne une très bonne place aux approches basées sur les Grammaires de Propriétés, face à des analyseurs de haut niveau développés en laboratoire ou dans des applications déjà commercialisées. Enfin, une évaluation ne vaut que si elle prépare la suivante. En TALN, il serait trop ambitieux de considérer qu'un outil puisse être définitivement achevé.

Cinquième partie

Applications

Chapitre 16

Développement et validation de grammaires

De nombreuses plateformes de développement d'analyseurs et de grammaires ont été développés pour les formalismes TAG, GPSG et HPSG. Citons XTAG ([Paroubek *et al.*, 1992]), Pargram ([Butt 1999]), LKB ([Copestake, 2002]) ou encore Lingo Grammar Matrix([Bender *et al.*, 2002]).

Les outils que nous avons présentés jusqu'ici fonctionnent de façon modulaire, mais aussi comme un tout, accessible par des applications de haut niveau. Avant de présenter deux logiciels qui utilisent tout ou partie de la suite de modules LPLSuite, nous allons présenter un outil dédié à la maintenance et à la validation des ressources grammaticales dans le formalisme des Grammaires de Propriétés. Les Grammaires de Propriétés, exprimées dans les formats que nous avons présentés plus tôt, se prêtent parfaitement à une représentation sous forme de graphe. Aussi bien la sémantique des grammaires que les grammaires elles-mêmes et enfin les résultats de l'analyse, sont représentables sous forme de graphe. Ces graphes peuvent devenir très complexes et il devient difficilement imaginable de maintenir un fichier grammatical de plus de 3000 contraintes à l'aide seule d'un éditeur de texte. Développer une grammaire est une tâche rigoureuse qu'il faut distinguer de la tâche plus ingrate qui consiste à respecter le format XML et la correspondance avec les DTDs. Il est devenu nécessaire de se doter d'un outil de développement de grammaires associé aux ressources que nous avons développées. C'est le rôle du programme Accolade, que nous présentons ici. Accolade a pu être conçu grâce à la réflexivité que nous avons imposée à nos structures de données. La grammaire peut se référer à sa spécification sémantique. La caractérisation peut se référer à la grammaire. Une certaine introspection est possible (la grammmaire peut s'auto-connaître), ce qui a d'ailleurs fondé notre algorithme d'analyse. Une fois le logiciel présenté, nous décrirons la démarche semi-automatisée que nous avons mise en place pour développer, contrôler et valider des Grammaires de Propriétés.

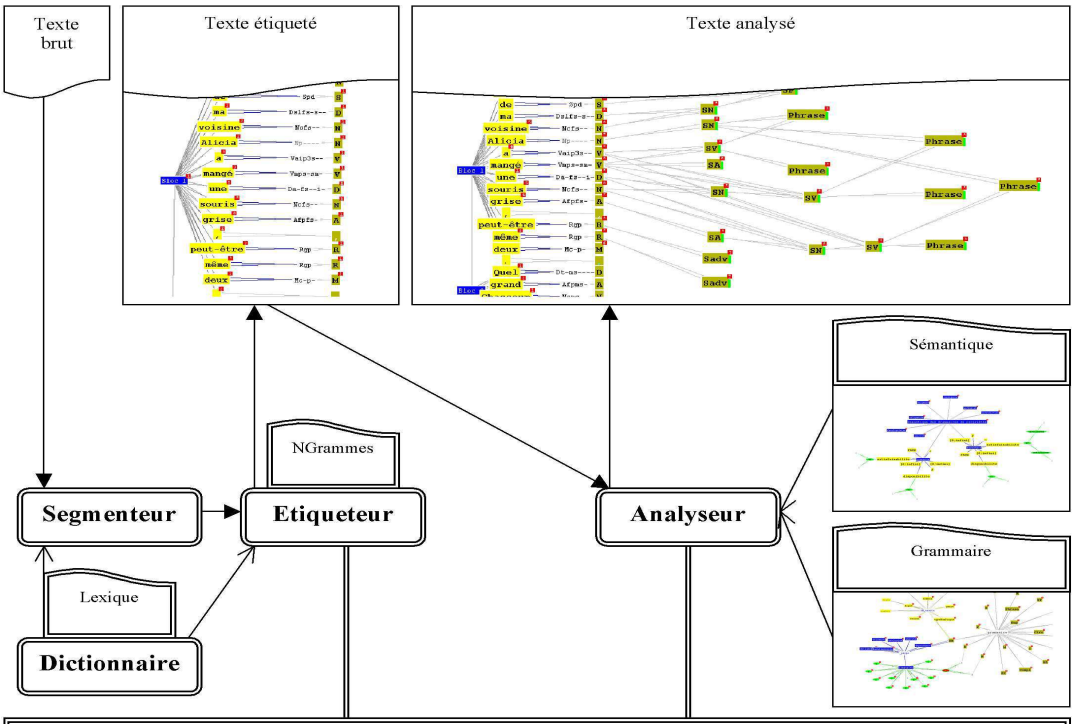
16.1 Un outil dédié à LPLSuite : Accolade

Le logiciel Accolade a été baptisé ainsi d'une part du fait qu'il regroupe l'ensemble des outils et ressources de LPLSuite pour en permettre une utilisation plus facile qu'à l'aide d'un enchainement de scripts de commande. En cela, il se destine à l'usage de personnes ayant à coeur de développer et d'utiliser des grammaires, plutôt qu'à un public informaticien. La

seconde raison de cette dénomination provient de l'usage courant du symbole accolade dans les descriptions syntaxiques. A la différence des parenthèses, les accolades sont employées dans le but de regrouper, de définir les bornes des syntagmes. La troisième raison est simplement pratique. Accolade signifie la même chose en Français et en Anglais, ce qui permet une compréhension plus rapide de l'utilité du logiciel. Parenthèse et crochet ne se traduisent pas du tout de la même façon.

L'outil Accolade, que décrit la figure 71 permet de sélectionner des ressources, des corpus d'entrée à analyser et de paramétrer les outils d'analyse. Il permet de modifier les Grammaires de Propriétés et leurs sémantiques dans un environnement graphique contrôlé par des routines qui surveillent la conformité des données avec le modèle de représentation interne. Ceci permet de modifier les ressources sans perdre de vue l'aspect linguistique. Il permet enfin de déclencher des analyses sur les entrées, soit de façon rapide dans le but de fournir des sorties d'analyse, soit étape par étape afin de visualiser chaque événement durant le processus d'analyse. L'ensemble des possibilités des modules de LPLSuite est accessible depuis Accolade. Nous allons concentrer notre présentation de l'outil sur ses fonctionnalités dans la perspective du développement de grammaires.

Pour représenter visuellement les Grammaires de Propriétés (notées GPs), les Spécifications Sémantiques des GPs (notées SSGPs) et les Caractérisations dans le formalisme des GPS (notées CGPs), Accolade se base sur le modèle décrit plus tôt dans la troisième partie. La figure 48 de cette partie montre le modèle qui permet la réutilisation graphique des nœuds et des arcs d'un graphe en leur associant un contrôleur et une figure. Le même type de graphe permet de représenter aussi bien les SSGPs et les GPs que les CGPs. Les nœuds de chaque graphe sont porteurs de l'information nécessaire au fonctionnement de l'ensemble. Donner une représentation graphique de ces graphes consiste en un affichage sélectif et interactif des éléments des graphes. Pour qu'une analyse soit effectuée, il faut charger un SSGP, une GP et un texte étiqueté qui sera transformé automatiquement en prégraphe de caractérisation. Comme nous l'indiquions dans la troisième partie, ce prégraphe est en quelque sorte une caractérisation vierge. Aucune analyse n'étant encore effectuée, ce graphe sert juste à initialiser le processus. L'interface graphique d'Accolade associe une fenêtre à chaque graphe. Il est possible de modifier chacun de ces graphes manuellement avec les commandes visuelles associées à ces graphes. Les modifications peuvent être enregistrées au format XML correspondant au type de la ressource. Comme les modifications sont contrôlées au fur et à mesure, les graphes restent consistants pour la tâche qu'ils permettront d'effectuer lors de l'analyse. De cette façon, un utilisateur peut travailler en déléguant la tâche de contrôle de conformité des ressources aux modèles. Le développement de ce logiciel est permis par le haut niveau d'abstraction des modules et des ressources. L'indépendance entre les programmes et les données permet de paramétrer finement les comportements. La cohésion entre le modèle informatique et les gestes manuels de modification des graphes est permise par l'introduction de la réflexivité dans la représentation interne des graphes. Chaque nœud de chaque graphe est typé au sein d'une hiérarchie d'objets très rigoureuse. Un nœud peut ou non être mis en relation avec d'autres nœuds en fonction des critères définis dans les DTDs. Cette cohésion est déjà à l'œuvre lors du processus d'analyse avec le module SeedParser. Comme nous l'avons vu, c'est grâce à elle que l'analyse est permise. Dans Accolade, l'introduction de figures graphiques associées à chaque nœud et chaque arc des graphes est permise par le modèle de contrôleur. La difficulté de représentation de tels graphes réside dans leur spécificité. Il ne s'agit pas d'arbres, la représentation arborescente est donc exclue. Il ne s'agit pas de graphes dont les arcs peuvent remonter à l'exception des arcs de référence. La représentation arborescente est



Accolade

- Utilisation interactive des modules de LPLSuite
  - Etiquetage
  - Analyse syntaxique
  - Formatage et sauvegarde des résultats
  - Mise en évidence des processus d'analyse
  - Mise en évidence des relations entre les niveaux de description :
    - Sémantique-Grammaire
    - Grammaire-Texte
- Manipulation des ressources de la suite
  - Développement de grammaires
  - Développement de sémantiques pour grammaires
- Modèle unique de représentation pour les graphes
  - Adéquation entre les objets manipulés et leur représentation graphique
  - Relation intime entre les formats (XML, DTD) et les graphes typés qui les représentent.
  - Le même modèle est utilisé pour les différentes ressources et les modules d'analyse (sémantique, grammaire, étiqueteur et analyseurs).

FIG. 71 – L'outil Accolade

donc valable pour presque tout le graphe, sauf pour les arcs qui relient un nœud catégoriel et les nœuds qui font référence à cette catégorie dans les descriptions de contraintes. Il a été intéressant de tirer parti de cette propriété. Les graphes sont des graphes acycliques dirigés, ce qui permet de les traiter graphiquement avec des outils de visualisation classiques. Notre recherche dans le cadre des possibilités de représentation des graphes de milliers de nœuds nous a menés à comparer différentes techniques. Il s'est rapidement avéré extrêmement difficile de choisir une technique générale. Certains outils de visualisation s'intégraient en effet assez bien dans l'architecture orientée objet de l'ensemble LPLSuite, mais leur lourdeur ou leur absence de prédestination au traitement de l'analyse par contraintes nous ont amenés à construire un modèle sur mesure pour la représentation et la manipulation de graphes. En utilisant la bibliothèque TouchGraph (développée et accessible gratuitement sous licence GPL à l'adresse http ://www.touchgraph.com/), en redéfinissant pratiquement la totalité des comportements des graphes de cette bibliothèque, nous avons mis pu mettre au point notre outil. Techniquement, la rédaction de routines de visualisation de graphes est difficile. Cette étape très longue permet cependant de manipuler les graphes des Grammaires de Propriétés sans connaissance informatique prérequise. C'est ainsi que le processus de développement de grammaires s'en trouve facilité. Un des avantages de l'outil Accolade réside dans la possibilité de visualiser pas à pas les étapes du processus d'analyse avec le parseur SeedParser décrit précédemment. Il est possible aussi de modifier le cours de l'analyse en agissant directement sur les graphes, afin de mettre en évidence des phénomènes particuliers. Enfin, l'état des nœuds du graphe variant beaucoup au fil de l'analyse, notamment parce que les mesures de densité de satisfaction évoluent à chaque étape, il a été intéressant de développer techniques visuelles permettant de n'afficher que certains éléments du graphe d'analyse en fonction de seuils de densité de satisfaction. On peut par exemple n'afficher que les nœuds dont le taux de satisfaction est supérieur/inférieur à un certain seuil. Ceci permet d'isoler des phénomènes et de vérifier précisément les effets de l'ajout d'une nouvelle contrainte dans la grammaire.

Avec cet outil et grâce aux travaux linguistiques autour des Grammaires de Propriétés, nous avons pu mettre en place une grammaire spécifique pour la campagne d'évaluation EASY. Les résultats actuellement connus pour cette campagne, qui ont été présentés dans la partie précédente, montrent l'importance de l'observation préalable à la constitution d'une grammaire. Avant de voir comment développer et valider une grammaire de propriétés avec l'ensemble des outils dont nous disposons, nous souhaitons présenter plus en détail l'interface de l'outil Accolade.

Accolade se lance par une commande java. Au chargement, un fichier d'initialisation (accolade.ini) est chargé, analysé et interprété afin de rétablir l'environnement d'utilisation que l'utilisateur a pu sauvegarder au cours d'une session d'utilisation précédente.

Nous donnons ci-dessous le contenu de ce fichier, modifiable manuellement (avec un éditeur de texte) ou depuis l'interface d'Accolade. Ce script permet à Accolade de précharger un ensemble de ressources (lexique, spécification sémantique des Grammaires de Propriétés, Grammaires de Propriétés, jeux de traits morphosyntaxiques associés, banque de NGrammes pour le désambiguiseur, et enfin éventuellement un fichier désambiguïsé à précharger.

```
**** ne modifiez par le texte qui précède les symboles '='

SEMANTIQUE=<USERDIR>/Ressources/Grammaires/SemantiqueGP.xml
GRAMMAIRE=<USERDIR>/Ressources/Grammaires/grammaire.easy.groupes.1.5.TVR.xml
TRAITS=<USERDIR>/Ressources/Etiqueteur/Traits_LPL.dat
DICTIONNAIRE=<USERDIR>/Ressources/Dictionnaire/DicoLPL.zip
ETIQUETEUR.NGRAMMES=<USERDIR>/Ressources/Etiqueteur/SyntaxeCM4.zip
CHARGER_DESAMBIGUISATION=<USERDIR>/../EntreeSortie/tests/test_01.txt.etiq.xml

**** constantes de manipulation des outils TAL ****
**** l'analyseur peut empêcher la repetition d'une categorie au dessus de la même souche
**** (gain de temps et de mémoire)
**** Ainsi, pour l'exemple Det(1)+Nom(2), mettez la variable à VRAI pour eviter
****
****    SN(1-2)
****      |
****    SN(1-2)
****    /    \
**** Det(1)  Nom(2)
**** mettez la variable à FAUX pour permettre les repetitions (attention, danger d'explosion)
ANALYSEUR.EMPECHER_REPETITION=VRAI

**** l'analyseur peut autoriser ou non les contraintes basées sur l'absence d'une categorie de la souche
**** à projeter des categories. Par exemple, l'exclusion d'un Y par un X dans les contraintes du SX
**** permettra de projeter un SX si un Y n'est pas present dans le contexte.
****
****                                SX
****                                 |
****    Exclusion violee       Exclusion satisfaite
****    /              \       /
****   X               Y       X               Z
**** Le contraintes pouvant être lacunaires sont l'exclusion, l'exigence et l'interdiction
**** mettez la variable à FAUX pour interdire ce genre de projection
**** Attention, danger d'explosion si vous passez à VRAI: en effet, les contraintes lacunaires sont
**** facilement satisfaisables et sur des souches très larges.
ANALYSEUR.AUTORISER_PROJECTION_SUR_CONTRAINTES_LACUNAIRES=FAUX

**** l'analyseur peut choisir une proposition parmi celles que fait l'etiqueteur (VRAI)
**** l'analyseur peut garder les propositions de l'etiqueteur (FAUX)
**** l'analyseur peut ne pas tenir compte des propositions de l'etiqueteur et tenter toutes les possibilités (AUCUNE)
ANALYSEUR.FORCER_DESAMBIGUISATION_DETERMINISTE=VRAI

**** l'analyseur peut choisir de ne garder que la categorie la plus large pour une souche donnée
**** (gain de temps et de mémoire)
**** Ainsi, pour l'exemple Det(1)+Nom(2), mettez la variable à FAUX pour obtenir {SN(1), SN(2), SN(1-2)}
**** mettez la variable à VRAI pour obtenir {SN(1-2)}
ANALYSEUR.GARDER_PROJECTION_LA_PLUS_LARGE=FAUX

**** l'analyseur peut choisir d'éliminer les categories dont la densité de satisfaction serait trop basse
**** (gain de temps et de mémoire)
**** choisissez une valeur entière de 0 à 100 %
ANALYSEUR.FILTRE_SATISFACTION=90
ANALYSEUR.FILTRE_SAT_PROPAGEE=85
```

Lorsque le script est chargé, son contenu est interprété et un ensemble de structures de données est construit en mémoire. La spécification sémantique et la grammaire sont convertis en graphes destinés au module d'analyse syntaxique SeedParser, selon le modèle décrit en quatrième partie. Un objet Dictionnaire est construit pour permettre à l'utilisateur de déclencher une analyse. L'interface graphique est ensuite construite.

La figure 72 montre l'interface d'Accolade au moment du lancement. L'application est composée d'un menu et de plusieurs cadres redimensionnables. Le cadre gauche permet de

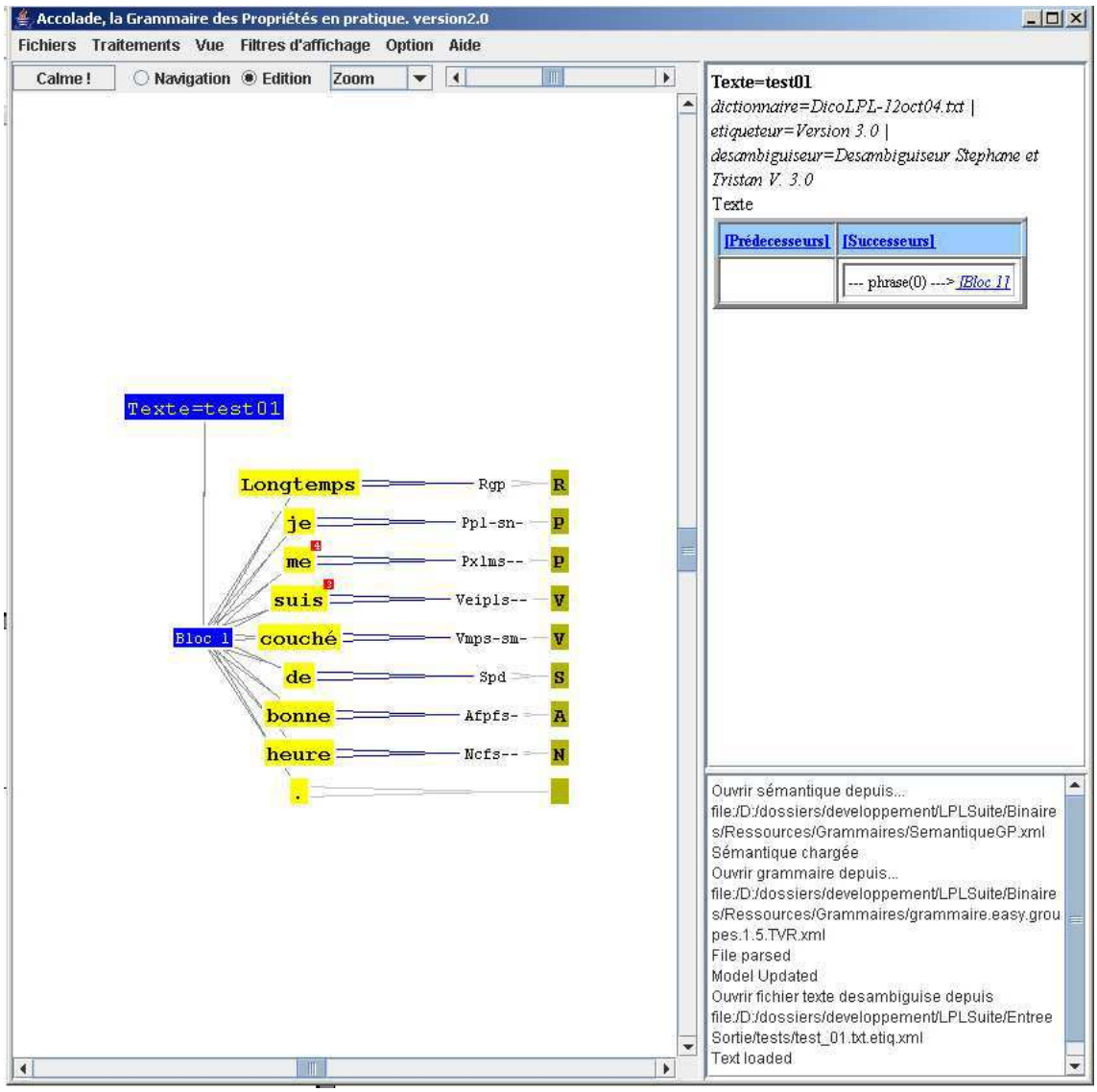


FIG. 72 – Interface graphique de l'application Accolade

représenter des graphes, aussi bien de sémantique, que de grammaire ou de caractérisation. Le cadre droit est divisé en deux sous-cadres. Celui du haut contient une information hypertextuelle sensible aux éléments sélectionnés dans le graphe du cadre gauche. Pour simplifier nos descriptions, nous appellerons ce cadre gauche la *vue* et le cadre en haut à droite sera appelé *cadre d'information contextuelle*. Le dernier cadre (en bas à droite) contient une information chronologique concernant chaque tâche accomplie au cours de la session. Le cadre de vue contient à son tour un ensemble de boutons de commande permettant de naviguer dans la vue représentant le graphe actuellement affiché.

Le menu d'Accolade et ses sous-menus sont présentés par la figure 73. Certains de ces sous menus ont des intitulés explicites et ne seront pas détaillés ici.

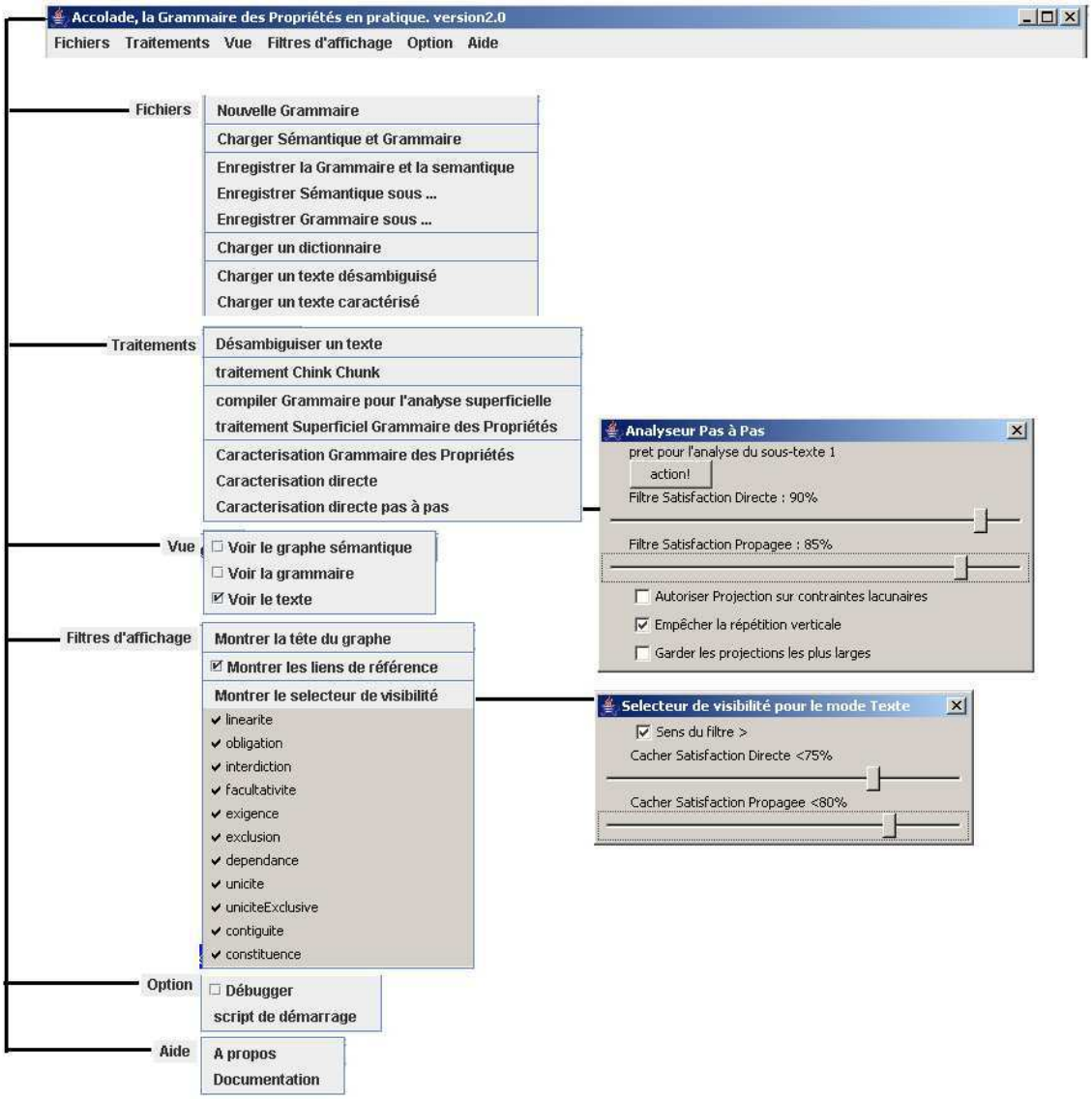


FIG. 73 – Menus et sous-menus de l'application Accolade

Le sous-menu *Fichier* donne accès aux fonctions de chargement et de sauvegarde pour les ressources utiles au fonctionnement d'Accolade.

Le sous-menu *Traitements* permet de déclencher les modules de LPLSuite tels que l'étiquetage et la désambiguisation, mais aussi trois types d'analyseurs syntaxiques. Le premier analyseur disponible est le *chunker* que nous avons décrit dans la partie consacrée à l'évaluation, basé sur une grammaire Chink/Chunk que nous avons donné à cette occasion. Le second analyseur a lui aussi été présenté dans la même partie. Il s'agit de l'analyseur fonctionnant à partir d'une forme précompilée de la grammaire. Accolade permet d'accéder à ces deux modules d'analyse, afin de permettre les comparaisons entre les techniques. Le troisième analyseur est *SeedParser*, que nous avons présenté dans la troisième partie. Ce dernier est accessible de plusieurs façons à partir d'Accolade : soit en déclenchant l'analyse rapidement sur un texte désambiguisé, sans afficher de sortie intermédiaire dans l'interface d'Accolade. Ceci permet de traiter de grands corpus sans perdre des ressources (mémoire) à construire des représentations graphiques qui seraient trop nombreuses. La seconde façon d'appeler SeedParser est plus interactive puisqu'elle déclenche le mode pas à pas de SeedParser et visualise chaque transformation dans le cadre de vue. En lançant une analyse pas à pas, l'utilisateur peut choisir dans la fenêtre qui s'affiche de paramétrer finement le comportement de SeedParser (seuils de densité de satisfaction, ainsi que les paramètres heuristiques tels que l'autorisation ou l'interdiction de construire des structures redondantes.

Le sous-menu *vue*, permet d'afficher dans le cadre de vue soit le graphe de la spécification sémantique, soit le graphe de la grammaire, soit le graphe de caractérisation. Lorsqu'un texte désambiguisé est chargé, il est transformé une structure de graphe que nous avons appelée *prégraphe de caractérisation*. Ce graphe est accessible depuis le sous-menu *texte* du menu *vue*. Au lancement de l'application, si le script d'initialisation contenait un ordre de chargement de texte désambiguisé, la vue bascule automatiquement sur le prégraphe de caractérisation qui en est déduit. Sinon, la vue bascule sur le graphe grammatical. Il est aussi possible de naviguer entre ces vues à partir du cadre d'information contextuelle.

Le sous-menu *Filtres d'affichage* est dépendant de la vue courante, ainsi que des ressources actuellement disponibles. Ce menu offre des filtres permettant de simplifier l'affichage des graphes, en autorisant ou en interdisant l'apparition des nœuds ayant trait à telle ou telle propriété ou des arcs de référence. Ce menu permet aussi de déclencher un processus dynamique d'affichage ou de masquage des nœuds créés au fil de l'analyse. Un processus d'analyse sur une phrase de quelques mots risque en effet de provoquer la création de beaucoup de nœuds et arcs qui rendent rapidement illisible la structure affichée. Avec ce filtre dynamique, il est possible de ne conserver à l'écran que les nœuds dont la valeur de densité de satisfaction est supérieure ou inférieure à une valeur donnée de densité de satisfaction. Ceci permet de n'afficher que les constructions les plus correctes ou bien les moins correctes. Lors de l'analyse pas à pas, ce filtre est indispensable pour appréhender le déroulement des opérations.

Les étapes de l'analyse avec SeedParser ont déjà été étudiées dans la troisième partie, où ont été présentées des structures de graphes de caractérisation issues de l'application Accolade. Nous n'avons cependant pas observé les structures de graphes sémantiques et grammaticales vues depuis Accolade. Les figures 74 et 75 montrent ces deux vues.

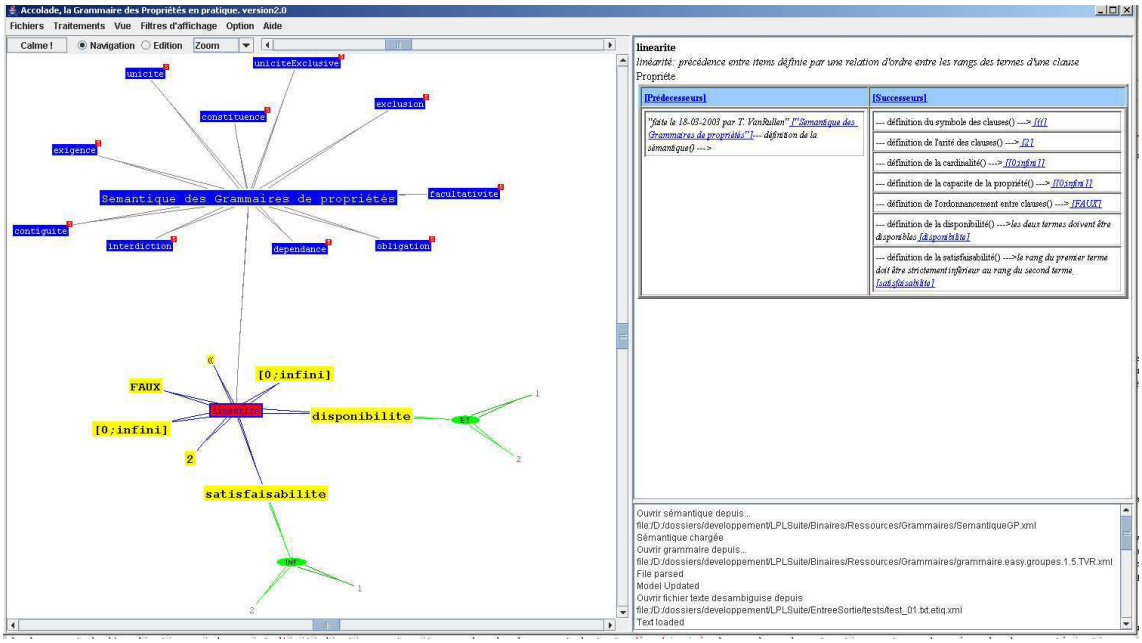


FIG. 74 – Accolade : vue de la spécification sémantique des GPs

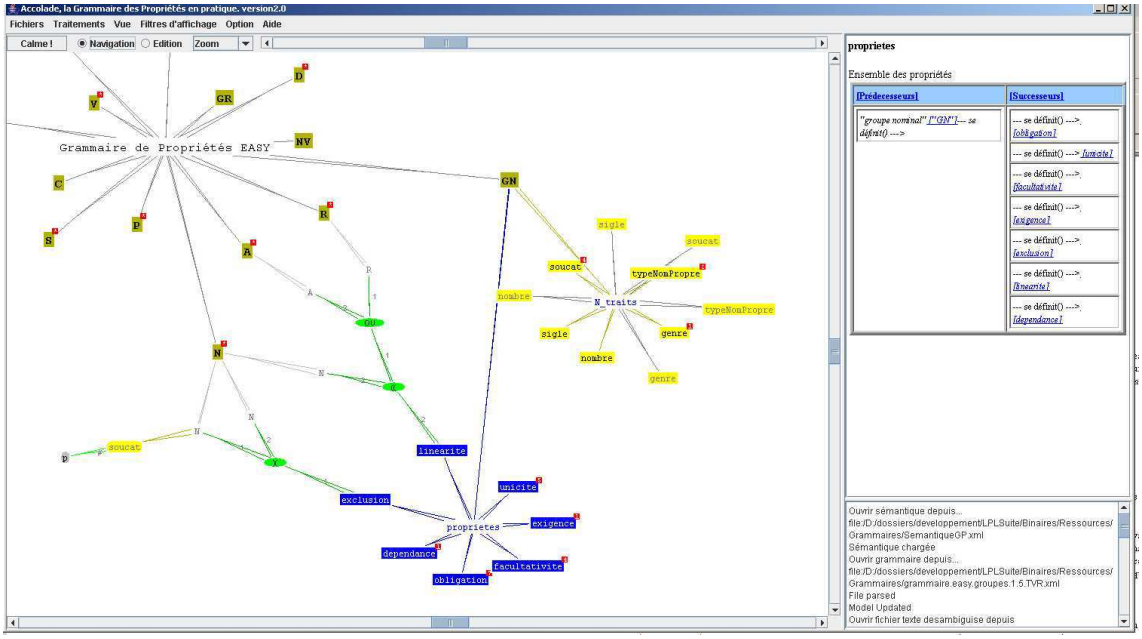


FIG. 75 – Accolade : vue de la grammaire

Enfin, la figure 75 montre un début d'analyse syntaxique avec SeedParser en *mode pas à pas*.

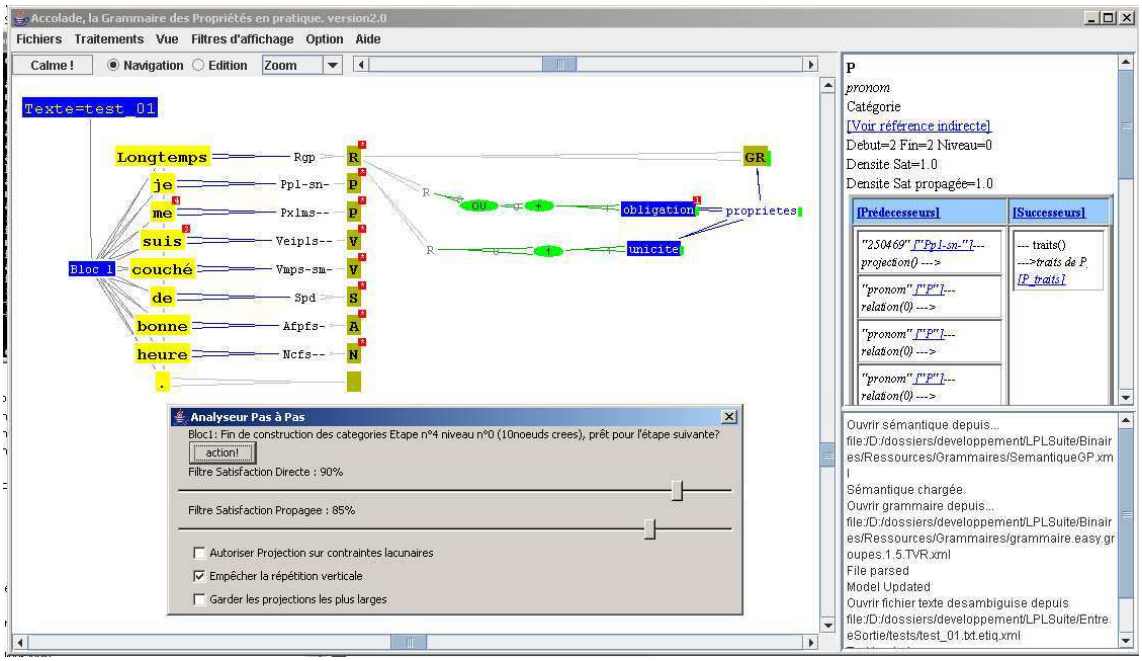


FIG. 76 – Accolade : vue du texte en cours de caractérisation

16.2 Développement et validation des grammaires sur corpus

Le processus empirique consistant à développer une grammaire peut être envisagé dans un environnement semi-automatisé. Grâce à l'outil Accolade, nous avons mis en place la technique que schématise la figure 77 : une séquence semi-automatique de mises au point empiriques de différentes versions de grammaires.

Le développement de grammaires à large couverture pour le Français dépend dans notre plateforme de plusieurs éléments : des corpus simples destinés aux tests sur des parties de grammaires, une grammaire en cours d'élaboration et un ou plusieurs analyseurs. Afin de tester plus en avant la grammaire, de grands corpus sont utilisés, comme le corpus journalistique Le Monde (3 millions de mots annotés et désambiguisés au LLF, Université de Paris VII).

L'analyseur SeedParser peut être employé pour évaluer la grammaire en cours de développement. Il est possible d'observer ainsi des réponses variant en fonction de l'entrée ou de la grammaire en étudiant soit une partie, soit la totalité de la grammaire. On peut par exemple étudier le poids et l'incidence d'un sous-ensemble particulier de la grammaire, afin d'en déduire des modifications éventuelles, si une erreur récurrente apparait, sans pour cela devoir prendre en compte la totalité de la grammaire. Cela peut être réalisé sans modifier la structure de la grammaire ou l'analyseur, simplement en neutralisant les données non désirées, grâce aux

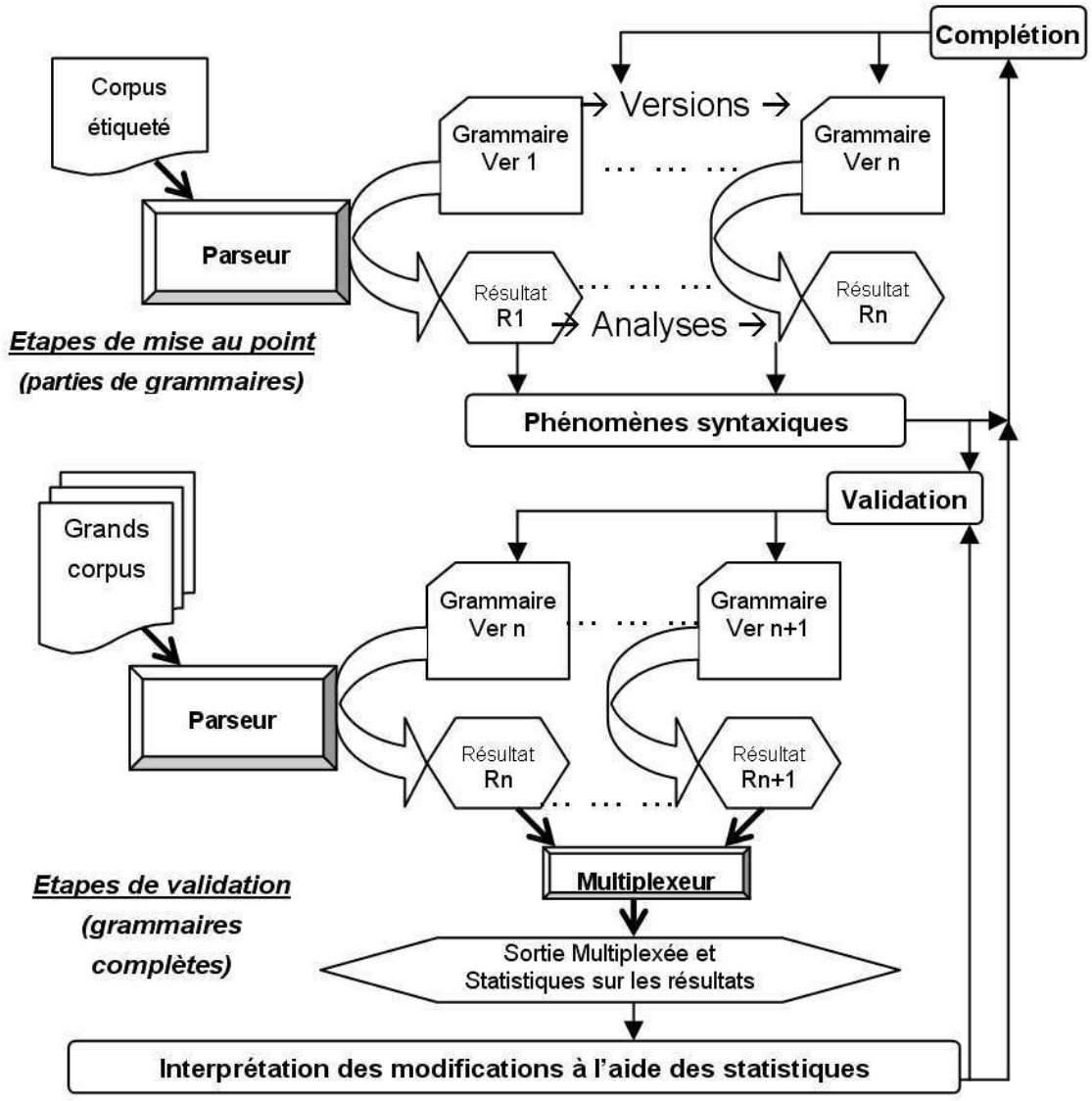


FIG. 77 – Séquences de mises au point semi-automatiques de grammaires

filtres d'affichage. Le non-déterminisme de l'analyseur permet en outre d'évaluer la qualité de la grammaire en regardant le nombre et la qualité des analyses proposées.

Le processus de développement grammatical peut alternativement être effectué de trois manières. Premièrement, d'un point de vue *descriptif* : dans ce cas, nous isolons du corpus différentes occurrences d'une construction sur laquelle nous voulons nous concentrer (clivées, coordinations). Nous réalisons tout d'abord une description linguistique fine basée sur l'observation des productions attestées et nous intégrons à notre grammaire les propriétés correspondant à nos conclusions théoriques. Ensuite, nous employons l'analyseur sur notre corpus de tests pour comparer les résultats avec ceux obtenus dans une version précédente de la grammaire. Nous pouvons alors observer l'évolution des résultats en fonction de l'introduction, la modification ou la suppression de propriétés grammaticales, les proportions de cette évolution et sa qualité permettent des modifications précises.

La seconde perspective de développement est liée à la *couverture* de la grammaire : dans ce cas, la première étape consiste en un isolement, parmi les résultats obtenus avec une version précédente de la grammaire, de plusieurs occurrences de certaines difficultés d'analyse (par exemple, les dépendances non bornées). On peut alors étudier en détail les erreurs causées par la grammaire, soit par omission soit par manque de précision. Une fois corrigée, la grammaire est à nouveau testée pour donner une nouvelle version qui pourra à nouveau être comparée à la précédente.

Troisièmement, en ce qui concerne l'*ensemble des propriétés* lui-même et sa *sémantique* : dans ce cas, nous isolons un type de propriété et nous observons son comportement. Cette dernière exploitation du SeedParser permet d'observer directement l'impact d'un type particulier de propriétés dans les résultats d'analyse. Ceci peut mener à modifier l'ensemble des contraintes lui-même. C'est à dire leur mécanisme et ou même leur présence. Par exemple, on peut comparer la qualité d'un résultat d'analyse obtenu avec un seul ensemble de propriétés d'une part, et un autre résultat obtenu avec un autre ensemble de propriétés d'autre part. Alors, en se référant aux résultats obtenus avec l'ensemble complet des propriétés de la grammaire, on peut en déduire le poids d'un type particulier de propriétés sur l'analyse. Par exemple, si la majorité des bons résultats est trouvée dans les résultats obtenus avec seulement ce type, ou encore sans ce type de propriétés (par exemple, si le taux de mauvaises réponses décroît en l'absence de ce type de propriétés). Ceci peut conduire à une redéfinition en profondeur de la sémantique d'un type particulier de propriétés, c'est à dire de son mécanisme de satisfaction, ou encore sa simple suppression si une erreur récurrente découle de cette propriété.

Considérons par exemple les résultats d'analyse pour la phrase suivante extraite d'un corpus de dialogues :
"*Alors on vous demandera aussi heu pourquoi c'est le meilleur*" Une première analyse avec une version de la grammaire donne (Figure 78) :

En observant ces résultats, nous avons effectué de nombreux changements dans la grammaire en utilisant alternativement chacune des trois techniques décrites ci-dessus. Une nouvelle proposition est obtenue avec une nouvelle version de la grammaire (Figure 79) :

FIG. 78 – Une proposition avec une première version de la grammaire

FIG. 79 – Une nouvelle proposition avec une version ultérieure de la grammaire

Il est évident que chaque modification de la grammaire, quelle que soit la technique, peut avoir des effets de bord sur le reste de l'analyse. Par exemple à cause des éléments que nous avons laissés de côté durant nos différentes étapes de correction sur des phénomènes isolés. Par exemple, la modification du graphe de contraintes décrivant une catégorie dans le but de permettre un certain traitement de structures peut mener à la prolifération d'analyses inadéquates pour le reste du corpus. Si ce phénomène se répète trop souvent, ceci peut amener à des résultats trop erronés. C'est pourquoi il est impératif, pour chaque modification locale de la grammaire, de lancer l'analyse sur l'intégralité du corpus, avec la totalité de la grammaire afin d'être avisés des effets indésirables. Si nécessaire, une nouvelle modification de la grammaire devra alors être mise en œuvre. Cette vérification peut être effectuée avec le parseur en le paramétrant en fonction des besoins pour plus de robustesse sur le corpus à analyser. L'avantage du SeedParser réside dans la possibilité de paramétrer la granularité de l'analyse en fonction des besoins.

Chaque nouvelle version de la grammaire est donc testée sur de grands corpus à l'aide du parseur. L'évaluation de cette étape s'appuie sur l'analyse des résultats par comparaison automatique des frontières gauches et droites des constructions proposées par l'analyseur avec celles obtenues avec les versions précédentes de la grammaire. L'outil de multiplexage que nous avons déja présenté permet de réaliser cette comparaison en fournissant des statistiques locales portant sur la taille, la nature et le nombre des types de syntagmes construits. Nous tirons de ce dernier outil la possibilité de comparer rapidement et efficacement des jeux de résultats, avec ou sans corpus de référence. De cette façon, nous pouvons préserver un point de vue général sur l'efficacité de la grammaire à travers cette élaboration pas à pas.

Les résultats de multiplexage donnés ci-dessous (Figure 80) montrent par exemple deux versions d'une grammaire qui ne varient pas dans leur description des groupes nominaux (NPs), mais qui donnent 25% de groupes verbaux différents et 15% de groupes prépositionnels

différents. Ces indications sont données à chaque opération de multiplexage parmi beaucoup d'autres statistiques. Il revient à l'observateur de juger empiriquement des effets d'un ajout, d'une modification ou d'une suppression entre deux grammaires comparées. Ceci devient rapidement efficace en interprétant les résultats globalement lorsqu'on effectue un multiplexage, et localement lorsqu'on revient aux étapes plus fines de correction.

Grammar	GP1	GP2
Phrases / sentence	19.04	18.97
Words / phrase	1.50	1.50

VP Stats	GP1	GP2
GP1	100%	**75**%
GP2		100%

NP Stats	GP1	GP2
GP1	100%	**100**%
GP2		100%

PP Stats	GP1	GP2
GP1	100%	**85**%
GP2		100%

FIG. 80 – Quelques résultats de multiplexage entre deux résultats fournis avec deux versions d'une grammaire

Les résultats de ces expériences sont intéressants pour le développement de grammaires autant que pour l'amélioration des analyseurs. Avec les modules de LPLSuite et l'outil Accolade, nous avons mis en place une plateforme de développement de grammaires. Même si le processus de développement reste empirique, il devient plus aisé d'envisager une grammaire comme un tout et de garder à l'esprit les préoccupations linguistiques en se détachant des considérations informatiques.

Chapitre 17

Intégration logicielle

Les outils et les ressources que nous avons développées sont destinés aux tâches usuelles dans le domaine du TALN. Des applications de haut niveau peuvent se servir de tout ou partie des outils de LPLSuite. Une intégration au logiciel de synthèse vocale Syntaix (Laboratoire Parole et Langage) est dans une phase très avancée. Nous pouvons espérer améliorer ainsi la métrique imposée aux constructions par l'ancien algorithme Chink/Chunk, en se basant sur des informations plus profondes fournies par un analyseur à granularité variable tel que SeedParser. Une autre intégration logicielle est quant à elle parfaitement achevée : il s'agit de la Plateforme de Communication Assistée PCA développée au Laboratoire Parole et Langage. Dans cette application, les modules de bas niveau ont été complètement incorporés au logiciel. Le lexique ainsi que les outils de désambiguisation permettent de piloter la prédiction syntaxique pour faciliter la communication de personnes lourdement handicapées.

17.1 Synthèse vocale : Syntaix

Le programme de synthèse vocale Syntaix a été développé par Philippe Di Cristo en 1998. Nous avons souhaité le rendre plus accessible aussi bien à d'autres applications qu'à une utilisation directe. Une première interface graphique en TCl/TK a été développée par P. Di Cristo, mais celle-ci n'était pas vraiment destinée à un usage public. D'autre part, l'accès aux modules de Syntaix depuis des programmes écrits en JAVA n'était pas simple et nécessitait de compiler plusieurs versions pour que le logiciel soit supporté par différents systèmes d'exploitation. En portant ce programme en langage JAVA, nous avons pu lui adjoindre quelques modifications et le rendre accessible depuis plusieurs systèmes et depuis d'autres applications JAVA. Le développement de notre version JAVA du programme Syntaix n'est pas achevé car quelques modules d'origine écrits en PERL ou en C nécessiteraient encore plusieurs mois de travail. Toutefois, la version actuelle permet d'effectuer une synthèse complète depuis une interface graphique dédiée ou depuis un programme JAVA appelant.

Nous ne souhaitons pas présenter ici la totalité de l'outil Syntaix, ce qui a été fait par Philippe Di Cristo dans sa thèse, mais montrer l'incorporation des outils de LPLSuite à la nouvelle version de l'application. Rappelons que Syntaix est une application de synthèse vocale basée sur le moteur de concaténation de diphones MBRola développé par Thierry Dutoit. Syntaix est modulaire, c'est à dire que chaque traitement réalisé dans la chaîne qui va du texte jusqu'au signal sonore, peut être considéré comme un élément d'un puzzle. Chaque module de traitement peut éventuellement être remplacé par un autre module pour peu que

celui-ci respecte les formats d'entrée/sortie.

Les intérêts de Syntaix sont nombreux, mais restent trop peu mis en valeur. Cette application a été essentiellement développée et utilisée au sein du Laboratoire Parole et Langage, mais sa compétitivité face aux autres systèmes de synthèse vocale restait en 1998 très bonne, notamment quant à la qualité de la prosodie. Le module de calcul prosodique de Syntaix reste très intéressant, car il est fondé sur un petit nombre de règles établies par Albert et Philippe Di Cristo. Une des lacunes de Syntaix réside dans le découpage syntaxique actuel (un simple découpage en Chunks sans information linguistique). Quelques heuristiques ont été ajoutées pour détecter les groupes nominaux et les groupes verbaux afin d'améliorer la métrique des chunks et d'éviter la synthèse de tronçons sans pause trop longs. Nous avons souhaité améliorer le calcul des chunks dans Syntaix en utilisant l'analyseur syntaxique SeedParser. Il reste encore de nombreux obstacles à l'intégration de cet outil comme module d'analyse syntaxique dans Syntaix. Comme nous le verrons dans la présentation ci-dessous, malgré le souci de modularité affiché lors de la mise au point de Syntaix, trop de formats et d'outils restent opaques et difficilement reprogrammables. Par exemple, le module de phonétisation, qui est un script Perl assez ancien doté de nombreuses règles, est difficilement maintenable. Le reprogrammer en JAVA n'entrait pas directement dans la portée de nos travaux. Nous gardons à l'esprit la perspective d'achever ce travail si les besoins s'en font ressentir.

La figure 81 montre la fenêtre principale de la nouvelle interface de l'application Syntaix. Cette interface permet une interaction directe, mais il reste possible d'accéder au programme depuis une ligne de commande ou encore depuis un programme appelant. On peut y saisir un texte à synthétiser, ou encore le charger depuis un fichier. On peut aussi y choisir la voix qui sera employée (trois voix masculines et deux féminines). Depuis cette interface graphique, le paramétrage des voix est préprogrammé, mais il est toujours possible pour un programme appelant de modifier finement les caractéristiques de ces voix. La version originale de Syntaix donnait accès uniquement à une voix masculine. Nous avons donc amélioré ce point en intégrant de nouvelles voix (bases de diphones) fournies avec les ressources du programme MBRola. Nous renvoyons le lecteur à la thèse de Philippe Di Cristo pour tous les détails concernant la synthèse vocale avec Syntaix.

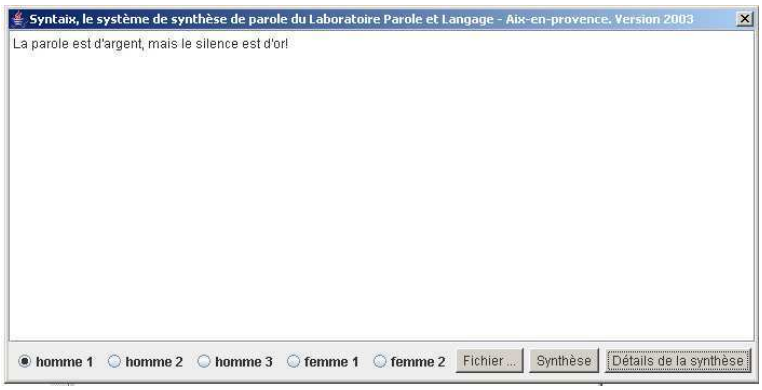


FIG. 81 – Interface graphique de l'outil Syntaix

Les étapes de la synthèse sont assez classiques : de la segmentation du texte brut en mots, phrases et groupes syntaxiques à la construction de représentations métriques et prosodiques

de cette entrée, jusqu'au fichier phonétique qui sera transmis au programme MBRola. Nous avons conservé la même succession de traitements dans la version actuelle. Les figures 82 à 91 mettent en évidence ces traitements à partir d'un texte donné en exemple (La parole est d'argent, mais le silence est d'or). Nous commenterons pour chaque figure les différences entre la version originale de Syntaix et la version actuellement développée.

L'intégration des outils de LPLSuite au programme Syntaix en est à ses débuts : nous souhaitons remplacer peu à peu les modules originaux par les modules de LPLSuite (notamment la segmentation, la désambiguisation et l'analyse syntaxique). En effet, le système de synthèse vocale a besoin, comme l'a lui-même indiqué P. Di Cristo, d'une qualité d'analyse importante pour rendre la voix plus naturelle. Le découpage en éléments constitutifs du texte entré est précisément le traitement le plus sensible à la qualité des ressources (lexique, données pour la désambiguisation et grammaires). De plus, la version initiale de Syntaix a des caractéristiques qui rendent la synthèse assez lente par souci d'informativité (la trace de chaque étape de la synthèse est accessible pour les besoins des utilisateurs, ce qui génère un assez grand nombre de fichiers intermédiaires redondants entre eux pour ce seul but informatif). Nous avons donc plusieurs raisons pour reconstruire la chaine des traitements de Syntaix.

La segmentation du texte en mots (figure 82 et 83) est basée sur le segmenteur (tokenizer) du projet MULTEXT. Cette segmentation de très bonne qualité s'appuie sur un lexique MULTEXT susceptible de fournir des informations morphosyntaxiques utiles aux processus de désambiguisation ultérieurs. Le lexique DICOLPL et ultérieurement le lexique MORPHALOU pourraient apporter plus de précision dans les connaissances apportées autant que dans le découpage. C'est pourquoi nous avons commencé à remplacer le module original par le segmenteur de LPLSuite.

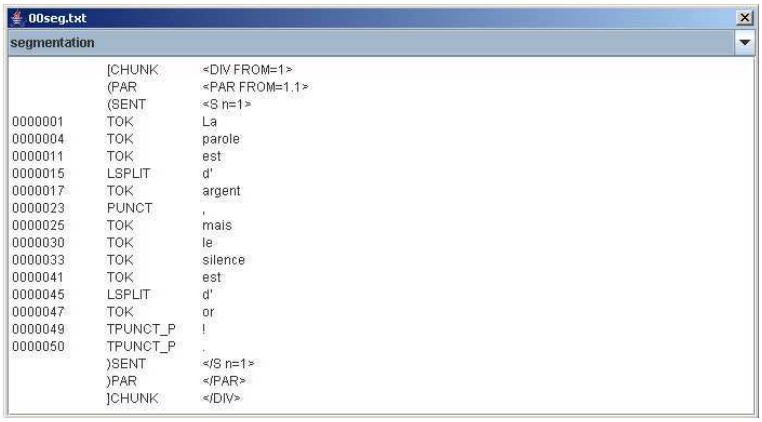


Fig. 82 – Details de la synthèse avec Syntaix : segmentation

La désambiguisation des catégories morphosyntaxiques (figure 84) est originalement basée sur un outil mixte alliant des NGrammes et des informations syntaxiques. Ce programme dont les NGrammes sont calculés sur des catégories simplifiées (peu de valeurs de traits) est néanmoins moins précis que le désambiguiseur de LPLSuite (dont les NGrammes sont calculés

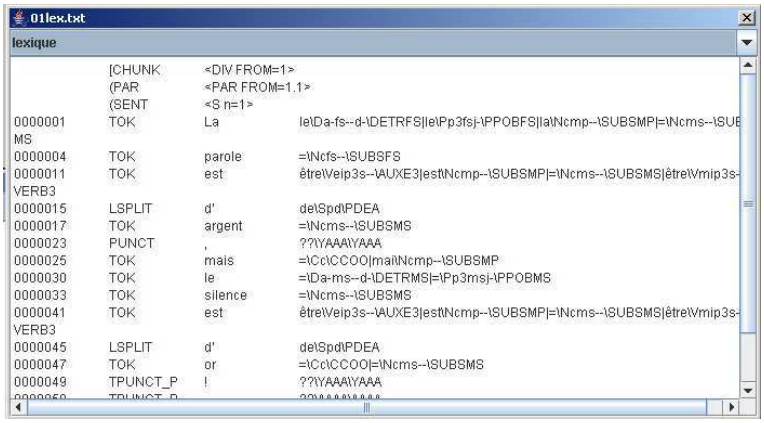


FIG. 83 – Details de la synthèse avec Syntaix : interrogation du lexique

en tenant compte des valeurs de trait des catégories). Là encore, nous avons commencé à intégrer l'outil de désambiguisation de LPLSuite à la nouvelle version de Syntaix. De nombreux problèmes de correspondance entre les outils restent à résoudre.

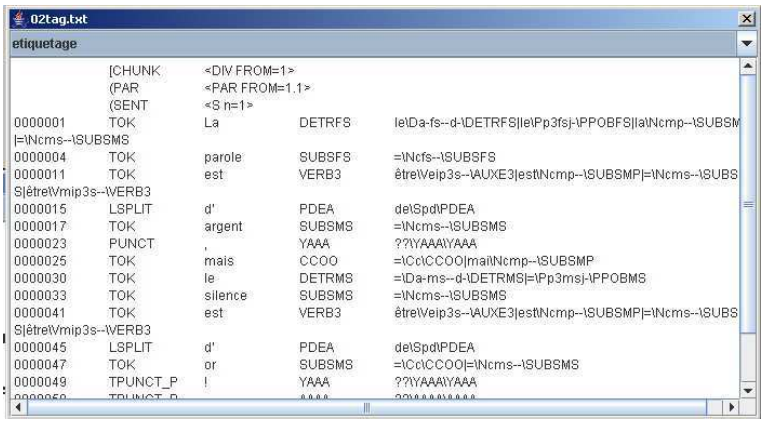


FIG. 84 – Details de la synthèse avec Syntaix : désambiguisation

L'étape de phonétisation (figure 85) dépend d'un script PERL assez difficile à maintenir dans des logiciels portables. La reconstruction du phonétiseur en JAVA est donc prévue dans la série des améliorations de Syntaix.

La métrique fondée sur un découpage proposé par un algorithme Chink/Chunk (figures 86 à 90) a prouvé ses qualités, mais manque encore de souplesse face à la variabilité et à la complexité des énoncés à traiter. Tout le travail réalisé pour développer des analyseurs à la fois robustes, riches en informations et rapides a été guidé en vue d'intégrer nos outils à des applications telles que Syntaix.

Des expériences préliminaires sont actuellement menées pour montrer les différences perceptives dans Syntaix entre un texte synthétisé à l'aide d'un outil de chunking et un texte

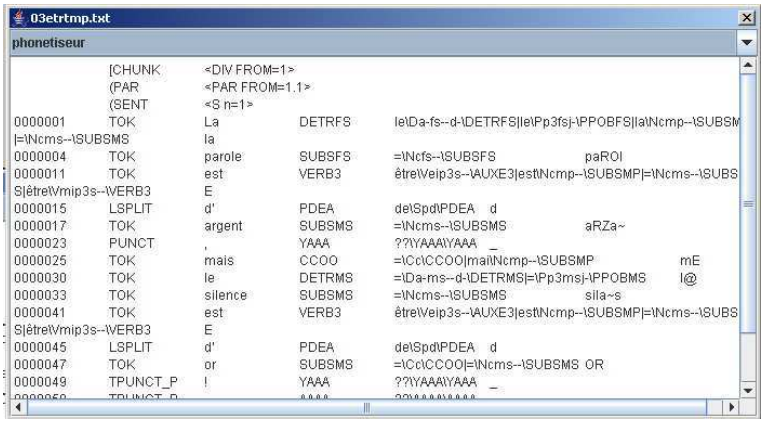


Fig. 85 – Details de la synthèse avec Syntaix : phonétisation

synthétisé à partir d'un parser plus profond tel que SeedParser. Nous en attendons une amélioration de la qualité métrique et prosodique de la voix synthétisée, tendant vers plus de naturel.

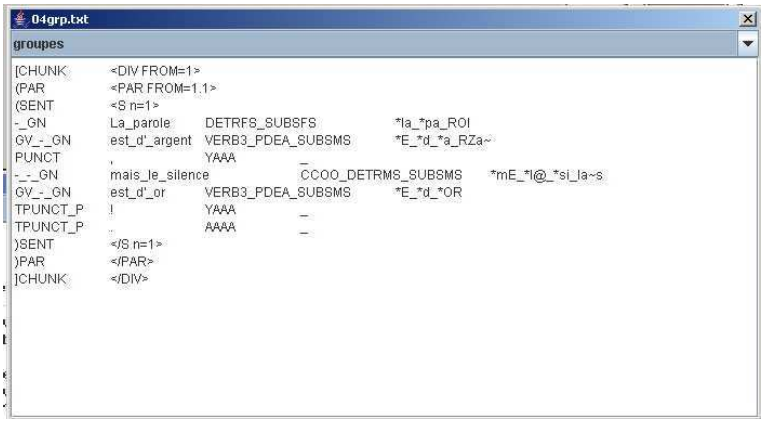


Fig. 86 – Details de la synthèse avec Syntaix : calcul des chunks

A la fin de tous ces traitements, les durées des phonèmes sont calculées et transformées au format d'entrée de MBRola. Les figures 89 et 90 mettent en évidence ces formats.

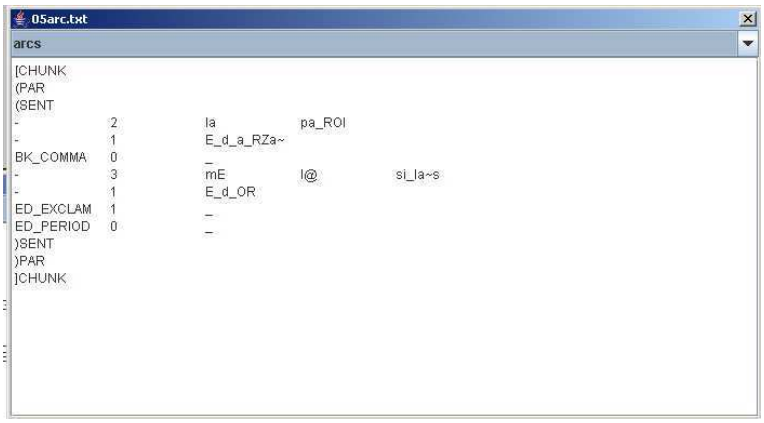


FIG. 87 – Details de la synthèse avec Syntaix : calcul des arcs

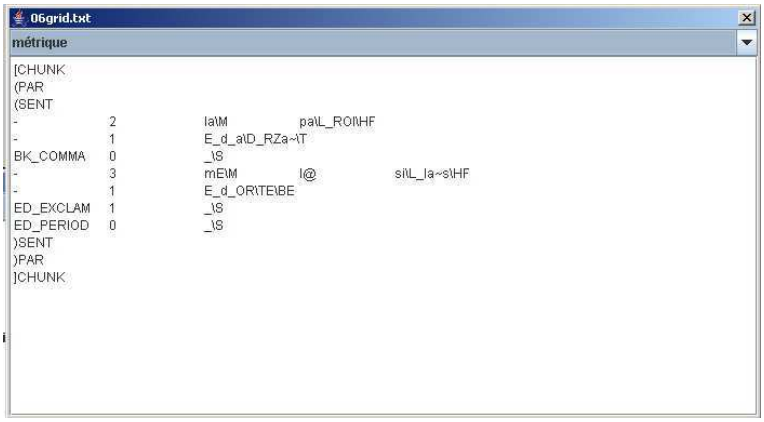


FIG. 88 – Details de la synthèse avec Syntaix : métrique

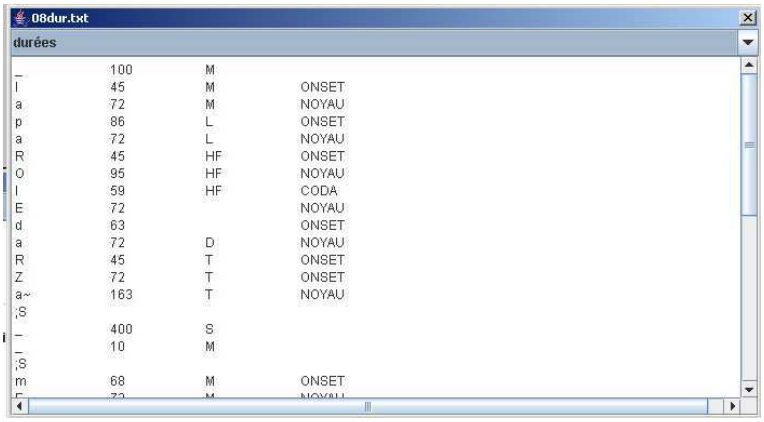


FIG. 89 – Details de la synthèse avec Syntaix : durées

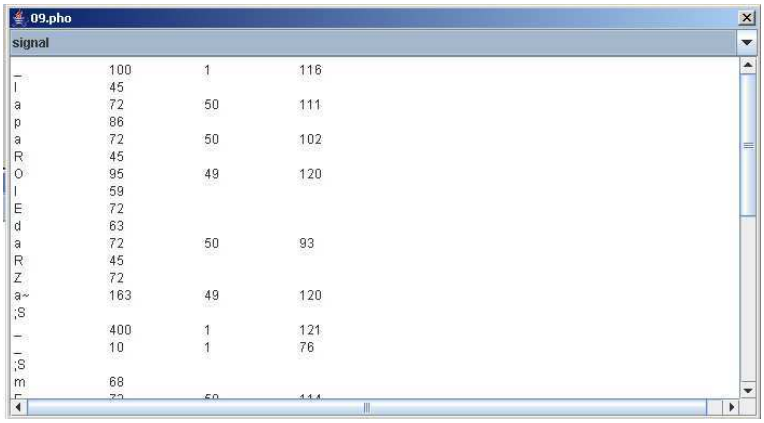


FIG. 90 – Details de la synthèse avec Syntaix : entrée du synthétiseur MBRola

Syntaix est très dépendant du synthétiseur MBRola auquel il transmet ses données finales pour obtenir la production du signal sonore. Cette dépendance ne peut être corrigée qu'au prix d'un développement complet de logiciel de synthèse par diphones. Nous pensons plus simplement permettre à la nouvelle plateforme Syntaix de transmettre ses données à différents systèmes existants de synthèse (par diphones, par formants ou enfin par mots).

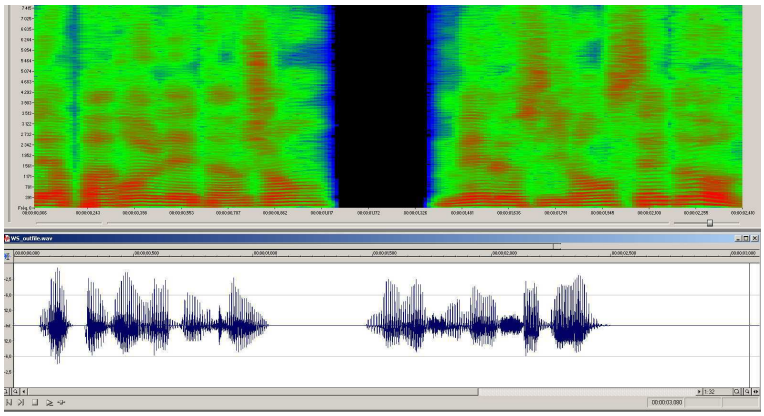


FIG. 91 – Details de la synthèse avec Syntaix : production du fichier audio

L'avancement de l'intégration des outils de LPLSuite au programme Syntaix n'en est qu'à ses débuts mais constitue un objectif à court terme pour valider la qualité des outils grâce à des expériences perceptives.

17.2 Plateforme de Communication Alternative : PCA

Parmi les outils développés au Laboratoire Parole et Langage, la plateforme de communication alternative (PCA) est au stade de la diffusion commerciale. On pourra consulter les articles [Bellengier, 2004] et [Blache *et al.*, 2005] pour plus d'informations sur cette plateforme.

L'application PCA répond à un grand nombre de besoins émis par le monde thérapeutique (médecins, orthophonistes, ergothérapeutes etc.) qui entoure le problème de la communication dans les handicaps lourds. Les personnes atteintes d'infirmité motrice quasi-totale, qu'il s'agisse de conséquences post-traumatiques ou encore de maladies neuro dégénératives telles que la sclérose amyotrophique latérale, en passant par les troubles liés à la perte partielle ou totale de la maitrise de certains phénomènes linguistiques (perte du sens ou de la compréhension, perte de la faculté de composer des messages orthographiés par exemple). Il existe un certain nombre de handicaps (temporaires ou définitifs) empêchant une communication orale ou écrite. C'est le cas par exemple des personnes paralysées et ne pouvant plus parler (pathologies neuro-dégénératives, accidents vasculaires cérébraux, IMC, etc.), mais également de certaines situations thérapeutiques comme des séjours prolongés en soins intensifs. Dans bien des cas, le patient garde sa capacité cognitive à communiquer mais ne peut la mettre en œuvre naturellement. Un certain nombre d'outils aidant la communication sont d'autre part proposés, voir notamment [Vaillant, 1997] et [Abraham, 2000]. Il s'agit d'ordinateurs intégrant dans leur forme de base un clavier virtuel qui, via un système de défilement, permet de contrôler un clavier à l'écran remplaçant le clavier physique. Cependant ces outils restent peu nombreux, peu paramétrables, et d'une efficacité encore limitée, aucun d'entre eux ne répondant à l'ensemble des besoins de la communication assistée. Le système PCA s'est fixé pour but la mise au point d'un procédé global d'aide à la communication permettant de composer des messages (verbaux ou non verbaux) à l'aide de capteurs en utilisant un ensemble de techniques de prédiction. Nous avons pour cela mis sur pied un groupe de travail composé de chercheurs en informatique et en linguistique, de psychologues, de médecins et rééducateurs, de centres d'accueil de handicapés ainsi que d'associations de handicapés. Ce groupe a permis début 2001 d'élaborer d'un cahier des charges stipulant les fonctionnalités à incorporer dans notre logiciel d'aide à la communication : la Plateforme de Communication Alternative (PCA). De 2001 à 2003, la PCA a été développée au sein du LPL, en partenariat avec le CNRS et le Ministère de la Recherche (cf. [Blache, 2003]). La version industrielle du logiciel PCA a été finalisée fin janvier 2004. Elle est distribuée par la société Aegys. Trois grands principes distinguent la PCA des autres logiciels d'aide à la communication : son homogénéité (réalisée particulièrement au niveau de l'interface du logiciel), sa généricité (calculée en terme de ressources linguistiques et para-linguistiques) et enfin son évolutivité (évaluée en fonction de la capacité du logiciel à s'adapter aux performances de l'utilisateur).

A la différence des autres systèmes, un travail particulier a de plus été conduit concernant le contrôle de l'environnement. Nous avons ainsi été amenés à mettre au point plusieurs capteurs ou contacteurs permettant de compléter de façon efficace l'existant. Nous proposons ainsi des solutions pour les personnes totalement paralysées, là où peu d'outils sont disponibles. Nous proposons également d'intégrer ces outils directement au sein de la plateforme, en particulier en offrant la possibilité d'une utilisation multiple. PCA permet ainsi une gestion de différentes modalités de contrôle simultanément.

La généricité des outils permet non seulement de s'adapter aux différentes configurations, mais offre de plus la possibilité d'une utilisation réversible du système. La même plateforme peut en effet être utilisée dans une perspective de remédiation, bien entendu, mais également à des fins de rééducation : il est par exemple possible d'utiliser des fonctionnalités conjointes de communication verbales et non verbales en tant que support dans des phases d'acquisition ou de réacquisition du langage.

Pour permettre à ses utilisateurs une communication assistée, PCA intègre déjà la plupart des outils et ressources de LPLSuite : le segmenteur, l'étiqueteur, un analyseur syntaxique

dédié, et un sous-lexique noyau du lexique DicoLPL. Deux modules (module orthographique et un module iconique présentés respectivement par les figures 92 et 93) emploient ces outils pour faciliter la tâche rédactionnelle des utilisateurs. En mode orthographique, une prédiction des mots susceptibles de terminer une phrase en cours d'écriture est disponible grâce à ces outils. Dans le mode iconique, chaque pictogramme est associé à un ensemble de formes lexicales qui permettent au logiciel de prédire à nouveau les mots/icones pouvant prolonger une expression en cours de saisie.

Le module de prédiction en mode orthographique dérive directement des outils *Dictionnaire* et *Etiqueteur* de LPLSuite présenté dans la seconde partie. Il s'agit en effet de prédire la suite d'un mot en cours de saisie autant que de prédire le mot le plus probable en fonction des mots déja saisis. Pour cela, le module de désambiguisation de l'étiqueteur est à même de proposer les catégories les plus probables à chaque étape de la saisie. Le dictionnaire fournit alors les mots appartenant aux catégories les plus probables et les classe par ordre de fréquence.

Le module de prédiction en mode iconique a fait l'objet d'une étude approfondie. Le programme qui gère cette prédiction est encore l'étiqueteur/désambiguiseur et il se fonde sur les informations stockées au préalable avec chaque icone (un mot et un ensemble de catégories morphosytaxiques). Un module de reformulation a été introduit pour obtenir des phrases syntaxiquement correctes à partir d'une succession d'icones. Nous ne le présentons pas ici car il est encore l'objet de nombreux développements.

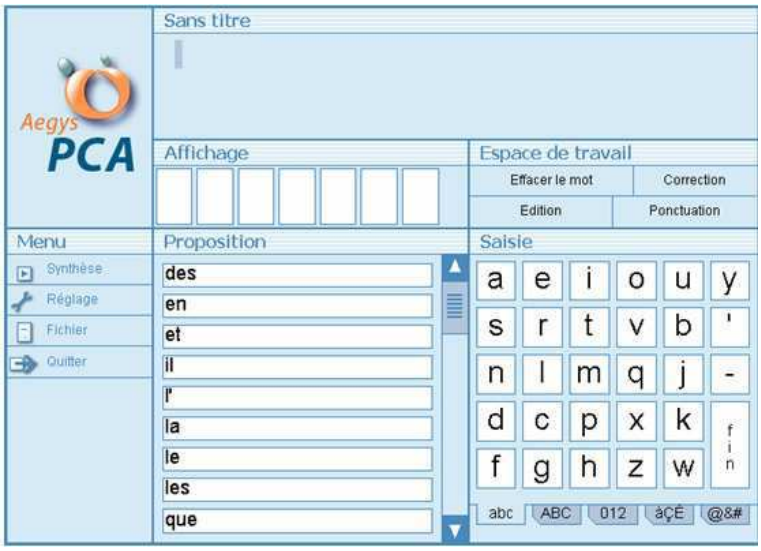


Fig. 92 – Application PCA : mode orthographique

Les possibilités offertes par des outils de bas niveau fiables sont innombrables. Nous attendons encore beaucoup d'améliorations, qui devraient permettre d'accroître encore les fonctionnalités des applications de haut niveau qui les emploient.

L'avancement du développement de l'application PCA permet de conclure à un gain en communication substanciel pour les utilisateurs. La charge cognitive liée à la saisie en environ-

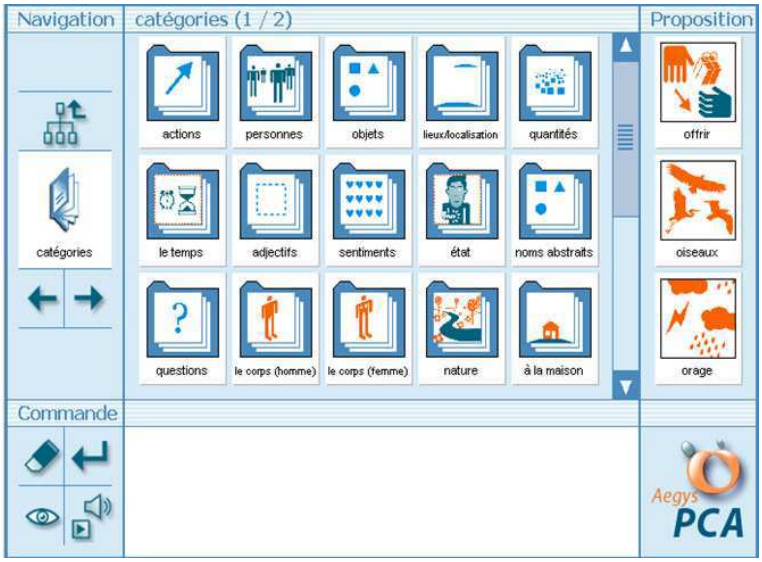


FIG. 93 – Application PCA : mode iconique

nement assisté a été au centre de chaque étape d'intégration. Il faut par exemple un maximum de trois interactions avec l'utilisateur pour saisir un mot, quelle que soit sa longueur.

17.3 Conclusion

Outre l'intérêt intrinsèque des applications que nous avons présentées, il est important de noter le fait que c'est en confrontant les outils de bas niveau avec la réalité d'un usage de haut niveau qu'il est possible de les améliorer en repérant directement leurs défauts et leurs lacunes.

Les applications de haut niveau présentent de fait une sensibilité directe à la lenteur ou à la charge mémoire dédiée aux modules qui la composent. L'application PCA, par exemple est extrèmement sensible à cette charge du fait de son embarquement dans de petits ordinateurs mobiles susceptibles d'accompagner les personnes handicapées dans leurs déplacements. Les répercussions dues à un contact direct avec des utilisateurs non informaticiens et non linguistes apporte un retour extrêmement utile quant à la réalité des besoins. Nous pouvons espérer de ce retour une plus grande efficacité dans l'amélioration des outils qui constituent LPLSuite.

L'analyse syntaxique, notamment, reste au coeur des réflexions. Tant du fait qu'elle participe au processus de prédiction dans PCA qu'au processus de découpage en unités et segments intonatifs dans Syntaix. La question de la granularité est constamment soulevée, comme celle de la robustesse. Les applications doivent fournir des informations fiables, aussi riches que possible face à des entrées de qualité quelconque.

Conclusion générale

Conclusion

Les différents travaux que nous avons présentés ici ont permis la mise en place d'une véritable plateforme de traitement automatique des langues naturelles. La modularité et l'évolutivité des outils et des ressources que nous avons développés permettent à des applications linguistiques d'accéder à une robustesse et une richesse d'informations tout à fait compétitives. Les résultats actuellement connus de la campagne d'évaluation EASY placent nos analysurs syntaxiques en troisième, quatrième et sixième places dans le groupe des quinze participants.

Outre ces observations pratiques, nous avons avant tout mis en place un cadre de développement pour des outils modulaires, robustes et adaptables autour du formalisme des Grammaires de Propriétés. La qualité d'une analyse syntaxique dépend pour une bonne part de celle des outils qui préparent l'entrée des analyseurs. Ces derniers ont été pensés, développés et évalués avec de nouvelles approches algorithmiques et de nouvelles structures de données permettant une analyse bottom-up à granularité variable en temps polynomial. L'analyseur *SeedParser* utilise pour cela une représentation des données linguistiques et grammaticales sous forme de graphes permettant un calcul de satisfaction des contraintes basé sur la mesure de *densité de satisfaction* que nous avons introduite.

Pour faciliter les développements, nous avons dû proposer une formalisation de la notion de propriétés (propre aux Grammaires de Propriétés) dans laquelle un ensemble restreint de caractéristiques logiques et mathématiques permet de définir une specification des types de contraites.

Parmi les éléments formels que nous avons introduits, la notion de *disponibilité* (*availability*), présentée en première partie, rend possible le calcul logique (satisfaction de contraintes) sur les contraintes complexes qui définissent les Grammaires de Propriétés.

Depuis la formalisation de notre problématique jusqu'au développement de l'ensemble d'outils LPLSuite, nous pensons avoir défendu l'idée d'une informatique respectueuse des problématiques linguistiques : quelle syntaxe, quels modèles permettent de représenter de façon suffisamment riche l'information linguistique contenue dans un texte à analyser ?

Avec l'ensemble de nos travaux, nous avons démontré la possibilité d'une analyse sensible à la qualité de l'entrée analysée. A travers le simple mot valise de *granularité*, nous avons envisagé et mis en évidence plusieurs caractéristiques importantes qui font parfois défaut aux analyseurs syntaxiques modernes. Même si de nombreux développements restent à prévoir, nous avons mis en place les bases pour ces recherches ultérieures.

La conclusion à laquelle nous aboutissons par rapport à la question posée initialement (une analyse à granularité variable est-elle possible ?) est positive. Nous avons pour cela introduit un système à trois niveaux (spécification sémantique des grammaires, grammaires et caractérisation). Des structures de données réflexives (capables de s'auto-interroger) ont été mises

en place pour permettre une constante remise en cause du processus d'analyse. La mesure de densité de satisfaction donne l'information nécessaire à cette remise en cause. Il est donc tout à fait possible de mettre en œuvre un programme de contrôle capable de décider une analyse approfondie là où l'énoncé analysé est peu correct, en mettant en place un seuil de tolérance à l'agrammaticalité révisable.

La notion de granularité nous a aussi amenés à envisager la possibilité de grammaires multimodales, ce qui a dirigé en partie notre stratégie visant à construire des structures de données réflexives. Un nœud donné du graphe de caractérisation peut être analysé simultanément au crible d'une grammaire de la syntaxe ou d'une grammaire prosodique ou encore d'une grammaire sémantique, pragmatique etc. Nous avons pour cela mis en place la notion de *grammaire colorée* et montré que la coloration permet l'analyse multimodale en une seule passe en interdisant des projections à partir de constituants de couleur différente. Au sein d'une même grammaire peuvent coexister plusieurs grammaires non contradictoires et non corrélées.

La granularité en tant que mot trop flou convient peu à un domaine aussi strict que celui de l'analyse syntaxique et du TAL en général. Cependant, son emploi est devenu si fréquent qu'une interrogation à son propos était devenue nécessaire. Nous pensons avoir répondu à ces interrogations en envisageant les possibilités communément entendues derrière cette appellation, puis en mettant en place la plateforme LPLSuite et l'analyseur SeedParser. Nous reconnaissons aussi n'avoir suivi qu'un ensemble assez réduit de pistes dans la recherche d'une analyse à granularité variable sensible au contexte. L'ouverture principale vers une réelle sensibilité au contexte se situe dans le calcul de densité de satisfaction, qui permet de repérer les difficultés locales. Des heuristiques peuvent être développées afin d'auto réguler le seuil de densité de satisfaction acceptable en fonction du contexte.

Parallèlement à cette question, nous avons été amenés à évaluer nos outils et à mettre en place des stratégies d'évaluation même en l'absence de corpus de référence. A l'aide de l'outil de multiplexage, il a été possible de comparer les différences entre diverses techniques aussi bien superficielles qu'approfondies. A travers des évaluations quantitatives, il a aussi été possible de vérifier la qualité de l'étiquetage, de la segmentation et de l'analyse syntaxique.

Le formalisme des Grammaires de Propriétés s'est avéré capable de supporter les développements et les questions que nous avons soulevées. Ceci est dû au fait que l'analyse y est considérée comme un processus de satisfaction de contraintes, que ces contraintes ne sont pas hiérarchisées entre elles et qu'il permet de représenter toutes sortes d'informations linguistiques (pas uniquement syntaxiques). Avec ce formalisme, nous avons pu mettre en place aussi bien des analyseurs superficiels que très profonds.

Pour atteindre ces objectifs, il a fallu construire un ensemble cohérent d'outils du TAL (segmenteur, étiqueteur, fréquenceur etc.) ainsi que des ressources riches et fiables (lexique et base de NGrammes). Ces outils ont intégré le coeur de deux applications (synthèse vocale et communication assistée) et permis de mettre en place une plateforme de développement de grammaires (Grammaires de Propriétés). Cette dernière (Accolade) est à la fois un outil d'édition assistée et de validation empirique semi-automatisée. La ressource lexicale DicoLPL, qui a été l'objet de beaucoup d'attentions, a rejoint le projet Morphalou[8] pour participer à la création d'un grand dictionnaire Français gratuit et destiné aux besoins du TALN.

En 1993, Jean-Marie Marandin ([Marandin, 1993]) exposait l'impossibilité épistémologique d'un parseur à devenir une instance de vérification pour la syntaxe. Nous pouvons dire à notre

[8]http ://loreley.loria.fr/morphalou/

tour que cette conclusion n'est plus totalement vraie, dès lors que l'analyseur offre suffisamment de réflexivité pour autoriser une vérification empirique des grammaires qu'il manipule. Nous lui donnons cependant raison dans le sens où les analyseurs ne permettent pas d'expérimentation transthéorique. A ce propos, le formalisme des Grammaires de Propriétés reste comparable à d'autres (utilisation de la précédence linéaire, de contraintes d'accord, d'une forme d'unification de traits, etc.), mais reste difficilement comparable aux autres formalismes par la simple observation des résultats d'analyse. Ceux-ci ne sont généralement pas comparables. Tout l'intérêt de la campagne EASY se situe justement dans cette tentative de comparer ces résultats en se donnant un format cible identique. Là encore, ce n'est pas seulement la qualité de l'analyseur qui est évaluée, mais celle de toute la chaîne de traitements qui précèdent et suivent l'analyse, y compris les heuristiques ad-hoc destinées au formatage vers le standard EASY.

Au delà des travaux que nous avons présentés ici, de nombreuses perspectives sont ouvertes pour des développements à court et moyen terme. L'interprétation des résultats d'évaluation permettra d'améliorer encore les outils. Leur intégration dans des applications de haut niveau permettra encore de les confronter à la réalité du traitement sur des données issues de situations réelles. Le développement de Grammaires de Propriétés pour le Français, mais aussi pour l'Anglais, est un travail en cours au Laboratoire Parole et Langage. Marie Laure Guénot s'intéresse particulièrement à cette question en abordant la problématique du point de vue de la théorie linguistique plutôt que du point de vue computationnel, ce qui a déjà grandement contribué à la tentative de programmation d'un analyseur générique (pour les Grammaires de Propriétés) capable de traiter avec des grammaires très différentes et des spécifications sémantiques réglables. Les types de contraintes telles que la contiguïté ont été introduites à partir de ses besoins. L'abstraction des structures de l'analyseur a aussi permis d'envisager un outil de développement de grammaires en environnement contrôlé (Accolade) manipulable par des personnes non-informaticiennes, mais linguistes. Jean-Marie Balfourier continue le développement de son analyseur (LPL1 dans la campagne d'évaluation EASY, qui se place actuellement en troisième position au classement entre les participants). Son travail le mène actuellement vers la construction de grammaires apprises automatiquement sur des corpus de référence. Le formalisme des Grammaires de Propriétés ne facilité pas cette tâche, car les grammaires ne sont pas dérivationnelles au sens classique. Un processus de subsomption, de factorisation et d'induction s'avère nécessaire, ce qui conduit à nouveau à mathématiser le formalisme. Jean-Philippe Prost travaille lui aussi à cette tâche en développant une technique d'analyse fondée sur une variante de la mesure de densité de satisfaction. Il travaille avec Christel Portes à la validation d'hypothèses grammaticales à l'aide d'expériences cognitives Emmanuel Bellengier développe pour sa part une grammaire sémantique du Français, ce qui l'amène à interroger le formalisme des GPs, à le développer dans une direction différente de la syntaxe. Les Grammaires de Propriétés offrent véritablement un formalisme linguistique général non générativiste, basé sur les contraintes et capable de rendre compte non seulement de la grammaticalité, mais aussi de l'acceptabilité d'une entrée.

Notre outil d'analyse à granularité variable (SeedParser) présente encore des défauts (Les résultats de la campagne EASY permettent déjà d'en cerner quelques uns), mais aussi des qualités exploitables et optimisables. Nous espérons en faire un outil accessible à la communauté afin de mieux partager notre approche et de promouvoir l'approche basée sur le formalisme des GPs.

Les questions de standardisation et de réutilisabilité ont été au coeur de nos développements. De nombreuses tentatives ont été nécessaires avant de fonder nos choix et notre

modélisation. Nous pouvons à présent compter sur cet ensemble d'outils mathématiques et informatiques pour améliorer et développer les applications sans revenir sur les choix préliminaires. En ceci, nous croyons avoir ébauché les bases d'une analyse syntaxique robuste et souple à granularité variable.

Annexe A

DTD des textes étiquetés

La DTD donné ici permet de représenter un texte étiqueté. Il s'agit de la version actuelle servant dans la suite LPLSuite.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- etiqueteur.dtd edited by Tristan Vanrullen  (LPL) -->


<!ELEMENT TEXTE_ETIQUETE (TEXTE,STATISTIQUES)>
<!ATTLIST TEXTE_ETIQUETE
    Dico CDATA #REQUIRED
    Desambiguiseur CDATA #REQUIRED
    Etiqueteur CDATA #REQUIRED>

<!ELEMENT TEXTE (ETIQUETTE*)>
<!ATTLIST TEXTE
    Fichier CDATA #REQUIRED>

<!ELEMENT ETIQUETTE (DESAMBIGUISE*,POSSIBILITE*)>
<!ATTLIST ETIQUETTE
    Type CDATA #REQUIRED
    Mot CDATA #REQUIRED
    Categorie CDATA #IMPLIED
    Categorie_potentielle CDATA #IMPLIED
    Validite CDATA #IMPLIED
    numEnonce CDATA #IMPLIED
    numForme CDATA #IMPLIED
    sur CDATA #IMPLIED>

<!ELEMENT DESAMBIGUISE EMPTY>
<!ATTLIST DESAMBIGUISE
    Index CDATA #REQUIRED
    Categorie CDATA #REQUIRED>

<!ELEMENT POSSIBILITE EMPTY>
<!ATTLIST POSSIBILITE
    Index CDATA #REQUIRED
    Categorie CDATA #REQUIRED>

<!ELEMENT STATISTIQUES (STAT*)>

<!ELEMENT STAT EMPTY>
<!ATTLIST STAT
    label CDATA #REQUIRED
    valeur CDATA #REQUIRED>
```

Annexe B

Grammaire BNF et DTD des Grammaires de Propriétés

B.1 Grammaire BNF

```
grammaire ::=
    {categorie}
categorie ::=
    nomCategorie,
    traits,
    proprietes
traits ::=
    {typeTrait}
typeTrait ::=
    nomTypeTrait
    {valeurTrait}
valeurTrait ::=
    valeur
proprietes ::=
    {typePropriete}
typePropriete ::=
    nomTypePropriete
    {propriete}
propriete ::=
    {membre}
membre ::=
    {refCateg | operationLogique}
refCateg ::=
    nom,
    {refValeur | refTrait}
operationLogique ::=
    (['NON'] (refCateg | operationLogique)) |
    (((refCateg | operationLogique)) ('OU'|'ET') (refCateg | operationLogique)))
refValeur ::=
    nomCategorie,
    operateur,
    valeur
refTrait ::=
    nomTypeTrait
```

B.2 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Grammaire.dtd edited by Tristan Vanrullen et Marie-Laure Guenot (LPL) -->
<!ELEMENT grammaire (categorie)*>
```

```
<!ATTLIST grammaire
    label CDATA #REQUIRED
    comment CDATA #IMPLIED
    type (LPLTRAITS | MLG) "LPLTRAITS">
<!ELEMENT categorie (traits, proprietes)>
<!ATTLIST categorie
    label ID #REQUIRED
    type (CONTINUE | DISCONTINUE) "CONTINUE"
    comment CDATA #IMPLIED>
<!ELEMENT traits (trait |propagation)*>
<!ATTLIST traits
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT trait (valeur*)>
<!ATTLIST trait
    label  CDATA #REQUIRED
    comment CDATA #IMPLIED
    type (CARACTERE | ENTIER) "CARACTERE">
<!ELEMENT propagation EMPTY>
<!ATTLIST propagation
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT valeur EMPTY>
<!ATTLIST valeur
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT proprietes (propriete)*>
<!ATTLIST proprietes
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT propriete (clause)*>
<!ATTLIST propriete
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT clause (membre)*>
<!ATTLIST clause
    comment CDATA #IMPLIED
    weight CDATA "1">
<!ELEMENT membre (refCateg | operationLogique)*>
<!ATTLIST membre
    comment CDATA #IMPLIED>
<!ELEMENT refCateg (refValeur | refTrait)*>
<!ATTLIST refCateg
    label IDREF #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT operationLogique (refCateg | operationLogique)+>
<!ATTLIST operationLogique
    label (OU|ET) "OU"
    comment CDATA #IMPLIED>
<!ELEMENT refValeur EMPTY>
<!ATTLIST refValeur
    label CDATA #REQUIRED
    operateur  CDATA #REQUIRED
    valeur CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT refTrait EMPTY>
<!ATTLIST refTrait
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
```

Annexe C

Grammaire BNF et DTD de la Spécification Sémantique des Grammaires de Propriétés

C.1 Grammaire BNF

```
Semantique ::= {definitionPropriete}
definitionPropriete ::=
        nom,
        symbole,
        arite,
        capaciteMin,
        capaciteMax,
        cardinaliteMin,
        cardinaliteMax,
        ordonnee,
        ConditionSatisfaisabilite,
        ConditionDisponibilite
ConditionSatisfaisabilite ::=
        expressionBooleenne | expressionLogique
ConditionDisponibilite ::=
        expressionBooleenne | expressionLogique
expressionLogique ::=
        facteurLogique | termeLogique | elementLogique
facteurLogique ::=
        (facteurLogique | termeLogique | elementLogique | expressionBooleenne ),
        'ET',
        (facteurLogique | termeLogique | elementLogique | expressionBooleenne)
termeLogique ::=
        (facteurLogique | termeLogique | elementLogique | expressionBooleenne),
        'OU',
        (facteurLogique | termeLogique | elementLogique | expressionBooleenne)
elementLogique ::=
        litteral |litteralNegative
litteralNegative ::=
        'NON',
        litteral
litteral ::=
        numeroTerme,
        ('DISPONIBLE' | 'SATISFAIT')
expressionBooleenne ::=
        expressionBooleenneBinaire | expressionBooleenneUnaire | constanteBooleenne
expressionBooleenneBinaire ::=
        expressionArithmetique ,
        ('INF' | 'SUP' | 'EQ' | 'DIF' | 'INFEQ' | 'SUPEQ')
```

```
        expressionArithmetique
expressionBooleenneUnaire ::=
        ('FONCTIONNEUTRE' | 'VERIFDOMAINES'),
        expressionArithmetique
constanteBooleenne ::=
        ('VRAI' | 'FAUX')
expressionArithmetique ::=
        (facteurArithmetique | termeArithmetique | elementArithmetique)
facteurArithmetique ::=
        (facteurArithmetique | termeArithmetique | elementArithmetique),
        ('MUL' | 'DIV'),
        (facteurArithmetique | termeArithmetique | elementArithmetique)
termeArithmetique ::=
        (facteurArithmetique | termeArithmetique | elementArithmetique),
        ('PLUS' | 'MOINS')
        (facteurArithmetique | termeArithmetique | elementArithmetique)
elementArithmetique ::=
        (terminal | terminalMoins | constanteArithmetique)
terminalMoins ::=
        'MOINS',
        terminal
terminal ::=
    numeroMembre,
    ('RANGDEBUT' | 'RANGFIN' | 'VALEUR' | 'DOMAINE')
constanteArithmetique ::=
    valeur
```

C.2 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- semantique.dtd edited by Tristan Vanrullen (LPL) -->
<!ELEMENT semantique (definitionPropriete)*>
<!ATTLIST semantique
    label CDATA #REQUIRED
    comment CDATA #IMPLIED>
<!ELEMENT definitionPropriete (conditionSatisfaisabilite, conditionDisponibilite)>
<!ATTLIST definitionPropriete
    nom CDATA #REQUIRED
    symbole CDATA #REQUIRED
    comment CDATA #IMPLIED
    arite CDATA #REQUIRED
    capacite CDATA #REQUIRED
    cardinalite CDATA #REQUIRED
    ordonnee (VRAI | FAUX) "FAUX">
<!ELEMENT conditionSatisfaisabilite (expressionBooleenne | expressionLogique)>
<!ATTLIST conditionSatisfaisabilite
    comment CDATA #IMPLIED>
<!ELEMENT conditionDisponibilite (expressionBooleenne | expressionLogique)>
<!ATTLIST conditionDisponibilite
    comment CDATA #IMPLIED>
<!ELEMENT expressionLogique (facteurLogique|termeLogique|elementLogique)>
<!ELEMENT facteurLogique ((facteurLogique|termeLogique|elementLogique|expressionBooleenne),
                          (facteurLogique|termeLogique|elementLogique|expressionBooleenne))>
<!ATTLIST facteurLogique
    operateur (ET) "ET">
<!ELEMENT termeLogique ((facteurLogique|termeLogique|elementLogique|expressionBooleenne),
                        (facteurLogique|termeLogique|elementLogique|expressionBooleenne))>
<!ATTLIST termeLogique
    operateur (OU) "OU">
<!ELEMENT elementLogique (litteral| litteralNegative)>
<!ELEMENT litteralNegative (litteral)>
<!ATTLIST litteralNegative
    operateur (NON) "NON">
<!ELEMENT litteral EMPTY>
<!ATTLIST litteral
```

```
    numeroMembre CDATA #REQUIRED
    accesseur (DISPONIBLE| SATISFAIT) "DISPONIBLE">
<!-- expressions mathematiques : -->
<!-- les variables doivent faire partie du contexte des clauses de propriete -->
<!-- c'est-a-dire le rang des elements disponibles ou/et leur valeur  ou/et leur domaine-->
<!ELEMENT expressionBooleenne (expressionBooleenneBinaire |
   expressionBooleenneUnaire |constanteBooleenne)>
<!ELEMENT expressionBooleenneBinaire (expressionArithmetique,expressionArithmetique)>
<!ATTLIST expressionBooleenneBinaire
    operateur (INF | SUP | EQ | DIF |INFEQ | SUPEQ) "EQ">
<!ELEMENT expressionBooleenneUnaire (expressionArithmetique)>
<!ATTLIST expressionBooleenneUnaire
    operateur (FONCTIONNEUTRE | VERIFDOMAINES) "FONCTIONNEUTRE ">
<!ELEMENT constanteBooleenne EMPTY>
<!ATTLIST constanteBooleenne
    valeur (VRAI | FAUX) "VRAI">
<!ELEMENT expressionArithmetique (facteurArithmetique|termeArithmetique|elementArithmetique)>
<!ELEMENT facteurArithmetique ((facteurArithmetique|termeArithmetique|elementArithmetique),
                               (facteurArithmetique|termeArithmetique|elementArithmetique))>
<!ATTLIST facteurArithmetique
    operateur (MUL | DIV) "MUL">
<!ELEMENT termeArithmetique ((facteurArithmetique|termeArithmetique|elementArithmetique),
                             (facteurArithmetique|termeArithmetique|elementArithmetique))>
<!ATTLIST termeArithmetique
    operateur (PLUS | MOINS) "PLUS">
<!ELEMENT elementArithmetique (terminal | terminalMoins | constanteArithmetique)>
<!ELEMENT terminalMoins (terminal)>
<!ATTLIST terminalMoins
    operateur (MOINS) "MOINS">
<!ELEMENT terminal EMPTY>
<!ATTLIST terminal
    numeroMembre CDATA #REQUIRED
    accesseur (RANGDEBUT | RANGFIN | VALEUR | DOMAINE) "VALEUR">
<!ELEMENT constanteArithmetique EMPTY>
<!ATTLIST constanteArithmetique
    valeur CDATA #REQUIRED>
```

Annexe D

Une représentation XML de spécification sémantique des Grammaires de Propriétés

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE semantique SYSTEM "semantique.dtd">
<?xml-stylesheet type="text/xsl" href="semantique.xsl"?>
<semantique label="Semantique des Grammaires de propriétés" comment="faite le 18-03-2003 par T. VanRullen">
    <!-- ********************************************************************************************** -->
    <definitionPropriete
            nom="linearite"
            comment="linéarité: précédence entre items définie par une relation d'ordre entre les
                rangs des termes d'une clause"
            symbole="(("
            arite="2"
            capacite="[0;infini]"
            cardinalite="[0;infini]">
        <conditionDisponibilite comment="les deux termes doivent être disponibles">
            <expressionLogique>
                <facteurLogique>
                    <elementLogique>
                        <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                    </elementLogique>
                    <elementLogique>
                        <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
                    </elementLogique>
                </facteurLogique>
            </expressionLogique>
        </conditionDisponibilite>
        <conditionSatisfaisabilite  comment="le rang du premier terme doit être strictement inférieur
                au rang du second terme">
            <expressionBooleenne>
                <expressionBooleenneBinaire operateur="INF">
                    <expressionArithmetique>
                        <elementArithmetique>
                            <terminal numeroMembre="1" accesseur="RANGFIN"/>
                        </elementArithmetique>
                    </expressionArithmetique>
                    <expressionArithmetique>
                        <elementArithmetique>
                            <terminal numeroMembre="2" accesseur="RANGDEBUT"/>
                        </elementArithmetique>
                    </expressionArithmetique>
                </expressionBooleenneBinaire>
            </expressionBooleenne>
        </conditionSatisfaisabilite >
```

```
</definitionPropriete>

<!-- ********************************************************************************************** -->
<definitionPropriete
        nom="obligation"
        comment="têtes potentielles de la catégorie"
        symbole="+"
        arite="1"
        capacite="[1;infini]"
        cardinalite="[1;infini]"
        ordonnee="VRAI">
    <conditionDisponibilite comment="le terme  doit être disponible pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles
            sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="interdiction"
        comment="elements ne devant pas figurer dans la categorie"
        symbole="!"
        arite="1"
        capacite="[0;0]"
        cardinalite="[0;infini]"
        ordonnee="FAUX">
    <conditionDisponibilite comment="il faut T1 ou pas T1 pour rendre la clause disponible">
        <expressionLogique>
            <termeLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteralNegative>
                        <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                    </litteralNegative>
                </elementLogique>
            </termeLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont violées sans condition T1 est disponible">
        <expressionLogique>
            <elementLogique>
                <litteralNegative>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </litteralNegative>
            </elementLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
<!-- ********************************************************************************************** -->
<definitionPropriete
        nom="facultativite"
        comment="elements facultatifs du syntagme"
        symbole="?"
        arite="1"
        capacite="[0;infini]"
        cardinalite="[0;infini]"
```

```
        ordonnee="FAUX">
    <conditionDisponibilite comment="le terme unique doit être disponible pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="exigence"
        comment="le terme 1 exige le terme 2"
        symbole="=)"
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
        <conditionDisponibilite comment="les deux termes doivent être disponibles">
            <expressionLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
            </expressionLogique>
        </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="exclusion"
        comment="le terme 1 exclut le terme 2"
        symbole=")("
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
    <conditionDisponibilite comment="Disponible si les deux membres le sont">
        <expressionLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="le premier terme doit être satisfait mais pas le second">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteralNegative>
                        <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
```

```
                    </litteralNegative>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
    <definitionPropriete
        nom="dependance"
        comment="accord (dépendance) entre deux termes"
        symbole="~~"
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
    <conditionDisponibilite comment="les deux termes doivent être disponibles">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites si les domaines de ses termes sont contrôlés">
        <expressionLogique>
            <facteurLogique>
                <expressionBooleenne>
                    <expressionBooleenneUnaire operateur="VERIFDOMAINE">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="1" accesseur="DOMAINE"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneUnaire>
                </expressionBooleenne>
                <expressionBooleenne>
                    <expressionBooleenneUnaire operateur="VERIFDOMAINE">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="2" accesseur="DOMAINE"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneUnaire>
                </expressionBooleenne>
            </facteurLogique>
        </expressionLogique>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="unicite"
        comment="catégories uniques dans un syntagme"
        symbole="1"
        arite="1"
        capacite="[0;1]"
        cardinalite="[0;infini]"
        ordonnee="FAUX">
    <conditionDisponibilite comment="le terme unique doit être disponible pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
```

```
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="uniciteExclusive"
        comment="catégories uniques mutuellement exclusives dans un syntagme"
        symbole="1!"
        arite="1"
        capacite="[1;1]"
        cardinalite="[0;1]"
        ordonnee="FAUX">
    <conditionDisponibilite comment="le terme unique doit être disponible pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="contiguite"
        comment="précédence immédiate entre items définie par une relation d'ordre entre les rangs des termes
            d'une clause"
        symbole="//"
        arite="2"
        capacite="[0;infini]"
        cardinalite="[0;infini]">
    <conditionDisponibilite comment="les deux termes doivent être disponibles">
        <expressionLogique>
            <facteurLogique>
                <elementLogique>
                    <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
                </elementLogique>
                <elementLogique>
                    <litteral numeroMembre="2" accesseur="DISPONIBLE"/>
                </elementLogique>
            </facteurLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite  comment="le rang du premier terme doit être égal au rang du second terme moins 1">
        <expressionLogique>
            <termeLogique>
                <expressionBooleenne>
                    <expressionBooleenneBinaire operateur="EQ">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="1" accesseur="RANGDEBUT"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                        <expressionArithmetique>
                            <termeArithmetique operateur="PLUS">
                                <elementArithmetique>
                                    <terminal numeroMembre="2" accesseur="RANGFIN"/>
                                </elementArithmetique>
                                <elementArithmetique>
```

```
                                    <constanteArithmetique valeur="1"/>
                                </elementArithmetique>
                            </termeArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneBinaire>
                </expressionBooleenne>
                <expressionBooleenne>
                    <expressionBooleenneBinaire operateur="EQ">
                        <expressionArithmetique>
                            <elementArithmetique>
                                <terminal numeroMembre="1" accesseur="RANGFIN"/>
                            </elementArithmetique>
                        </expressionArithmetique>
                        <expressionArithmetique>
                            <termeArithmetique operateur="MOINS">
                                <elementArithmetique>
                                    <terminal numeroMembre="2" accesseur="RANGDEBUT"/>
                                </elementArithmetique>
                                <elementArithmetique>
                                    <constanteArithmetique valeur="1"/>
                                </elementArithmetique>
                            </termeArithmetique>
                        </expressionArithmetique>
                    </expressionBooleenneBinaire>
                </expressionBooleenne>
            </termeLogique>
        </expressionLogique>
    </conditionSatisfaisabilite >
</definitionPropriete>

    <!-- ********************************************************************************************** -->
<definitionPropriete
        nom="constituance"
        comment="catégories devant figurer au moins et au plus une fois dans le syntagme"
        symbole="."
        arite="1"
        capacite="[1;1]"
        cardinalite="[0;infini]"
        ordonnee="FAUX">
    <conditionDisponibilite comment="le terme unique doit être disponible pour que la clause le soit">
        <expressionLogique>
            <elementLogique>
                <litteral numeroMembre="1" accesseur="DISPONIBLE"/>
            </elementLogique>
        </expressionLogique>
    </conditionDisponibilite>
    <conditionSatisfaisabilite comment="Les clauses sont satisfaites sans condition si elles sont disponibles">
        <expressionBooleenne>
            <constanteBooleenne valeur="VRAI"/>
        </expressionBooleenne>
    </conditionSatisfaisabilite>
</definitionPropriete>
</semantique>
```

Annexe E

Une représentation XML de grammaire de propriétés

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE grammaire SYSTEM "grammaire.dtd">
<?xml-stylesheet type="text/xsl" href="grammaire.xsl"?>
<grammaire label="grammaire" type="LPLTRAITS" comment="grammaire exemple 1" >
    <!-- ********************************************************************************************** -->
    <categorie label="D" comment="determinant">
        <traits label="D_traits" comment="traits du déterminant">
            <trait label="soucat" comment="pos2">
                <valeur label="a" comment="article?"/>
                <valeur label="d" comment="démonstratif"/>
                <valeur label="i" comment="indéfini"/>
                <valeur label="s" comment="possessif"/>
                <valeur label="t" comment="interrogatif"/>
            </trait>
            <trait label="ordre" comment="pos3">
                <valeur label="1" comment="première personne"/>
                <valeur label="2" comment="deuxième personne"/>
                <valeur label="3" comment="troisième personne"/>
            </trait>
            <trait label="genre" comment="pos4">
                <valeur label="m" comment="masculin"/>
                <valeur label="f" comment="féminin"/>
            </trait>
            <trait label="nombre" comment="pos5">
                <valeur label="s" comment="singulier"/>
                <valeur label="p" comment="pluriel"/>
            </trait>
            <trait label="possesseur" comment="pos7'">
                <valeur label="s" comment="singulier"/>
                <valeur label="p" comment="pluriel"/>
            </trait>
            <trait label="resultatFusion" comment="pos8">
                <valeur label="d" comment="défini"/>
                <valeur label="i" comment="indéfini"/>
            </trait>
            <trait label="fusionAvecQuoi" comment="pos9">
                <valeur label="a" comment="à + le"/>
                <valeur label="d" comment="de + le"/>
            </trait>
        </traits>
        <proprietes label="_props" comment="pas de propriétés">
        </proprietes>
    </categorie>
    <!-- ********************************************************************************************** -->
    <categorie label="N" comment="Nom">
```

```
    <traits label="N_traits" comment="traits du nom">
        <trait label="soucat" comment="pos2">
            <valeur label="c" comment="commun"/>
            <valeur label="d" comment="propre avec déterminant"/>
            <valeur label="p" comment="propre sans déterminant"/>
            <valeur label="l" comment="latin?"/>
        </trait>
        <trait label="genre" comment="pos3">
            <valeur label="m" comment="masculin"/>
            <valeur label="f" comment="féminin"/>
        </trait>
        <trait label="nombre" comment="pos4">
            <valeur label="s" comment="singulier"/>
            <valeur label="p" comment="pluriel"/>
        </trait>
        <trait label="sigle" comment="pos5">
            <valeur label="s" comment="sigle (abréviation, etc.)"/>
        </trait>
        <trait label="typeNomPropre" comment="pos6">
            <valeur label="c" comment="pays"/>
            <valeur label="h" comment="habitants"/>
            <valeur label="s" comment="société"/>
        </trait>
    </traits>
    <proprietes label="_props" comment="pas de propriétés">
    </proprietes>
</categorie>
<!-- ********************************************************************************************** -->
<categorie label="SN">
    <traits label="SN_traits" comment="traits du syntagme nominal">
        <trait label="soucat">
            <valeur label="c" comment="commun"/>
            <valeur label="d" comment="propre avec déterminant"/>
            <valeur label="p" comment="propre sans déterminant"/>
            <valeur label="l" comment="latin"/>
        </trait>
        <trait label="genre">
            <valeur label="m" comment="masculin"/>
            <valeur label="f" comment="féminin"/>
        </trait>
        <trait label="nombre">
            <valeur label="s" comment="singulier"/>
            <valeur label="p" comment="pluriel"/>
        </trait>
        <trait label="sigle">
            <valeur label="s" comment="sigle (abréviation, etc.)"/>
        </trait>
        <trait label="typeNomPropre">
            <valeur label="c" comment="pays"/>
            <valeur label="h" comment="habitants"/>
            <valeur label="s" comment="société"/>
        </trait>
        <propagation label="soucat"/>
        <propagation label="genre"/>
        <propagation label="nombre"/>
        <propagation label="sigle"/>
        <propagation label="typeNomPropre"/>
    </traits>
    <proprietes label="_props" comment="propriétés">
        <propriete label="obligation">
            <clause>
                <membre>
                    <refCateg label="N" />
                </membre>
            </clause>
        </propriete>
```

```
<propriete label="facultativite">
    <clause>
        <membre>
            <refCateg label="D"/>
        </membre>
    </clause>
</propriete>

<propriete label="unicite">
    <clause>
        <membre>
            <refCateg label="D"/>
        </membre>
    </clause>
</propriete>

<propriete label="exclusion">
    <clause>
        <membre>
            <refCateg label="N">
                <refValeur label="soucat" operateur="=" valeur="p"/>
            </refCateg>
        </membre>
        <membre>
            <refCateg label="D"/>
        </membre>
    </clause>
</propriete>

<propriete label="linearite">
    <clause>
        <membre>
            <refCateg label="D"/>
        </membre>
        <membre>
            <refCateg label="N" />
        </membre>
    </clause>
</propriete>

<propriete label="exigence">
    <clause>
        <membre>
            <refCateg label="N">
                <refValeur label="soucat" operateur="=" valeur="c"/>
            </refCateg>
        </membre>
        <membre>
            <refCateg label="D"/>
        </membre>
    </clause>
</propriete>

<propriete label="dependance">
    <clause>
        <membre>
            <refCateg label="D">
                <refTrait label="genre"/>
                <refTrait label="nombre"/>
            </refCateg>
        </membre>
        <membre>
            <refCateg label="N">
                <refTrait label="genre" />
                <refTrait label="nombre" />
```

```
                        </refCateg>
                    </membre>
                </clause>
            </propriete>
        </proprietes>
    </categorie>
</grammaire>
```

Annexe F

Format des entrées du lexique

Nous regroupons ici les tables qui présentent le format de codage des entrées lexicales dans DicoLPL. Cette annexe précise la mise en forme de ces données exposée dans le chapitre 5.

Catégorie	Schéma de traits	position	type de trait	valeurs possibles
Adjectif	A - - - - -	1	sous catégorie	f qualificatif i indéfini o ordinal s possessif
		2	type	c comparatif p positif
		3	genre	m masculin f féminin
		4	nombre	s singulier p pluriel
		5	non attribué	
Conjonction	C -	1	sous catégorie	c coordination s subordination
Déterminant	D - - - - - - - -	1	sous catégorie	a article d démonstratif i indéfini s possessif t interrogatif
		2	ordre	1 première pers. 2 deuxième pers. 3 troisième pers.
		3	genre	m masculin f féminin
		4	nombre	s singulier p pluriel
		6	possesseur	s singulier p pluriel
		7	resultatFusion	d défini i indéfini
		8	fusionAvecQuoi	a à + le d de + le

Suite page suivante

Catégorie	Schéma de traits	position	type de trait	valeurs possibles
Numéral	M - - -	1	sous catégorie	c cardinal o ordinal
		2	genre	m masculin f féminin
		3	nombre	s singulier p pluriel
Nom	N - - - - -	1	sous catégorie	c commun d propre avec dét. p propre sans dét. l latin
		2	genre	m masculin f féminin
		3	nombre	s singulier p pluriel
		4	sigle	s sigle (abrév., etc.)
		5	typeNomPropre	c pays h habitants s société
Pronom	P - - - - - -	1	sous catégorie	d démonstratif i indéfini p personnel r relatif s possessif t interrogatif x réfléchi
		2	ordre	1 première pers. 2 deuxième pers. 3 troisième pers.
		3	genre	m masculin f féminin
		4	nombre	s singulier p pluriel
		5	typePronom	n nominatif o oblique (coi) j objet (cod)
		6	possesseur	s singulier p pluriel
Adverbe	R - -	1	sous catégorie	g normal p negation
		2	type	c comparatif p normal d PAS n NE
Préposition	S - -	1	sous catégorie	p préposition
		2	type	a à d de
Verbe	V - - - - - -	1	sous catégorie	m principal o modal a auxiliaire avoir e auxiliaire être

Suite page suivante

Catégorie	Schéma de traits	position	type de trait	valeurs possibles
		2	mode	n infinitif i indicatif m impératif c conditionnel s subjonctif p participe
		3	temps	p présent s passé i imparfait f futur
		4	ordre	1 première pers. 2 deuxième pers. 3 troisième pers.
		5	genre	m masculin f féminin
		6	nombre	s singulier p pluriel
Interjection	I			

Tab. 1: (Annexe) Valeurs de traits du dictionnaire

Symbole	Situation	Exemple
0	Majuscule facultative sur le premier caractère sauf en début de phrase	nous
1	Majuscule obligatoire sur le premier caractère	Paul
2	Majuscule obligatoire pour tous les caractères du mot	NASA

Tab. 2 – (Annexe) Valeurs de l'octet MAJ

Symbole	Durée	Type	Exemple	Symbole	Durée	Type	Exemple
_	100	SILENCE		t	90	PLOSIVE	temps
i	70	VOYELLE	si	d	70	PLOSIVE	dans
e	80	VOYELLE	ses	k	80	PLOSIVE	quand
E	80	VOYELLE	seize	g	55	PLOSIVE	gant
a	80	VOYELLE	patte	f	90	FRICATIVE	femme
A	80	VOYELLE	pâte	v	80	FRICATIVE	vent
O	75	VOYELLE	comme	s	85	FRICATIVE	sans
o	80	VOYELLE	gros	z	85	FRICATIVE	zone
u	85	VOYELLE	doux	S	85	FRICATIVE	champ
y	75	VOYELLE	du	Z	80	FRICATIVE	gens
2	90	VOYELLE	deux	j	40	VOYELLE	ion
9	60	VOYELLE	neuf	m	75	CONSONNE	mont
@	75	VOYELLE	justement	n	60	CONSONNE	nom
e~	95	NASALE	vin	J	80	CONSONNE	oignon
a~	90	NASALE	vent	N	70	CONSONNE	camping
o~	90	NASALE	bon	l	50	CONSONNE	long
9~	90	NASALE	brun	R	50	CONSONNE	rond
p	95	PLOSIVE	pont	w	65	VOYELLE	voiture
b	75	PLOSIVE	bon	H	40	VOYELLE	huile

Tab. 3 – (Annexe) Alphabet phonétique SAMPA

Bibliographie

[Abdennadher *et al.*, 1996] S. Abdennadher, T. Frühwirth, and H. Meuss. On confluence of constraint handling rules. *Lecture Notes in Computer Science*, 1118, 1996.

[Abdennadher, 1998] S. Abdennadher. *Analyse von regelbasierten Constraintlösern.* PhD thesis, Ludwig-Maximilians-Universität München, 1998.

[Abeillé and Blache, 1999] A. Abeillé and P. Blache. *Grammaires et analyseurs syntaxiques*, pages 51–76. Traité IC2, volume Ingénierie des Langues. Hermès, 1999.

[Abeillé and Candito, 2000] A. Abeillé and M.H. Candito. FTAG : a lexicalized tree adjoining grammar for french. in Abeillé and Rambow (ed), Tree Adjoining Grammars : formalism, linguistic analysis and processing, Stanford :CSLI, pp305-330, 2000.

[Abeillé and Rambow, 2000] A. Abeillé and O. Rambow. Tree adjoining grammars : formalism, linguistic analysis and processing. Stanford :CSLI, 2000.

[Abeillé *et al.*, 2003] A. Abeillé, L. Clément, and F. Toussenel. Building a treebank for french. in Abeillé (ed), Treebanks : building and using parsed corpora, Dordrecht :Kluwer, pp165-185, 2003.

[Abeillé, 1991] A. Abeillé. Une grammaire électronique du français. Paris : Editions du CNRS, 1991.

[Abeillé, 1993] A. Abeillé. Les nouvelles syntaxes : grammaires d'unification et analyse du français. Armand Colin, Paris, 1993.

[Abeillé, 1995] A. Abeillé. The flexibility of french idioms : a representation with LTAG. in M. Everaert, E. Van der Linden, A. Schenk, Schreuder (eds) Idioms : structural and psychological perspectives, LEA : New York. p. 15-42, 1995.

[Abeillé, 2003] A. Abeillé. Treebanks : building and using parsed corpora. Dordrecht :Kluwer, 2003.

[Abeillé *et al.*, 1994] Abeillé, Cavazza, and Rastier. *Sémantique pour l'analyse.* Sciences cognitives. Masson, 1994.

[Abeillé, 2002] Anne Abeillé. *Une grammaire électronique du français.* CNRS Editions — coll. Sciences du langage, Paris, 2002.

[Abney, 1987] Steven Abney. *The English noun phrase in its sentential aspect.* Phd thesis, MIT, Cambridge, 1987.

[Abney, 1991] S. Abney. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-based parsing*, pages 257–278. Kluwer Academic Publishers, Dordrecht, 1991.

[Abney, 1996] S. Abney. Partial parsing via finite-state calculus. 1996.

[Abney, 1997] S. Abney. Part of speech tagging and partial parsing. In S. Young and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*, pages 118–136. Kluwer Academic Publishers, Dordrecht, 1997.

[Abraham, 2000] M. Abraham. Reconstruction de phrases oralisées à partir d'une écriture pictographique. in actes de Handicap 2000, 2000.

[Aho and Ullman, 1972] A. Aho and J. Ullman. The theory of parsing, translation and compiling. Prentice-Hall, 1972.

[Allen *et al.*, 1979] J. Allen, S. Hunnincutt, R. Carlson, and B. Granström. MITalk-79 : The 1979 MIT text-to-speech system. In Wolf and Klatt, editors, *Speech Communications Papers Presented at the 97th Meeting of the ASA*, pages 507–510, 1979.

[Allen *et al.*, 1987] J. Allen, S. Hunnincutt, and D. Klatt. From text to speech : the mitalk system. *Cambridge University Press*, 1987.

[Alonso *et al.*, 2001] Miguel A. Alonso, Eric Villemonte de la Clergerie, and Manuel Vilares. A formal definition of bottom-up embedded push-down automata and their tabulation technique. *Lecture Notes in Computer Science*, 2099 :44–, 2001.

[Appelt *et al.*, 1993] D. Appelt, J. Hobbs, J. Bear, D. Israel, and M. Tyson. FASTUS : a finite state processor for information extraction from real-world texts. IJCAI, Chambéry, 1993.

[Appelt, 1987] D. Appelt. Bidirectional grammars and the design of natural language generation systems. 3rd Theoretical issues in NLP, Las Cruces, p. 185-191, 1987.

[Archangeli and Langendoen, 1997] D. Archangeli and D.T. Langendoen. *Optimality Theory.* 1997.

[Association des Bibliophiles Universels, 2000] Association des Bibliophiles Universels. ABU. dictionnaire des mots communs. in La Bibliothèque Universelle (http ://abu.cnam.fr/DICO/mots-communs.html) CNAM, 2000.

[Aït-Mokhtar and Chanod, 1997] S. Aït-Mokhtar and J.P. Chanod. Incremental finite-state parsing. 5e ANLP, Washington, p. 72-79, 1997.

[Austin, 1962 1970] J. Austin. *How to do things with words (Quand dire c'est faire : traduction française).* Oxford University Press (Seuil), 1962 (1970).

[Ayache, 1995] L. Ayache. *Analyse et Synthèse de phrases simples à noms d'unités de mesure.* Mémoire de DEA, Faculté des sciences de LUMINY, 1995.

[Baccus, 2002] N. Baccus. *Grammaire française.* Flammarion — coll. Librio, Paris, 2002.

[Balfourier *et al.*, 2002] J-M. Balfourier, P. Blache, and T. VanRullen. From shallow to deep parsing using constraint satisfaction. pages 36–42. COLING'2002, 2002.

[Balfourier *et al.*, 2005] J-M. Balfourier, P. Blache, M.-L. Guénot, and T. VanRullen. analyseurs syntaxiques et grammaires du LPL pour la campagne dévaluation EASY. TALN-2005, 2005.

[Bar-Hillel, 1964] Y. Bar-Hillel. *Language and information.* Addison Wesley, Reading (Mass.), 1964.

[Barton *et al.*, 1987] G. Barton, R. Berwick, and E. Ristad. Computational complexity and natural language. MIT Press, 1987.

[Basset and Perennec,] L. Basset and M. Perennec. *Les classes de mots.* Traditions et perspectives. Presses Universitaires de Lyon.

[Becker, 1988] T. Becker. An efficient kernel for multilingual generation in speech to speech dialogue translation. Proceedings ACL-COLING, p. 110-116, 1988.

[Bellengier, 2004] E. Bellengier. PCA : Un système de communication alternative évolutif et réversible. in actes des ISAAC-04, 2004.

[Bender *et al.*, 2002] E. Bender, D. Flickinger, and S. Oepen. The grammar matrix : An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. in Proceedings of COLING-02, (Workshop on Grammar Engineering and Evaluation), 2002.

[Benveniste, 1966] E. Benveniste. *Problèmes de linguistique générale — Tome 1.* Gallimard, Paris, 1966.

[Berwick *et al.*, 1991] R. Berwick, S. Abney, and T. Tenny. Principle-based parsing. In P. Sells (et al.), editor, *Foundational issues in natural language processing*, pages 115–226. MIT Press, Cambridge (Mass.), 1991.

[Bès and Blache, 1999] G. Bès and P. Blache. Propriétés et analyse d'un langage. TALN'99, 1999.

[Blache and Balfourier, 2001] P. Blache and J.-M. Balfourier. Property grammars : a flexible constraint-based approach to parsing. in proceedings of IWPT-2001, 2001.

[Blache and Caelen, 1997] P. Blache and G. Caelen. Prosodie et syntaxe. T.A.L, 38 :1, 1997.

[Blache and Guénot, 2002] P. Blache and M.-L. Guénot. Flexible corpus annotation with property grammars. workshop on *Treebanks and Linguistic Theories*, 2002.

[Blache and Hirst, 2001] P. Blache and D. Hirst. Aligning prosody and syntax in property grammars. in *proceedings of EuroSpeech 2001*, 2001.

[Blache and Morawietz, 2001] P. Blache and F. Morawietz. *Travaux Interdisciplinaires du Laboratoire Parole et Langage d'Aix-en-Provence*, volume 19, chapter A Non-Generative Constraint-Based Formalism, pages 11–26. Issn 1621-0360 edition, 2001.

[Blache and Morin, 1990] P. Blache and J.-Y. Morin. Bottom-up filtering : a parsing strategy for GPSG. in proceedings of COLING'90, 1990.

[Blache and Paquelin, 1996] P. Blache and J.-L. Paquelin. Active constraints for a direct interpretation of HPSG. HPSG'96, 1996.

[Blache and Prost, 2005] P. Blache and J.P. Prost. Gradience, constructions and constraint systems. in Constraint Solving and Language Processing, H. Christiansen et al. (eds), LNAI 3438, Springer, 2005.

[Blache and VanRullen, 2002] P. Blache and T. VanRullen. An evaluation of different symbolic shallow parsing techniques,. in *proceedings of LREC-02*, 2002.

[Blache *et al.*, 2003] P. Blache, M.L. Guénot, and T. Vanrullen. Corpus-based grammar development. pages 124–131. in *proceedings of Corpus Linguistics 2003*,, 2003.

[Blache *et al.*, 2005] P. Blache, S. Rauzy, and N. Richardet. Système multimodal de communication alternative. in actes du colloque Société de l'Information, 2005.

[Blache, 1999a] P. Blache. Filtering and fusion : A technique for parsing with properties. NLPRS'99, 1999.

[Blache, 1999b] P. Blache. Shared trees : Representing a parse forest with a single tree. VEXTAL-99, 1999.

[Blache, 2000a] P. Blache. Constraints, linguistic theories and natural language processing. *Lecture Notes in Artificial Intelligence*, 1835, 2000.

[Blache, 2000b] P. Blache. Property grammars and the problem of constraint satisfaction. pages 91–99. ESSLLI-2000 Workshop on Linguistic Theory and Grammar Implementation, 2000.

[Blache, 2001] P. Blache. *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles*. Hermès Sciences Publications, Paris, 2001.

[Blache, 2003] P. Blache. Vers une théorie cognitive de la langue basée sur les contraintes. in *actes de TALN-2003*, 2003.

[Blache, 2005] P Blache. Property grammars : A fully constraint-based theory". in Constraint Solving and Language Processing, H. Christiansen et al. (eds), LNAI 3438, Springer, 2005.

[Black *et al.*, 1991] E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, and B. Santorini Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of english grammars. Actes ARPA Workshop, IBM, New York, p. 306-311, 1991.

[Black *et al.*, 1993] E. Black, R. Garside, and G. Leech. Statistically-driven computer grammars for english : the IBM-lancaster approach. Amsterdam : Rodopi, 1993.

[Blanche-Benveniste *et al.*, 1991] C. Blanche-Benveniste, C. Rouget, and K. Van den Eynde. Le français parlé - etudes grammaticales. Paris : Editions du CNRS, 1991.

[Blanche-Benveniste, 1997] C. Blanche-Benveniste. Approches de la langue parlée en français. Gap : Ophrys, 1997.

[Bloomfield, 1933] Leonard Bloomfield. *Language*. Holt, New York, 1933. (première édition : 1914).

[Bod, 1998] R. Bod. Beyond grammar : an experiment based theory of language. CSLI Publications, 1998.

[Borsley, 1996] R. Borsley. *Modern Phrase Structure Grammars*. 1996.

[Boullier *et al.*, 2005] P. Boullier, B. Sagot, and L. Clément. Un analyseur LFG efficace : SxLFG. in actes de TALN-05, 2005.

[Bouma *et al.*,] G. Bouma, R. Malouf, and I. Sag. *Natural Language and Linguistic Theory*, volume 19 :1, chapter Satisfying Constraints on Extraction and Adjunction. Kluwer.

[Bourigault and Fabre, 2000] D. Bourigault and C. Fabre. Approche linguistique pour l'analyse syntaxique de corpus. in Cahiers de Grammaire 25, 2000.

[Bresnan and Kaplan, 1982] Joan Bresnan and Ronald Kaplan. *The mental representation of grammatical relations*. MIT Press, Cambridge, 1982.

[Bresnan, 2001] J. Bresnan. Lexical functional syntax. Blackwell, 2001.

[Brill, 1992] E. Brill. A simple rule-based part of speech tagger. Conférence ANLP, Trento, Italie, p. 151-155, 1992.

[Briscoe and Carroll, 2002] E. Briscoe and J. Carroll. Robust accurate statistical annotation of general text. in Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-02), 2002.

[Butt *et al.*, 1999] M. Butt, T. King, M. Nino, and F. Segond. *A Grammar Writer's Cookbook*. CSLI Publications, 1999.

[Candito, 1999] M.-H. Candito. Organisation modulaire et paramétrable de grammaires électroniques lexicalisées. application au français et à l'italien. Thèse de doctorat, Université Paris 7, 1999.

[Carpenter and Penn, 1995] B. Carpenter and G. Penn. *Current Issues in Parsing Technologies*, chapter Compiling Typed Attribute-Value Logic Grammars. Kluwer, h. bunt and m. tomita edition, 1995.

[Carpenter, 1991] R. Carpenter. The generative power of categorial grammars and HPSG with lexical rules. in Computational Linguistics, 17 :3, p. 301-314, 1991.

[Carpenter, 1992] R. Carpenter. The logic of typed feature structures. Cambridge University Press, 1992.

[Carroll and Weir, 1997] J. Carroll and D. Weir. Encoding frequency information in lexicalized grammars. in IWPT'97, 1997.

[Carroll *et al.*, 2003] J. Carroll, G. Minnen, and T. Briscoe. Parser evaluation using a grammatical relation annotation scheme. in Abeillé (ed) Treebank, 2003.

[Chandrasekar *et al.*, 1996] M. Chandrasekar, C. Doran, and B. Srinivas. Motivations and methods for text simplification. 15ème COLING, Copenhague, 1996.

[Chanod and Aït-Mokhtar, 1996] J.P. Chanod and S. Aït-Mokhtar. Incremental finite state parsing. Technical report, Rank Xerox research center, 1996.

[Chanod and Tapanainen, 1995] J.-P. Chanod and P. Tapanainen. Tagging french : comparing a statistical and a constraint-based method. 7ème Conférence European ACL, Dublin, p. 149-156, 1995.

[Chanod and Tapanainen, 1996] J.P. Chanod and P. Tapanainen. A non deterministic tokenizer for finite state parsing. 1996.

[Chanod and Tapanainen, 1997] J.P. Chanod and P. Tapanainen. A robust finite state parser. Technical report, Xerox Research Center, 1997.

[Chanod, 1993] J.-P. Chanod. Problèmes de robustesse en analyse syntaxique. 2ème Conférence ILN, Nantes, p. 223-244, 1993.

[Chanod, 2000] J.-P. Chanod. *Robustness in Language Technology*, chapter Robust Parsing and Beyond. Kluwer, 2000.

[Chen and Vijay-Shanker, 1997] J. Chen and K. Vijay-Shanker. Towards a reduced commitment, d-theory style TAG parser. in proceedings of the International Workshop on Parsing Technologies, 1997.

[Chomsky, 1957] Noam Chomsky. *Syntactic Structures.* Mouton, la Hague, 1957.

[Chomsky, 1965] Noam Chomsky. *Aspects of the theory of syntax.* MIT Press, Cambridge, 1965.

[Chomsky, 1968] Noam Chomsky. *le langage et la pensée.* Paris. Seuil, 1968.

[Chomsky, 1981] Noam Chomsky. *Lectures on Government and Binding.* Dordrecht : Foris, 1981.

[Chomsky, 1995] Noam Chomsky. *The Minimalist Program.* MIT Press, 1995.

[Church, 1988] K. Church. A stochastic parts program and noun phrase parser for unrestricted text. 2de ANLP Conference, Austin, p. 136-143, 1988.

[Clément *et al.*, 2004] L. Clément, B. Sagot, and B. Lang. Morphology-based automatic acquisition. in proceedings of LREC-04, 2004.

[Clément, 2002] Lionel Clément. Construction et exploitation d'un corpus syntaxiquement annoté pour le français. Thèse, Université de Paris 7, 2002.

[Coch, 1996] José Coch. Overview of alethgen. In *Proceedings of 8th International Workshop on Natural Language Generation — INLG'96*, volume 2, pages 25–28. Herstmonceux, 1996.

[Colmerauer, 1975] A. Colmerauer. Les grammaires de métamorphose. rapport interne, Département d'informatique, Université d'Aix-Marseille II. in [L. Bolc (ed) (1978) Natural language communication with computers, Springer Verlag], 1975.

[Copestake and Flickinger, 2000] A. Copestake and D. Flickinger. An open source grammar development environment and broad coverage english grammar using HPSG. LREC Proceedings, 2000.

[Copestake, 2002] A. Copestake. Implementing typed feature structure grammars. CSLI, 2002.

[Crouch *et al.*, 2004] R. Crouch, T. Holloway King, J. T. Maxwell III, S. Riezler, and A. Zaenen. Exploiting f-structure input for sentence condensation. Proceedings LFG Conference, CSLI online publications, 2004.

[Crysmann and al., 2002] B. Crysmann and al. An integrated architecture for shallow and deep processing. in proceedings of ACL-02, 2002.

[Daille, 1995] B. Daille. Repérage et extraction de terminologie par une méthode mixte statistique et linguistique. in TAL, 36 :1-2, p. 101-118, 1995.

[Danlos, 2000] L. Danlos. G-TAG : a lexicalizd formalism for text generation inspired by TAG. in Abeillé and Rambow (eds), Tree adjoining grammars, CSLI, p. 343-370, 2000.

[De Saussure, 1916] F. De Saussure. *Cours de linguistique générale*. Payot, Paris, 1916. réédition de 1990.

[De Saussure, 1960] F. De Saussure. *Cours de linguistique générale*. Payot, 1960.

[Descles, 1990] J.-P. Descles. Langages applicatifs, langues naturelles et cognition. Paris : Hermès, 1990.

[Di Cristo *et al.*, 2000] A. Di Cristo, P. Di Cristo, E. Campione, and J. Veronis. A prosodic model for text to speech synthesis in french. rapport interne, 2000.

[Di Cristo, 1998] P. Di Cristo. *Génération automatique de la prosodie pour la synthèse à partir du texte*. PhD thesis, Laboratoire Parole et Langage, Aix-en-Provence, 1998.

[Doran and Srinivas, 2000] C. Doran and B. Srinivas. Developing a wide coverage CCG system. in Abeillé, Rambow (eds), Tree Adjoining Grammars, CSLI, Stanford 405-426, 2000.

[Doran *et al.*, 1994] C. Doran, D. Egedi, B. Hockey, B. Srinivas, and M. Zaidel. The XTAG system : a wide coverage grammar for english. 15e COLING, Kyoto, p. 922-928, 1994.

[Doran *et al.*, 2000] C. Doran, B.-A. Hockey, A. Sarkar, B. Srinivas, and F. Xia. Evolution of the XTAG system. in Abeillé, Rambow (eds), Tree Adjoining Grammars, CSLI, Stanford 371-404, 2000.

[Doran, 1998] C. Doran. Incorporating punctuation into the sentence grammar : a LTAG perspective. Thèse de PhD, University of Pennsylvania, Philadelphia, 1998.

[Duchier and Debusmann, 2001] D. Duchier and R. Debusmann. Topological dependency trees : A constraint-based account of linear precedence. ACL'01, 2001.

[Duchier and Thater, 1999] D. Duchier and S. Thater. Parsing with tree descriptions : a constraint based approach. NLULP'99, 1999.

[Duchier *et al.*, 2005] D. Duchier, J. Leroux, and Y. Parmentier. XMG un compilateur de métagrammaires extensible. Proceedings TALN, Dorudan, 2005.

[Duchier, 2000] D. Duchier. Constraint programming for natural language processing. Lecture Notes, ESSLLI 2000, 2000.

[Earley, 1970] J. Earley. An efficient context-free parsing algorithm. Communication of the ACM, 13 :2, p. 94-102. reprint in B. Grosz, K. Sparck Jones, B. Webber (eds) (1986) Readings in Natural Language Processing, Morgan Kaufman, p. 25-34, 1970.

[Eccles, 1995] J. Eccles. *Evolution du cerveau et apparition de la conscience*. 1995.

[Emele and Zajac, 1990] M. Emele and R. Zajac. Typed unification grammars. 13e COLING, Helsinki, p. 293-298, 1990.

[Estival and Lehmann, 1997] D. Estival and S. Lehmann. TSNLP : des jeux de phrases test pour le TALN. TAL, 38 :1, p. 155-172, 1997.

[Estival, 1993] D. Estival. Une grammaire pour l'analyse et la génération. TAL, 34 :1, p. 83-100, 1993.

[Flaux and de Velde, 2000] Nelly Flaux and Danièle Van de Velde. *Les noms en français : esquisse de classement*. Ophrys — coll. L'essentiel français, Paris, 2000.

[Flickinger, 1996] D. Flickinger. Time expressions in an HPSG grammar. in T. Gunji (ed) Studies on the Universality of Constraint-based phrase structure grammar, Osaka University, p. 1-8, 1996.

[Foth *et al.*, 2005] K. Foth, M. Daum, and W. Menzel. A broad coverage parser for german based on defeasible constraints. in H. Christiansen, P. R. Skadhauge, and J. Villadsen, (eds), Constraint Solving and Language Processing, LNAI 3438, Springer-Verlag, 2005.

[Frank *et al.*, 2003a] A. Frank, M. Becker, B. Crysmann, B. Kieffer, and U. Schafer. Integrated shallow and deep parsing : TopP meets HPSG. in proceedings of ACL-03, 2003.

[Frank *et al.*, 2003b] A. Frank, L. Sadler, J. Van Genabith, and A. Way. From treebank resources to LFG f-structures. in Abeillé (ed) Treebanks, Kluwer, 367-390, 2003.

[Frazier and Fodor, 1978] L. Frazier and J. Fodor. The sausage machine : a new two-stage parsing model. Cognition, 6, p. 291-325, 1978.

[Frühwirth, 1998] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37 :1–3, 1998.

[Fuchs and Goffic, 1992] C. Fuchs and P. Le Goffic. *Les linguistiques contemporaines — Repères théoriques*. Hachette Université — coll. Langue, Linguistique, Communication, Paris, 1992. (première édition 1975).

[Fuchs *et al.*, 1993] C. Fuchs, L. Danlos, A. Lacheret-Dujour, D. Luzzati, and B. Victorri. *Linguistique et traitements automatiques des langues*. Hachette Université — coll. Langue, Linguistique, Communication, Paris, 1993.

[Gaiffe *et al.*, 2002] B. Gaiffe, B. Crabbé, and A. Roussanaly. A new metagrammar compiler. *Proceedings of TAG+6*, 2002.

[Gaifman, 1961] H. Gaifman. Dependency systems and phrase-structure systems. Technical Report RM-2315, Rand Corporation, 1961.

[Gaifman, 1965] H. Gaifman. Dependency systems and phrase-structure systems. *Information and control*, 18 :304–337, 1965. (initialement rapport technique de 1961).

[Gardent and Kow, 2005] C. Gardent and E. Kow. Generating and selecting grammatical paraphrases. Proceedings 10th European Workshop on Natural Language Generation, 2005.

[Gardes-Tamine, 1988] J. Gardes-Tamine. *La grammaire — Tome 2 : Syntaxe*. Armand Colin, Paris, 1988.

[Gaussier and Lange, 1995] E. Gaussier and J-M. Lange. Modèles statistiques pour l'extraction de lexiques bilingues. TAL, 36 :1-2, p. 133-156, 1995.

[Gazdar *et al.*, 1985] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Blackwell, Oxford, 1985.

[Gibson, 2000] E. Gibson. The dependency locality theory : A distance-based theory of linguistic complexity. In Image, Language, Brain, A. Marantz, Y. Miyashita and W. O'Neil (eds), MIT Press, 2000.

[Giguet, 1997] E. Giguet. Méthode pour l'analyse automatique de structures formelles sur documents multilingues. Thèse, Université de Caen, 1997.

[Goffic, 2000] P. Le Goffic. *Grammaire de la phrase française*. Hachette Supérieur — coll. Langue française, Paris, troisème edition, 2000. (première édition : 1993).

[Götz *et al.*, 1997] T. Götz, D. Meurers, and D. Gerdemann. The controll manual. Technical report, SFS Report, 1997.

[Grefenstette, 1996] G. Grefenstette. Light parsing as finite state filtering. 1996.

[Grinberg *et al.*, 1995] D. Grinberg, J. Lafferty, and D. Sleator. A robust parsing algorithm for link grammars. Technical report, CMU-CS-95-125, Carnegie Mellon University, 1995.

[Gross, 1975] M. Gross. Méthodes en syntaxe. Paris : Hermann, 1975.

[Gross, 1986] M. Gross. *Grammaire transformationnelle du français — tome 2 : Syntaxe du nom*. Cantilène — coll. Systématique de la langue française, Paris, 1986.

[Götz and Meurers, 1997] T. Götz and D. Meurers. The controll system as large grammar development platform. Proceedings of the Workshop Computational Environments for Grammar Development and Linguistic Engineering (ENVGRAM), 1997.

[Guenthner, 1988] F. Guenthner. Features and values. Technical report, Universität München, 1988.

[Harris, 1951] Z. Harris. *Structural linguistics*. Chicago University Press, Chicagik, 1951.

[Harris, 1968] Z. Harris. *Mathematical structures of English*. Wiley, New York, 1968. (trad. fr. Structures mathématiques du langage, Dunod, Paris).

[Harris, 1976] Z. Harris. *Notes du cours de syntaxe*. Seuil, Paris, 1976.

[Harris, 1982] Z. Harris. *A grammar of English on mathematical principles*. Wiley, New York, 1982.

[Harris, 1991] Zellig Harris. *A theory of language and information — A mathmatical approach*. Clarendon, Oxford, 1991.

[Hays, 1960] D. Hays. Grouping and dependency theories. Technical Report RM-2646, Rand Corporation, 1960.

[Hays, 1964] D. Hays. Dependency theory : a formalism and some observations. *Language*, 40(4) :511–525, 1964.

[Hindle, 1983] D. Hindle. Deterministic parsing of syntactic non fluencies. 21e ACL Meeting, 1983.

[Hindle, 1989] D. Hindle. Acquiring disambiguation rules from text. 27e ACL Meeting, Vancouver, 1989.

[Hockett, 1958] C.F. Hockett. *A course in modern linguistics*. Macmillan, New York, 1958.

[Hopcroft and Ullman, 1979] J. Hopcroft and J. Ullman. Introduction to automata theory, languages and computation. Addison-Wesley, 1979.

[Intelligent Systems Laboratory, 1995] Intelligent Systems Laboratory. *SICStus Prolog User's Manual*. Technical report, SICS, 1995.

[Jackendoff, 1977] R. Jackendoff. *X-bar syntax — A study of phrase structure*. MIT Press, 1977.

[Jacobs, 1990] P. Jacobs. To parse or not to parse : relation-driven text skimming. 13e COLING, Helsinki, vol. 2, p. 194-198, 1990.

[Jacquemin and Tzoukermann, 1999] C. Jacquemin and E. Tzoukermann. NLP for term variant extraction : Synergy between morphology lexicon and syntax. in Natural Language Information retrieval, T. Strzalkowski (ed), Kluwer, 1999.

[Jakobson, 1963] R. Jakobson. *Essais de Linguistique générale*. Paris. Editions de minuit, 1963.

[Johnson and Lappin, 1999] D. Johnson and S. Lappin. Local constraints vs. economy. Technical report, CSLI, 1999.

[Johnson, 2002] M. Johnson. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. in proceedings of ACL-02, 2002.

[Joshi and Srinivas, 1994] A. Joshi and B. Srinivas. Disambiguation of super parts of speech (or supertags) : almost parsing. 16th COLING, Kyoto, p. 154-160, 1994.

[Joshi *et al.*, 1991] A. Joshi, D. Weir, and K. Vijay-shanker. The convergence of mildly context-sensitive formalisms. in P. Sells, S. Shieber, T. Wasow (eds), Foundational issues in NLP, MIT Press, 1991.

[Joshi, 1987] Aravind Joshi. Introduction to tree adjoining grammar. In Manaster Ramer, editor, *The mathematics of language*, pages ,87–114. Benjamins, Amsterdam, 1987.

[Järvinen, 1994] T. Järvinen. Bank of english and beyond. in Abeillé (ed) Treebanks, Kluwer, 43-60, 1994.

[Kager, 1999] R. Kager. *Optimality Theory*. Cambridge University Press, 1999.

[Kaplan *et al.*, 2002] R. Kaplan, T. King, and J. Maxwell. Adapting existing grammars : the XLE experience. Workshop on Grammar Engineering and Evaluation (COLING-02), 2002.

[Karlsson *et al.*, 1995] F. Karlsson, A. Voutilainen, J. Aikkilä, and A. Antilla. Constraint grammar : a language independent system for parsing unrestricted texts. Berlin : Mouton de Gruyter, 1995.

[Karlsson, 1990] F. Karlsson. Constraint grammar as a framework for parsing running text. COLING'90, 1990.

[Kasami, 1965] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Mass, 1965.

[Kasper *et al.*, 1995] R. Kasper, K. Netter, and K. Vijay-shanker. Compiling a HPSG grammar into LTAG. 33e ACL Meeting, Santa-Cruz, 1995.

[Kay *et al.*, 1994] M. Kay, J.-M. Gawron, and P. Norvig. Verbmobil : a translation system for face- to-face dialog. CSLI, Chicago Press, 1994.

[Kay, 1980] M. Kay. Algorithm and data structure in syntactic processing. Actes Nobel Symposium on Text Processing, Gotheborg,. reprint in B. Grosz, K. Sparck Jones, B. Webber (eds) (1986) Readings in Natural Language Processing, Morgan Kaufman, p. 35-70, 1980.

[Kay, 1989] M. Kay. Head-driven parsing. International Workshop on Parsing Technologies, Pittsburg, 1989.

[Kermes and Evert, 2003] H. Kermes and S. Evert. Text analysis meet corpus linguistics. pages 402–411. Corpus Linguistics 2003, 2003.

[Kinyon and Prolo, 2002] A. Kinyon and C. Prolo. A classification of grammar development strategies. Workshop on Grammar Engineering and Evaluation (COLING-02), 2002.

[Kinyon, 1999] A. Kinyon. Hiérarchisation d'analyses basées sur des informations dépendancielles dans le cadre des LTAG. de TALN'99, 1999.

[Kittredge and Lehrberger, 1982] R. Kittredge and J. Lehrberger. Sublanguages. structures of language in restricted semantic domains. Walter de Gruyter, 1982.

[Kittredge and Polguère, 1991] R. Kittredge and A. Polguère. Dependency grammars for bilingual text generation. in International Conference on Current Issues in Computational Linguistics, Penang, p. 318-330, 1991.

[Klavans and Resnik, 1996] J. Klavans and P. Resnik. The balancing act - combining symbolic and statistical approaches to language. MIT Press, 1996.

[Kornai, 1997] A. Kornai. Finite state systems. Journal of language engineering, 2 :4, 1997.

[Kübler and Hinrichs,] S. Kübler and E. Hinrichs. From chunks to function-argument structure : A similarity-based approach. ACL-01.

[Kuno and Oettinger, 1962] S. Kuno and A. Oettinger. A multiple path syntactic analyser. Information processing, 1962.

[Laezlinger and Wehrli, 1991] C. Laezlinger and E. Wehrli. Un analyseur interactif pour le français. TA Informations, 32 :2, p. 35-50, 1991.

[Lange and Gaussier, 1995] J.-M. Lange and E. Gaussier. Alignement de corpus multilingues au niveau des phrases. TAL, 36 :1-2, p. 67-80, 1995.

[Laporte, 1997] E. Laporte. Les mots. un demi-siècle de traitements. TAL, 38 :2, 1997.

[Leclère and Subirats-Rüggeberg, 1991] C. Leclère and C. Subirats-Rüggeberg. A bibliography of studies on lexicon-grammar. *Linguisticae Investigationes*, XV(2) :347–409, 1991.

[Lecomte and Paroubek, 1996] J. Lecomte and P. Paroubek. Le catégoriseur d'e. brill : mise en œuvre de la version entraînée pour le français. Rapport Interne, INALF : Nancy, 1996.

[Lecomte, 1996] A. Lecomte. Grammaires and théorie de la preuve. TAL, 37 :2, 1996.

[Le Guern, 1972] M. Le Guern. *Sémantique de la métaphore et de la métonymie*. Larousse, Paris, 1972.

[Le Pesant,] D. Le Pesant. Syntaxe de certaines phrases à noms d'unités de mesure. Rapport de recherches 20, Laboratoire de Linguistique Informatique URA CNRS 1576.

[Liberman and Church, 1992] M. Liberman and K. Church. text analysis and word pronunciation in text-to- speech synthesis. In S. Furui and M.M. Sondhi, editors, *Advances in Speech Signal Processing*, pages 791–831. Dekker, New York, 1992.

[Lin, 2003] D. Lin. Dependency based evaluation using minipar. in Abeillé (ed) Treebanks, Kluwer, 317-330, 2003.

[Loper and Bird, 2002] E. Loper and S. Bird. NLTK : The natural language toolkit. in Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL, 2002.

[Ludwig, 1997] P. Ludwig. *Le Langage*, volume 3027 of *Corpus*. Flammarion, 1997.

[Lytinen and Gershman, 1986] S. Lytinen and A. Gershman. ATRANS automatic processing of money transfer messages. 5e National Conference on artificial intelligence, Philadelphia, 1986.

[Magerman, 1995] M. Magerman. Statistical decision-tree models for parsing. 23e ACL Meeting, 1995.

[Mani and Maybury, 1999] I. Mani and M. Maybury. Advances in automatic text summarization. MIT Press, 1999.

[Manning and Carpenter, 1997] C. Manning and B. Carpenter. Probabilistic left corner grammars. in IWPT'97, 1997.

[Marandin, 1993] J.-M. Marandin. Analyseurs syntaxiques, équivoques et problèmes. in T.A.L., vol. 1, pp. 5-33, 1993.

[Marcus *et al.*, 1993] M. Marcus, B. Santorini, and M.-A. Marcinkiewicz. Building a large annotated corpus of english : the penn treebank. Computational Linguistics, vol. 19, n2, 313- 330, 1993.

[Marcus, 1980] M. Marcus. A theory of syntactic recognition for natural language. MIT Press, 1980.

[Mariani and Veronis, 1999] J. Mariani and J. Veronis. Evaluation en TAL. Numéro spécial de la revue Langues, 1999.

[Marriott and Stuckey, 1998] K. Marriott and P.J. Stuckey. *Programming with Constraints*. MIT Press, 1998.

[Martinet, 1967] A. Martinet. *Eléments de linguistique générale*. Armand Colin, Paris, 1967.

[Martinet, 1979] A. Martinet. *Grammaire fonctionnelle du français*. Didier, Paris, 1979.

[Martinet, 1985] A. Martinet. *Syntaxe générale*. Armand Colin, Paris, 1985.

[Martinet, 1987] A. Martinet. *Le Langage*. La Pléiade. Gallimard, 1987.

[Maruyama, 1990] H. Maruyama. Structural disambiguation with constraint propagation. ACL'90, 1990.

[Maurel, 1991] D. Maurel. Préanalyse des adverbes de date en français. TA Informations, 32 :2, p. 5-18, 1991.

[Maxwell and Kaplan, 1991] J. Maxwell and R. Kaplan. A method for disfunctive constraint satisfaction. in M. Tomita (ed.) Current Issues in Parsing Technologies, Kluwer, 1991.

[Mckeown, 1986] K. Mckeown. Text generation using discourse strategies and focus constraints to generate natural language text. Cambridge University Press, 1986.

[Mel'Čuk, 1988] I. Mel'Čuk. Dependency syntax : theory and practice. SUNY Press, Albany, 1988.

[Meurers and Minnen, 1998] D. Meurers and G. Minnen. *Lexical and Constructional Aspects of Linguistic Explanation*, chapter Off-line Constraint Propagation for Efficient HPSG Processing. CSLI, g. webelhuth, j.-p. koenig, a. kathol (eds.) edition, 1998.

[Müller, 1999] S. Müller. Deutsche syntax deklarativ. head-driven phrase structure grammar für das deutsche. Linguistische Arbeiten, No. 394, Tübingen : Max Niemeyer Verlag, 1999.

[Montague, 1974] R. Montague. English as a formal language. In R. H. Thomason, editor, *Formal philosophy : selected papers of Richard Montague*. Yale University Press, 1974.

[Moortgat, 1989] M. Moortgat. *Categorial investigations : logical and linguistic aspects of the Lambec calculus*. Foris, Dordrecht, 1989.

[Morawietz and Cornell, 1997] F. Morawietz and T. Cornell. Representing constraints with automata. ACL/EACL, 1997.

[Nagel *et al.*, 1989] Ernest Nagel, James R. Newman, Kurt Gödel, and Jean-Yves Girardé. *Le théorème de Gödel*. Sciences. Seuil, 1989.

[Namer and Schmidt, 1993] F. Namer and P. Schmidt. Une grammaire du français dans un formalisme à structures de traits typées. 2e Conference ILN, Nantes, p. 105-128, 1993.

[Oepen *et al.*, 2000] S. Oepen, U. Callmeier, A. Copestake, D. Flickinger, and R. Malouf. Scalable grammar software for computational linguists. Demonstration,Hong Kong, *ACL 2000.*, 2000. (see http ://www.cs.ust.hk/acl2000/Demo/22_oepen.pdf).

[Paroubek *et al.*, 1992] P. Paroubek, Y. Schabes, and A. Joshi. XTAG : a graphical workbench for developing TAGs. 4° Conf. on Applied NLP, Trento, p. 223-227, 1992.

[Paroubek *et al.*, 2005] P. Paroubek, L.-G. Pouillot, I. Robba, and A. Vilnat. EASY : Campagne d'évaluation des analyseurs syntaxiques. in actes de TALN-05, 2005.

[Pavia, 2003] N. Gala Pavia. *Un modèle d'analyseur syntaxique robuste fondé sur la modularité et la lexicalisation de ses grammaires*. Thèse de doctorat en informatique, Université de Paris-Sud, 2003.

[Pereira and Schabes, 1992] F. Pereira and Y. Schabes. Inside-outside reestimation from partially bracketed corpora. 30e ACL meeting, 1992.

[Pereira and Shieber, 1987] F. Pereira and S. Shieber. Prolog and natural language analysis. CSLI n°10, Stanford, 1987.

[Pereira and Warren, 1980] F. Pereira and D. Warren. Definite clause grammars for natural language analysis : a survey of the formalism and a comparison with ATN. Artificial Intelligence, 13, p. 231-278, 1980.

[Pereira and Warren, 1983] F. Pereira and D. Warren. Parsing as deduction. 21e ACL Meeting Cambridge, p. 137-144, 1983.

[Pesant,] Denis Le Pesant. LEXIQUE grandeurs et mesures. Liste 40, Laboratoire de Linguistique Informatique URA CNRS 1576.

[Pesant and Colas, 1998] D. Le Pesant and M. Mathieu Colas. Les classes d'objets. *LANGAGES*, (131) :6–125,, septembre 1998.

[Phillips and Thompson, 1985] J. Phillips and H. Thompson. GPSGP - a parser for GPSG. Linguistics, 23, 1985.

[Piaget *et al.*, 1979] J. Piaget, N. Chomsky, and Massimo Piattelli-Palmarini. *Theories Du Langage, Theories De L'apprentissage : Le débat entre Jean Piaget et Noam Chomsky.* Centre Royaumont pour une science de l'homme. Seuil, 1979.

[Pierrel, 1997] J.-M. Pierrel. Bilan des grandes orientations de recherche en traitement automatique du langage parlé en france. TAL, 38 :2, 1997.

[Pittman, 1948] R. Pittman. Nuclear structure in linguistics. *Language*, 24 :287–292, 1948.

[Pla *et al.*, 2000] F. Pla, A. Molina, and N. Pietro. Tagging and chunking with bigrams. Technical report, Universitat Politecnica de València, 2000.

[Pollard and Sag, 1987] C. Pollard and I. Sag. *Information-Based Syntax and Semantics — Volume 1 : Fundamentals.* Center for the Study of Language and Information (CSLI), Stanford, 1987. lecture notes no. 13.

[Pollard and Sag, 1994] C. Pollard and I. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, 1994. lecture notes no. 13.

[Pollard, 1996] K. Pollard. The nature of constraint-based grammar. PACLIC Conference, reprinted in *Constructions : an HPSG Perspective*, ESSLLI'98 Lecture Notes, 1996.

[Pottier, 1962] B. Pottier. *Systématique des éléments de relation.* Klincksieck, Paris, 1962.

[Pottier, 1967a] B. Pottier. Au-delà du structuralisme en linguistique. *Critique*, 237 :266–274, février 1967.

[Pottier, 1967b] B. Pottier. *Présentation de la linguistique.* Klincksieck, Paris, 1967.

[Pottier, 1974] B. Pottier. *Linguistique générale, théorie et description.* Klincksieck, Paris, 1974.

[Pottier, 1987] B. Pottier. *Théorie et analyse en linguistique.* Hachette, Paris, 1987.

[Prince and Smolensky, 1993] A. Prince and P. Smolensky. Optimality theory : Constraint interaction in generative grammar. Technical report, Rutgers University Centre for Cognitive Science, 1993.

[Rajman, 1995] M. Rajman. Approche probabiliste de l'analyse syntaxique. TAL 36-1-2. pp 157-200, 1995.

[Roche and Schabes, 1997] E. Roche and Y. Schabes. Finite-state language processing. MIT Press, 1997.

[Roche, 1993] E. Roche. Analyse transformationnelle du français par transducteurs et lexiques-grammaires, thèse de doctorat, université paris 7, 1993.

[Romary *et al.*, 2004] L. Romary, G. Francopoulo, and S. Salmon-Alt. Standards going concrete : from LMF to morphalou. COLING workshop, 2004.

[Rosenkrantz and Lewis, 1970] D. Rosenkrantz and P. Lewis. Deterministic left-corner parser. Annual Symposium on Switching and Automata Theory, 1970.

[Russel, 1901] B. Russel. *De principia mathematica.* 1901.

[Sabatier and Pasero, 1997] P. Sabatier and R. Pasero. Concurrent processing for sentences analysis, synthesis and guided composition. Rapport interne 221, Laboratoire d'Informatique de Marseille URA CNRS 1787, 1997.

[Sabatier and Pasero, 1998] P. Sabatier and R. Pasero. GNF : une grammaire noyau du français. Rapport, Laboratoire d'Informatique de Marseille, 1998.

[Sabatier, 1987] P. Sabatier. Contribution au développement d'interfaces en langage naturel. Thèse d'Etat, Université Paris VII, 1987.

[Sag and Wasow, 1999] I. Sag and T. Wasow. Syntactic theory. a formal introduction. *CSLI*, 1999.

[Sag *et al.*, 2002] I. Sag, T. Baldwin, F. Bond, A. Copestake, and D. Flickinger. Multiword expressions : A pain in the neck for NLP. In Proceedings of the Third International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2002), Mexico City, Mexico, pp. 1-15, 2002.

[Sag *et al.*, 2003] I. Sag, T. Wasow, and E. Bender. Formal syntax, an introduction. CSLI publication, 2003.

[Sag, 1997] I. Sag. English relative clause constructions. Journal of Linguistics, 1997.

[Sager, 1981] N. Sager. *Natural language information processing : a computer grammar of English and its applications*. Addison-Wesley, Reading, 1981.

[Salkoff, 1973] M. Salkoff. *Une grammaire en chaîne du français : analyse distributionnelle*. Dunod, Paris, 1973.

[Salkoff, 1979] M. Salkoff. *Analyse syntaxique du français : grammaire en chaîne*. Benjamins, Amsterdam, 1979.

[Sampson, 1994] G. Sampson. Suzanne : a domesday book of english grammar. in Corpus-based Research into language, N. Oostdijk and P. de Haan (eds), Rodopi : Amsterdam, p. 169-187, 1994.

[Schabes, 1994] Y. Schabes. Left to right parsing of LTAG. in Computational Intelligence, 1994.

[Shieber,] S. Shieber. Introduction to unification-based theories of grammar. CSLI series, University of Chicago Press.

[Shieber, 1984] S. Shieber. Direct parsing of ID/LP grammars. Linguistics and Philosophy, 7, 1984.

[Shieber, 1985] S. Shieber. Using rectriction to extend parsing algorithms for complex feature-based formalisms. 23e ACL Meeting, Chicago, p. 145-152, 1985.

[Shieber, 1992] S. Shieber. *Constraint-Based Grammar Formalisms*. MIT Press, 1992.

[Silberztein, 1993] M. Silberztein. Dictionnaires électroniques et analyse automatique de textes : le système INTEX. Paris, Masson, 1993.

[Simov *et al.*, 2001] K. Simov, Z. Peev, M. Kouylekov, A. Simov, M. Dimitrov, and A. Kiryakov. CLaRK – an XML-based system for corpora development. pages 558–560. proceedings of Corpus Linguistics 2001, 2001.

[Simov *et al.*, 2002] K. Simov, M. Kouylekov, and A. Simov. Cascaded regular grammars over XML documents. proceedings of the 2nd Workshop on NLP and XML (NLPXML-2002), 2002.

[Sleator and Temperley, 1993] D. Sleator and D. Temperley. Parsing english with a link grammar. Internat. Workshop on Parsing Technologies, 1993.

[Smith, 1999] J. Smith. English number names in HPSG. in Webelhuth et al. (eds) Lexical and constructional aspects of linguistic explanation, CSLI, Stanford, p. 145-160, 1999.

[Sorace and Keller, 2005] A. Sorace and F. Keller. Gradience in linguistic data. Lingua115 :11, 2005.

[Spark-Jones and Galliers, 1996] K. Spark-Jones and J. Galliers. Evaluating natural language processing systems. Lecture Notes in Artificial Intelligence, Springer, 1996.

[Srinivas, 1997] B. Srinivas. Complexity of lexical descriptions and its relevance for partial parsing. Thèse de PhD, University of Pennsylvania, Philadelphia, 1997.

[Stabler, 1992] E. Stabler. The logical aproach to syntax. MIT Press, 1992.

[Tesnière, 1934] L. Tesnière. Comment construire une syntaxe. *Bulletin de la Faculté des Lettres de Strasbourg*, 12(7) :219–229, 1934.

[Tesnière, 1959] L. Tesnière. *Eléments de syntaxe générale*. Klincksieck, Paris, 1959.

[Thomasset and Villemonte De La Clergerie, 2005] F. Thomasset and E. Villemonte De La Clergerie. Comment obtenir plus des méta-grammaires. in actes de TALN-05, DOURDAN, 2005.

[Tomita, 1987] M. Tomita. An efficient augmented context-free parsing algorithm. Computational Linguistics, 13, 1987.

[Tomita, 1991] M. Tomita. Parsing 2-dimensional language. in Current Issues in Parsing Technologies, Kluwer, 1991.

[Touratier, 2002] C. Touratier. *Morphologie et Morphématique — Analyse en morphèmes*. Publications de l'Université de Provence — coll. Langues et Langages, Aix en Provence, 2002.

[Toussaint, 1967] M. Toussaint. Gustave guillaume et l'actualité linguistique. *Langages*, 6 :93–100, 1967.

[Tseng, 2003] J. Tseng. A french HPSG grammar. ESSLI workshop on grammar development, Vienna, 2003.

[Uszkoreit, 2002] H. Uszkoreit. New chances for deep linguistic processing. in proceedings of COLING-02, 2002.

[Uzkoreit, 1986] H. Uzkoreit. Categorial unification grammars. In *Proceedings of COLING-86*, pages 187–194, Bonn, 1986.

[Vaillant, 1997] P. Vaillant. Interaction entre modalités sémiotiques : de l'icône à la langue. Thèse de l'Université Paris XI, Orsay, France, 1997.

[Van Noord, 1997] G. Van Noord. An efficient implementation of the head-corner parsing. Computational Linguistics, 1997.

[Van Noord, 1999] G. Van Noord. Pauses location and duration calculated with syntactic dependencies and textual considerations for t.t.s. system. de ICPhS, 1999.

[Vanrullen *et al.*, 2003] T. Vanrullen, M.L. Guénot, and E. Bellengier. Formal representation of property grammars,. proceedings of ESSLLI 2003 Student Session, 2003.

[Vergne, 2000] J. Vergne. *Etude et modélisation de la syntaxe des langues à l'aide de l'ordinateur — Analyse syntaxique automatique non combinatoire*. thèse d'hdr, Université de Caen, 2000.

[Vijay-Shanker and Weir, 1993] K. Vijay-Shanker and D. Weir. Parsing some constrained grammar formalisms. Computational Linguistics, 19 :4, p. p. 591-636, 1993.

[Villemonte de la Clergerie and Rajman, 2003] E. Villemonte de la Clergerie and M. Rajman. Evolutions en analyse syntaxique. *Traitement Automatique des Langues (T.A.L.)*, 44(3) :7–15, 2003.

[Villemonte de la Clergerie, 2002] E. Villemonte de la Clergerie. Construire des analyseurs avec DyALog. In *Proc. of TALN'02*, June 2002.

[Villemonte de la Clergerie, 2005] E. Villemonte de la Clergerie. Concevoir des analyseurs syntaxiques avec DyALog à partir de méta-grammaires, January 2005. Slides presented at TALaNa, University Paris 7.

[Villemonte de la Clergerie, 1991] E. Villemonte de la Clergerie. A tool for abstract interpretation : Dynamic programming. In *Actes JTASPEFL'91*, pages 151–156, Bordeaux (FRANCE), October 1991.

[Villemonte de la Clergerie, 1994] E. Villemonte de la Clergerie. Modulated Call/Return evaluation strategies for logic programs. Nov 1994.

[Villemonte de la Clergerie, 2001] E. Villemonte de la Clergerie. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, pages 167–174, CMU, Pittsburgh, PA, USA, June 2001.

[Vincenz, 1969] I. Vincenz. Gustave guillaume et la théorie transformationnelle. *Cahiers de linguistique théorique et appliquée*, IV :113–118, 1969.

[Wehrli, 1985] E. Wehrli. Design and implementation of a lexical data base. In *ACL Proceedings, Second European Conference*, pages 146–153, 1985.

[Wehrli, 1988] E. Wehrli. Parsing with a GB grammar. In U. Reyle and C. Rohrer, editors, *Natural languauge parsing and linguistic theories*, pages 177–201. Reidel, Dordrecht, 1988.

[Wehrli, 1992] E. Wehrli. The IPS system, 1992.

[Wehrli, 1997] E. Wehrli. L'analyse syntaxique des langues naturelles. Masson, 1997.

[Wilmet, 1972] M. Wilmet. *Gustave Guillaume et son école linguistique*. Nathan, Paris, 1972.

[Wilmet, 1998] M. Wilmet. *Grammaire critique du Français*. Duculot. Hachette, 2ème edition, 1998.

[Winograd, 1983] T. Winograd. *Language as a cognitive processs — Vol. 1 : Syntax*. Addison Wesley, Reading (Mass.), 1983.

[Woods, 1970] W. A. Woods. Transition network grammars for natural language analysis. *CACM*, 13 :591–606, 1970.

[Woods, 1980] W. A. Woods. Cascaded ATN's. *Journal of the Association for Computational Linguistics*, 6 :1–12, 1980.

[Xia, 2001] F. Xia. Automated grammar development from two different perspectives. University of Pennsylvania. PhD thesis, 2001.

[Younger, 1967] D. Younger. Recognition and parsing of context-free languages in time n3. Information and Control, 10 :2, 1967.

Résumé

Il est souhaitable qu'une analyse syntaxique -en traitement automatique des langues naturelles - soit réalisée avec plus ou moins de précision en fonction du contexte, c'est-à-dire que sa granularité soit réglable. Afin d'atteindre cet objectif, nous présentons ici des études préliminaires permettant d'appréhender les contextes technique et scientifique qui soulèvent ce problème. Nous établissons un cadre pour les développements à réaliser et pour leur évaluation. Nous choisissons un formalisme d'analyse par satisfaction de contraintes (celui des Grammaires de Propriétés) ayant l'avantage de permettre l'utilisation des mêmes ressources linguistiques avec un degré de précision réglable. Nous introduisons une reformulation mathématique du formalisme des Grammaires de Propriétés et nous définissons une mesure (la densité de satisfaction), qui permet de contrôler la granularité de l'analyse. Puis nous décrivons un ensemble d'outils modulaires (LPLSuite) et de ressources (lexique et sous-lexiques DicoLPL) développés pour permettre une analyse syntaxique et susceptibles d'être embarqués dans des applications de haut niveau. Nous présentons et évaluons ensuite plusieurs analyseurs syntaxiques dans ce formalisme, le dernier (SeedParser) étant destiné à mettre en œuvre une véritable analyse à granularité variable. L'évaluation de ces outils est l'objet d'une étude approfondie. Enfin, nous présentons quelques applications développées à l'aide de nos outils.

Mots-clés: Analyse syntaxique automatique, robustesse, granularité variable, Grammaires de Propriétés