Vers une génération automatique en ROBRA d'analyseurs et de générations syntaxiques pour des systèmes de traduction automatique

Amélie BOSC

Mémoire de MASTER

Ingénierie Multilingue

Directeurs de recherche :

Vincent BERMENT

Frédérique SEGOND

Présenté et soutenu le 21 novembre 2014

Remerciements

J'adresse tous mes remerciements aux personnes ayant contribué à ce mémoire et tout particulièrement à mes directeurs de recherche pour leur encadrement et leurs bons conseils, à mon maître de stage Vincent Berment, pour sa disponibilité, son soutien et son aide, aux personnes du GETALP et de Malaisie qui m'ont aidée dans ma compréhension des formalismes linguistiques et pour la réalisation de l'outil de saisie, à tous mes professeurs et mes camarades de master à l'INaLCO ainsi qu'à mon collègue de stage.

En dernier lieu, je tiens à remercier mes parents pour tout leur soutien et leur accompagnement.

Résumé

Ariane est un environnement de développement de systèmes de traduction automatique par règles. Les grammaires statique sont un élément-clé pour le développement de tels systèmes dans la mesure où en décrivant la langue, elles servent de spécification pour les phases structurales écrites en ROBRA : les programmes d'analyse de l'énoncé en langue source et de génération de l'énoncé en langue cible. Ces grammaires statiques présentent néanmoins deux inconvénients qui sont d'être difficiles à prendre en main et pas assez formelles.

L'objectif de ce travail est donc de proposer une alternative aux grammaires statiques afin de permettre de rédiger plus facilement des spécifications linguistiques et d'intégrer de nouvelles langues dans un système. La solution que nous proposerons devra par ailleurs être manipulable informatiquement, tant par sa formalisation que dans son format de stockage interne.

Mots-clés : traduction automatique, RBMT, Ariane, Héloïse, ROBRA, grammaires statiques, GSCS, STCG, SSTC, GETA, GETALP, INaLCO, langues peu dotées, grammaires d'unification, langage de spécification, XML, Bidirectional Correspondence Grammar (BCG)

Sommaire

Remerciements....2
Résumé....3
Sommaire....4
Liste des Figures....5
1. Introduction....6
1.1 Contextualisation historique....6
1.2. Problématique....7
1.3. Démarche....8
2. État de l'art....9
2.1. Description linguistique et origines des grammaires d'unification....9
2.2. Des grammaires d'unification constituant des modèles linguistiques....12
2.3. Quelques analyseurs récents comme exemples d'application de tels formalismes....15
3. Méthodologie et formalisme linguistiques du GETA....19
3.1. Héritage linguistique....19
3.2. La structure multiniveaux....22
3.2.1. Qu'est-ce que la structure multiniveaux ?....22
3.2.2. Description formelle et exemples....24
3.2.3. Le rôle de la structure multiniveaux dans Ariane-G5....31
3.3. Les grammaires statiques, le formalisme du GETA....36
3.3.1. Description générale des grammaires statiques....36
3.3.2. Description formelle des grammaires statiques....37
3.3.3. Intérêt des grammaires statiques....41
4. Proposition d'un nouveau formalisme....44
4.1. Rappel du double enjeu de ce formalisme....44
4.1.1. Un formalisme plus abordable que celui des grammaires statiques....44
4.1.2. Permettre la génération de code ROBRA....45
4.2. État des recherches autour des grammaires statiques....48
4.3. Description des STCG....51
4.4 Solution proposée : Bidirectional Correspondence Grammar....52
4.4.1 Synthèse entre les grammaires statiques et les STCG....52
4.4.2. Présentation de la solution adoptée....54
4.4.3. Limites et développements envisageables....57
4.5 Format de stockage interne des planches....58
4.6. Développement d'Eurydice, interface de saisie pour les planches de BCG....60
5. Conclusion....64
Glossaire....65
Bibliographie....67
Liens....70
Annexes....71

Liste des Figures

Fig. 1 : description d'un groupe nominal en FUG.....11

Fig. 2 : description de « Jean aime Marie » en LFG.....12

Fig. 3 : règle correspondant à « Jean aime cette maison » en TAG.....14

Fig. 4 :analyse de « Jean aime cette maison » en TAG.....14

Fig. 5 : niveau syntaxique de la structure multiniveaux.....26

Fig. 6 : niveau des fonctions syntaxiques de la structure multiniveaux.....27

Fig. 7 : niveau des relations logiques de la structure multiniveaux.....28

Fig. 8 : niveau des relations sémantiques de la structure multiniveaux.....30

Fig. 9 : structure multiniveaux de « il donne un livre à Marie ».....31

Fig. 10 : le triangle de Vauquois.....32

Fig. 11 : structure logique du verbe « prendre ».....35

Fig. 12 : hiérarchie des groupes syntagmatiques du français.....33

Fig. 13 : zone III de grammaire statique.....41

Fig. 14 : description d'un groupe nominal en LSCS.....48

Fig. 15 : capture d'écran d'Eurydice - exemple d'arbre.....62

Fig. 16 : capture d'écran d'Eurydice - boite de dialogue.....62

1. Introduction

1.1 Contextualisation historique

La traduction automatique est la discipline mère du traitement automatique des langues. Elle apparaît en contexte de guerre froide et le tout premier système de traduction automatique est issu d'une collaboration entre IBM et l'université de Georgetown en 1954. Ce système est capable de traduire une soixantaine de phrases du russe à l'anglais [Chéragui, 2012]. À ce moment, la traduction automatique apparaît comme une réalité, on assiste à un véritable engouement pour la recherche dans ce domaine et on espère voir rapidement des systèmes fonctionnels produisant d'aussi bonnes traductions que celles des traducteurs humains. En 1958 a lieu le premier congrès sur la traduction automatique à Moscou et une douzaine de laboratoires de recherches dans ce domaine-là fleurissent rien qu'aux États-Unis.

La recherche en matière de traduction automatique débute à la fin des années 50 en France d'une part avec la création de l'ATALA et la fondation de sa revue *Traduction automatique* et d'autre part avec la création du CETA en 1959, Centre d'Étude pour la Traduction Automatique, rattaché au CNRS. Le CETA se divise en deux unités : le CETAP à Paris sous la direction d'Aimé Sestier et le CETAG à Grenoble sous la direction de Bernard Vauquois. Trois ans plus tard divers facteurs mènent à la démission d'Aimé Sestier et à la dissolution du CETAP [Léon, 2002]. Le CETAG devient alors CETA et poursuit seul ses recherches, toujours rattaché au CNRS. En 1966, un rapport pessimiste sur l'avenir de la traduction automatique publié par l'ALPAC, une commission internationale pour le traitement automatique des langues, met un frein au développement de la recherche en matière de traduction automatique dans le monde. Pour autant, la dynamique du CETA n'en est pas affectée. Le GETA, département issu d'une division du CETA en 71, développe alors ce qui deviendra Ariane-78, système de traduction automatique par règles [Vauquois, 79]. Il s'agit d'un système pionnier dans le domaine et c'est dans le cadre de ce système que se situent notre étude. En 2009, Vincent Berment de l'entreprise Taranis Software, spécialisée en traduction automatique, reprend alors ce système, rédige les compilateurs permettant d'exploiter réellement le système développé sous IBM. La nouvelle version d'Ariane désormais disponible en ligne a été nommée Héloïse [lien 1]. C'est précisément sur ce système que nous

travaillerons au cours de nos recherches.

1.2. Problématique

Ariane-78, puis Ariane-G5, sont des générateurs de systèmes de traduction par règles fonctionnant en trois étapes : analyse, transfert, génération. Le texte source est analysé, puis l'analyse ainsi produite est transformée pour enfin permettre la génération du texte en langue cible. Des règles de description linguistique doivent être rédigées pour permettre les étapes d'analyse et de génération structurales. Ces règles sont implémentées dans le système en ROBRA, un LSPL, c'est-à-dire un langage spécialisé pour la programmation linguistique. Chaque module d'analyse ou de génération programmé en ROBRA comprend de nombreuses lignes de code afin de couvrir l'intégralité de la langue traitée. En conséquence, et afin de maîtriser les développements complexes des différents modules de chaque langue, il faut passer par une spécification, en l'occurrence une description purement linguistique. Cette spécification, qui a pour vocation d'être compréhensible à la fois par des linguistes et par des programmeurs, a été créée pour les besoins du système Ariane. Le GETA l'a nommée « grammaire statique ».

Les grammaires statiques sont un outil puissant, elles permettent de faire une description complète d'une langue mais elles présentent un premier inconvénient : elles sont dans un format trop peu formel pour permettre de les manipuler automatiquement, elles sont même le plus souvent manuscrites. Mais il est envisageable de définir un format de description linguistique qui rende possible de générer automatiquement ou semi-automatiquement du code ROBRA. Ceci a déjà été réalisé à l'université de Penang en Malaisie avec les travaux de Zaharin Bin Yusoff notamment, sur lesquels nous reviendrons. La génération automatique de code ROBRA faciliterait le développement de nouveaux modules permettant au système de traiter de nouvelles langues.

Nous chercherons donc à établir un formalisme qui soit à la fois suffisamment puissant pour décrire des langues et suffisamment mathématique et formel pour permettre de la génération de code. Mais le formalisme en question doit avant tout être facile à appréhender, en particulier pour une personne ne possédant pas de formation poussée en informatique. En

effet, ceci représentera la partie la plus importante de notre travail qui se situe également dans l'optique de faciliter le développement de ressources linguistiques pour les langues peu dotées informatiquement. Notre formalisme s'adressera donc en grande partie à des spécialistes en langues et en linguistique et tâchera de leur offrir une solution simple leur permettant de développer une spécification opératoire pour leurs langues. Une réflexion plus poussée sur l'intérêt des systèmes de traduction automatique par règles comme celui sur lequel nous travaillons, pour la traduction de langues peu dotées est développée dans [de Malézieux, 2014].

1.3. Démarche

La recherche d'un outil de description de la langue à la fois puissant et formel rappelle les enjeux du courant des grammaires d'unification. Nous allons donc étudier ce courant en première partie de ce mémoire, dans le chapitre 2, et tâcher de comprendre ce qui existe dans le domaine pour s'en inspirer et l'utiliser dans notre tâche. Puis dans le chapitre 3, nous expliquerons ce en quoi consistent les grammaires statiques, ce formalisme adopté par le GETA sur lequel nous travaillons. Nous isolerons ainsi ses particularités afin d'identifier nos besoins. Nous pourrons ensuite présenter dans le chapitre 4 la proposition de formalisme à laquelle nous avons pensé, cette proposition s'inspirant d'un des multiples formalismes dérivés des grammaires statiques, les STCG. Enfin, en dernier lieu nous pourrons exposer notre travail de réalisation d'un outil de saisie pour ce nouveau formalisme. Cet outil représente l'aboutissement de notre travail, la possibilité pour un linguiste sans formation en informatique de soumettre des règles linguistiques parfaitement normalisées, stockées dans un format numérique facilement manipulable afin de générer du code ROBRA.

2. État de l'art

2.1. Description linguistique et origines des grammaires d'unification

Notre étude consiste en la recherche d'un formalisme de description linguistique adapté au système de traduction automatique sur lequel nous travaillons. En plus de permettre une véritable description linguistique, ce formalisme devra être rigoureusement défini, formel et mathématique. Nous trouverons des sources d'inspiration du côté des grammaires d'unification et avec des exemples d'application à travers de quelques analyseurs syntaxiques. Nous allons en passer en revue quelques exemples, afin d'en identifier les points forts et les points faibles et de s'en inspirer, pour établir une solution adaptée à nos besoins.

Le courant des grammaires d'unification est issu d'une remise en question de l'étude de la syntaxe comme un objet à part, détachée de la langue dans son ensemble [Abeillé, 93]. Elle ont été pensées pour permettre de traiter de manière unie la syntaxe, le lexique et la sémantique, trois niveaux d'étude de la langue qui sont traditionnellement étudiés par des disciplines différentes. Elles constituent des théories linguistiques à part entière et s'inscrivent dans la lignée de Chomsky, en tant que grammaires mathématiques, formelles, manipulables informatiquement. Elles ont pu se développer grâce à un apport important de domaines comme la logique, l'informatique, l'intelligence artificielle, etc. Cette discipline se situe donc au croisement entre informatique et linguistique.

La théorie standard de Chomsky a pu apporter un cadre formel aux études linguistiques et une première réponse au besoin de décrire la langue de manière manipulable informatiquement. Certains points cependant ont pu être remis en question : la place de la sémantique a porté à débat [Abeillé, 96]. Valait-il mieux déduire la sémantique depuis la structure profonde de la langue comme dans la théorie standard ou valait-il mieux déduire la sémantique de la structure de surface selon les propositions théoriques plus récentes de Chomsky ? C'est l'hypothèse d'une sémantique de surface qui a été retenue par les grammaires d'unification dont la phase initiale d'analyse en constituants est dépourvue de caractère

sémantique. Elle est fondée sur l'interprétation du texte et est représentée par des graphes. C'est ensuite le fondement même des grammaires génératives et l'appareil transformationnel qui ont été remis en question. Ces grammaires n'apparaissaient plus à même de décrire le langage humain, par exemple, les langues avec un ordre des mots libre posent problème.

Le courant des grammaires d'unification, impulsé par des développements en informatique comme l'arrivée de Prolog, s'est donc proposé de reprendre et enrichir les grammaires de réécriture pour les rendre implémentables [Marandin, 2001]. Trois grammaires sont de bons exemples du résultat obtenu :

- Les grammaires de clauses définies, établies en 1980 par Fernando Peirera et David Warren, autrement appelées DCG, sont des grammaires de type logique. L'analyse fonctionne comme une déduction. Chaque règle de grammaire est comparable à un axiome et est constituée de prédicats et d'arguments qui peuvent contenir des variables. Ces variables seront instanciées à l'application de l'axiome. La validité de la phrase analysée se déduit des axiomes de la grammaire. Il s'agit donc une grammaire particulièrement bien adaptée pour une implémentation dans des langages comme Prolog. On peut par exemple décrire ainsi un groupe nominal constitué d'un déterminant et d'un nom, les deux éléments s'accordant entre eux en genre et en nombre.

  Exemple :

  GN(u,v) ==> D(x,y), N(z,w) {Accord(x,z), Accord(y,w), Accord(x,u), Accord(y,v)}

  u, x et z sont les variables représentant le genre, elles doivent donc être accordées entre elles, de la même manière, v, y et w sont les variables représentant le nombre et elles doivent être accordées entre elles. Si toutes les conditions se vérifient, le groupe nominal a été construit.

- Les grammaires fonctionnelles d'unification (FUG) de Martin Kay datent de 1984, elles fonctionnent avec des ensembles de traits. Chaque élément est un atome, caractérisé par un ensemble de traits qui peuvent chacun prendre certaines valeurs. Les groupes syntagmatiques sont alors constitués en unifiant les traits de chacun de leurs composants. Un groupe nominal peut se constituer de la manière suivante :

Exemple : un groupe nominal composé d'un déterminant et d'un nom se décrit ainsi :

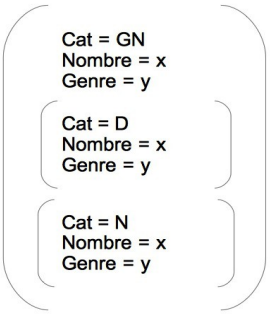


Fig. 1 : description d'un groupe nominal en FUG

L'accord en nombre et en genre se vérifie lorsque les valeurs attachées à Nombre et Genre sont identiques. L'analyse produite est celle d'une structure de traits.

- PATR date de 1983. Son concepteur, Stuart M. Shieber, cité par Anne Abeillé [Abeillé, 96], définit PATR comme « la plus simple des grammaires d'unification ». Ce formalisme se compose de règles de réécritures, augmentées de traits qui sont rattachés à l'élément qui leur correspond grâce à un identifiant.

  Exemple : Un groupe nominal composé d'un déterminant suivi d'un nom.

  X0 ==> X1 X2
  <X0 Cat> = GN
  <X1 Cat> = D
  <X2 Cat> = N
  <X0 Accord> = <X1 Accord> = <X2 Accord>

  Le trait Accord regroupe les informations sur l'accord en nombre et l'accord en genre à la fois.

Ces trois formalismes, Tang Enya Kong les décrit dans sa thèse comme bidirectionnels, flexibles et implémentables. Mais leur problème majeur, est que pour arriver à une description exhaustive d'une langue, il faut multiplier les règles et traits utilisés pour

chaque phénomène linguistique ajouté à la grammaire. Il devient donc difficile de distinguer les principes généraux linguistiques qui régissent la langue décrite. Les grammaires d'unification proposés ci-dessous, se définissent en revanche comme des « modèles véritablement linguistiques » [Abeillé, 96].

2.2. Des grammaires d'unification constituant des modèles linguistiques

Nous allons à présent passer en revue quelques exemples parlants de grammaires d'unification :

En 1982, les grammaires lexicales-fonctionnelles de Joan Bresnan et Ronald Kaplan, les LFG, ont été construites dans le but d'offrir une alternative simple aux grammaires génératives transformationnelles pour diverses raisons [Abeillé, 96], entre autre : permettre de caractériser les langues de manière à dégager plus facilement les universaux de langage, produire une grammaire plus légère donc plus facile à implémenter. Les LFG superposent une structure de constituants, appelée structure-c, et une structure fonctionnelle, autrement appelée structure-f. La structure de constituants est représentée sous forme d'un arbre, elle correspond aux groupes syntagmatiques de la phrase. La structure-f est représentée sous forme de traits ajoutés à l'arbre, et représente les fonctions grammaticales des éléments de la phrase.

- Exemple : « Jean aime Marie »

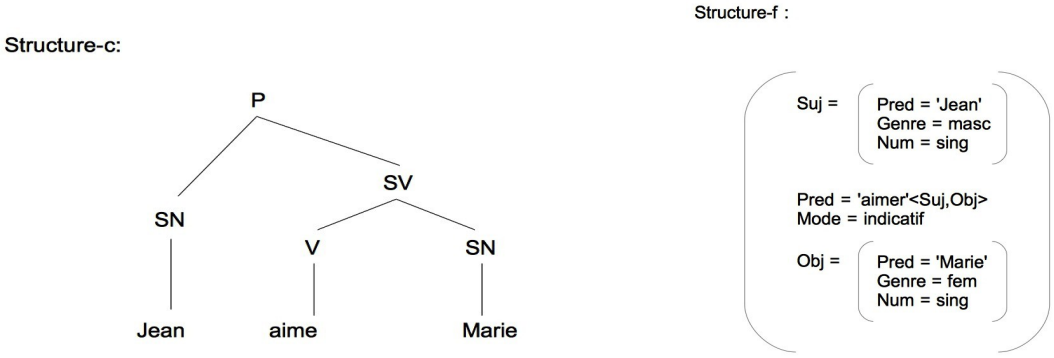


Fig. 2 : description de « Jean aime Marie » en LFG

Ainsi, tandis que la structure de constituants permet d'identifier les groupes syntagmatiques de la phrase, la structure-f permet d'identifier le prédicat, son sujet et son objet. On notera que les arguments attendus par le prédicat sont précisés entre chevrons dans la structure-f. Ici, le verbe « aimer » exige un sujet et un complément.

La grammaire syntagmatique généralisée, GPSG, a été développée au début des années 80 par Gerald Gazdar, Ewan Klein, Ivan Sag et Geoffrey Pullum. Elle consiste en un ensemble constitué d'une grammaire hors-contexte et d'une structure de traits. Le fonctionnement de cette grammaire est celui de la généralisation. Il n'y est pas fait usage de classes syntagmatiques, mais de traits de description. L'utilisation de ces traits permettent de généraliser certains éléments au sein d'une règle. Aux règles s'ajoutent des meta-règles, qui permettent d'exprimer des conditions et de généraliser les relations entre les phrases et éviter la redondance des règles indépendantes. La langue est ainsi décrite par des modules qui peuvent se généraliser. Les grammaires syntagmatiques guidées par les têtes, HPSG, ont été développées dans leur continuité au début des années 80 par Carl Pollard et Ivan A. Sag. Elles reprennent les mêmes représentations en traits des éléments que dans les FUG de Martin Kay. Les HPSG se veulent également simplifiées par rapport aux GPSG.

Les grammaires d'arbres adjoints, TAG, ont été définies dans le milieu des années 70 par A. Joshi. L'unité de base manipulée par ce formalisme est celle des arbres élémentaires, qui sont combinés selon deux opérations : la substitution et l'adjonction. Avec la première, il s'agit de remplacer un nœud d'un arbre élémentaire par un autre arbre élémentaire, de sorte à obtenir un arbre plus complexe. La seconde consiste simplement à joindre des arbres entre eux. À chaque élément des arbres peuvent être associés des traits. La force de ce formalisme est de faire dépendre la description syntaxique de la description lexicale. Il n'y a donc pas de règles syntaxique qui fonctionne indépendamment des propriétés combinatoires propres aux éléments du lexique.

Exemple : Jean aime cette maison.

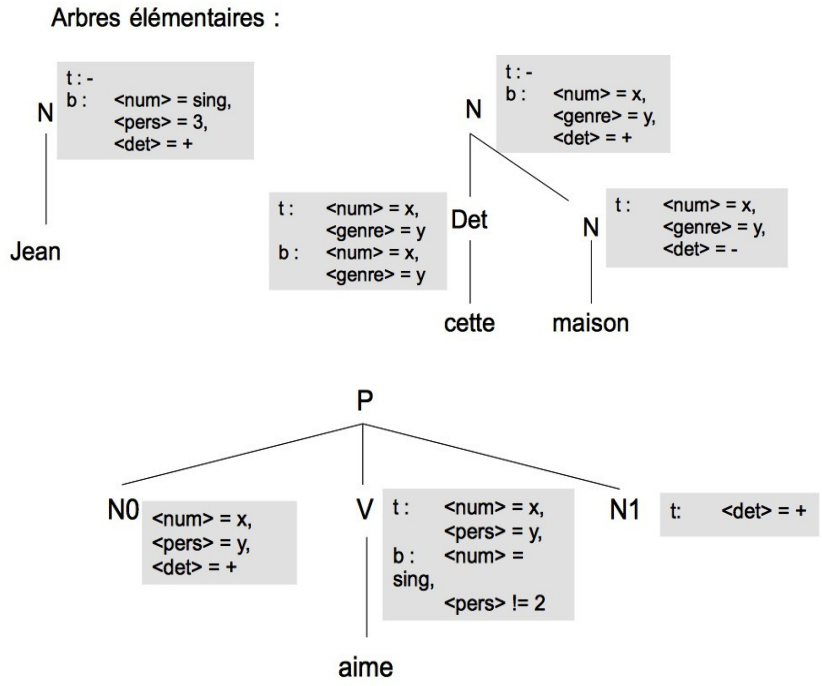


Fig. 3 : règle correspondant à « Jean aime cette maison » en TAG

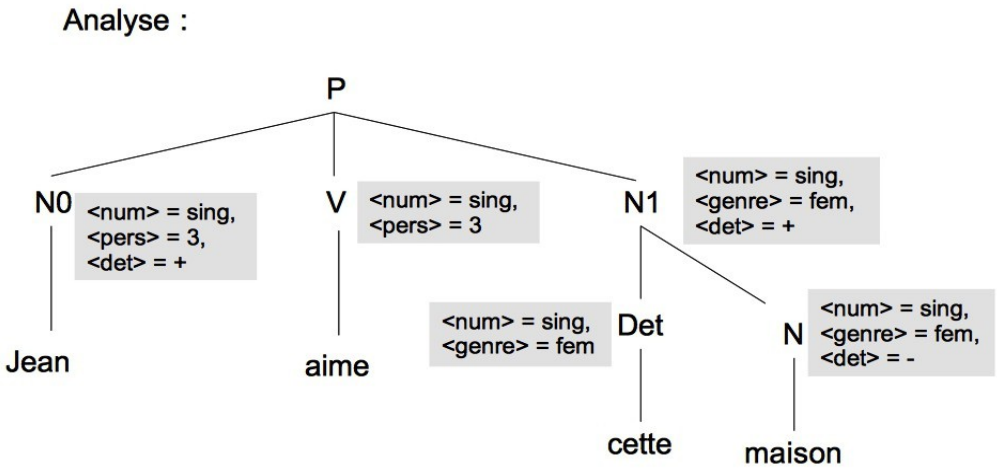


Fig. 4 : analyse de « Jean aime cette maison » en TAG

Trois arbres élémentaires se regroupent pour produire l'analyse de la phrase. Les arbres de N0 et N1 s'ajoutent à l'arbre du prédicat par substitution. En effet, c'est le nœud terminal qui est remplacé par un arbre élémentaire. Les substitutions sont valides parce que les traits correspondent. Par exemple, un déterminant est attendu pour N1, et l'arbre n'est valide que parce que le déterminant est présent.

Parmi les quatre grammaires que nous venons d'exposer, la première, LFG et la dernière, TAG, ont retenu notre attention. Elles se rapprochent de la solution de formalisme que nous cherchons : elles présentent en particulier une gestion des relations logiques de la phrase à travers la gestion des arguments des prédicats, des idées intéressantes dans la façon de gérer les traits.

Une limite importante, dans le cadre de nos recherches, des formalismes que nous venons de présenter, est qu'ils ont été développés en anglais et pour l'anglais et sont parfois difficiles à adapter à d'autres langues [Abeillé, 96]. Or il nous faut un formalisme à capacité multilingue. Mais les grammaires d'unification auront fourni un cadre de recherche et de développement pour nombre des analyseurs actuels. Nous avons lieu de nous intéresser à ces analyseurs pour compléter notre état de l'art et constater l'évolution des théories que nous venons de présenter.

2.3. Quelques analyseurs récents comme exemples d'application de tels formalismes

Il existe aujourd'hui plusieurs analyseurs efficaces qui sont autant d'exemples de réussites face à ce défi de représentation et d'abstraction mathématisée de la langue. En voici quelques exemples que nous allons développer en essayant d'insister sur les formalismes de spécification utilisés à chaque fois que cela apportera un élément d'inspiration pour la suite de nos travaux.

FRMG [lien 2] est l'analyseur développé par l'équipe ALPAGE de l'INRIA depuis 2004. Il est programmé à l'aide d'une grammaire TAG qui est ensuite transformée en

analyseur syntaxique par un système appelé DYALOG. Les grammaires TAG présentent l'inconvénient de devenir difficile à maintenir lorsqu'elles deviennent trop importantes, en raison du nombre de combinaisons d'arbres possibles [de la Clergerie, 2009]. Les métagrammaires, ou MG, ont été conçues en conséquence pour servir de spécification aux grammaires TAG.

Les métagrammaires permettent de générer les grammaires TAG, et Éric de la Clergerie les décrit comme « une grammaire source descriptive, servant à produire une grammaire cible opérationnelle ». Les métagrammaires sont faites de classes contenant une liste de propriétés. Les classes sont ensuite combinées entre elles pour générer les arbres. Il n'y a pas de composante graphique dans cette spécification, son format semble particulièrement plus adapté à la manipulation pour génération d'autres grammaires.

Le LIMA est l'analyseur multilingue du CEA LIST [lien 3], développé depuis 2002 et réalisé dans la lignée des travaux de Christian Fluhr. Il est capable de traiter neuf langues assez différentes les unes des autres (l'anglais, le français, l'allemand, l'espagnol, l'italien, le russe, l'arabe, le chinois et le hongrois). Il s'agit d'un analyseur en dépendance robuste. Romaric Besançon explique dans [Besançon 2005] que son analyse se compose de deux étapes : d'abord une recherche des chaînes nominales et verbales, ensuite une recherche des relations de dépendance implémentées à l'aide d'automates à états finis. C'est la stratégie d'application de ces automates qui assure la robustesse du système. Une analyse sera donc toujours produite, quel que soit l'énoncé. Les automates à états finis ainsi décrits correspondent en fait à des règles, qui sont regroupées en plusieurs ensembles appliqués les uns après les autres, dans un ordre bien défini pour produire l'analyse qui se construit par accumulation de relations.

Diverses applications de cet analyseur ont été faites : au sein de la plateforme de recherche de documents du CEA LIST, pour la constitution semi-automatique d'ontologies, lors du développement d'annotateur sémantiques pour de la désambiguïsation ou l'étiquetage de rôle sémantique, le résumé automatique, etc.

SYGMART [lien 4] est un moteur de traitement structurel développé par Jaques Chauché. Il dispose d'un analyseur appelé SYGFRAN. Dans [Chauché 2012], Jaques

Chauché définit SYGMART comme « indécidable et [ayant] la puissance d'une machine de Turing ». Le traitement est une suite de réécritures suivant les algorithmes de réécriture de Markov adaptés à la manipulation d'objets plus complexes que de simples chaînes. En effet, les objets manipulés par SYGMART sont des « éléments structurés ». Ils sont spécifiés par la grammaire structurelle, le formalisme spécifique à SYGMART.

La grammaire structurelle est faite d'un réseau conditionnel de grammaires élémentaires, une grammaire élémentaire étant une suite ordonnée de règles de transformation. On retrouve là une solution de traitement proche de celle que nous allons adopter.

Leopar [lien 5] est l'analyseur de l'équipe du projet Sémagramme des laboratoires de LORIA et INRIA. Il produit en sortie des analyses en dépendances aussi bien que des arbres de constituants. Guy Perrier explique dans [Perrier 2009] que le choix du cadre formel pour le développement de Leopar a été fait selon trois critères : « il [devait] être suffisamment expressif pour représenter les généralisations linguistiques, facilement lisible par des linguistes et traitable automatiquement ». Soulignons ici l'importance de ces critères que nous reprendrons lorsque nous devrons établir notre propre formalisme.

Pour répondre à ces trois critères, les chercheurs de Sémagramme se sont tournés vers les grammaires interactives ou GI, formalisme récent reposant sur la notion de polarité et dérivé des grammaires d'unification polarisées (PUG) définies par Sylvain Kahane. Dans [Kahane, 2004], il explique que la phrase peut être considérée « comme une molécule dont les mots seraient les atomes, chaque mot étant doté d'une certaine valence (terme explicitement emprunté par la linguistique à la chimie) qui l'oblige ou lui permet de s'assembler à un certain nombre d'autres mots ». En effet, chaque élément peut être négatif, positif ou neutre et on peut les combiner pour obtenir une représentation de la phrase neutre et saturée. Lorsqu'on dit qu'une phrase est « saturée », cela signifie qu'elle est complète et valide. Pour être valides, les phrases doivent être formées de groupes complets. Par exemple, un verbe conjugué (hors impératif) devra être avec son sujet. Grâce à la polarité, on peut traiter cette question de la validité des phrases et de l'exhaustivité des arguments des groupes qui la composent, directement dans le formalisme. On combine les éléments entre eux jusqu'à obtenir une structure saturée.

Des exemples d'analyseurs robustes peuvent se trouver du côté de l'analyseur de Standford ou encore de MaltParser, développé dans les universités d'Uppsala et de Växjö en Suède par Johan Hall, Jens Nilsson et Joakim Nivre. Pour ce dernier en particulier, l'analyse d'une phrase se fait par apprentissage à partir d'une banque de données d'arbres. Mais ces approches statistiques, bien que robustes et possédant leurs qualités propres, ne sont pas notre propos ici. En effet, le système dans le cadre duquel se situent nos recherches, est un système à base de règles.

Tous les analyseurs cités ici produisent des analyses soit en dépendance, soit en constituants. Nous verrons que le fonctionnement des systèmes développés sous Ariane dépend de l'utilisation d'une approche qui superpose à la fois analyse en constituants et analyse en dépendance. Ce formalisme, conçu par le GETA, repose sur une représentation linguistique : celle de la structure multiniveaux.

3. Méthodologie et formalisme linguistiques du GETA

3.1. Héritage linguistique

Le formalisme utilisé par le GETA repose principalement sur trois théories linguistiques que nous allons passer en revue.

La première est la théorie standard de Chomsky et en particulier sa théorisation de l'analyse en constituants. Le but de cette grammaire est de pouvoir rendre compte de toutes les phrases grammaticalement correctes d'une langue et seulement de ces phrases [Fuchs, 96]. En ce sens, sa démarche se rapproche de celle entreprise dans le cadre du formalisme que nous allons étudier, puisque nous verrons qu'il s'agira de mettre en correspondance un énoncé de langage naturel avec sa représentation arborescente, jusqu'à énumérer toutes les phrases possibles de la langue décrite.

Parmi les langages formels, Chomsky présente la grammaire de constituants, ou grammaire hors-contexte, comme la seule grammaire permettant vraiment de modéliser les structures syntaxiques de la langue. Elle se compose d'un axiome, d'un vocabulaire auxiliaire et d'un vocabulaire terminal et de règles de réécritures. Les règles étant récursives, il est possible de générer une infinité de phrases. Chomsky introduit alors la grammaire transformationnelle qui s'inscrit dans le prolongement de la grammaire de constituant et intervient dans un deuxième temps pour permettre de restreindre les possibilités de phrases engendrées afin de ne conserver que les phrases grammaticales. Cette grammaire s'inscrit dans le prolongement de la grammaire de constituants. Des règles de réécriture permettent alors de générer une structure profonde qui est ensuite transformée pour obtenir l'énoncé correct. Nous retrouverons dans nos recherches la même dichotomie entre analyse profonde et analyse de surface et les concepts de réécriture.

La deuxième théorie sur laquelle se base le GETA, et probablement celle qui l'a le plus inspirée est celle de Tesnière qui a introduit l'analyse en dépendances. En redéfinissant les

niveaux d'analyse de la langue, il re-délimite l'objet de la syntaxe et donne la sémantique comme fondement de la syntaxe lorsqu'il déclare « le structural n'a de raison d'être que dans la sémantique » [Tesnière, 59] cité par [Fuchs, 96]. Il distingue ensuite deux niveaux de syntaxe : un niveau de surface, la syntaxe statique, et un niveau profond, la syntaxe dynamique. La syntaxe statique traite des catégories des mots et Tesnière distingue bien les catégories syntagmatiques de leurs fonctions syntaxiques [Fuchs 96]. Chaque catégorie pourra donc remplir une ou plusieurs fonctions au niveau de la syntaxe dynamique. Les analyses au niveau de la syntaxe dynamique sont transcrites sous forme d'une représentation binaire, que Tesnière appelle stemmas. Ils ont pour racine le verbe. Les actants, c'est-à-dire l'auteur, l'objet etc. du procès décrit par le verbe sont représentés sous le verbe au même niveau les uns avec les autres. Un verbe peut également être accompagné de circonstants, ces compléments non-obligatoires et cumulables qui servent à exprimer les circonstances de l'action (temps, lieu, etc.). La relation simple qui unit un verbe à ses compléments s'appelle la jonction. Cette analyse des phrases en dépendances sera un des niveaux d'analyse utilisés par le GETA.

Revenons plus précisément sur la question des actants et des circonstants qui accompagnent le verbe. Leur présence est décrite par la théorie de la valence. Inspirée de la valence en chimie, cette théorie permet de prédire quels éléments doivent accompagner un prédicat, c'est-à-dire un verbe chez Tesnière. Selon Christian Touratier dans [Touratier, 2005], la nature de cette théorie oscille entre syntaxique, sémantique et logique. Christian Touratier compare la théorie de la valence de Tesnière et la logique des prédicats qu'il qualifie de « vraiment parallèle », le prédicat étant l'équivalent du verbe chez Tesnière, et les arguments l'équivalents des actants. Mais il montre ensuite que les choses ne sont pas si réductrices et que la valence est avant tout un concept fondamentalement sémantique qui ne peut pas être simplement ramené à une notion logique. Il explique par exemple que là où en logique le nombre d'argument associé à un prédicat caractérise ce prédicat au point qu'omettre un des arguments change la nature des arguments, en langue, lorsqu'un locuteur parle, il est possible d'omettre un des actants. Christian Touratier ajoute ensuite que les circonstants décrits par Tesnière n'ont pas d'équivalents dans la logique des prédicats. Leur nature est donc fondamentalement sémantique et non syntaxique car leur présence n'est pas exigée par le verbe ni aucune autre composante syntaxique. Ils n'existent que pour moduler le sens du prédicat, ajouter des précisions optionnelles sur les circonstances du prédicat. Christian

Touratier modère ensuite ses propos en soulignant la « portée syntaxique » de la théorie de la valence due à l'existence de critères structuraux permettant de représenter la différence entre actants et circonstants : les actants ayant une construction exocentrique et les circonstants une construction endocentrique.

On voit donc que les frontières entre les niveaux de description syntaxique, sémantique et logique ne sont pas imperméables. Ces considérations nous permettront de mieux comprendre le niveau de description logico-sémantique du formalisme du GETA et son rôle.

Enfin, le GETA a pu trouver des sources d'inspiration chez Mel'čuk avec la théorie sens-texte qui développe et approfondit des éléments de la théorie de Tesnière. Mel'čuk distingue le plan du sens et du plan du texte, que sa théorie vise à mettre en relation. Cette mise en relation pour une langue donnée est un modèle sens-texte. À un sens correspond plusieurs formulations, plusieurs paraphrases et le sens est défini comme « l'invariant des paraphrases langagières, c'est-à-dire la seule propriété commune à tous les énoncés ayant le même sens » [Mel'čuk, 88]. Cette notion d'invariant se retrouvera dans les travaux du GETA.

Dans sa leçon inaugurale au Collège de France [Mel'čuk, 97], Mel'čuk présente la théorie sens-texte et décrit les trois caractéristiques sur lesquelles elle est fondée : (1) un modèle sens-texte est équatif, c'est-à-dire qu'il permet de dire qu'un énoncé correspond à un sens donné ; (2) le fonctionnement de la théorie sens-texte repose sur le principe de paraphrasage ; (3) un modèle sens-texte doit décrire une langue de façon globale. Ces caractéristiques se retrouvent également dans les travaux du GETA.

Enfin, Mel'čuk décompose les modèles sens-texte en plusieurs strates : description phonétique, morphologique, syntaxique et sémantique. Les trois premiers de ces niveaux se dédouble ensuite en analyse profonde et de surface. Ce sont ensuite ces différentes analyses qui seront articulées entre elle pour obtenir une représentation de l'énoncé. En effet, Mel'čuk explique que « chaque aspect identifiable d'un fait à décrire reçoit une représentation autonome, pour établir ensuite des règles de correspondance entre ces diverses représentations » [Mel'čuk, 97]. La théorie de Mel'čuk repose donc essentiellement sur un système de correspondances que l'on ne retrouve pas aussi marqué dans les travaux du GETA.

Nous noterons également la présence d'éléments de logique aristotélicienne modernisée par le GETA. Avec les trois théories décrites plus haut, nous avons esquissé une description de l'héritage linguistique sur lequel se fonde les recherches du GETA. Cette description qui tente de faire le lien entre les théories linguistiques connues et les travaux du GETA est originale et est le fruit d'une réflexion élaborée à partir de la lecture des documents traitant d'Ariane. Elle pourrait présenter des inexactitudes ou des lacunes concernant le lien avec les travaux du GETA. Cette description nous permettra néanmoins d'aborder plus facilement et de mieux comprendre le formalisme mis au point dans ses murs.

3.2. La structure multiniveaux

3.2.1. Qu'est-ce que la structure multiniveaux ?

Le GETA a repris les quatre théories évoquées plus haut pour obtenir sa représentation linguistique propre, une représentation ainsi adaptée aux besoins du laboratoire et implémentable dans le cadre du système Ariane. Cette représentation, appelée structure multiniveaux, utilise la théorie de Chomsky d'analyse en constituants immédiats et l'analyse en dépendance de Tesnière, auxquelles ont été superposées une structure inspirée de la logique aristotélicienne et des éléments de la théorie sens-texte de Mel'čuk.

La vocation de la structure multiniveaux est de permettre de représenter à la fois le sens et la forme d'un texte avec la plus grande précision possible. À cette fin, la description de l'énoncé se fait selon quatre aspects, autrement appelés niveaux de description :

- **regroupement en classes syntagmatiques :** ce niveau correspond aussi à ce que Nicolas Nédobejkine appelait : classe morphologique avec ses dépendances [Chappuy, 2003]. Les éléments de la phrases sont ainsi regroupés et hiérarchisés en fonction de leur nature (identification des noms, verbes, adjectifs entre autres et constitution des différents groupes nominaux, etc.).

- **relations syntaxiques :** il s'agit d'un deuxième type d'informations contenues dans la structure multiniveaux. Généralement appelé niveau des fonctions syntaxiques et proche des stemmas de Tesnière, ce niveau décrit le rôle des groupes syntagmatiques dans la syntaxe de la phrase. En grammaire classique ce niveau correspond aux fonctions sujet, complément d'objet etc. Ces fonctions sont reprises et accompagnées d'autres informations propres à la structure multiniveaux, informations qui permettent de compléter la description du comportement syntaxique de certaines catégories. Par exemple, les verbes peuvent porter des informations sur l'auxiliaire utilisé pour construire le passé composé.
- **relations logiques :** les relations logiques permettent d'articuler entre eux certains des éléments de la phrase, tels que décrits plus haut. On définit donc ainsi la structure argumentaire de chaque élément de la phrase analysée. On retrouve ici des éléments de la théorie de la valence de Tesnière et des éléments de logique aristotélicienne, dans la mesure décrite dans la partie précédente.
- **relations sémantiques :** en dernier lieu, ce niveau de description permet de donner les informations concernant le sens des relations établies entre différents éléments de la phrase décrite : indications sur la cause, la matière, la partie etc.

Les deux premiers niveaux concernent la surface du texte. Grâce à ces deux niveaux, la structure multiniveaux peut refléter la formulation du texte, les choix stylistiques de l'auteur, l'ordre des éléments dans la phrase, etc. On garde ainsi une trace de la formulation choisie pour un énoncé [Vauquois, 78].

Les deux autres niveaux sont ceux de la description profonde du texte. Ils sont largement indépendants de la formulation des phrases étudiées. On cherche, avec ces deux niveaux, à décrire le sens profond de la phrase. Il peut y avoir plusieurs façons d'exprimer un même sens, plusieurs paraphrases possibles pour une même idée ainsi que la possibilité d'exprimer la même idée et de dire la même chose dans différentes langues. L'intérêt de ces deux niveaux de description est que quelle soit la phrase analysée, on peut calculer les mêmes informations concernant son sens pour plusieurs paraphrases différentes. Ceci sera utile pour la phase de transfert d'une langue à l'autre, où les niveaux de description du fond pourront constituer un point de repère. Ces deux niveaux fonctionnent ensemble et sont liés, au point

qu'on les considère plus souvent comme un seul et même niveau, le niveau logico-sémantique, décrit dans [Vauquois, 80].

Ces quatre niveaux de représentation de la forme et du fond des énoncés se superposent dans une arborescence unique pour former la structure multiniveaux grâce aux variables portées pas les nœuds..

3.2.2. Description formelle et exemples

Prenons un exemple plus concret et tâchons de définir la structure multiniveaux de la phrase simple « Il donne un livre à Marie ». Pour construire cette structure multiniveaux, nous aurons besoin de « décorations », c'est-à-dire d'un certain nombre de variables qui ont chacune un certain nombre de valeurs possibles. Les variables développées ci-dessous à titre d'exemple sont les variables utilisées par le GETA pour la structure multiniveaux du français. Elles ont en grande partie été reprises pour les descriptions d'autres langues dans le cadre d'Ariane. Mais elle résultent d'un choix de linguiste.

Le premier niveau de description est celui des classes syntagmatiques. On les décrit à l'aide d'un arbre de constituants. Pour constituer l'arbre correspondant à notre phrase, nous aurons besoin de deux variables, K et CAT. K (classes syntagmatiques) sert à exprimer la nature d'un groupe syntagmatique et CAT la catégorie syntagmatique d'un élément. K, dans notre exemple, pourra avoir pour valeur :

- **GN :** désigne un groupe nominal.
- **PVB :** désigne une proposition verbale conjuguée.

CAT pourra avoir comme valeur :

- **N :** désigne un nom. Un nom peut avoir différentes sous-catégories :
  - **NP :** désigne un nom propre.
  - **NC :** désigne un nom commun.

On écrit alors ici N/NC ou N/NP. Le nom peut ensuite être précisé par d'autres variables comme :

  - **NB :** renseigne le nombre du nom. Il peut être SG (singulier) ou PL (pluriel).

- **GNR :** renseigne le genre du nom. Il peut être MASC (masculin) ou FEM(féminin).

- **V:** désigne un verbe. Un verbe peut avoir différentes sous-catégorie comme dans notre exemple en particulier :
  - **VF :** désigne un verbe conjugué.

Lorsqu'un verbe est conjugué, il peut alors porter d'autres variables pour préciser son sens :

- **NB :** comme pour un nom, renseigne le nombre.
- **PERS :** renseigne la personne à laquelle est conjugué le verbe. Peut prendre les valeurs 1, 2 ou 3.
- **MODE :** renseigne le mode du verbe (IND pour indicatif, SUBJ pour subjonctif etc.).
- **TEMPS :** renseigne le temps du verbe (PRES pour présent etc.).

- **R :** désigne un représentant, plus souvent appelé un pronom. Il est précisé par la décoration :
  - **NB :** comme pour un verbe ou un nom, renseigne le nombre.
  - **PERS :** comme pour un verbe, renseigne la personne référencée par le pronom.

- **S :** désigne un subordonnant. Ceci rassemble entre autres les prépositions et conjonctions de subordination.

Chaque subordonnant régit un type de groupe syntaxique qui lui est propre. Ce groupe est précisé par la décoration :

- **KREG :** il peut, comme en l'occurrence, régir un GN (groupe nominal).

- **D :** désigne un déicteur (par exemple, articles et adjectifs possessifs, démonstratifs). Comme le nom, il peut-être précisé par les décorations :
  - **NB :** comme pour un verbe ou un nom, renseigne le nombre.
  - **GNR :** comme pour un nom, renseigne le genre.

On construit ainsi la structure suivante :

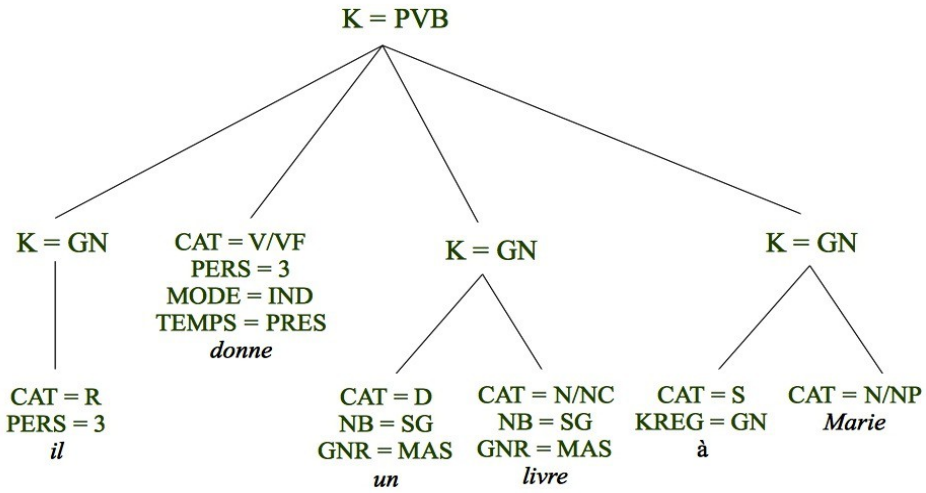


Fig. 5 : niveau syntaxique de la structure multiniveaux

On reconnait bien la structure de constituants selon Tesnière de la phrase « il donne un livre à Marie », le syntagme verbal est composé d'un verbe et de trois groupes nominaux. On note ici le choix opéré par le GETA de ne pas utiliser de syntagmes prépositionnels dans leur description, le syntagme « à Marie » est alors compté comme groupe nominal.

Le deuxième niveau est celui des fonctions syntaxiques, FS. On représente ce niveau sous forme d'un graphe, où les fonctions sont exprimées au niveau des arcs. Pour ce niveau, ces variables seront nécessaires :

- **GOV :** signale le gouverneur du groupe syntaxique, encore appelé régisseur. Nous verrons par la suite que la représentation retenue pour la structure multiniveaux complète est celle d'un arbre au sein duquel les étiquettes sont donc portées par les nœuds. La difficulté de cette représentation finale sera de représenter les arcs porteurs d'informations au sein de l'arbre. À cette fin, la variable GOV permet de créer un point de repère et toutes les relations décrites au sein d'un groupe seront orientées depuis le GOV. Il ne peut donc y avoir qu'un GOV par groupe. Il est le point de départ de toutes les flèches des graphes. Les éléments des groupes vers lesquels pointent les flèches pourront porter les variables suivantes :
- **SUBJ :** désigne le sujet de la phrase.

- **OBJD :** désigne l'objet direct.
- **OBJI :** désigne l'objet indirect.
- **DES :** fonction de désignation. Ceci signifie que le mot sert à désigner quelque chose. Ici « un » précise « livre ».
- **REG :** exprime que l'élément d'où part la flèche est régi par l'élément vers lequel pointe la flèche. En l'occurrence, « Marie » dépend de la préposition « à ».

Le niveau des fonction syntaxiques est représenté par le graphe suivant :

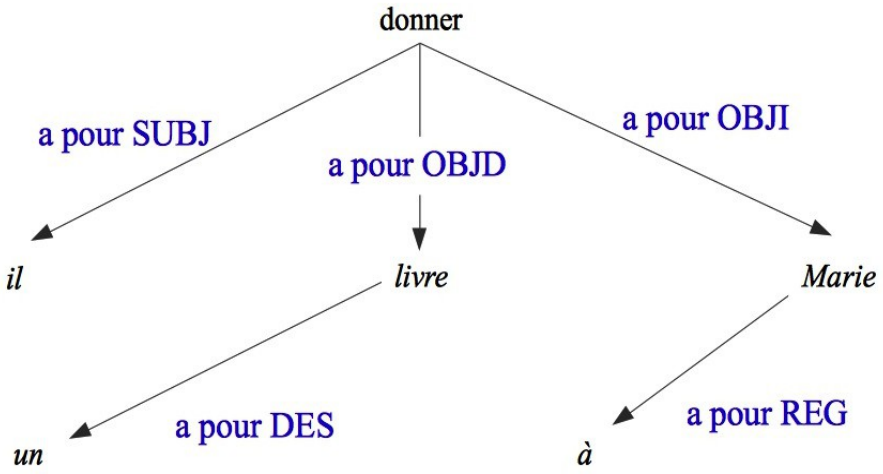


Fig. 6 : niveau des fonctions syntaxiques de la structure multiniveaux

Le troisième niveau touche à la description logique de la phrase. Appelé RL, il est celui des relations logiques. Il permet d'identifier le prédicat, ses actants et ses circonstants. Deux variables ont ce rôle :

- **GOV :** correspond au GOV du niveau des fonctions syntaxiques. Les éléments rattachés au GOV par une relation pourront porter une variable ARG.
- **ARGx :** les différents arguments de la phrase sont donc numérotés de ARG0 à ARGn en fonction du nombre d'arguments attendus par le prédicat. ARG0 correspond à l'auteur de l'action décrite par le prédicat.

Ces deux variables sont complétées par deux autres, portées par le prédicat, qui servent à

préciser la valence et la sémantique des arguments attendus.

- **VALn :** des valences peuvent être associés aux arguments. Dans notre exemple, en l'occurrence, ARG2, le troisième argument du verbe « donner » doit être un groupe nominal introduit par « à » (car on dit « donner quelque chose à quelqu'un). VAL2, la valence correspondante est donc AN (« à » + nom).
- **SEMk :** des sémantiques sont également associées aux arguments. Pour reprendre le cas du troisième argument de notre exemple, non seulement il doit être introduit par « à » comme l'indique la valence, mais il doit aussi avoir une sémantique d'animé. On utilise la décoration SEM2 pour décrire la sémantique de l'argument ARG2.

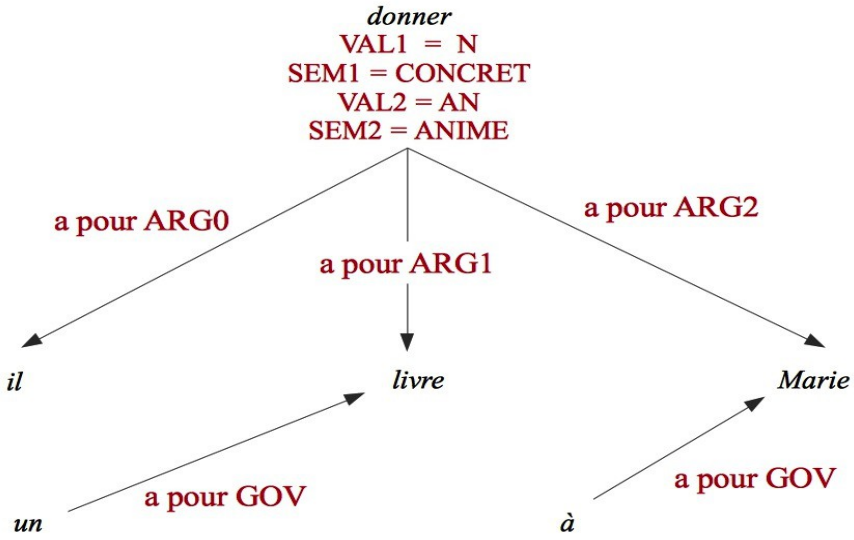


Fig. 7 : niveau des relations logiques de la structure multiniveaux

Ce troisième niveau, combiné au précédent sert par exemple à traiter les cas comme celui du passif : « le chat mange la souris » et « la souris est mangée par le chat » auront la même représentation au niveau logique, « chat » sera argument 0 et « souris » argument 1, mais la représentation des relations syntaxiques permettra de différencier le passif, où la souris est sujet, de l'actif, où elle est objet. On remarque la présence de « un » et « à » sur le

graphe. Ces deux éléments ne devraient pas apparaître dans une description logique classique, puisqu'il ne sont pas en eux-mêmes des arguments du verbe « donner ». Leur présence est liée à la création d'une variable SAUV au niveau logique, introduite par le GETA afin de conserver la trace de ces unités lexicales au niveau logico-sémantique. Ceci présentera son intérêt à la fin de l'étape d'analyse, puisque seul le niveau logico-sémantique est utilisé en génération pour construire l'énoncé en langue cible. Garder une trace de ces unités lexicales permet d'assurer la robustesse du système.

Enfin, le quatrième niveau de notre exemple, celui des RS, les relations sémantiques. Nous les distinguons ici pour la clarté de l'exemple, mais rappelons, comme nous l'avons dit plus haut, que ce niveau est généralement combiné avec le précédent. On parle alors de niveau logico-sémantique.

Ce niveau des relations sémantiques regroupe deux types d'informations dans le cadre de la structure multiniveaux du français. D'abord, des informations qui fonctionnent comme des relations logiques. Nous n'en avons qu'une occurrence dans notre exemple :

- **BENEF :** désigne le bénéficiaire de l'action. Ici, la personne à qui le livre est donné.

Puis des informations appelées « potentialité de RS ». Ces informations ont un double usage : elles sont un intermédiaire qui permet de calculer les relations logico-sémantiques. Prenons le cas du groupe nominal « un livre » dans notre exemple. Les informations sémantiques liées à « livre » sont déjà programmées, enregistrées dans le dictionnaire qui permet l'analyse morphologique de la phrase. Lorsque l'analyse structurale débute, on a donc déjà des informations sémantiques associées aux éléments terminaux. Mais les relations logico-sémantiques doivent se faire au niveau au dessus, au niveau du groupe nominal. On calcule donc une potentialité de RS pour « un livre » qui ici correspond aux mêmes informations que celles associées à « livre » : c'est un concret et plus précisément un objet matériel. Le deuxième usage des potentialités de RS est celui de la désambiguïsation, lorsque le dictionnaire d'analyse morphologique indique plusieurs sémantiques possibles pour un même nom par exemple. On vérifie si l'une des valeurs sémantiques attribuées pour ce nom dans le dictionnaire morphologique correspond avec la valeur sémantique attendue au vue du rôle logique qui lui a été attribué. Par exemple si la valeur sémantique attendue pour l'argument 0 du prédicat est d'être animé, on en déduit que le nom qui est l'argument 0 devra avoir une

sémantique d'animé pour que la relation argumentaire soit réalisée. Des hypothèses de sémantique peuvent ainsi être faites sur un mot, elles sont donc contenues dans la variable POTRS.

Enfin, des variables permettent aussi d'indiquer la valeur sémantique des éléments du texte, afin de calculer les relations sémantiques :

- **SEMx :** permet d'exprimer la sémantique associée à un mot. Ici pour exprimer la sémantique associée à un nom, la décoration appropriée est SEMN.
- **SSx :** permet de préciser encore la sémantique énoncée plus haut.Par exemple ici pour une sémantique CONCRET, on peut préciser que le SOUSCONCRET est un OBJET.

On obtient ainsi le graphe suivant pour notre exemple :

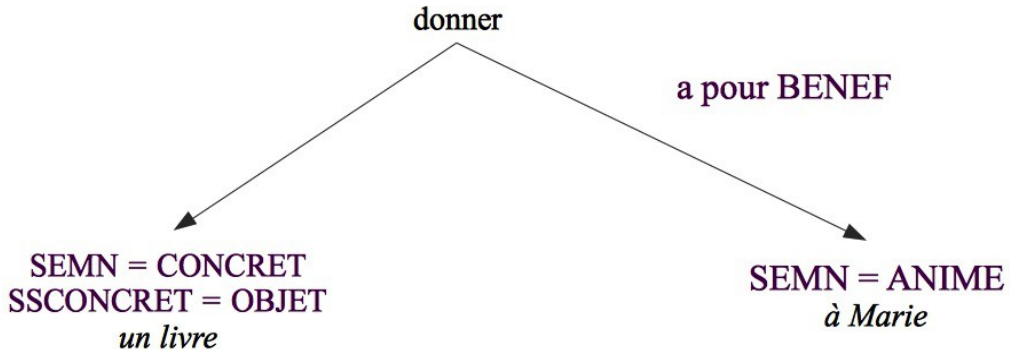


Fig. 8 : niveau des relations sémantiques de la structure multiniveaux

En rassemblant les quatre niveaux de description détaillés plus haut, on obtient donc la structure multiniveaux suivante :

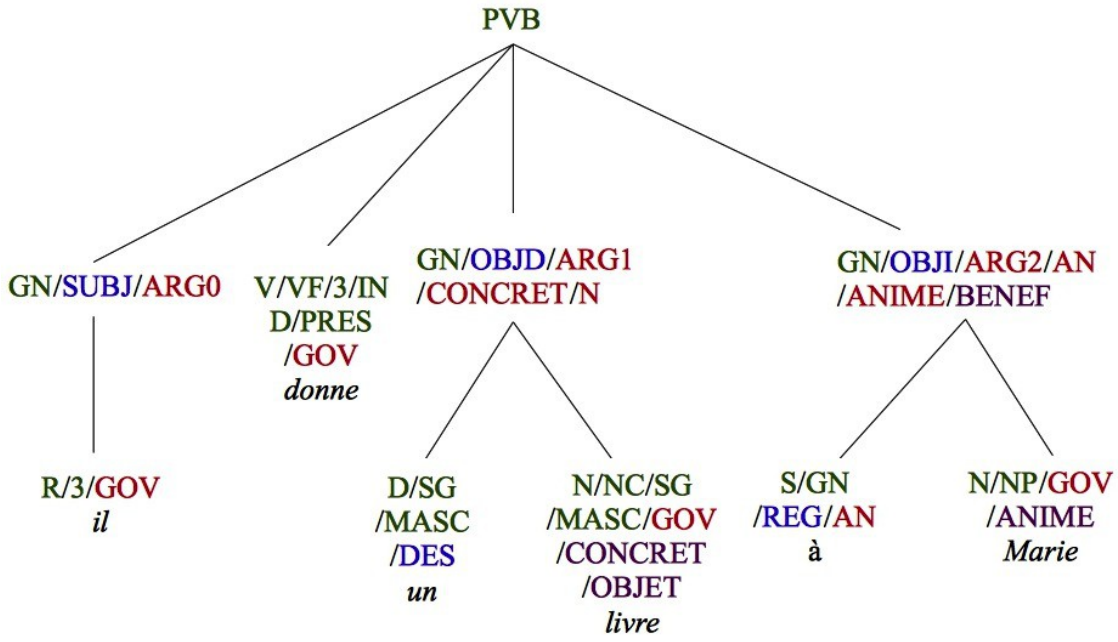


Fig. 9 : structure multiniveaux de « il donne un livre à Marie »

On reconnait bien les quatre niveaux décrits plus haut : niveau des classes syntagmatiques en vert, niveau des fonctions syntagmatiques en bleu, le niveau logico-sémantique en rouge et violet. La forme retenue pour représenter la structure multiniveaux est celle de l'arbre de constituants immédiats du premier niveau de description. Le gouverneur de chaque groupe syntagmatique est indiqué au niveau logico-sémantique (GOV en rouge). Il y en a bien un par fratrie, comme expliqué précédemment, il s'agit du point de départ de chaque relation décrite par les niveaux des fonctions syntaxiques et logico-sémantiques.

3.2.3. Le rôle de la structure multiniveaux dans Ariane-G5

La structure multiniveaux telle que décrite précédemment joue un rôle central dans les systèmes développés sous Ariane. Il s'agit en effet d'une représentation abstraite qui peut être associée à tout texte en langage naturel. Dans l'optique d'un système de traduction automatique, cet aspect est crucial. Il l'est en fait d'autant plus dans le cadre des systèmes développés sous Ariane où la traduction s'opère le plus souvent en trois étapes :

- Analyse du texte d'entrée et construction de la structure multiniveaux correspondante.
- Transfert de cette structure multiniveaux du texte d'entrée en sa structure image représentant le texte de sortie. Ici rappelons-nous que les deux premier niveaux de description de la structure multiniveaux correspondent à la description de surface (groupes syntagmatiques, ordre des éléments dans la phrase et syntaxe). Ces deux niveaux sont donc différents d'une langue à l'autre.
- Génération du texte de sortie à partir de la structure obtenue à l'issue du transfert.

Bernard Vauquois représente l'enchaînement de ces étapes par une pyramide que nous allons utiliser pour mieux illustrer l'importance de la structure multiniveaux.

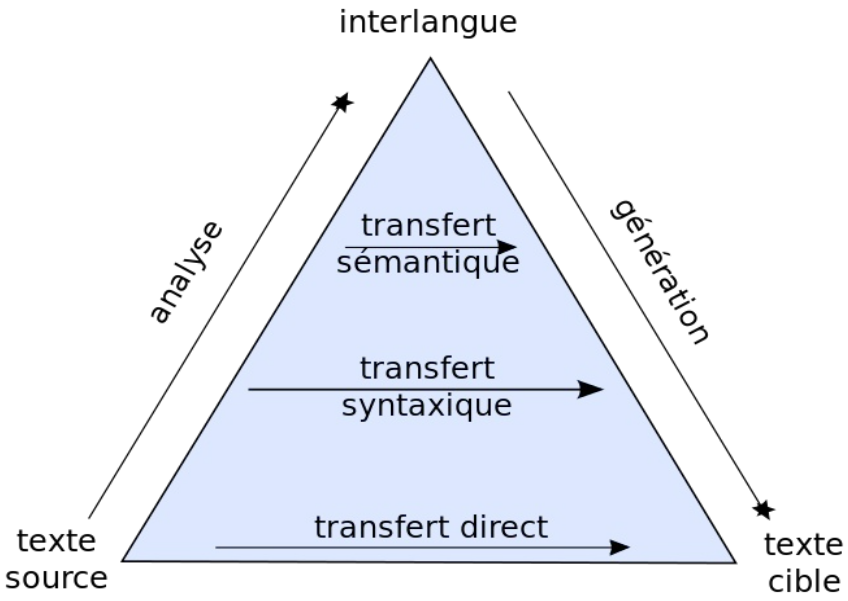


Fig. 10 : le triangle de Vauquois

Ce triangle représente les différentes étapes du processus de traduction d'un système comme celui développé sous Ariane. En bas à gauche, se trouve le texte en langue source, en bas à droite, le texte en langue cible. Traduire c'est donc traverser la pyramide de gauche à droite. Une traduction basique consisterait à faire du transfert direct, un transfert mot-à-mot

de la langue source à la langue cible. Ce travail qui est le plus simple à réaliser, est celui qui donne le résultat le moins bon. Mais il est possible de monter dans le triangle afin de réduire le travail de transfert et d'améliorer le résultat de la traduction. Pour cela, il faut analyser le texte pour en construire une représentation abstraite désambiguïsée. Il peut s'agit d'une analyse syntaxique seule, ou doublée d'une analyse sémantique de plus ou moins haut niveau. Plus l'analyse est profonde, plus le résultat de la traduction sera précis. Dans Ariane, avec la présence d'un niveau logico-sémantique dans la structure multiniveaux, on se situe un peu au dessus de la flèche « transfert sémantique » dans le schéma. Bien sûr, après l'étape de transfert, il faut prévoir une étape de génération qui sera d'autant plus longue que l'analyse a été profonde. La qualité de la traduction produite en sera théoriquement meilleure, puisque la langue générée à partir d'une représentation profonde de haut niveau est censée être plus naturelle, libérée de l'influence de la langue source.

Cet enchaînement d'analyse-transfert-génération permet au système de fonctionner avec trois étapes indépendantes. Prenons l'exemple d'un système traduisant trois langues, arbitrairement l'anglais, le français et le khmer. Sans le passage par la représentation intermédiaire que constitue la structure multiniveaux et l'étape de transfert, il faudrait créer un système complet pour chaque couple de langues (français-anglais, anglais-français, français-khmer, khmer-français, anglais-khmer, khmer-anglais). Donc six systèmes complets. Pour ajouter une seule langue il faudrait créer six nouveaux systèmes, et huit pour en rajouter une de plus etc. Tandis que la décomposition des étapes et l'indépendance des modules entre eux permettent de faciliter l'ajout d'une nouvelle langue dans le système. Il suffit de créer une analyse et une génération qui seront toujours valables pour la langue concernée et de réutiliser les analyses et générations déjà existantes d'autres langues. Il ne reste ensuite qu'à créer les étapes de transfert, ce qui représente une quantité de travail moindre. Grâce au passage par une représentation intermédiaire du texte, tous les modules d'analyse et de génération sont donc indépendants et ré-utilisables, ce qui permet d'optimiser le travail sur le développement du système.

Une fois qu'une analyse et une génération de profondes ont été produites, l'étape suivante pour réduire au maximum l'étape de transfert, consisterait à passer par un langage pivot, un langage ou même un formalisme commun à toutes les langues. UNL est un exemple de formalisme qui se donne cette vocation-là. Il s'agit d'un langage formel permettant de coder la représentation du sens d'un énoncé. Fondé en 1996 à l'université des Nations Unies

de Tokyo, ce formalisme a été créé afin de permettre des faire une représentation manipulable informatiquement du sens d'un énoncé donné. Ce passage par un langage pivot est symbolisé par le sommet de la pyramide de Vauquois.

Mais la structure multiniveaux d'Ariane présente deux avantages qui font la force de ce système : (1) la notion de filet de sécurité qui fait du système sur lequel nous travaillons un système robuste et (2) la notion de paraphrasage qui permet la génération d'une langue plus naturelle.

Le filet de sécurité représente en fait les deux premiers niveaux de description de la structure multiniveaux : la niveau des catégories et groupes syntagmatiques et celui des fonctions syntaxiques. Si la traduction échoue parce que l'un des termes à traduire est absent dans le dictionnaire, la génération peut alors prendre appui sur les deux niveaux de description de surface évoqués plus haut pour proposer une traduction pour les groupes correctement analysés. Les mots traduits seront alors correctement rassemblés au sein du même groupe, accordés entre eux, etc. On aura toujours un résultat avec le mot de la langue source laissé comme tel à sa place dans la phrase et mis entre chevrons. On obtient ainsi toujours un résultat, ce qui permet de qualifier le système sur lequel nous travaillons de robuste [Nedobejkine, 80].

Le paraphrasage quant à lui repose en partie sur la notion de dérivation qui permet de rapprocher les dérivés d'une même unité lexicale. Par exemple, « prendre » et « prise » pourront être rassemblés sous l'unité lexicale commune « prendre ». L'information principale est celle concernant le prédicat, il s'agit du prédicat « prendre ». On pourra donc identifier les arguments correspondants etc. Grâce à cela, il devient possible de générer plusieurs paraphrases pour un même sens. Par exemple, la structure logique suivante peut se réaliser de différentes manières, avec un prédicat verbal ou nominal.

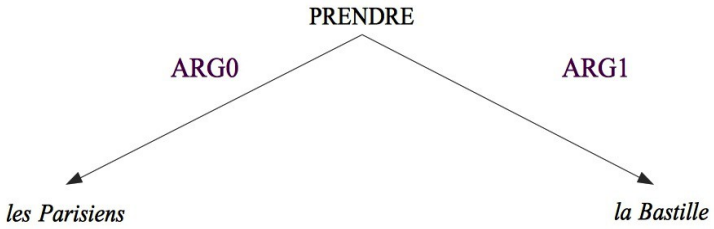


Fig. 11 : structure logique du verbe « prendre »

Il y a donc plusieurs manières de formuler le sens décrit par ce graphe, et c'est le contexte qui va permettre de choisir une paraphrase. Le segment décrit peut par exemple apparaître au sein de la phrase « En 1789 les Parisiens ont pris la Bastille », auquel cas PRENDRE se traduit par un verbe, mais il peut aussi apparaître au sein de la phrase « Le 14 juillet, on fête la prise de la Bastille par les Parisiens », auquel cas on pourra choisir de traduire le prédicat par un nom. La choix se fait donc grâce au contexte, et ce choix a lieu dans un processus de traduction automatique avec Ariane, lors du transfert, au cours duquel des « prédictions » de génération sont faites.

Le transfert qui consiste à passer de la structure multiniveaux décrivant le texte source à une structure décrivant le texte cible, se découpe en deux étapes :

- **Le transfert lexical** est l'étape où chaque UL est traduite et donnée avec toutes ses potentialités de dérivation. Pour reprendre l'exemple donné plus haut et dans le cadre d'une traduction français-anglais, « donner » sera traduit par « give » et à ce verbe seront rattachées les potentialités de dérivation**.**
- **Le transfert structural** est l'étape où les phénomènes syntaxiques propres à une langue sont transformés pour correspondre à ceux de la langue cible. C'est à cette étape également que sont faites les prédictions de générations en fonction du contexte.

Le transfert se veut donc comme une étape minimale, où les caractéristiques particulières à chaque langue sont identifiées et transformées dans une approche contrastive. Le texte en langue cible est ensuite généré à partir de la structure profonde de la description obtenue

[Guibauld, 87].

Pour résumer, le paraphrasage est le fait de pouvoir formuler un texte de façons différentes. Dans Ariane, c'est ce qui permet, lors de l'étape de génération, de trouver la formulation la mieux adaptée au contexte. Sylviane Chappuy déclare à ce propos : « Cette capacité de paraphrasage portée par le choix de la m-structure est ce qui donne leur puissance aux systèmes développés sous Ariane » [Chappuy, 2013]. C'est en effet le concept de structure multiniveaux (ou m-structure) décrit plus haut, et en particulier la présence d'un niveau logico-sémantique, qui permet le paraphrasage. L'autre force du passage par la structure multiniveaux, comme nous l'avons vu, est qu'elle permet aux systèmes développés sous Ariane d'être robustes. On peut alors mieux comprendre l'enjeu de notre recherche qui consiste à trouver un formalisme qui permette de décrire les règles faisant le lien entre un texte dans une langue donnée et sa représentation en structure multiniveaux. À cette fin, nous commencerons par étudier le formalisme utilisé par le GETA et le GETALP jusqu'à aujourd'hui.

3.3. Les grammaires statiques, le formalisme du GETA

3.3.1. Description générale des grammaires statiques

Les grammaires statiques sont un formalisme imaginé par Bernard Vauquois et développé par Sylviane Chappuy dans sa thèse soutenue en 1983. Elles sont également appelées GSCS depuis une révision plus récente sur laquelle nous basons nos travaux. Pensées à l'origine comme une spécification du code ROBRA pour les analyses et les générations sous Ariane, elles constituent pour autant de véritables outils de description linguistique. Leur fonctionnement repose sur un système de correspondances, chaque règle établissant une correspondance entre une chaîne donnée et une structure multiniveaux, ou plus précisément une famille de chaînes et une famille de structure multiniveaux. En effet, afin d'obtenir une description réellement linguistique, il faut que les traits de langage généraux soient reconnaissables dans les règles. Une règle peut par exemple décrire les groupes nominaux composés d'un déterminant suivi d'un nom et établir la correspondance

entre les structures multiniveaux et les chaînes correspondant à cette description.

Nous remarquerons ici que les grammaires statiques ont été créées pour le développement de système sous Ariane et que par conséquent, elles se placent dans le contexte de la méthodologie linguistique du GETA avec la structure multiniveaux. Mais ce formalisme pourrait aussi bien être utilisé avec toute autre méthodologie linguistique associant des énoncés à des arbres, comme les syntagmes de Mel'čuk.

Le terme de grammaire statique a été choisi par le GETA par opposition aux « grammaires dynamiques », nom donné aux programmes d'analyse et génération rédigés en ROBRA. Par comparaison, les grammaires statiques sont une spécification linguistique et non un programme effectuant des transformations. Leur nom leur a donc été donné en fonction de leur rôle et il ne faut pas y voir de rapprochement avec d'autres théories linguistiques du même nom, comme chez Tesnière où le terme de « grammaire statique » désigne les constituants de surface de la phrase. Mais voyons à présent plus concrètement en quoi consiste une grammaire statique.

3.3.2. Description formelle des grammaires statiques

Une grammaire statique se compose de plusieurs planches, c'est-à-dire des règles, organisées entre elles en réseaux car elles ont la possibilité de faire référence les unes aux autres, c'est-à-dire de s'appeler. Elles sont organisées selon deux critères : elles sont d'abord regroupées en fonction du groupe syntagmatique qu'elles décrivent (groupe nominal, groupe verbal etc.). Les différents groupes syntagmatiques sont également hiérarchisés entre eux, cette hiérarchie résultant du choix du linguiste qui décrit sa langue. Par exemple, la hiérarchie adoptée pour le français est la suivante [Chappuy 83] :

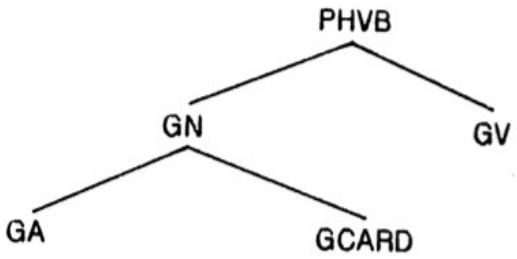


Fig. 12 : hiérarchie des groupes syntagmatiques du français

Dans cette hiérarchie les groupes adjectivaux (GA) et cardinaux (GCARD) sont plus bas dans la hiérarchie, dominés par les groupes nominaux (GN) qui sont au même niveau que les groupes verbaux (GV). Le niveau le plus haut classé est celui des propositions verbales conjuguées (PHVB). Cette hiérarchie est utilisée pour classer les planches en fonction de leur niveau de complexité. On compte trois niveaux de complexité [travail collectif, 83] :

- **les planches élémentaires** décrivent les groupes syntagmatiques minimaux, ceux qui vont ensuite servir de base au linguiste pour construire des groupes plus complexes.
- **les planches simples** décrivent des groupes en se reposant sur des groupes de hiérarchie plus faible. Pour reprendre l'exemple du français, les groupes nominaux simples ne peuvent être composé que d'éléments terminaux (noms, déterminants etc.) et de groupes adjectivaux ou cardinaux. Ex : « Les trois petits chats ».
- **les planches complexes** décrivent des groupes qui peuvent être composés de n'importe quels éléments quelle soit leur place dans la hiérarchie.

Les planches statiques sont donc reliées entre elles et ordonnées pour constituer une grammaire. Tâchons maintenant de voir comment est constituée une planche en prenant comme exemple la planche GN8c en annexe 1. Cette planche tirée d'une grammaire du français écrite en 1990 (FR3), décrit les groupes nominaux du type de « un tas de feuille ».

Rappelons qu'une planche décrit une correspondance entre une chaîne et une structure

multiniveaux. Elle se divise en différentes zones. Les trois dernières zones expriment des conditions. Si toutes les conditions se réalisent, la correspondance est valide.

- La **ZONE I**, encore appelée ZGRAPH dans [Chappuy 2013], permet d'avoir une vue d'ensemble sur la correspondance décrite par la planche. Elle se divise en deux parties : une représentation parenthésée de la structure multiniveaux suivie d'une représentation graphique. La représentation parenthésée ne contient que les identifiants des nœuds, les décorations qui lui sont associées seront exprimées dans les zones suivantes.

  Dans l'arbre, un type de nœud spécial pourra attirer notre attention. Il s'agit des nœuds de contexte. Encadrés de trois points d'exclamation, ils représentent un ensemble de nœuds ou d'arborescences. Ils apparaissent pour délimiter les références. Dans la planche GN8c, le nœud 0 est une référence à un groupe nominal déjà décrit dans d'autres planches. Ce groupe nominal peut être simple, comme un déterminant suivi d'un nom (« le tas »), être plus complexe et comporter d'autres éléments (« le gros tas »). Ce qui importe pour la règle de la planche GN8c est que le nom de ce groupe nominal ait une sémantique de collectif. Le contexte permet de dire qu'il peut se trouver avec ce nom divers éléments, tant que le groupe correspond bien au groupe décrit par la référence. Enfin, la présence d'un contexte gauche et droit permet de délimiter la référence, et de savoir précisément quels éléments dans notre arbre renvoient à une autre planche.

- La **ZONE IIa**, appelée zone contrainte sur la chaîne d'arbres ou ZCA dans [Chappuy 2013] est la zone où les conditions concernant la chaîne sont exprimées. En réalité, Sylviane Chappuy introduit ici une notion nouvelle avec la chaîne d'arbres. Elle est issue du constat que la chaîne manipulée dans les planches statiques n'est pas un objet simple. Elle contient aussi des références, des raccourcis d'expression. Le nœud 1 de la planche GN8c correspond par exemple à un groupe nominal. On trouve donc dans la chaîne des éléments simples comme des références à des groupes syntagmatiques qui correspondent à des arbres.

Les conditions sont exprimées de la manière suivante :

```
K(0)=GN
```

Ici la catégorie syntagmatique du nœud 0 doit être GN. Ces conditions peuvent être

combinées entre elles avec des opérateurs logiques comme dans l'exemple ci-dessous.

```
FS(G)=GOV ⋀ CAT(G)=N ⋀ SUBN(G)=NC ⋀ SEMN(G)=ABSTRAIT
    ⋀ (SSABST(G)=COLLECTA ⋁ SSABST(G)=COLLECTNA)
```

Lorsqu'il arrive que des éléments soient optionnels dans la chaîne, il est possible d'exprimer dans cette zone des conditions d'existence.

- La **ZONE IIb**, zone contrainte sur l'arbre ou ZA dans [Chappuy2013], fonctionne exactement comme la zone précédente. Elle sert à exprimer les conditions sur la structure.
- Enfin, **la ZONE III** encore appelée ZCORR dans [Chappuy 2013], « exprime les correspondances entre la chaîne d'arbres et l'arbre décrits dans les zones précédentes ». Ces correspondances sont de deux types différents :
  - les correspondances directes permettent d'affecter une variable d'une valeur à l'arbre et à la chaîne. Ici, les informations typographiques du nœud 0 doivent correspondre à l'intervalle des informations typographiques des nœuds 0 et 1.

    ```
    TYPOG(0)=$inter.typog(0,1)
    ```

  - les correspondances exprimées sous forme de réseaux permettent d'exprimer des affectations qui ne se réalisent que sous certaines conditions. Les informations sur les arcs concernent la chaîne et les informations dans les blocs concernent l'arbre. Si la condition sur la chaîne se vérifie, alors les conditions du bloc doivent se vérifier aussi. La planche donnée en annexe étant en texte brut, le réseau a été reproduit avec des caractères. Une suite de trois points d'interrogation délimite le bloc.

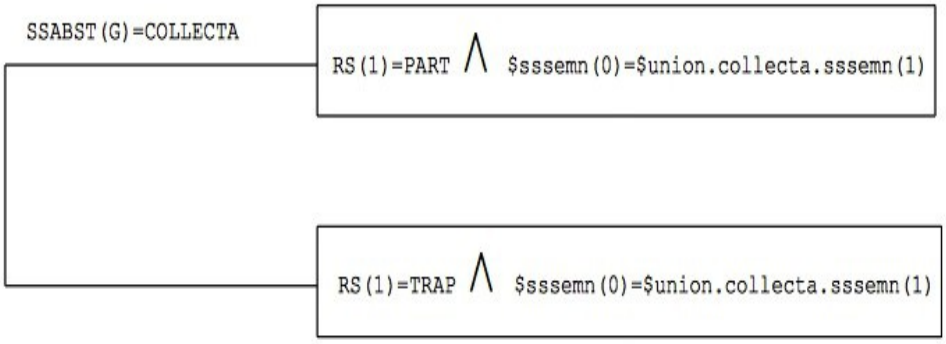


Fig. 13 : zone III de grammaire statique

En plus de ces zones de description, chaque planche statique comporte une zone d'exemples et une zone de métadonnées où sont stockées son nom, une explication sur son utilité, la liste de ses références.

Maintenant que nous avons pu donner un aperçu sur la composition des grammaires statiques, voyons à présent quel est leur intérêt.

3.3.3. Intérêt des grammaires statiques

Les grammaires statiques telles que exposées plus haut présentent plusieurs intérêts. Elles sont au croisement entre représentation linguistique et spécification informatique. En effet, tout en facilitant l'écriture et la maintenance d'analyseurs ou de générations, elles permettent d'avoir une vue d'ensemble sur les phénomènes linguistiques qu'elles décrivent [Vauquois, 85]. Les grammaires statiques permettent de faire des descriptions linguistiques complètes avec la structure multiniveaux. Ceci est possible grâce à divers facteurs.

La grande variété de conditions portées par chaque nœud de l'arbre ou élément de la chaîne permet de décrire de façon à la fois suffisamment détaillée et générique toute sorte de réalités linguistiques.

De plus, c'est grâce à la chaîne présente dans les planches statiques que le formalisme des grammaires statiques peut être qualifié de non-orienté et de bidirectionnel. En effet,

puisqu'une règle ne consiste qu'en une correspondance chaîne-arbre, cette correspondance peut être interprétée pour en déduire au choix une règle d'analyse ou une règle de génération. La description obtenue est donc bel et bien de nature linguistique, et elle permet d'identifier les principes généraux sur lesquels repose la langue décrite. Cette description linguistique sera particulièrement efficace dans le cadre de notre système de traduction, puisque le processus se découpe en trois étapes d'analyse, de transfert et de génération, donc un passage de l'énoncé en langue source, à une structure multiniveaux, à un énoncé en langue cible. La présence de la chaîne dans les grammaires statiques facilitera donc l'écriture des programmes d'analyse et de génération. Pour aller plus loin, il faudrait pouvoir générer automatiquement le code des programmes correspondants. Mais, en l'état, les grammaires statiques ne permettent pas la génération automatique de code parce qu'elles ne sont pas dans un format adéquat.

Une réponse à ce besoin peut être trouvée entre autre du côté du formalisme PATR brièvement présenté dans l'état de l'art, pour lequel un parallèle peut être fait avec les grammaires statiques. En effet, ce formalisme consiste à décrire une dérivation, suivie de conditions et des règles. Reprenons l'exemple développé en état de l'art, pour illustrer nos propos :

X0 ==> X1 X2
<X0 Cat> = GN
<X1 Cat> = D
<X2 Cat> = N
<X0 Accord> = <X1 Accord> = <X2 Accord>

La transformation sur la première ligne permet de décrire un arbre. Les lignes en dessous rappellent les conditions de zone II et III. Des conditions sont exprimées pour chaque partie de la règle de réécriture, puis une règle permet de relier les différentes parties pour transmettre des information des terminaux vers le nœud racine. Trois différences avec les grammaires statiques sont à noter. D'abord PATR avec ce système de balises est manipulable automatiquement, ce qui n'est pas le cas des grammaires statiques telles qu'elles sont aujourd'hui, et qui sont au mieux des documents saisis sous word, mais sans réelle uniformité, voire la plupart du temps des documents papier. De plus, les grammaires statiques ne décrivent pas une transformation mais une correspondance. Enfin, et ceci constitue le deuxième point de différence avec les grammaires statiques, il n'y a pas de composante

graphique dans les grammaires PATR. L'arbre compris par les grammaires statiques est un point de repère important pour la personne qui lirait les planches statiques et il contribue à faire des grammaires statiques un formalisme abordable.

Nous avons posé le cadre de nos recherches, présenté la structure multiniveaux sur laquelle se fonde l'une des méthodologies de traduction automatique du GETA et présenté le formalisme créé pour rédiger la spécification de leurs programmes d'analyse et de génération. Nous avons insisté sur les enjeux de ce formalisme et ses lacunes. Voyons maintenant quelle proposition d'amélioration nous pouvons faire.

4. Proposition d’un nouveau formalisme

4.1. Rappel du double enjeu de ce formalisme

4.1.1. Un formalisme plus abordable que celui des grammaires statiques

Le but de notre recherche est, rappelons-le, de proposer un formalisme en alternative aux grammaires statiques présentées plus haut. Ce formalisme doit répondre à deux besoins essentiels : être simple d'accès et traitable automatiquement. En parallèle, il ne doit pas non plus dénaturer la force des grammaires statiques qui est de permettre de décrire le langage humain.

Un point qui peut rendre difficile la lecture des planches statiques est le fait que la représentation de la chaîne et celle de l'arbre se confondent dans la représentation graphique. De plus, les deux expressions parenthésées décrivant la chaîne et l'arbre sont simplement juxtaposées, ne disposent pas vraiment d'outils pour marquer de manière explicite la correspondance entre un élément de l'arbre et un élément de la chaîne. Nous chercherons donc un autre moyen d'exprimer les correspondances chaîne-arbre.

La répartition en zones avec description de l'arbre et de la chaîne, zone de condition sur la chaîne, zone de conditions sur l'arbre et zone de construction de nouvelles conditions en fonction des correspondances chaîne-arbre complique un peu la lecture des planches. Il devient difficile d'avoir une vue d'ensemble permettant de dégager des règles linguistiques régissant la langue. La zone I avec la représentation graphique de la planche permet seulement en partie d'avoir cette vue d'ensemble. En effet, toutes les décorations ne sont pas exprimées, celles qui correspondent aux nœuds plus haut dans l'arbre ou plus profond dans la chaîne notamment. Nous tâcherons donc de rattacher les informations au nœud auquel elles sont associées.

Nous jugeons également le nom des nœuds trop obscurs. Le fait d'utiliser des identifiants numériques n'aide pas le lecteur a obtenir une vue d'ensemble sur la planche. Nous préférerons utiliser des noms de nœuds explicites.

Enfin, une difficulté que nous n'avons pas encore résolue est causée par des planches

trop compliquées, trop longues et donc difficiles à comprendre. Prenons l'exemple de la planche GN5, tirée de la même grammaire du français que la planche expliquée plus haut. Cette planche tient sur onze pages, son arbre est composé de onze nœuds dont dix sont optionnels. Ces onze nœuds peuvent donc se combiner de façons différentes pour former des dizaines de possibilités d'arbres. Des conditions d'existence sont exprimées, mais elles ne limitent pas tant les arbres possibles. Enfin, le graphe de conditions de la zone III s'étale sur six pages. Difficile ainsi de comprendre quel trait de langage est décrit, d'autant plus que dans le cadre de cette planche, différents types de groupes nominaux sont décrits, comprenant des groupes cardinaux, ordinaux, des adjectifs avant ou après le nom, des prépositions, etc. Une part de notre travail a donc consisté à réfléchir à la manière de simplifier de telles planches. Notre formalisme doit pouvoir en offrir la possibilité.

Telles sont donc les contraintes linguistiques auxquelles nous souhaitons répondre, voyons maintenant quelles sont les contraintes formelles.

4.1.2. Permettre la génération de code ROBRA

Commençons par expliquer brièvement en quoi consiste un programme d'analyse ou de génération dans un système développé sous Ariane : un tel programme est écrit en ROBRA (« arbre » à l'envers), qui est un langage créé par Christian Boitet, Pierre Guillaume et Maurice Quezel-Ambrunaz en 1976 [Travail collectif, ROBRA]. Ce langage est une extension du langage CETA conçu par Jacques Chauché dans le cadre de sa thèse d'état. Il permet d'effectuer des transformations d'arbres décorés. Un programme ROBRA doit contenir un fichier de variables et un fichier de formats sur lesquelles nous ne ferons pas de commentaires et un ou plusieurs fichiers de grammaire. Le fichier de grammaire comporte à son tour plusieurs éléments listés en détail dans [Travail collectif, ROBRA] :

- des **procédures** de différentes sortes. Les procédures sont des éléments prédéfinis, des conditions et des affectations qui seront réutilisés dans les règles de grammaire. Elles seront alors préfixées du symbole $.
- un **graphe de contrôle**, aussi appelé « grammaire » qui indique dans quel ordre et sous quelles conditions appeler les différents ensembles de règles
- les **règles** qui réalisent des transformations sur les arborescences décorés.

Prenons un exemple de règle pour mieux comprendre comment celles-ci sont faites. Nous prendrons la règle d'analyse déduite de la planche présentée plus haut, la GN8c. Cette règle permet de construire les groupes nominaux du type GN de GN où le premier GN a une sémantique de collectif (« un tas de feuilles »).

```
COLLECT4: (*0, &NIV=1)
              0(P($O,G,$L),*,F(S,$M,N),*,C) /** S 05/12/85.
          P: $GNC;
          G: $GOV -ET- $ABSTCOL;
          F: $ULETGN -ET- $VLCDEN;
          S:$S -OU- $DELEDES;
          N: $GOV -ET- -N-$MOTCOMP1;
          C:-N-$ULETGN -OU- $VLN -OU- $RREL
       ==0(P($O,G,$L,F(S,$M,N)),C) /  /
          F:F, FS:=COMP, VAL1:=DEN,
            -SI- SSANIME(G)-INC-ANIMAL
            -ALORS- RS(F):=PART
            -SINON- RS(F):=TRAP
            -FSI-;
          G:G,$AFSEMN(N);
          P:P,$AFSEMN(N),
            -SI- UL(P)-E-'*GN'
            -ALORS- UL(P):='*GNCOL'
            -SNSI- UL(P)-E-'/GN'
            -ALORS- UL(P):='/GNCOL'
            -SNFSI-;
          C:C,-SI- UL(C)-E-'/GN'
              -ALORS- UL(C):='*GN'
              -SNFSI-.
```

La première ligne contient le nom de la règle puis une expression parenthésée qui permet de situer la règle où et comment s'applique la règle. *0 signifie que la transformation s'applique sur le nœud 0 et qu'il peut participer à plusieurs transformations en parallèle. &NIV=1 signifie que la racine 0 est à une profondeur de 1 sous l'arbre transformé par la règle. Ceci a son intérêt car les programmes développés dans le cadre du système que nous étudions

produisent une analyse au niveau du texte, ce qui permet de réaliser de la désambiguïsation d'une phrase à l'autre.

La règle se divise ensuite en deux parties, séparées par le symbole ==. La partie gauche s'appelle le schéma et concerne l'arbre d'entrée de la règle, la partie droite s'appelle l'image et concerne l'arbre de sortie. Le schéma comporte sur la première ligne la description géométrique du schéma à rechercher sous forme d'expression parenthésée, suivie des conditions à vérifier pour chaque nœud. Quand un sous-arbre correspondant au schéma est trouvé, il est transformé en l'arbre décrit dans la partie droite de la règle. Les affectations à faire pour chaque nœud sont décrites à droite du symbole d'affectation :=. On retrouve le graphe de la zone III sous forme d'affectations conditionnelles. Par exemple, « *-SI-SSANIME(G)-INC-ANIMAL »* signifie que si la liste des sous-catégories sémantiques du nœud G comprennent « ANIMAL », l'affectation qui suit le mot-clé « *-ALORS-* » sera réalisée. Le nœud F prendra une relation sémantique de partie pour le tout.

Cette description du fonctionnement d'une règle ROBRA nous permet de comprendre qu'il est théoriquement possible de réaliser une génération de règles ROBRA à partir d'un formalisme du type de celui des grammaires statiques. Mais pour cela, il faudrait bénéficier de fichiers de grammaires statiques qui soient manipulables informatiquement. Il resterait alors à articuler ces règles entre elles, c'est-à-dire à générer un graphe de contrôle. Tang Enya Kong évoque dans sa thèse [TANG 94], comment générer ce graphe de contrôle à partir du graphe d'appel des planches entre elles. Tang Enya Kong évoque cette solution pour générer du code ROBRA à partir du formalisme STCG que nous exposerons plus tard, mais cette solution est aussi applicable aux grammaires statiques de Vauquois-Chappuy car elle ne concerne que le principe d'appel des planches entre elles, qui est le même que celui des grammaires statiques. Les planches sont classées en fonction de leur références (planches élémentaires sans références, planches simples dont les références ne bouclent pas, planches cycliques dont les références bouclent en revenant sur elles-mêmes et planches complexes dont les références bouclent sur une autre planche). En fonction de ce classement et d'autres facteurs, un niveau de complexité leur est attribué de 1 à n. Le graphe de contrôle correspond à l'appel des planches niveau par niveau. On construit ainsi une grammaire où les règles les plus simples sont appliquées en premier afin de servir d'appui aux règles plus complexes. Mais tout ceci reste théorique, dans la mesure où les grammaires statiques sont écrites dans un format qui n'est en l'état pas manipulable informatiquement. En effet, elles sont, rappelons-

le, imprimées sur du papier ou saisies sous Microsoft Word, et donc conçues pour être lues. Elles se composent de dessins et de textes non analysables.

Nous voyons donc maintenant quelles sont les raisons qui nous poussent à étudier un nouveau formalisme, et ce que nous avons besoin de mettre en place. Ce constat d'insuffisance a été opéré plusieurs fois au sujet des grammaires statiques et a donné lieu a un certain nombre de proposition de formalismes dérivé des grammaires statiques. Nous allons les passer en revue, pour tenter de trouver des réponses à nos besoins.

4.2. État des recherches autour des grammaires statiques

Il existe plusieurs tentatives de réécriture des grammaires statiques. Nous allons les présenter chronologiquement pour mieux situer notre démarche.

Rémi Zajac linéarise les grammaires statiques dans ses travaux de thèse en 1986. Il définit ainsi un nouveau langage de spécification linguistique, LSCS, permettant de réécrire les grammaires statiques dans un format éliminant toute composante graphique. Le résultat est donc manipulable informatiquement, mais difficile à lire. Voici un exemple cité par Yves Lepage dans sa thèse [Lepage, 84] :

```
TYPE : GROUPE NOMINAL COMPLEXE
   ...
REFERENCE .0 : (np)
REFERENCE .1 : (np)

ARBRE  .0 ( $5,
            .2
            $6,
            .1 ( $7, .3, $8 )
          )
    CPROPRES ...

FORET  .0($5, .2, $6) , .1($7, .3, $8)
    CPROPRES ...
    CSCHEMA  ...

CTXTD  .4($9)
```

Fig. 14 : description d'un groupe nominal en LSCS

Ce formalisme ne répond donc pas au besoin d'avoir un formalisme linguistique, facilement abordable et permettant d'exprimer une description linguistique. Il n'a pas non plus vraiment donné lieu à des applications ou à des recherches plus poussées.

En 1986 Zaharin Bin Yusoff propose dans sa thèse une formalisation d'une partie des GSCS : les STCG, *String-Tree Correspondence Grammar*. Il s'agit d'une représentation à la fois formelle et graphique des correspondances entre une chaîne et sa représentation arborescente. Les traits linguistiques décrits se comprennent facilement. Le principe se rapproche donc de celui des grammaires statiques, mais répond efficacement aux lacunes des grammaires statiques évoquées plus haut. En 1988, Zaharin Bin Yussof et Christian Boitet proposent également un raffinement des STCG, les SSTC (*Structured String-Tree Correspondences*), afin de mieux décrire les correspondances qui constituent les STCG [Tsai, 2004].Après un bref historique des recherches autour des STCG, nous en présenterons le fonctionnement plus en détails.

Yves Lepage, dans ses travaux de thèse en 1984, reprend le principe des STCG pour, entre autres, formaliser la notion d’identification [Lepage, 84]. L'identification est un type particulier d'unification qui sera repris par Tang Enya Kong dans sa proposition d'amélioration des STCG. Yves Lepage a également mis au point un éditeur graphique pour les STCG.

Tang Enya Kong dans ses travaux de thèse en 1994 [Tang, 94], a fait la synthèse des publications traitant des STCG. Ce sont les STCG dans leur présentation la plus aboutie, telles que exposées dans sa thèse que nous prendrons comme référence par la suite dans nos travaux. De plus, Tang Enya Kong a pu proposer un algorithme permettant d'obtenir un analyseur fonctionnel à partir des STCG.

La recherche autour des STCG s'est poursuivie et plus récemment, en 2003, Mosleh H Al-Adhaileh en a proposé avec Tang Enya Kong une version bilingue [Al-Adhaileh, 2003] : les S-SSTC (*Synchronous Structured String-Tree Correspondences*). Chaque S-SSTC se compose de deux SSTC, une pour chaque langue décrite. Ce formalisme a été appliqué par Tang Enya Kong dans le système SiSTeC qui inclut un éditeur d'arbres, afin de construire une base de données de correspondances bilingues.

La raison pour laquelle la recherche autour des correspondances chaîne-arbre s'est poursuivie si longtemps est la puissance qui leur est inhérente. Tout en étant claires et simples à comprendre, les grammaires construites à partir des STCG permettent de décrire des phénomènes linguistiques complexes et reconnus comme des difficultés du TAL. Tang Enya Kong dans sa thèse présente trois de ces difficultés :

- L'ellipse (*lexicalisation*) se produit quand l'auteur d'un énoncé sous-entend un élément de la phrase pour éviter une redondance. Par exemple dans la phrase « Jean a mangé une pomme et Marie une poire », c'est le verbe MANGER qui n'est pas répété et est sous-entendu entre « Marie » et « une poire ». Les STCG permettent de traiter ce phénomène en dessinant un arbre qui restitue la présence de MANGER dans la deuxième partie de la phrase afin de produire un arbre complet et atteste de la validité de la phrase, tout en faisant correspondre cet arbre à la chaîne où le deuxième verbe est absent. Le phénomène de l'ellipse est ainsi reconnu et correctement interprété.
- La non-projectivité (*cross-dependencies*) est le phénomène qui se produit en français par exemple avec l'utilisation de l'adverbe « respectivement ». Par exemple, « Jean et Marie mangent respectivement une pomme et une poire ». L'analyse doit montrer que Jean mange la pomme et que Marie mange la poire. Les STCG permettent de produire la description adaptée car elles offrent une liberté dans la composition des arbres pour déplacer et dupliquer les éléments, tout en gardant la trace de l'ordre des éléments.
- La variabilisation (*featurisation*) résulte du choix du linguiste, qui peut juger que certains éléments de la phrase ne doivent pas être présents dans l'arbre. Ils sont alors représentés sous forme de traits rattachés à un élément de l'arbre. Prenons l'exemple d'une phrase négative, le linguiste peut choisir de ne pas représenter les mots qui expriment la négation dans l'arbre, et préférer ajouter un trait « negation » par exemple sur le prédicat de la phrase.

Les STCG retiendront donc notre attention dans notre démarche. Nous y trouvons des éléments qui correspondent à nos besoins. Étudions donc à présent les STCG plus en détails.

4.3. Description des STCG

Une grammaire de correspondance chaîne-arbre est constituée de règles. Les règles se divisent en deux parties, une partie gauche, ou « correspondance principale » (*main-corr*), et une partie droite, ou partie des « sous-correspondances » (*sub-corr*). À ces deux zones de description de correspondances s'ajoutent des zones où sont exprimées les décorations portées par les nœuds et la chaîne.

Prenons comme exemple la règle donnée en annexe 2. Comme dans les grammaires statiques, elle commence par une zone ZDOC avec les méta-informations concernant la règle et une zone pour les exemples en fin de règle.

Vient ensuite une zone ZGRAPH contenant les correspondances. Elle se divise en deux parties, une avec la correspondance principale et une avec les sous-correspondances. Ces dernières viennent préciser la première. Une correspondance aligne une chaîne et un arbre. La chaîne est soulignée, tous les éléments de la chaîne sont séparé par un point. Les éléments de la chaîne sont symboliques. Chaque élément symbolise un token du type de groupe décrit. Une variable préfixée par le symbole # peut être utilisée pour décrire une liste non définie d'éléments de chaîne (un ensemble de tokens qui se suivent). Les arbres peuvent également comporter des variables. Préfixées par le symbole $, elles sont équivalentes à une forêt, c'est-à-dire une suite d'arbres ou de nœuds simples.

La correspondance principale permet d'exprimer quels sont la chaîne et l'arbre mis en correspondance. Mais, ainsi que Tang Enya Kong l'explique dans sa thèse [TANG, 1994] il peut y avoir un écart important entre la chaîne et l'arbre qui la modélise. L'usage systématique des sous-correspondances permet donc de bien préciser quel morceau d'arbre correspond à quel morceau de chaîne. Par exemple dans la règle R1 donnée en annexe 2, la deuxième sous-correspondance permet de rattacher la particule UP au verbe. Ainsi, le nœud correspondant au verbe VK est composé d'une liste d'éléments : le verbe et sa particule UP. Ces deux éléments ne sont pas réalisés côte à côte dans la chaîne.

Les zones qui suivent permettent d'exprimer les décoration qui s'appliquent aux éléments des correspondances. Toutes les conditions sur un même nœud sont exprimées au sein d'une paire d'accolades.

La zone ZSTRING permet d'exprimer les conditions concernant la chaîne de la correspondance principale. Par exemple, dans la règle R1, des conditions sont exprimées uniquement sur l'élément « up » qui doit être une particule et bien correspondre à l'unité lexicale « up ».

La zone ZTREEREF permet d'exprimer les conditions qui s'appliquent sur les arbres des sous-correspondances (aussi appelés TREEREF), tandis que la zone ZTREE permet d'exprimer les conditions qui s'appliquent sur l'arbre de la correspondance principale (aussi appelé TREE). Enfin, pour éviter la redondance, les informations communes aux zones ZTREEREF et ZTREE sont exprimées dans la zone ZCOMMON.

Voilà le fonctionnement d'une grammaire STCG, elle présente différents avantages et inconvénients dont nous allons faire la synthèse pour proposer une solution alternatives aux grammaires statiques.

4.4 Solution proposée : Bidirectional Correspondence Grammar

4.4.1 Synthèse entre les grammaires statiques et les STCG

Les STCG comblent donc les lacunes que nous avions pu exposer concernant les grammaires statiques : le formalisme des STCG est bien un formalisme relativement facile à aborder pour un linguiste, qui permet de décrire la langue en mettant en évidence des traits de langage. De plus, le formalisme des STCG est bien manipulable informatiquement, parce qu'il repose sur un principe de réécriture, grâce auquel les correspondances sont bien marquées entre la chaîne et l'arbre. La manière d'exprimer les décorations sur les nœuds, sans ambiguïtés et bien délimités par les accolades participe également à rendre ce formalisme

manipulable. De plus, Zaharin Bin Yussof et Tang Enya Kong ont pu montrer qu'il était possible de générer du code ROBRA à partir des STCG [Zaharin, 91], [Tang, 94].

En contrepartie, cette façon d'exprimer les décorations n'est pas des plus pratique lorsqu'il s'agit de lire la règle. Nous essaierons donc de trouver une solution au niveau de l'affichage des décorations pour que chaque nœud soit accompagné de ses décorations.

La façon dont sont exprimées les décorations sur les nœuds pose un autre problème. Nous avons vu que dans les grammaires statiques, une zone appelée zone III ou ZCORR permettait d'exprimer des conditions inter-sommets. Ces conditions doivent se vérifier sur plusieurs sommets à la fois (comme par exemple lorsqu'il s'agit de vérifier si le nom est bien accordé en genre et en nombre avec son déterminant). Aucune zone prédéfinie ne permet d'exprimer de telles conditions. Dans sa thèse, Tang Enya Kong montre qu'il est toujours possible de trouver un moyen d'exprimer ce type de correspondance, mais leur expression peut très vite devenir complexe. Par exemple, pour exprimer que deux nœuds, les nœuds 1 et 2, ont le même nombre, la condition de grammaire statique :

NB(1) = NB(2)

peut se réécrire en passant par une variable intermédiaire X en STCG :

{NOD = 1, NB = X, ...}

{NOD = 2, NB = X, ...}

Il est donc possible d'exprimer des conditions inter-sommets en STCG, mais cela est moins évident. Et l'égalité entre les valeurs des décorations des deux nœuds n'est plus aussi évidente. En effet, la variable NB se retrouve énumérée parmi toutes les autres variables rattachées aux nœuds en question. La propriété linguistique selon laquelle un déterminant doit s'accorder avec le nom qu'il accompagne n'apparaît plus à la même échelle. L'effet sera le même en ce qui concerne le traitement informatique de la règle et la génération de code, mais cela fait des STCG un formalisme plus informatique que linguistique. Rappelons donc ici le double objectif de notre réflexion sur le formalisme de spécification, qui est à la fois de faciliter la production de code ROBRA, mais surtout d'être un formalisme facile à aborder et permettant une véritable description linguistique des langues. Il doit permettre de théoriser les traits spécifiques à chaque langue.

De plus, un autre type d'information se retrouve dans la zone III des grammaires

statiques qui n'est pas évident à exprimer en STCG, il s'agit des affectations conditionnelles, pour lesquelles, rappelons-le, les conditions portent sur la chaîne et les affectations portent sur l'arbre correspondant. L'expression des conditions sur la chaîne et celles sur l'arbre étant répartie entre les zones ZSTRING et ZTREE dans les STCG, il devient difficile d'exprimer des conditions qui dépendent l'une de l'autre d'une zone à l'autre.

Le but de notre recherche étant de définir un formalisme qui permette d'exprimer les mêmes informations qu'une grammaire statique, nous devons trouver une formulation efficace qui corresponde à nos besoins. Nous allons donc maintenant exposer notre proposition et montrer comment l'utiliser pour exprimer les mêmes informations qu'une grammaire statique.

4.4.2. Présentation de la solution adoptée

Le choix de notre solution a été inspiré par un article de Christian Boitet [Boitet 88]. Lors d'une rencontre au GETALP le 9 juillet 2014, ce dernier a pu nous exposer plus en détails le contenu de l'article. Par la suite, des échanges avec le GETALP nous ont aidée à parvenir au formalisme que nous proposons.

Dans l'article mentionné plus haut, Christian Boitet explique en quoi il est possible de décrire des phénomènes linguistiques complexes en utilisant une grammaire de réécriture Ce format a l'avantage d'être manipulable informatiquement.

Lors de la réunion du 9 juillet, Christan Boitet, reprenant le même exemple que dans l'article, a ajouté des conditions aux règles de réécriture, montrant qu'il est possible de penser un formalisme qui permette d'exprimer autant de conditions que les grammaires statiques :

R1 : (S (a' , b' , c' , S ($F)), a $A b $B c $C)

==> (S ($F) , $A $B $C) & (a' , a) & (b' , b) & (c' , c) & ($A dans a*) & ($B dans b*) & ($C dans c*)

R2 : (S (a' , b' , c') , a b c)

==> (a' , a) & (b' , b) & (c' , c)

Nous avons donc considéré, pour notre proposition, un formalisme proche de celui décrit par Christian Boitet, mais nous avons tenu à y ajouter une composante graphique, comme pour les STCG, afin d'obtenir un formalisme plus intuitif. Ce formalisme, nous l'avons appelé BCG pour « Bidirectional Correspondence Grammar » Nous avons transcrit plusieurs planches statiques en BCG, que nous mettons en annexe avec à chaque fois la planche statique correspondante. Nous avons fait ce travail avec les planches statiques du système anglais, afin de pouvoir présenter notre travail aux chercheurs Malais ayant travaillé sur les STCG et de mieux correspondre avec eux. Nous commenterons ici plus particulièrement la planche R-NPc9, qui est la transcription de la planche NPc9. Cette planche est l'équivalent de la planche française GN8c présentée plus haut. Elle décrit les groupes nominaux composés avec nom collectif du type de « un groupe de personnes ».

Une planche en BCG se compose de cinq zones. La première zone est une zone de méta-informations et la dernière une zone exemple. Ce sont les mêmes que celles des grammaires statiques et des STCG.

Les trois autres zones sont les zones de description de la correspondance. Vient d'abord une zone de correspondance principale, comme pour les STCG, suivie d'une zone de sous-correspondances et enfin une zone équivalente à la zone III des grammaires statiques.

Les correspondances sont équivalentes aux correspondances des STCG. La chaîne est exprimée en bas, chacun de ses éléments est séparé des autres par un point, et les éléments complexes sont préfixés par le symbole #. Tous les éléments simples de l'arbre sont entourés par des cercles, tandis que les contextes sont représentés par des triangles. Nous avons choisi de faire cette différence à cause de la différence de nature qu'il existe entre ces deux types de nœuds. Les nœuds de contexte sont une liste d'éléments indéfinie. Aucune condition ne peut être exprimée dessus. Le contexte n'apparaît que lorsqu'il y a une référence, pour indiquer les frontières de la référence et pour signifier que l'arbre peut contenir plusieurs éléments en plus à l'endroit où se situe le contexte.

Dans la chaîne comme dans l'arbre, des opérateurs d'itération peuvent être utilisés :

- **? :** pour signifier que l'élément peut apparaître 0 ou 1 fois
- **+ :** pour signifier que l'élément peut apparaître 1 ou plusieurs fois
- *** :** pour signifier que l'élément peut apparaître 0 ou plusieurs fois

Lorsque la présence d'un élément est optionnelle et que des conditions d'existence régissent l'apparition de cet élément, ces conditions sont exprimées en dessous de la correspondance principale, dans un cadre bleu, comme c'est le cas pour la R-NPs6 en annexe 4.

La correspondance principale correspond à la partie gauche de la grammaire de réécriture, tandis que les sous-correspondances correspondent à la partie droite. Les sous-correspondances montrent ainsi de quoi est composée la correspondance principale. C'est au niveau des sous-correspondances que sont exprimées les décorations qui concernent la chaîne, les conditions de zone IIa des grammaires statiques, et c'est également l'endroit où les références à d'autres planches sont explicitées. Les conditions de zone IIb des grammaires statiques, c'est-à-dire les conditions concernant l'arbre sont exprimées dans la correspondance principale, grâce à un encadré rouge placé près du nœud qui le concerne.

Enfin, nous avons choisi de maintenir la zone III telle qu'elle existe dans les grammaires statiques, car elle nous semblait nécessaire. En effet, les conditions de cette zone lient l'arbre et la chaîne. Puisque nous avons choisi d'exprimer les conditions concernant l'arbre dans la zone de la correspondance principale et les conditions concernant la chaîne dans les sous-correspondances, il nous faut conserver une zone à part pour exprimer ces conditions qui sont d'une nature différente. Nous avons tout de même modifié l'expression des affectations conditionnelles, en remplaçant le graphe (peu pratique à dessiner et à traiter informatiquement) par des conditions en *if then* de type C.

Nous avons ensuite tenté de donner une représentation non-graphique de notre formalisme, représentation proche de celle donnée dans l'article de Christian Boitet. Cette représentation s'articule donc comme une règle de réécriture, chaque correspondance se place dans une paire de parenthèses et chaque élément de la correspondance, entre crochets.

Pour chaque correspondance, la première paire de crochet contient la description de la correspondance : l'arbre suivi de la chaîne soulignée. Chacun des autres crochets correspond à

l'expression d'une variable et sa valeur associée.

Pour plus de détails sur les BCG, un document de présentation des BCG est fourni en annexe 3. Voyons maintenant quelles sont les limites et améliorations possibles pour ce formalisme.

4.4.3. Limites et développements envisageables

Un premier inconvénient est vite apparu pour notre formalisme des BCG, il concerne les sous-correspondances. Celles-ci manquent d'une numérotation qui indiquerait l'ordre dans lequel les enchaîner pour reconstituer la correspondance principale.

La zone III a soulevé et soulève encore beaucoup de questions dans notre recherche d'un formalisme à la fois clair, facile à comprendre et manipulable informatiquement. En effet, la zone III contient à la fois les décorations qui concernent les nœuds de l'arbre et celles qui concernent les éléments de la chaîne. En faire une zone à part a paru comme une évidence, mais cela présente un inconvénient majeur : cela complique la lecture des planches. En effet, l'utilisateur doit alors faire des allers-retours entre la zone III et les autres zones de la planche. Trouver un moyen de répartir les informations contenues dans la zone III entre la correspondance principale et les sous-correspondances permettrait alors une lecture plus facile de la règle BCG.

Nous avons choisi de conserver les noms de zone des STCG et des grammaires statiques, afin de constituer des points de repère pour traduire les planches existantes dans le nouveau formalisme. Cependant, afin de distinguer notre formalisme et de le rendre plus adapté à une prise en main par un développeur linguiste sans connaissances particulières liées à l'historique du système sur lequel nous travaillons et de sa spécification, il serait peut-être plus pratique de trouver des noms de zones plus précis pour chaque zone.

Il reste ainsi beaucoup de modifications à apporter à notre formalisme et beaucoup de sujets qui portent à réflexion. Mais il reste surtout à mettre les BCG à l'épreuve d'une utilisation concrète pour développer une nouvelle langue. Nous pourrons alors les améliorer pour les rendre pleinement fonctionnelles. Maintenant que nous avons vu en détails le fonctionnement théorique des BCG et leur application sur un exemple de planches de l'anglais, il nous reste à présent à aborder le dernier point de présentation de notre formalisme, à savoir le format de stockage des planches.

4.5 Format de stockage interne des planches

Les planches écrites dans le formalisme des BCG ont vocation à être manipulées pour produire le code ROBRA d'une analyse et d'une génération. En conséquence, elles doivent être stockées dans un format manipulable et ne pas être une simple image ou un pdf. Notre choix s'est porté vers le langage XML, facile à manipuler pour récupérer les informations et générer du code ROBRA. En plus de cela, en utilisant XML, il suffit d'utiliser une feuille de style XSL pour ensuite générer une version des planches destinée simplement à la lecture, en pdf par exemple. Quant aux arbres en particulier, ils peuvent être générés en SVG, qui est un format graphique basé sur XML. XML nous a donc paru comme le format qui s'imposait pour stocker nos planches. Nous avons alors défini une structure type pour nos planches et rédigé une DTD en conséquence. Cette DTD se trouve en annexe 9. Nous avons également rédigé quelques planches dans ce format (voir annexe 10), afin de vérifier son adéquation au besoin de stocker fidèlement les règles.

Les fichiers XML se divisent en deux parties : toutes les méta-informations et les exemples sont contenus dans la balise d'en-tête <meta>, et tout le reste de la règle est contenu entre les balises <rule>. Celle-ci se divise ensuite en trois parties :

- **<main_corr> :** la partie de la correspondance principale. Cette balise peut avoir trois fils : <root> qui est la racine de la correspondance principale, <string> qui contient la chaîne et une balise optionnelle <existence_conditions> où sont exprimées le conditions d'existence des nœuds quand il y en a. Nous avons choisi d'avoir une balise spécifique pour la racine de la correspondance principale afin de l'identifier plus vite

lors du traitement des planches. Ainsi, pour trouver l'arbre décrit par la règle, l'expression XPath « //root » suffit. Le fonctionnement de la correspondance principale est ensuite le même que celui des sous-correspondances décrit ci-après.

- **<sub_corr> :** cette partie est constituée de plusieurs sous-correspondances <sub>, chacune composée d'un arbre <sub_root> et d'une chaîne <string>. Les arbres contenus sous la racine <sub_root> contiennent des nœuds <nodes> qui peuvent éventuellement contenir une balise <tag> dans laquelle sont données toutes les conditions correspondant à ce nœud.
- **<zoneIII> :** ces balises contiennent les conditions de zone III.

Différent types de conditions existent dans notre structure XML :

- **<decoration> :** contient les informations les plus simples. Elles sont exprimées de la façon suivante :

Ex : K(NP) = NP

```
<decoration name="K" node="NP" value="NP"/>
```

- **<assignment> :** correspond aux affectations, lorsque la valeur d'une variable est affectée à une autre.

Ex: SEM1(NP) = SEM1(noun)

```
<assignment name="SEM1" node="NP" name_value="SEM1" node_value="noun"/>
```

- **<predicate> :** correspond aux prédicats exprimés avec le symbole $ dans les grammaires statiques. Le prédicat permet de vérifier une condition, il est prédéfini.

Ex1: $agr.semn/sem1(noun,ap2)

```
<predicate value="$agr.semn/sem1">
<applied_to node="noun"/> <applied_to node="ap2"/>
</predicate>
```

Ex2: SEM1(ap2) = $inter.semn/sem1(noun,ap2)

```
<predicate name="SEM1" node="ap2" value="$inter.semn/sem1">
        <applied_to node="noun"/>
<applied_to node="ap2"/> </predicate>
```

Dans tous les cas, nous avons essayé de faire en sorte de laisser le moins de texte brut possible et de bien structurer les informations au sein de balises.

Certains points resteraient cependant à améliorer dans notre format XML, comme, entre autres, la partie exemples. Les commentaires n'étant pas toujours dissociés des exemples, certains exemples peuvent être accompagnés de commentaires pour en expliciter la valeur. Enfin, il resterait surtout à mettre notre format XML à l'épreuve d'une application et d'une transformation pour en tester l'efficacité, mais ceci ne fait pas partie de l'objet de nos recherches.

Un exemple de planche ainsi stockée se trouvent en annexe 10. Une fois le formalisme établi et le format de stockage défini, nous avons pu commencer à travailler sur l'étape qui découle directement de notre étude : la conception d'une interface de saisie qui permettra de transformer des grammaires statiques ainsi que d'écrire de nouvelles grammaires de description dans notre formalisme des BCG.

4.6. Développement d'Eurydice, interface de saisie pour les planches de BCG

Afin de réaliser un outil de saisie adapté, il nous a fallu identifier les besoins spécifiques aux BCG. Ces besoins sont les suivants :

- Nous avons besoin de saisir des arbres de constituants.

- Les nœuds de l'arbre saisi devront pouvoir porter des étiquettes.
- Il faut aussi prévoir de saisir une chaîne sous l'arbre.
- L'outil doit permettre de générer plusieurs sorties : une sortie au format XML, une sortie pour la lecture dans un format image ou en PDF par exemple.
- L'arbre produit doit être en SVG.
- L'outil doit pouvoir fonctionner en ligne, afin de permettre un accès facile aux contributeurs du système. Les planches ainsi produites peuvent être directement enregistrées dans le système.

Des recherches sur les outils existants dédiés à la saisie d'arbres de description linguistique ont pu nous permettre de conclure au besoin de développer notre propre outil. Aucun ne satisfaisait aux besoins énoncés plus haut.

Nous avons donc dû chercher une solution qui nous soit propre. Cette solution, nous l'avons trouvée lors de notre rencontre avec les membres du GETALP, avec l'aide de Raphaël Jakse, développeur de l'application Aude [Lien 6] sous la direction d'Yliès Falcone. Cet outil publié en open source est dédié à la saisie d'automates, mais grâce à une collaboration avec son auteur, nous avons pu envisager de l'adapter à la saisie d'arbre. Aude présente les caractéristiques techniques que nous recherchons dans notre outil de saisie : en open source et facilement adaptable, l'outil fonctionne en ligne, les automates ou arbres saisis sont au format SVG, l'outil prend déjà en charge l'export aux formats DOT et SVG, il est possible d'ajouter des étiquettes sur les états, les interactions avec la souris de l'utilisateur sont prédéfinies. Notre travail sur l'outil se résume donc en son adaptation à la saisie d'arbres. Nous obtiendrons ainsi un outil assez abouti maintenable et sur mesure en relativement peu de temps.

L'outil Aude est développé en JavaScript et en AudeScript, langage dérivé de JavaScript et conçu spécialement pour le développement de cet outil. Une première familiarisation avec l'outil et les bibliothèques utilisées s'avère donc nécessaire avant de pouvoir commencer le travail d'adaptation du code. Une fois cette première étape passée, nous avons pu obtenir les premiers résultats :

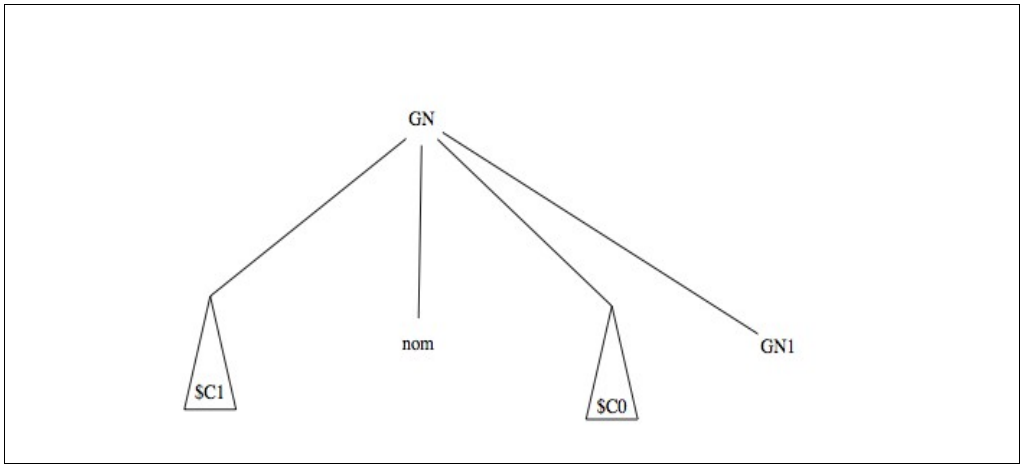


Fig. 15 : capture d'écran d'Eurydice - exemple d'arbre

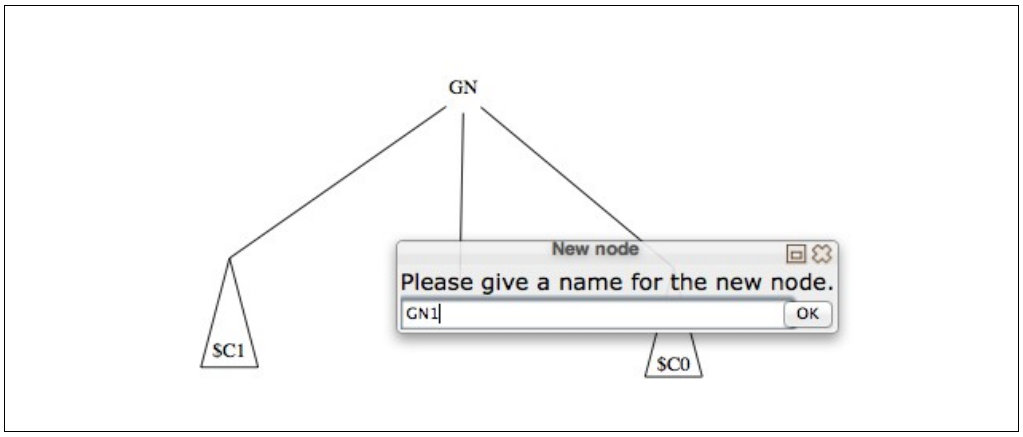


Fig. 16 : capture d'écran d'Eurydice - boite de dialogue

Il est possible de distinguer sur la figure 15 les nœuds simples des nœuds de contexte. La saisie des arbres est très rapide, un simple double-clic suffit à générer un nœud, un double-clic avec la touche shift maintenue enfoncée permet de générer un nœud de contexte, une boite de dialogue permet de saisir le nom du nœud (Fig. 16), et il suffit de relier deux nœuds entre eux avec le curseur pour que la branche se place à l'endroit optimal pour relier les deux nœuds par une droite. Lorsque l'utilisateur déplace un nœud, toutes les branches qui lui sont

accrochées suivent. Dessiner un arbre devient ainsi un jeu d'enfant et la saisie de nouvelles planches en sera facilitée.

Malheureusement, nous n'avons pas eu l'occasion d'aller plus loin dans ce travail, mais nous avons montré qu'il était possible d'adapter rapidement (une soixantaine d'heures pour découvrir JavaScript, l'extension AudeScript et les bibliothèques de l'outil et apporter les modifications décrites plus haut) l'outil afin de saisir des arbres au lieu d'automates. Le reste du développement pourra suivre et constituera une des suites logiques de notre travail. Nous pouvons imaginer à terme d'obtenir un outil qui permettrait non seulement la saisie de planches et leur export en XML, SVG, PDF ou autre format, mais un outil qui permettrait également de générer et tester le code ROBRA correspondant à la règle saisie, un outil qui rentrerait directement dans la chaîne de développement du système. Ceci est envisageable grâce au formalisme des BCG.

5. Conclusion

Au cours de cette étude nous avons cherché à répondre à un besoin lié au développement d'un système de traduction automatique par règles : il y a nécessité d'un formalisme de spécification pour les modules d'analyse et de génération. Ce formalisme doit être un formalisme de description linguistique suffisamment formel pour permettre la génération de code ROBRA et surtout facile à appréhender pour un linguiste sans qualification en informatique.

Dans notre démarche de recherche de ce formalisme, nous avons exploré le courant des grammaires d'unification et passé en revue quelques analyseurs fonctionnels pour trouver des exemples de réponses à notre problématique.

Nous avons ensuite contextualisé nos recherches en présentant l'héritage linguistique dans lequel s'inscrit les travaux du GETA et donc le système sur lequel nous travaillons. Nous avons tenté de clarifier le lien entre les théories de Chomsky, Tesnière et Mel’čuk et le formalisme linguistique du GETA, les grammaires statiques. Ceci nous a permis présenter ce formalisme, ses points forts et ses lacunes.

Enfin, nous avons trouvé source d'inspiration avec les STCG, un formalisme dérivé des grammaires statiques pour proposer une alternative plus formelles aux grammaires statiques. Les STCG nous ont aidé à proposer une solution adaptée à nos besoins. Ce formalisme de description linguistique que nous avons obtenu, les BCG, nous avons pu le mettre en application sur des exemples concrets en transformant quelques planches statiques en règles BCG. Nous avons également amorcé le travail autour d'une interface de saisie pour les règles BCG. Des améliorations, que nous avons listées, restent à faire sur les BCG, mais il reste surtout à mettre notre formalisme à l'épreuve du temps et de son utilisation.

Glossaire

Ariane : Environnement de développement de systèmes de traduction automatique développé au GETA entre 1978 et 1985. C'est sous cet environnement qu'ont été développés les systèmes de traduction automatique sur lesquels nous travaillons : les systèmes français-anglais FR3-AN3 et anglais-français ANG-FRA.

BCG : *Bidirectional Correspondence Grammars*. Formalisme proposé par cette étude.

GETA : Groupe d'Étude pour la Traduction Automatique. Laboratoire de Grenoble, issu de la réorganisation du CETA en 1971. En 2007 le GETA fusionne avec GEOD pour devenir le GETALP.

Grammaire statique : Formalisme de description linguistique développé par Bernad Vauquois et Sylviane Chappuy. Ce formalisme joue le rôle de spécification pour les programmes d'analyse et de génération écrits en ROBRA.

GSCS : Grammaire Statique de Correspondances Structurales. Nom donné aux grammaires statiques après leur révision.

Héloïse : Environnement de développement de systèmes de traduction automatique assurant la compatibilité ascendante avec les langages d'Ariane. Cet environnement, développé par Vincent Berment, est disponible sur Internet depuis fin 2009.

Planche statique : Un des éléments des grammaires statiques. Une grammaire statique est un ensemble de planches statiques qui peuvent faire référence les unes aux autres.

ROBRA : Langage Spécialisé pour la Programmation Linguistique d'Ariane permettant de programmer des transducteurs d'arborescences. C'est le langage dans lequel sont écrits les modules d'analyse et de génération structurales.

SSTG : *String-Tree Correspondence Grammar.* Formalisme développé Zaharin Bin Yussoff, Christian Boitet et Tang Enya Kong reprenant les grammaires statiques de manière plus formelle permettant la génération de code. Ce formalisme repose sur un système de correspondance entre chaînes et arbres.

Structure multiniveaux : Théorie linguistique originale mise au point par le GETA. Une structure multiniveaux associée à un énoncé mêle deux niveaux de description de surface de l'énoncé (arbre de constituants et arbre de dépendances) avec un niveau logico-sémantique de description du sens.

Bibliographie

Ouvrages

Abeillé A., *Les Nouvelles Syntaxes*, 1993

Fuchs C. et le Goffic P., *Les Linguistiques contemporaines*, 1996

Tesnière L., *Éléments de syntaxe structurale*, 1959

Articles

Besançon R., de Chalendar G., « L'analyseur syntaxique de LIMA dans la campagne d'évaluation EASY », TALN 2005

Boitet C., « Bernard VAUQUOIS' contribution to the theory and practice of building MT systems : a historical perspective », Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, 1988

Chéragui M.A., « Theoretical Overview of Machine translation », 2012

de La Clergerie É, Sagot B, Nicolas L, Guénot M-L, « FRMG: évolutions d'un analyseur syntaxique TAG du français », Journée de l'ATALA sur : Quels analyseurs syntaxiques pour le français ?, 2009

Guibauld JP., « Descripteurs multiniveaux et génération de texte en Ariane-78 », Journée ATHENA sur la traduction automatique, Liège, 1987

Kahane S., « Grammaires d'unification polarisées », TALN 2004

Léon J., « Le CNRS et les débuts de la traduction automatique en France », La revue pour l'histoire du CNRS, 2002

de Malézieux G., Bosc A., Berment V., « RBMT as an alternative to SMT for under-resourced languages »

Marandin JM., Cory M., « La linguistique au contact de l'informatique : de la construction des grammaires aux grammaires de construction », *Histoire Epistémologie Langage,* 2001

Mel'čuk I., Paraphrase et lexique dans la théorie linguistique sens-texte », *Lexique*, 1988

Mel'čuk I., « Vers une linguistique Sens-Texte. Leçon inaugurale », Paris: Collège de France, 1997

Nedobejkine N., Vauquois B., « Étude de la validité du formalisme choisi pour représenter

la structure linguistique interface », Contrat CEE, 1980

Perrier G., Guillaume B., Marchand J., « La chaîne d'analyse syntaxique de LEOPAR », International Workshop on Parsing Technologies 2009

Vauquois B., « Description de la structure intermédiaire », Colloque de Luxembourg, 1978

Vauquois B., « Aspects of mechanical translation in 1979 », Conference for Japan IBM scientific program, 1979

Vauquois B., Chappuy S., « STATIC GRAMMARS, A formalism for the description of linguistic models », Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, New York, 1985

Zaharin Y., TANG E.K., « STGC as a base for generation of analysis grammar in ROBRA », Washington, 1991

Thèses

Al-Adhaileh, M.H, *Synchronous Structured String-Tree Correspondence (S-SSTC) and its applications for machine translation*, Universiti Sains Malaysia, 2003

Chappuy S., *Formalisation de la description des niveaux d'interprétation des langues naturelles. Étude menée en vue de l'analyse et de la génération au moyen de transducteurs*, Grenoble, juillet 1983.

Lepage Y., *Un système de grammaires correspondancielles d'identification*, Grenoble 1, juillet 1984

Tang E.K., *Natural languages Analysis in machine translation (MT) based on the STCG*, PhD thesis, Sains Malaysia University, Penang, Mars 1994

TSAI WJ., *La coédition langue↔UNL pour partager la révision entre langues d'un document multilingue*, Grenoble 1, juillet 2004

Documentation

Chappuy S., *Une approche industrielle au développement des systèmes de TAO experts, Application à l'analyse du français, version FR4*, Projet Traouiéro, mars 2013

Chappuy S., Entretien avec Nicolas Nedobjeskine, 2003, document interne GETALP

Chauché J., *La Grammaire structurelle*, 2012

Chauché J., *SYGMART : manuel de référence*, 2008

Travail collectif, *Maquette pédagogique BEX-FEX. Grammaires statiques de l'anglais,*

morphologique et structurale, GETA Document, 1983

Travail collectif, *ROBRA*, Document GETALP

Cours

Touratier C., *Cours Fonctions de la phrase simple,* 2005, http://sites.univ-provence.fr/wclaix/cours.htm

Liens

1. Environnement de traduction automatique Héloïse

http://www.taranis-software.com/Heloise/GETA/Heloise.php

2. Analyseur FRMG d'ALPAGE (INRIA)

http://alpage.inria.fr/frmgwiki/frmg_main/frmg_server

3. Analyseur LIMA du CEA / LIST

http://www-list.cea.fr/fr/vision-et-multimedia/306-traitement-du-langage-naturel

4. Analyseur SYGMART

http://www.sygtext.fr/ExempleAnl.html

5. Analyseur Leopar de l'équipe sémagramme (LORIA, INRIA)

http://wikilligramme.loria.fr/doku.php?id=leopar:demo

6. Aude, outil de saisie d'automates

http://automata.forge.imag.fr

Annexes

Annexe 1 72
GN8c : planche statique décrivant un groupe nominal du français 72
Annexe 2 73
Règle STCG décrivant une phrase dont le verbe principal est un verbe à préposition 73
Annexe 3 74
Document d'introduction aux BCG 74
Annexe 4 81
Planche statique NPs6 81
Planche BCG R-NPs6 84
Transcription de la planche BCG R-NPs6 88
Annexe 5 91
Planche statique NPc9 91
Planche BCG R-NPc9 93
Transcription non graphique 96
Annexe 6 97
Planche statique NPc14a 97
Planche BCG R-NPc14a 99
Annexe 7 102
Planche statique AP1 102
Planche BCG R-AP1 108
Annexe 8 111
Planche statique NUMP1 111
Planche BCG R-NUMP1 113
Annexe 9 116
DTD du format de stockage interne des planches 116
Annexe 10 121
Exemple de planche stockée en XML : R-NPs6 121

Annexe 1

GN8c : planche statique décrivant un groupe nominal du français

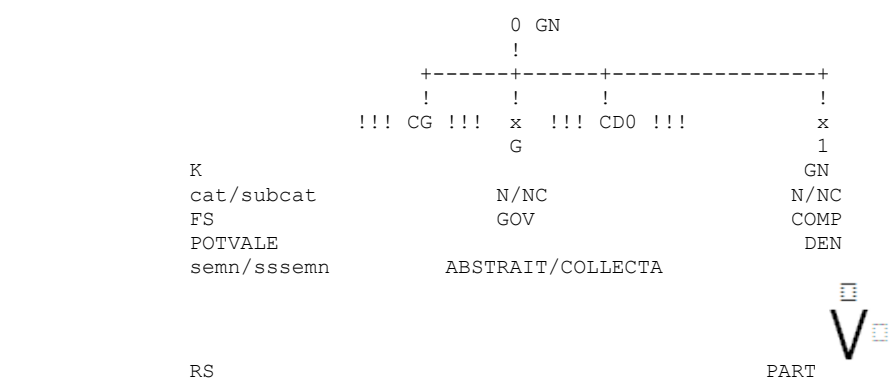
Grammaire statique du français

Numéro de la planche: GN8c
Type: GNc groupe nominal complexe
Cas traités: groupe nominal gouverné par un nom commun de sémantique COLLECTIF dominant à droite un groupe nominal deN

---

références: GN : 5a pour le nœud 0
GN : 5a, 6a, 6b, 7a, 7b, 7c, 8c, 9b, 11 pour le nœud 1

ZONE I
Chaîne : 0($L1,G), 1
Arbre : 0($L1,G,1)



ZONE IIa
K(0)=GN
FS(G)=GOV ⋀ CAT(G)=N ⋀ SUBN(G)=NC ⋀ SEMN(g)=ABSTRAIT ⋀ (SSABST(G)=COLLECTA ⋁ SSABST(G)=COLLECTNA)
K(1)=GN ⋀ CAT(1)=N ⋀ SUBN(1)=NC ⋀ POTVALE(1)=DEN
SSABST(G)=COLLECTA --> NB(1)=PLUR

ZONE IIb
K(0)=GN
K(1)=GN ⋀ FS(1)=COMP ⋀ VALE(1)=DEN

ZONE III
SEMN(0)=$union.abstrait.semn(1)
TYPOG(0)=$inter.typog(0,1)

+ SSABST(G)=COLLECTA !!!RS(1)=PART ⋀ $sssemn(0)=$union.collecta.sssemn(1)!!!
!
+------------------- !!!RS(1)=TRAP ⋀ $sssemn(0)=$union.collecta.sssemn(1)!!!

EXEMPLES ET COMMENTAIRES
l'ensemble des tuyauteries
un tas de feuilles
la totalité des gens
un groupe de personnes
une part de gâteau
une part de bonheur

Sylviane Chappuy
Version février 1990
Remise en forme août 2011

Annexe 2

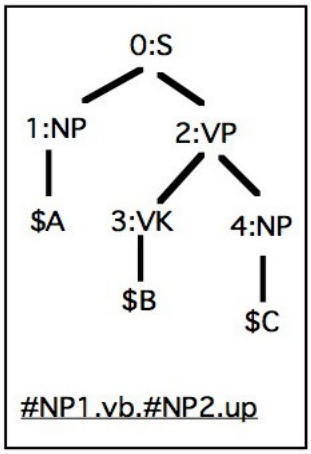
Règle STCG décrivant une phrase dont le verbe principal est un verbe à préposition

ZDOC

Rule Name : R1" ◉ **Axiom**
Type : Discontinuous sentence
Date : 14/2/1993
Author : TEK
Description : Treatment of discontinuous particle "up".

ZGRAF

Main Correspondence



Subcorrespondence

1:NP
$A
#NP1
with : RNP1,...,etc.

3:VK
$B
vb.up
with : RVK,...,etc.

4:NP
$C
#NP2
with : RNP2,...,etc.

ZSTRING {NOD = up, UL = 'up', CAT = adj, SUBA = pcl}

ZTREEREF {NOD = 1, NUM = ...X..., SSEM = ...@Y..};
{NOD = 3,TENSE = V, NUM = ...X..., SEM0 = ...@Y.., SEM1 = ...@Z..,
VL1 = ...W...};
{NOD = 4, SSEM = ...@Z..}

ZCOMMON {NOD = 1, K = np, (SVL = n or SVL = r)};
{NOD = 3, CAT = v, JPCL = up};
{NOD = 4, K = np, SVL = N, SVL = ...W...}

ZTREE {NOD = 0, K = vcl, UL = 'VCL', NUM = X, SSEM = @Y, VOICE = active, VL1 = W};
{NOD = 1, SF = subj, NUM = X, RL = arg0, SEM = @Y};
{NOD = 2, K = vp, TENSE = V, UL = 'VP', SF = gov, NUM = X, VL1 = W};
{NOD = 3, SF = gov, NUM = X, SSEM = @Y, VL1 = W};
{NOD = 4, SF = obj1, RL = arg1, SSEM = @Z}

ZEXAMPLE
John picks the ball up.
(He is the one) who picks the ball up.
Who picks the ball up ?

Annexe 3

Document d'introduction aux BCG

Bidirectional Correspondence Grammar
Introduction to the formalism

I. General introduction..... 1
II. Presentation of the formalism.....2
II.1. Meta-information.....2
II.2. Main-correspondence.....2
II.3. Sub-correspondences.....3
II.4. “Zone III”..... 4
III. Non-graphical transcription..... 4
IV. XML representation..... 5

I. General introduction

The formalism introduced in this document is called Bidirectional Correspondence Grammar (BCG). It is a STCG-based derivation of Bernard Vauquois and Sylviane Chappuy's static grammars, also called Structural Correspondences Static Grammar (SCSG). The point of it is to allow a faster understanding of the static boards as well as the automatic generation of ROBRA code.

The formalism we came up with is inspired from Christian Boitet's article

« Bernard Vauquois' contribution to the theory and practice of building MT systems: a historical perspective ». We defined a graphical and hopefully more intuitive version of the formalism that we will present in the first part of the document. We will also present the plain text version of the same formalism in a second part.

Finally, in order to store these new boards in a parsable format, we chose to use XML. We considered XML would offer an easy enough way to parse the board in order to generate ROBRA code and at the same time would allow viewing the board using a XSL stylesheet in order to generate the corresponding graphic or plain text display.

II. Presentation of the formalism

II.1. Meta-information

The formalism divides into five zones. The first and last zones only contain meta-information. The very first zone contains generic meta-information such as:

- **Rule number:** The rule number always starts with « R- » in order not to confuse the reference if, for example, rule R-AP2 is called on the second adjective phrase of the structure named AP2 too. The fact that every rule name is now starting with « R- » thus allows knowing immediately when we talk about a rule.
- **Type:** Whether the board is an elementary, simple or complex one.
- **Handled cases:** A description of the family of phrases handled by the board.
- **References:** A list of all the boards referenced by each node.
- **Corresponding rules:** The list of the ROBRA rules written to implement the board.

For purpose of readability, the last two pieces of meta-information, that is all the examples and comments, were placed at the end of the board.

II.2. Main-correspondence

The main-correspondence is a STCG-type correspondence between a string

and a structure. A tree is made of two kinds of nodes[1]:

- triangle nodes for the contexts of the referenced rules
- round nodes for all the other nodes

Such a difference is being made because the context nodes are of a different kind then the other nodes: no conditions can be expressed on those nodes. Indeed a context is just a non-specific list of trees and nodes. This list can be empty and can contain an undefined amount of elements. It is the equivalent of the ROBRA forests and as a consequence, we also chose to prefix the context names with the symbol $.

In order to make the tree easier to understand at first glance, we chose to give the nodes real names.

Nodes can occur several times. In order to represent that, we used the same formalism that is used for regular expressions:

?: a question mark means that the node is optional, it can occur 0 or 1 time.

***:** the Kleene star means that the node can occur 0 or several times

+: the plus sign means that the node occurs at least once.

Sometimes, when a node is optional, it can be needed to express conditions on when the node is going to occur. We expressed such existence conditions underneath the tree of the main correspondence, in a blue box.

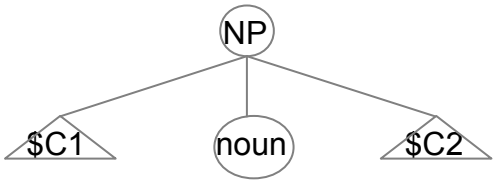
Here must also be found all the information expressed in the static grammars zone IIb, also called “Zone Arbre” in a recent revisiting of the SCSG made by Sylviane Chappuy, that is to say the information about the tree. This information has been put in a red box next to the affected node. The logical operators & and | are used to articulate the decorations. Parentheses are only used for the complex expressions, involving disjunction for example.

Finally, the string is made of different elements, which are all separated by a full stop. String variables, prefixed by the symbol « # » are used to replace several string elements. For example when using a context node which content is undefined,

1 Note that a node of the structure can represent a forest.

the matching string needs to be described using a string variable.

Example:



#C1.noun.#C2

II.3. Sub-correspondences

The sub-correspondences are made the same way as the main-correspondence: with a tree and a string. Each sub-correspondence is in a box and next to it; a red box contains all the information corresponding to the static charts zone IIa or “Zone Chaîne d'Arbres” in the recent revisiting made by Sylviane Chappuy.

In concrete terms, this zone allows two things:

- To characterise the string with all the decorations associated with each of its elements. It allows to develop the references used and to express the constraints on the different nodes of the string.
- To match the different elements of the tree with the corresponding section of the string. This is how the complex features of a language are treated, such as cross-dependencies, lexicalisation and featurisation.

II.4. “Zone III”

This zone corresponds to the zone III in the static grammars. Here are expressed all internodal conditions and conditional affectations. The graphics used by Sylviane Chappuy in her static grammars formalism are replaced here by if/else conditions as the one used in programming languages like C.

III. Non-graphical transcription

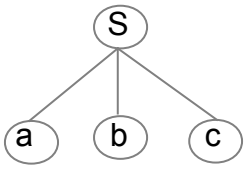
The non graphical transcription contains exactly the same information as the graphical version of the rules.

The first part, above the arrow is a transcription of the main-correspondence. Everything under the arrow is the equivalent of the sub-correspondences zone except for the last pair of brackets which matches zone III. Every decoration or string-tree correspondence is in between square brackets. And all the strings are underlined.

As mentioned above, apart from the last pair of brackets, every pair represents a correspondence. Here is how a correspondence is made:

- The first square brackets contain the transcription of the graphical correspondence. First, the tree structure, and after the coma, the corresponding string, which is also underlined.

Ex: transcription of a small string-tree correspondence



[S(a,b,c), a.b.c]

- After this, each following square brackets contains a decoration associated with the tree described above. The decoration is written in the exact same formalism as in the graphical representation.

Ex: S is a noun phrase

[K(S) = NP]

- In the sub-correspondence, the last pair of square brackets may contain an expression starting with « references ». This is where all the references called by the node are listed. They are all to be listed in between the brackets, separated by a coma.

Ex: S is a reference to a noun phrase described by board R-NPs6

[references(R-NPs6)]

All this information has been put next to each other using the logical operators:

([S(a,b,c), a.b.c]
& [K(S) = NP]
& [references(R-NPs6)]
)

To sum up, the representation can be briefly described by the following

grammar:

<rule> == <main_corr> ===> <sub_corrs> <zoneIII>

<main_corr> == "[" <tree> "," <string> "]" <conditions>

<sub_corrs> == <sub_corr> [<sub_corrs>]

<sub_corr> == "[" <tree>, <string> "]" [<conditions>]<conditions> == &

<condition> [<conditions>]

<condition> == ...

<zone_III> == ...

...

C-style commentaries can be used:

Ex:

// commentary

IV. XML representation

The general organisation of our XML format reflects the organisation of the formalism. It first divides into two parts:

- **<meta>** contains all of the meta-information, including the examples and comments.
- <**rule>** is the core of the board, it contains the linguistic description.

Then the rule is divided into three main parts:

- **<main_corr>** matches the main correspondence. It has three children: <root>, <string> and optional <existence_conditions> element. The <root> element contains the main tree of the main-correspondence. We chose to give this element a unique name “root” because it is the grammatical group described by the rule, the reason why the rule was written. Apart from this specific element <root> and the optional <existence_conditions> element, the main-correspondence works just the same as any of the sub-correspondence described below.
- **<sub_corr>** is composed of several sub-correspondences. They are made of a tree starting with a <sub_root> element and a <string> element. In the

tree, each element <node> can contain decorations of different kind.

- **<zoneIII>** has internodal constraints and conditional assignments. Each are made of decorations of different kinds.

There are three kinds of decorations in our XML representation:

- **decoration:** They are the simplest kind of information. They are represented as such with the different elements of the decoration as attribute values:

Ex: K(NP) = NP

```
<decoration name="K" node="NP" value="NP"/>
```

- **assignment:** They occur when the value of a node is assigned to another node.

ex: SEM1(NP) = SEM1(noun)

```
<assignment name="SEM1" node="NP" name_value="SEM1" node_value="noun"/>
```

- **predicate:** A predicate is an internodal constraint starting with the symbol $. Its value can be assigned to a node decoration or not. If that is the case, the node and decoration name are mentioned as attributes of the element <predicate>. And the nodes the predicate applies to are given as children of the predicate.

ex1:

$agr.semn/sem1(noun,ap2)

```
<predicate value="$agr.semn/sem1">
    <applied_to node="noun"/>
    <applied_to node="ap2"/>
</predicate>
```

ex2:

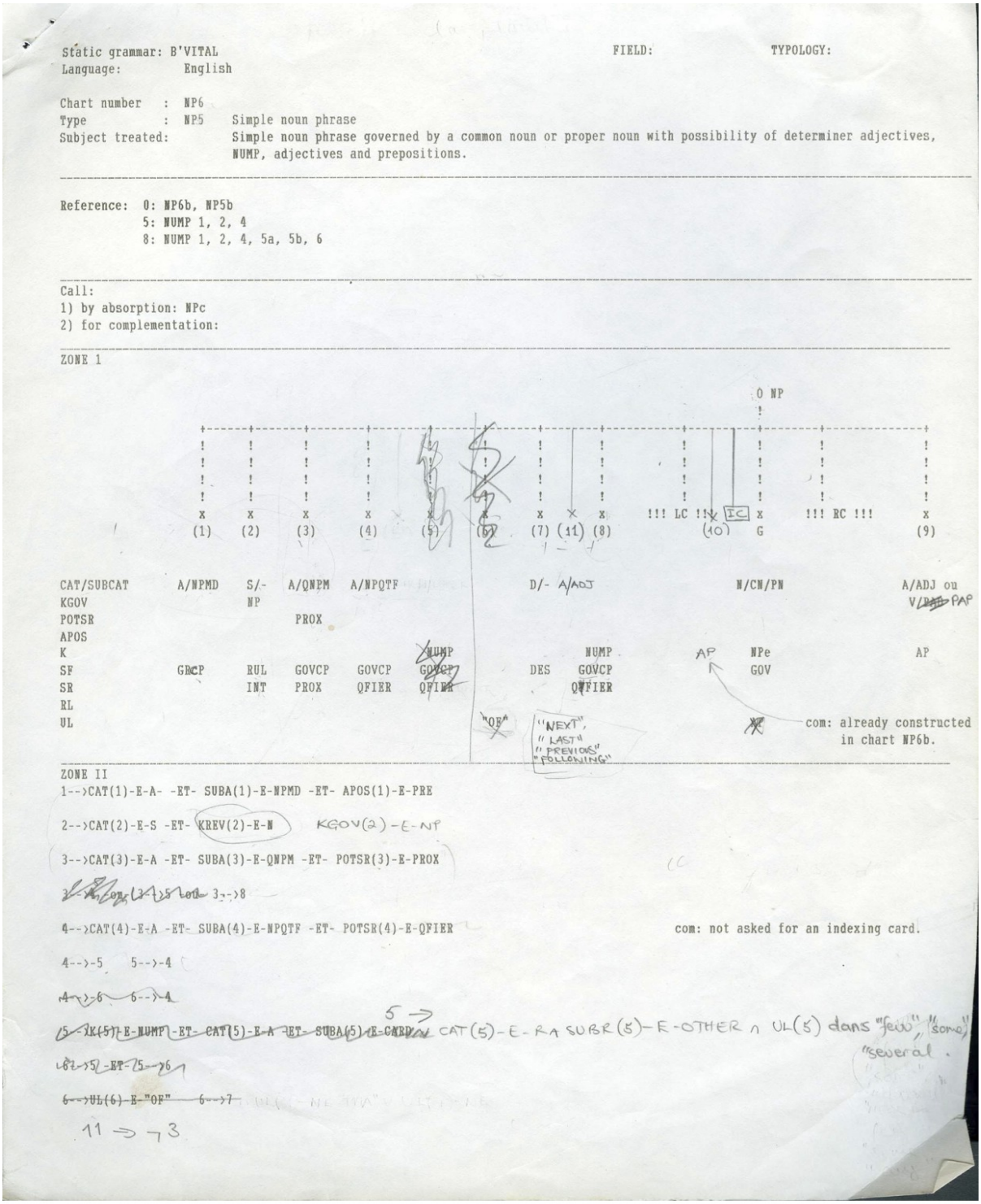
SEM1(ap2) = $inter.semn/sem1(noun,ap2)

```
<predicate name="SEM1" node="ap2"
value="$inter.semn/sem1">
    <applied_to node="noun"/>
    <applied_to node="ap2"/>
</predicate>
```

For more details, please see the DTD.

Annexe 4

Planche statique NPs6



Static grammar: B'VITAL — FIELD: — TYPOLOGY:
Language: English

Chart number : NP6
Type : NP5 Simple noun phrase
Subject treated: Simple noun phrase governed by a common noun or proper noun with possibility of determiner adjectives, NUMP, adjectives and prepositions.

Reference: 0: NP6b, NP5b
5: NUMP 1, 2, 4
8: NUMP 1, 2, 4, 5a, 5b, 6

Call:
1) by absorption: NPc
2) for complementation:

ZONE 1

0 NP

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(11)	(8)	(10)	G	(9)
	x	x	x	x	x	x	x	x	x	!!! LC !!! x IC	x	!!! RC !!! x
CAT/SUBCAT	A/NPMD	S/-	A/QNPM	A/NPQTF			D/- A/ADJ			N/CN/PN		A/ADJ ou V/~~PAS~~ PAP
KGOV		NP										
POTSR			PROX									
APOS												
K					NUMP					AP	NPe	AP
SF	GRCP	RUL	GOVCP	GOVCP	GOVCP		DES	GOVCP			GOV	
SR		INT	PROX	QFIER	QFIER			QFIER				
RL												
UL						"OF"						

"NEXT", "LAST", "PREVIOUS", "FOLLOWING"

com: already constructed in chart NP6b.

ZONE II

1-->CAT(1)-E-A- -ET- SUBA(1)-E-NPMD -ET- APOS(1)-E-PRE

2-->CAT(2)-E-S -ET- KREV(2)-E-N KGOV(2)-E-NP

3-->CAT(3)-E-A -ET- SUBA(3)-E-QNPM -ET- POTSR(3)-E-PROX

~~3-->5 5-->6~~ 3-->8

4-->CAT(4)-E-A -ET- SUBA(4)-E-NPQTF -ET- POTSR(4)-E-QFIER

com: not asked for an indexing card.

4-->5 5-->4

~~4-->6 6-->4~~

~~5-->K(5)-E-NUMP -ET- CAT(5)-E-A -ET- SUBA(5)-E-CARD~~ 5 → CAT(5)-E-A SUBR(5)-E-OTHER ∩ UL(5) dans "few", "some", "several".

~~L82-->5) -ET- 5-->6~~

~~6-->UL(6)-E-"OF" 6-->7~~

11 → 3

7-->CAT(7)-E-D

8-->K(8)-E-NUMP -ET- CAT(8)-E-A -ET- (SUBA(8)-E-ORD -ou- SUBA(8)-E-CARD)

G-->CAT(G)-E-N -ET- (SUBN(G)-E-CN -ou- SUBN(G)=PN

$agr.ncount(1,6)

$agr.num(5,G)

9-->K(9)=AP -ET- (SUBA(9)=ADJ -ET- APOS(9)=POST -ou- SUBV(9)=PAP)

11-->CAT(11)-E-A ∩ SUBA(11)-E-ADJ ∩ UL(11) dans "next", "last", "previous", "following"

ZONE III
K(0)=NP

$egal.cat.subn.drv.val1.val2.sem1.sem2.semn.semc.gnr(o,g)

NB(0)=$int(4,5,8,g)
NCOUNT(0)=$int(4,5,8,g)

+ CAT(7)=D -ET- UL(7)="my" -ET- DRV(G)=VN -ET- POTVALE(G)-INC(N) !!! VAL1(0)=N n SEM1(0)=SEM1(G) n SEMN(7)=SEM1(G) !!!
!
+ CAT(7)=D -ET- UL(7)="my" -ET- DRV(G)=N -ET- POTVAL(G)-NINC(N) !!! VAL1(0)=POTVAL(0) n SEM1(0)=SEM(G) n SEMN(7)=SEM1(0) !!!

DEG(7)-=DEG0 !!! DEG(0)=DEG7 !!!

SF(1)=GRCP n SR(1)=QUAL
SF(2)=RUL n SR(2)=INT
SF(3)=GOVCP n SR(3)=PROX
SF(4)=GOVCP n SR(4)=QFIER
SF(5)=GOVCP n SR(5)=QFIER
SF(6)=RUL n SR(6)=INT
SF(7)=DES
SF(8)=GOVCP n SR(8)=QFIER
SF(9)=GOVCP
SF(10)=GOVCP n RL(10)=GRA0

+ POTSR(7)=QFIER !!! SR(7)-E-QFIER
!
+ ------------!!! SR(7)-E-DET !!! ?

+ SUBV(9)=PAP n $agr.semn/sem1(g,9) !!! SEM1(9)=$inter.semn/sem1(g,9) !!!
!   !!! n SEMN(0)=$inter.semn/sem1(g,9) n LR(G)=GRA1 !!!
!
+---------------------------------!!! LR(9)=GRA0 !!!

EXAMPLES AND REMARKS
CORPUS 3
- procedure
- the aircraft
- under shelter
- onto its wheels
- about two minutes
- especially in the summer
- almost all of the green apples

UIT
volume.1 pg.174
- the last three new draft recommendations of the amended recommendations

volume.1 pg.142
- the USSR
- the CCITT

CHART USE
System: STANDARD
Ariane Phase: AS
Language: ENG

Textes de tests - CORPUS CHECK - ZNP6 (pas d'adj. past part.)

Grammars: NOUNP1
rules: CARDNP (--> noeud n]8) (appel r{cursif = DNP, QNPM)
rules: DNP (--> noeud n]7) (appel r{cursif = NUMOFNP, NPQTF)
rules: ~~NUMOFNP (--> noeuds n]5 et 6) (appel r{cursif = QNPM)~~
Grammars: CALL (r}gles appel{es r{cursivement uniquement)
rules: NPQTF (--> noeud n]4) (appel r{cursif = QNPM)
rules: QNPM (--> noeud n]3)
Grammars: PREPN
rules: PREPNP (-> noeud 2)
rules: NPMD (-> noeud 1)

Remarque: les appels r{cursifs r{solvent les conditions d'existence:
4-->7 = NPQTF est appel{ r{cursivement par DNP uniquement
~~4-->5 = NPQTF n'appelle pas QNPM~~ (idem pour 4-->6)
~~5-->4 = NPQTF n'appelle pas QNPM~~ (idem pour 6-->4)
3-->4 -ou- 3-->5 -ou- 3-->8 = QNPM appel{ uniquement par CARDNP, ~~NUMOFNP~~ et NPQTF
~~8-->5 = CARDNP n'appelle pas NUMOFNP~~
6-->7 = ... pas vraiment impos{ (pb avec le Repr{sentant...)
(cf. AS-EI1 Cahier n]1 page 11)
11-> 8 = APN P2est appelé récursivement par CARONP

Planche BCG R-NPs6

R-NPs6:

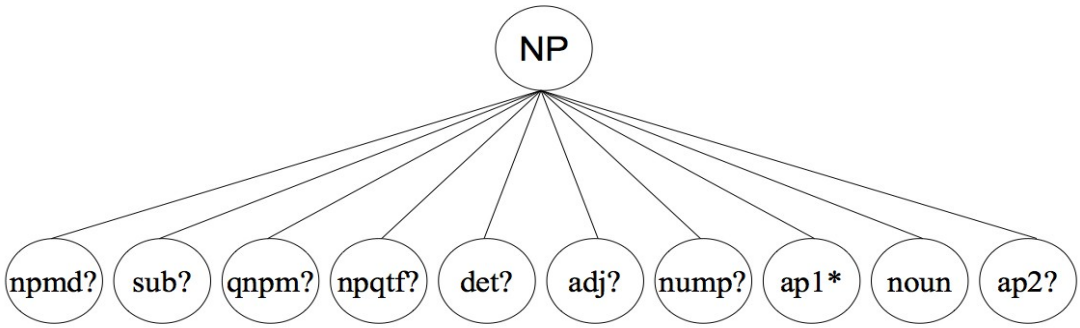
Type: NPs simple noun phrase

Handled cases: Simple noun phrase governed by a common noun or proper noun with possibility of determiner adjectives, NUMP, adjectives and prepositions.

References: R-NUMP1 for nump
R-AP1 for ap1 and ap2

Corresponding rules: CARDNP, DNP of grammar COMPLN (Structural Analysis)
NPQTF, QNPM of grammar CALL (Structural Analysis)
PREPNP, NPMD of grammar PREPN (Structural Analysis)

Main correspondence:

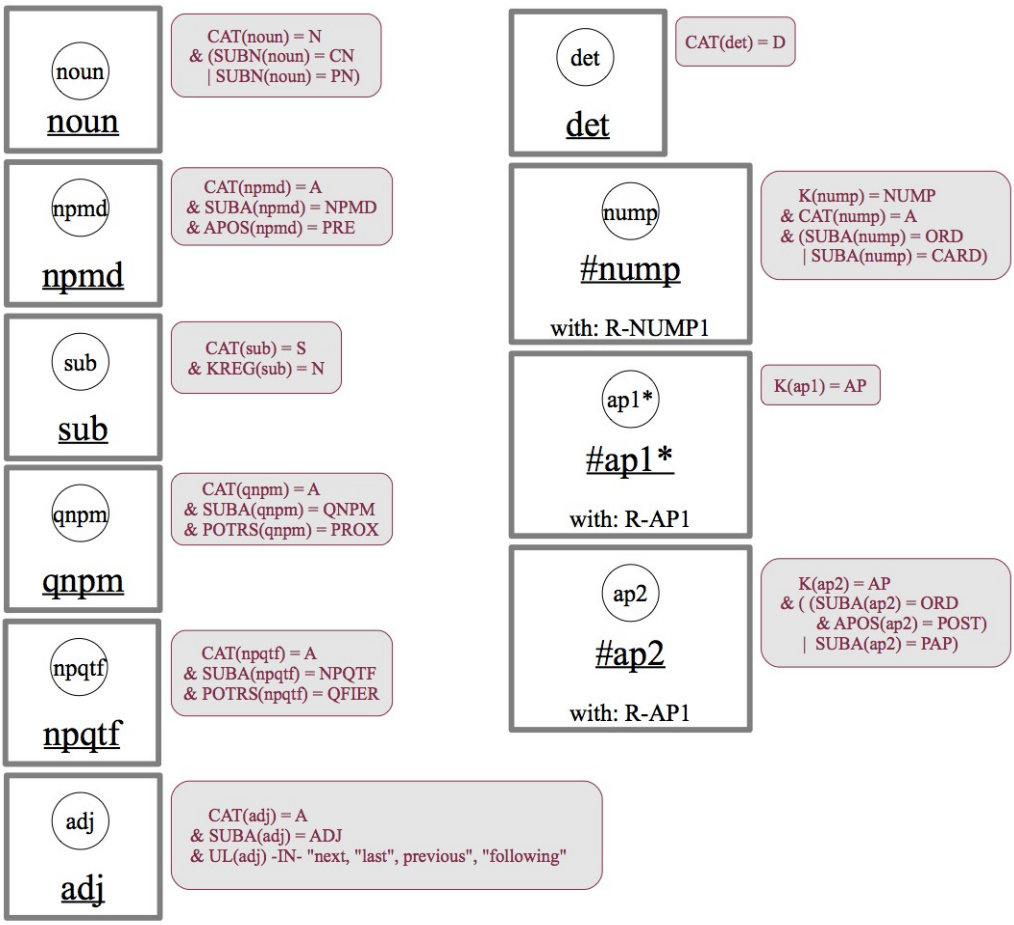


npmd?.sub?.qnpm?.npqtf?.det?.adj?.#nump?.#ap1*.noun.#ap2?

adj --> !qnpm & det & nump

R-NPs6:

Sub-correspondence:



R-NPs6:

Zone III:

```
K(NP) = NP
& $egal.cat.subcat.drv.val1.val2.sem1.sem2.semn.semnc.gnr(NP,noun)
& NB(NP) = $int(npqtf,adj,nump,noun)
& NCOUNT(NP) = $int(npqtf,adj,nump,noun)

if ( (CAT(det) = D) & (UL(det) = "my") &  (DRV(noun) = VN) & (POTVALE(noun) -INC- N) ) {
        VAL1(NP) = N
        & SEM1(NP) = SEM1(noun)
        & SEMN(det) = SEM1(noun)
} else if ( (CAT(det) = D) & (UL(det) = "my") &  (DRV(noun) = VN) & (POTVALE(noun) -NINC- N) ) {
        VAL1(NP) = POTVAL(NP)
        & SEM1(NP) = SEMN(noun)
        & SEMN(det) = SEMN(noun)
}

if (DEG(det) != DEG0) {
        DEG(NP) = DEG(det)
}

( (SF(npmd) = GRCP) & (SR(npmd) = QUAL) )
& ( (SF(sub) = RUL) & (SR(sub) = INT) )
& ( (SF(qnpm) = GOVCP) & (SR(qnpm) = PROX) )
& ( (SF(npqtf) = GOVCP) & (SR(npqtf) = QFIER) )
& (SF(det) = DES)
& ( (SF(nump) = GOVCP) & (SR(nump) = QFIER) )
& (SF(ap2) = GOVCP)
& ( (SF(ap1*) = GOVCP) & (SR(ap1*) = GRA0) )

if (POTRS(det) = QFIER) {
        SR(det) = QFIER
} else {
        SR(det) = DET
}

if ((SUBV(ap2) = PAP) & ($agr.semn/sem1(noun,ap2)) ) {
        SEM1(ap2) = $inter.semn/sem1(noun,ap2)
        & SEMN(NP) = $inter.semn/sem1(noun,ap2)
        & LR(noun) = GRA1
} else {
        LR(ap2) = GRA0
}
```

R-NPs6 (examples):

EXAMPLES AND COMMENTS

- procedure *(noun)*
- the aircraft *(det, noun)*
- under shelter *(sub, noun)*
- onto its wheels *(sub, det, noun)*
- about two minutes *(sub, nump, noun)*
- especially in the summer *(npmd, sub, det, noun)*
- the last three new preliminary recommendations *(det, adj, nump, ap1, ap1, noun)*

B'Vital
01 juin 88

Transcription de la planche BCG R-NPs6

```
( [NP(npmd?, sub?, qnpm?, npqtf?, det?, adj?, $nump?, $ap1*, noun, #ap2?),
npmd?.sub?.qnpm?.npqtf?.det?.adj?.#nump?.#ap1*.noun.#ap2?]
        & [adj --> !qnpm & det & nump1]
)


===>
// Sub-correspondences are described below
// One sub-correspondence for each parenthesis


( [noun, noun]
        & [CAT(noun) = N]
        & ( [SUBN(noun) = CN]
                | SUBN(noun) = PN]
        )
)
& ( [npmd, npmd]
        & [CAT(npmd) = A]
        & [SUBA(npmd) = NPMD]
        & [APOS(npmd) = PRE]
)
& ( [sub, sub]
        & [CAT(sub) = S]
        & [KREG(sub) = KREG]
)
& ( [qnpm, qnpm]
        & [CAT(qnpm) = A]
        & [SUBA(qnpm) = QNPM]
        & [POTRS(qnpm) = PREOX]
)
& ( [npqtf, npqtf]
        & [CAT(npqtf) = A]
        & [SUBA(npqtf) = NPQTF]
        & [POTRS(npqtf) = QFIER]
)
& ( [adj, adj]
        & [CAT(adj) = A]
        & [SUBA(adj) = ADJ]
        & [UL(adj) -ISINC- "next, "last", previous", "following"]
)
& ( [det, det]
        & [CAT(det) = D]
```

```
)
& ( [nump, #nump]
        & [K(nump) = NUMP]
        & [CAT(nump) = A]
        & ( [SUBA(nump) = ORD]
                | [SUBA(nump) = CARD]
        )
        & [references(R-NUMP1)]
)
& ( [ap1*, #ap1*]
        & [K($ap1) = AP]
        & [references(R-AP1)]
)
& ( [ap2, #ap2]
        & [K(ap2) = AP]
        & (     ( [SUBA(ap2) = ORD]
                        &[APOS(ap2) = POST]
                )
                | [SUBA(ap2) = CARD]
        )
        & [references(R-AP1)]
)

// ----------------------------------------
// Zone III starting here


& (
        [K(NP) = NP]
        & [$egal.cat.subcat.drv.val1.val2.sem1.sem2.semn.semnc.gnr(NP,noun)]
        & [NP(NP) = $int(npqtf,nump,noun)]
        & [NCOUNT(NP) = $int(npqtf,nump,noun)]


        if ( [CAT(det) = D] & [UL(det) = "my"] & [DRV(noun) = VN] & [POTVALE(noun)
-INC- N] ) {
                [VAL(NP) = N]
                & [SEM1(NP) = SEM1(noun)]
                & [SEMN(det) = SEM1(noun)]
        } else if ( [CAT(det) = D] & [UL(det) = "my"] & [DRV(noun) = 0] &
[POTVALE(noun) -INC- N] ) {
                [VAL(NP) = POTVAL(NP)]
                & [SEM1(NP) = SEM1(noun)]
                & [SEMN(det) = SEM1(noun)]
        }


        if ([DEG(det)!= DEG]) {
                [DEG(NP) = DEG(det)]
```

```
        }

        ( [SF(npmd) = GRCP] & [SR(npmd) = QUAL] )
        & ( [SF(sub) = RUL] & [SR(sub) = INT] )
        & ( [SF(qnpm) = GOVCP] & [SR(qnpm) = PROX] )
        & ( [SF(npqtf) = GOVCP] & [SR(npqtf) = QFIER] )
        & [SF(det) = DES]
        & ( [SF(nump) = GOVCP] & [SR(nump) = QFIER] )
        & [SF(ap2) = GOVCP]
        & ( [SF(ap1*) = GOVCP] & [SR(ap1*) = GRA0] )

        if ([POTRS(det) = QFIER]) {
                [SR(det) = QFIER]
        } else {
                [SR(7) = DET]
        }

        if ([SUBV(ap2) = PAP] & [$agr.semn/sem1(noun,ap2)] ) {
                [SEMN(ap2) = $inter.semn/sem1(noun,ap2)]
                & [SEMN(noun) = $inter.semn/sem1(noun,ap2)]
                & [LR(noun) = GRA1]
        } else {
                [LR(ap2) = GRA0]
        }
)
```

Annexe 5

Planche statique NPc9

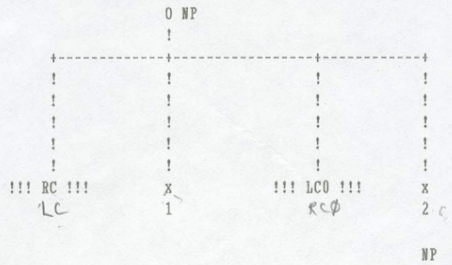
Static grammar: B'VITAL FIELD: TYPOLOGY:
Language: English

Chart number : NP9
Type : NPc Complex noun phrase
Subject treated: Noun phrase governed by a predicate whose semantic feature is collective dominating a noun phrase "of N".

Reference: (1- NP6
(2- NP6, NP5 ? NP5a, NP5b?
et toutes les NPc, sauf ~~elle-même~~. NP9)

Call:
1) by absorption:
2) for complementation:

ZONE 1



K		NP
CAT/SUBCAT	N/CN	
POTVAL		OF
SEMN/SUBSEMN	ABST/COLLECTIVE	
SF	GOV	GOVCP
SR		PART ? n'existe pas. CONT ?

ZONE II

K(0)=NP -ET- SEMN(0)=ABST -ET- SSABST(0)=COLLECT

CAT(1)=N -ET- SUBN(1)=CN -ET- SF(1)=GOV -ET- SEMN(1)=ABST -ET- SSABST(1)=COLLECT

K(2)=NP -ET- (CAT(2)=N -ET- SUBN(2)=CN) -ET- POTVAL(2)=OF -ET- (NB(2)=PLUR -ET- NCOUNT=COUNT -OU- NB(2)=SING -ET- NCOUNT=MASS)
-OU- (CAT(2)=R -ET- SUBR(2)=~~PRES~~ -ET- ROLE(2)=SUBJ -ET NB(2)=PLUR)
plutôt 'other'

Context

Noun phrase 2 is complete.

ZONE III

SF(2)=GOVCP n SR(2)=CONT

$egal.semn(0,2)

EXAMPLES AND REMARKS
CORPUS 3

- a collection of butterflies

- a pile of clothes

CHART USE
System: STANDARD
Ariane Phase: AS
Language: ENG

Textes de tests - CORPUS CHECK -

Grammars: COMPLN
rules: NPDNPCOL - la gram. COMPLN est en exhaustif libre - c'est la r}gle DEC qui assure le "non bouclage"

Planche BCG R-NPc9

R-NPc9:

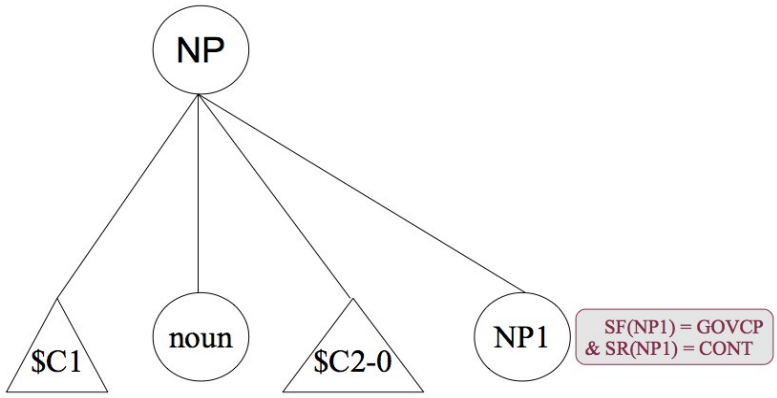
Type: NPc complex noun phrase

Handled cases: Noun phrase governed by a predicate whose semantic feature is collective, dominating a noun phrase "of N".

References: NPs6 for NP
NPs6 for NP1

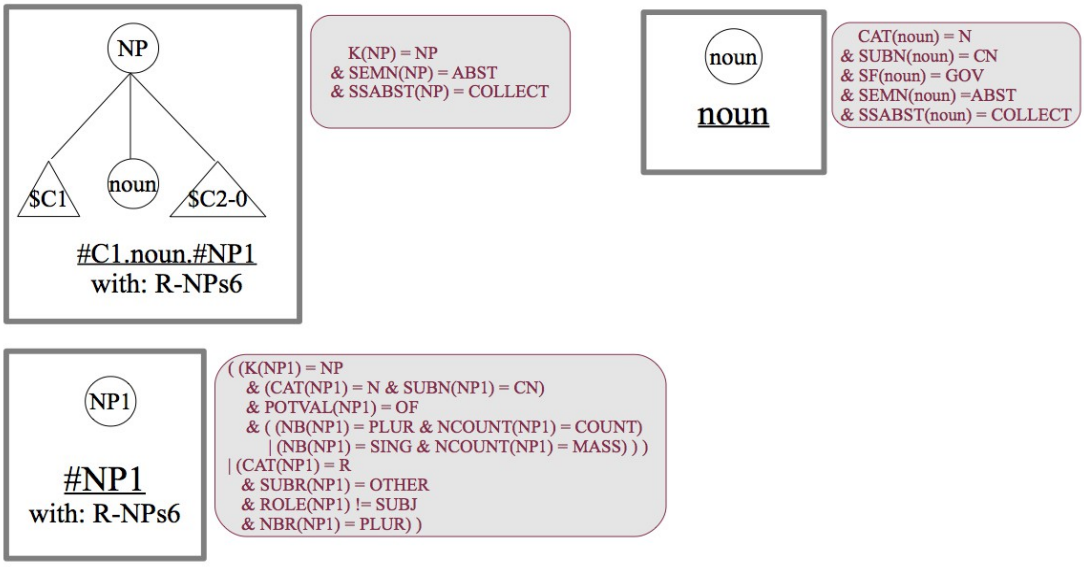
Corresponding rules: NPDNPCOL of grammar COMPLN (Structural Analysis)

Main correspondence:



#C1.noun.#NP1

Sub-correspondence:



R-NPc9:

Zone III:

SF(NP1) = GOVCP
& SR(NP1) = CONT
& $egal.SEMN(NP,NP1)

R-NPc9 (examples):

EXAMPLES AND COMMENTS

A collection of butterflies
A pile of clothes

B'Vital
01 juin 88

Transcription non graphique

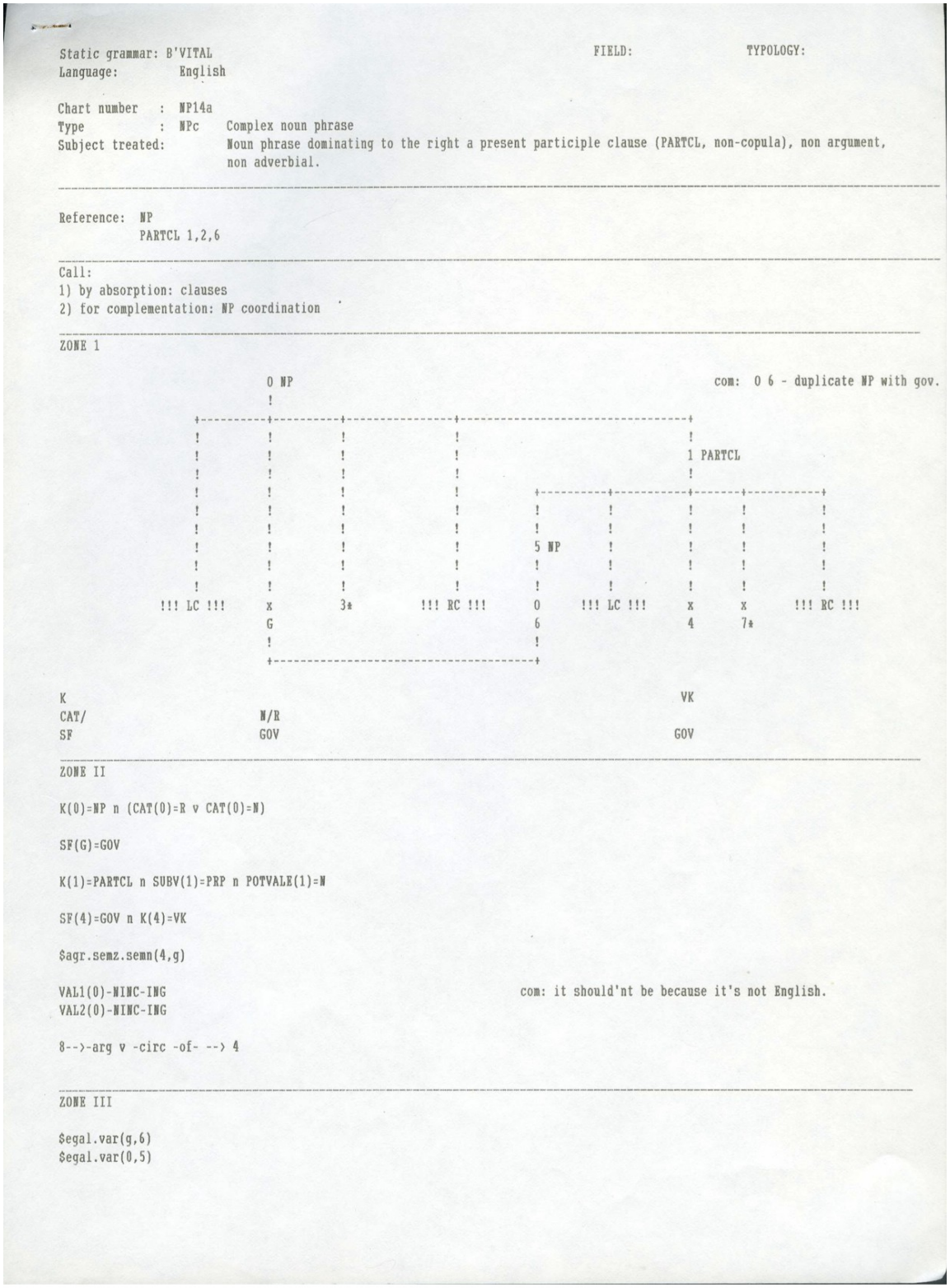
```
( [NP($C1, noun, $C2-0, NP1), #C1.noun.#NP1]
        & [SF(NP1) = GOVCP]
        & [SR(NP1) = CONT]
)

===>

( [NP($C1, noun, $C2-0, NP1), #C1.noun.#NP1]
        & [K(NP) = NP]
        & [SEMN(NP) = ABST]
        & [SSABST(NP) = COLLECT]
        & [references(R-NPs6)]
)
& ( [noun, noun]
        & [CAT(noun) = N]
        & [SUBN(noun) = NC]
        & [SF(noun) = GOV]
        & [SEMN(noun) = ABST]
        & [SSABST(noun) = COLLECT]
)
& ( [NP1, #NP1]
        & ( [K(NP1) = NP]
                & [CAT(NP1) = N]
                & [SUBN(NP1) = CN]
                & [POTVAL(NP1) = OF]
                & ( ( [NB(NP1) = PLUR] & [NCOUNT(NP1) = COUNT] )
                        | ( [NB(NP) = SING] & [NCOUNT(NP1) = MASS] )
                )
        )
        | ( [CAT(NP1) = R]
                & [SUBR(NP1) = PERS]
                & [ROLE(NP1) != SUBJ]
                & [NBR(NP1) = PLUR]
        )
        & [references(R-NPs6)]
)
& (
        [SF(NP1) = GOVCP]
        & [SR(NP1) = CONT]
        & [$egal.SEMN(NP,NP1)]
)
```

Annexe 6

Planche statique NPc14a



Static grammar: B'VITAL FIELD: TYPOLOGY:
Language: English

Chart number : NP14a
Type : NPc Complex noun phrase
Subject treated: Noun phrase dominating to the right a present participle clause (PARTCL, non-copula), non argument, non adverbial.

Reference: NP
PARTCL 1,2,6

Call:
1) by absorption: clauses
2) for complementation: NP coordination

ZONE 1

ZONE II

K(0)=NP n (CAT(0)=R v CAT(0)=N)

SF(G)=GOV

K(1)=PARTCL n SUBV(1)=PRP n POTVALE(1)=N

SF(4)=GOV n K(4)=VK

$agr.semz.semn(4,g)

VAL1(0)-MINC-ING com: it should'nt be because it's not English.
VAL2(0)-MINC-ING

8-->-arg v -circ -of- --> 4

ZONE III

$egal.var(g,6)
$egal.var(0,5)

```
VALE(1)=N
VALE2(1)=N

+ SYM(4)=SYM01 n $narg1(7)
! n NUM(0)=PLUR
! n $agr.semn/semz,sem1(0,4) !!! LR(1)=GRA01 n SF(1)=ATG n SEMZ(1)=$inter.semn/semz,sem1(0,4)            !!!
!                            !!! n SEMZ(4)=$inter.semn/semz/sem1(0,4) n SF(5)=SUJ n LR(5)=ARG01          !!!
!                            !!! n SEMN(0)=$inter.semn/semz,sem1(0,4) n SEMN(5)=$inter.semn/semz,sem1(0,4) !!!
!
+ $agr.semz,semn(4,g)
  n $narg0(7)         !!! SF(5)=SUJ n LR(5)=ARG0 n SEMZ(1)=$inter.sem2/semn(4,0) n SEMZ(4)=$inter.semz/semn(4,g)   !!!
                      !!! n SEMN(5)=$inter.semz/semn(4,0) n SEMN(0)=$inter.semz/semn(4,0) n LR(1)=GRA0 n SF(1)=ATG !!!
```

com: by reducing semantic features of already constructed groups, it will be necessary to modify semantics of already contructed infinitive or relative clauses, ie test on 3 intersection of semantic features if clause.

EXAMPLES AND REMARKS
FAS VI.II ANNEX. D pg.147
- ~~those/signals/forms/a signal list~~
- documents covering the whole system definition

FAS VI.13. pg. 32
- Output conveying information about the state of an input

FAS VI.9. pg. 59
- user action requesting call establishment

FAS VI.10. pg. 65
- the signal interworking between process instances

CHART USE
System: STANDARD
Ariane Phase: AS
Language: ENG

Textes de tests - CORPUS CHECK -

Grammars:
rules:

Planche BCG R-NPc14a

R-NPc14a:

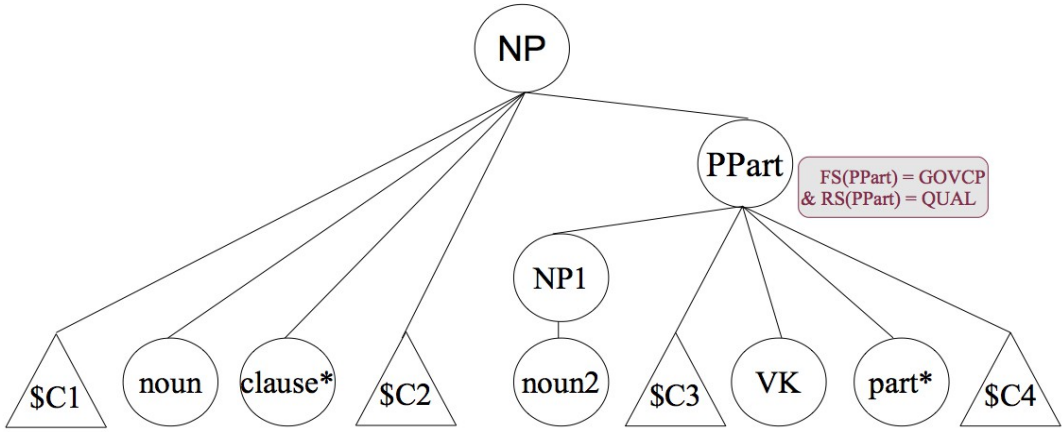
Type: NPc Complex noun phrase

Handled cases: Noun phrase dominating to the right a present participle clause (PARTCL), (non-copula), non-argument, non-adverbial

References: NPs6 for NP
PARTCL1 for PPart

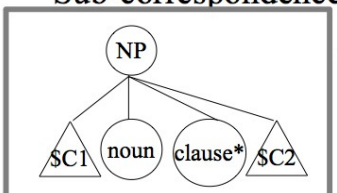
Corresponding rules:

Main correspondence:



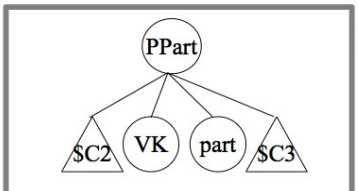
#C1.noun.#clause*.#C2.#C3.#VK.#part*.#C4

Sub-correspondence:



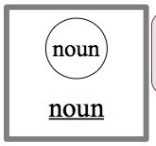
#C1.noun.#clause*.#C2
#C3.#VK.#part*.#C4
with: R-NPs6

K(NP) = NP
& (CAT(NP) = R
| CAT(NP) = N)
& VAL1(NP) -NINC- ING
& VAL2(NP) -NINC- ING
& CAT(noun) = N
& SUBN(noun) = CN
& SF(noun) = GOV
& $arg.semz.semn(VK,noun)



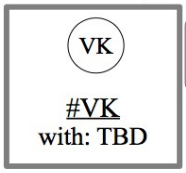
#C3.#VK.#part*.#C4
with: PARTCL1

K(PPart) = PARTCL
& SUBV(PPart) = PRP
& POTVALE(PPart) = N
& K(VK) = VK
& SUBV(VK) = PRP
& FS(VK) = GOV
& $arg.semz.semn(VK,noun)



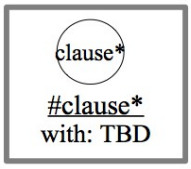
noun

CAT(noun) = N
& SUBN(noun) = CN
& FS(noun) = GOV

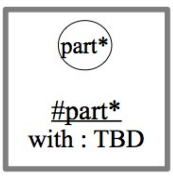


#VK
with: TBD

K(VK) = VK
& SUBV(VK) = PRP
& FS(VK) = GOV



#clause*
with: TBD



#part*
with : TBD

R-NPc14a:

Zone III:

```
$egal.var(noun,noun2)
$egal.var(NP,NP1)
VALE(PPart) = N
VALE2(PPart) = N

if ( (SYM(VK) = SYM01) & (narg1(part)) &  (NUM(NP) = PLUR) & ($agr.semn/semz,sem1(NP,VK)) ) {
        LR(PPart) = GRA01
        & SF(PPart) = ATG
        & SEMZ(PPart) = $inter.semn/semz,sem1(NP,VK)
        & SEMZ(VK) = $inter.semn/semz,sem1(NP,VK)
        & SF(NP1) = SUJ
        & LR(NP1) = ARG01
        & SEMN(NP) = $inter.semn/semz,sem1(NP,VK)
        & SEMN(NP1) = $inter.semn/semz,sem1(NP,VK)
} else if ($agr.semz.semn(VK, noun) & ($narg0(part)) ) {
        SF(NP1) = SUJ
        & LR(NP1) = ARG0
        & SEMZ(PPart) = $inter.sem2/semn(VK, NP)
        & SEMZ(VK) = $inter.semz/semn(VL, noun)
        & SEMN(NP1) = $inter.semz/semn(VK, NP)
        & SEMN(NP) = $inter.semz/semn(VK, NP)
        & LR(PPart) = GRA0
        & SF(PPart) = ATG
}
```

R-NPc14a (examples):

EXAMPLES ET COMMENTS

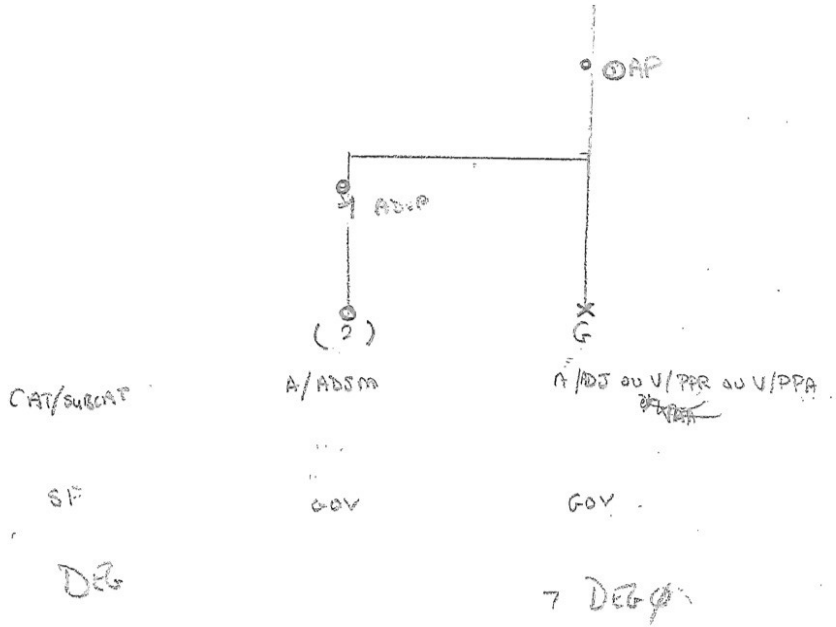
- Documents covering the whole system definition
- Output conveying information about the state of an input
- User action requesting call establishment
- The signal interworking between process instances

Sylviane Chappuy
Version février 1990
Remise en forme août 2011

Annexe 7

Planche statique AP1

ZONE 1


①AP
ADJP
(2)
G
CAT/SUBCAT
A/ADJM
A/ADJ ou V/PPR ou V/PPA
SP
GOV
GOV
DEG
7 DEG

ZONE 2

ZA 2 ssi DEG(G) ≠ DEG0 .

FS(G) = GOV

K(0) = AP , CAT(0) = A ,

K(1) = ADVP, SUBA(1) = ADEM , CAT(1) = A , SF(1) = GOVCP , SR(1) = QUAL

SUBA(1) := ADEM , CAT(1) = A , SF(1) = GOV , DEG(G) = DEG0

ZCOR

DEG(G) = C. COMP

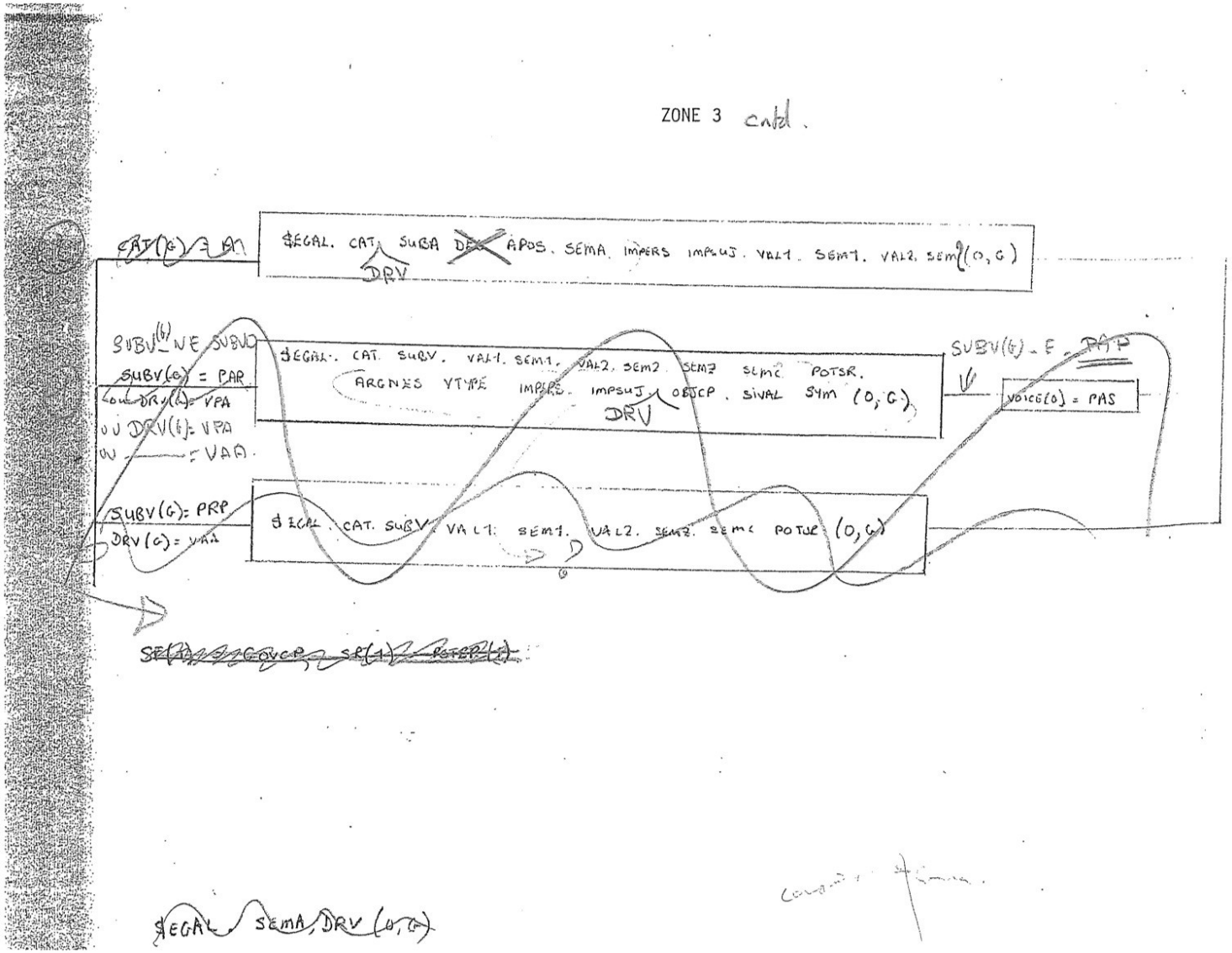
DEG(0) = COMP, LH(2) = 'MORE'
DEG(1) = COMP , DEG(2) = COMP

DEG(G) = C. SUP

DEG(0) = SUP, LH(2) = 'MOST'
DEG(1) = SUP . DEG(2) = COMP.

P.TO

ZONE 3 cntd.


$EGAL. CAT. SUBA APOS. SEMA. IMPERS IMPSUJ. VAL1. SEM1. VAL2. SEM2 (0, G)
DRV
SUBV(6) NE SUBVO
SUBV(G) = PAR
DRV(6) = VPA
DRV(6) = VPA
= VAA.
$EGAL. CAT. SUBV. VAL1. SEM1. VAL2. SEM2. SEM3 SEMC POTSR. ARGNES VTYPE IMPERS. IMPSUJ. OBJCP. SIVAL SYM (0, G)
DRV
SUBV(6) = E
VOICE(6) = PAS
SUBV(G) = PRP
DRV(G) = VAA
$EGAL. CAT. SUBV. VAL1. SEM1. VAL2. SEM2. SEMC POTSR (0, G)
$EGAL SEMA DRV (0, G)

rider
better
best.

opened
interesting
applicable
restrictive

? d'après la zone 1 on a $deg \neq deg\phi$ – zone 2

Planche BCG R-AP1

R-AP1:

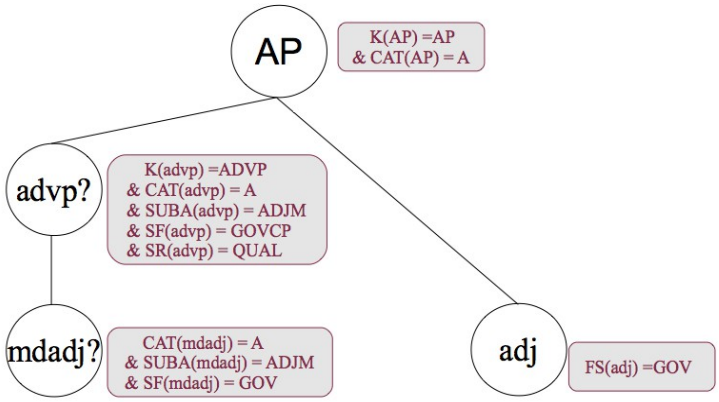
Type: Simple adjective phrase

Handled cases: -er camparative adjectives and simple adjective

References:

Corresponding rules: LVK of grammar ELEV (Structural Analysis)
ADJP1 of grammar COMPAP (Structural Analysis)

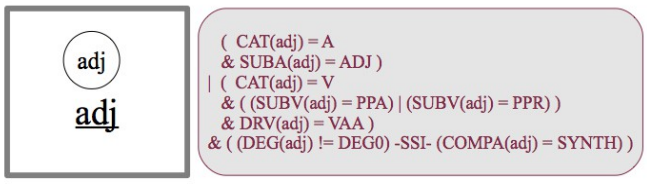
Main correspondence:



adj

mdadj -SSI- (DEG(adj) != DEG0)

Sub-correspondence:



R-AP1:

Zone III:

```
if (DEG(adj) = COMP) {
        DEG(AP) = COMP
        & UL(mdadj) = 'MORE'
        & DEG(advp) = COMP
        & DEG(adj) = COMP
} else if  (DEG(adj) = SUP) {
        DEG(AP) = SUP
        & UL(mdadj) = 'MOST'
        & DEG(advp) = SUP
        & DEG(adj) = COMP
}
```

R-AP1 (examples):

EXAMPLES AND COMMENTS

- richer
- better
- best
- opened
- interesting
- applicable

B'Vital
revu le 4 janvier 89

Annexe 8

Planche statique NUMP1

Static grammar: B'VITAL FIELD: TYPOLOGY:
Language: English

Chart number : NUMP1 (Simplified treatment)
Type : NUMPs Simple numerical phrase
Subject treated: Numerical phrase formed with a cardinal (in digits)
ordinal

Reference: Charts: None

Call:
1) by absorption:
2) for complementation:

ZONE 1

0 NUMP
!
!
!
!
!
!
x
G

CAT/SUBCAT A/CARD v ORD
SF GOV
TYPOG DIGIT

ZONE II

CAT(G)-E-A n (SUBA(G)-E-CARD v SUBA(G)-E-ORD) n TYPOG(G)-E-DIGIT

ZONE III

K(0)-E-NUMP n $egal.cat.suba.typog.num(0,g)

SF(G)-E-GOV

EXAMPLES AND REMARKS
CORPUS 3
- at least 25 volts
- doors 1708 and 2708
- approximately 85 bars
- figure 14
- to exceed 50 kts

OTHERS
- 1st
- 2nd
- 3rd
- 5th

CHART USE
System: STANDARD
Ariane Phase: AS
Language: ENG

Textes de tests - CORPUS CHECK -

Grammars: NUMBER
rules: CARD
Grammars: ELEV
rules: LVK

Planche BCG R-NUMP1

R-NUMP1:

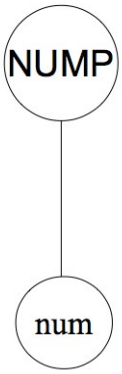
Type: Simple numerical phrase

Handled cases: Numerical phrase formed with a cardinal or ordinal (in digits)

References:

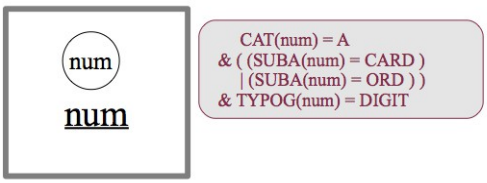
Corresponding rules: LVK of grammar ELEV (Structural Analysis)

Main correspondence:



num

Sub-correspondence:



R-NUMP1:

Zone III:

K(NUMP) = NUMP
& $egal.cat.subcat.typog.num(NUMP, num)
& SF(num) = GOV

R-NUMP1 (examples):

EXAMPLES AND COMMENTS

- at least **25** volts
- doors **1708** and **2708**

B'Vital
7 juin 88

Annexe 9

DTD du format de stockage interne des planches

```
<?xml version="1.0" encoding='UTF-8' ?>

<!-- Modifications de la DTD selon la nouvelle proposition de formalisme -->

<!--General architecture-->
<!ELEMENT board (meta, rule)>

<!--**************************************************-->
<!--Content of the meta tag-->
<!ELEMENT meta (board_name, type, handled_cases, references, corresponding_rules, examples, comments)>

<!ELEMENT board_name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT handled_cases (#PCDATA)>
<!ELEMENT references (reference*)>
<!ELEMENT corresponding_rules (grammar*)>
<!ELEMENT examples (#PCDATA)>
<!ELEMENT comments (#PCDATA)>

<!ELEMENT reference (ref+)>
<!ATTLIST reference node CDATA #REQUIRED>
<!ELEMENT ref (#PCDATA)>
```

<!ELEMENT grammar (robra_rule+)>

<!ATTLIST grammar name CDATA #REQUIRED phase (analysis|generation| analysis_and_generation) "analysis_and_generation">

<!ELEMENT robra_rule (#PCDATA)>

<!--**-->

<!--Content of the rule tag-->

<!--General organization : a rule is divided in three zones (the main correspondance, all the sub correspondances and the zoneIII)-->

<!ELEMENT rule (main_corr, sub_corr, zoneIII)>

<!--The main correspondance is made of a tree and the corresponding string. The tree starts with an element root, which is unique in the file, and corresponds to the type of grammatical group described by the board. -->

<!ELEMENT main_corr (root, string, existence_conditions?)>

<!ELEMENT root (tag?, node+)> <!--The root works almost the same as all the other nodes of the board. It has one single tag <tag> contains all the decorations and has at least one node as a child. -->

<!ATTLIST root type CDATA #REQUIRED>

<!ELEMENT string (#PCDATA)> <!--Contains a string-->

<!ELEMENT existence_conditions (existence_condition+)> <!-- Contains several <existence_condition> elements, one for each condition -->

<!ELEMENT existence_condition (implication+)> <!-- Each child of the element contains information on the existence or non-existence of one node of the structure. -->

<!ATTLIST existence_condition if_node CDATA #REQUIRED> <!-- The attribute of the element allows to express what node the existence of other nodes depends upon. -->

<!ELEMENT implication EMPTY>

<!ATTLIST implication exists (yes|no) #REQUIRED node CDATA #REQUIRED> <!-- Each implication node allows to express the existence or non-existence of one node. -->

<!--The sub-correspondances part is made of several sub-correspondances, one for each node of the main tree. -->

<!ELEMENT sub_corr (sub+)>

<!--A sub-correspondance is made of a tree and the corresponding string. The tree starts with an element <sub-root>. -->

<!ELEMENT sub (sub-root, string)>

<!ELEMENT sub-root (tag?, node*)> <!--The sub-root works almost the same as the main-corr <root> element. It has one single tag <tag> contains all the decorations and can have as many children as needed, even 0. -->

<!ATTLIST sub-root name CDATA #REQUIRED reference CDATA #IMPLIED> <!-- The <sub-root> bears the references if the sub-correspondence has any. -->

<!-- The <string> element is described above. -->

<!--The nodes contain zoneII type information-->

<!ELEMENT node (tag?, node*)> <!-- The <node> element contains informations on its decorations in the <tag> element. It can also contain other node when it is not a terminal node. -->

<!ATTLIST node name CDATA #REQUIRED occurrence (0or1|1orMore|0orMore) #IMPLIED context (true|false) "false"> <!-- The <node> element has two attributes : one name attribute to express the node name and a context attribute which is false by default but is true when the node is a context node. -->

<!-- The <tag> element contains one or several constraint. They can be articulated using logical operators-->

<!ELEMENT tag ((constraint, (or, constraint)?)+)>

<!-- The <zoneIII> element can express internodal constraints but also conditional constraints-->

<!ELEMENT zoneIII ((internodal_constraint*, (if, else_if*, else?)*)+)>

<!ELEMENT internodal_constraint (((decoration|predicate|assignment), (or|and)?)+)> <!--works like any constraint. cf. infra-->

<!ELEMENT if (string_constraint, structure_constraint)> <!--The if element expressed conditions on the string. If they are true, then the constraints on the structure must be true too-->

<!ELEMENT else_if (string_constraint, structure_constraint)> <!--The else_if element works as the if element-->

<!ELEMENT else (structure_constraint)> <!--The else element contains constraints on the structure, without any conditions on the string-->

<!ELEMENT string_constraint (((decoration|predicate), (or|and)?)+)> <!--works like any constraint. cf. infra-->

<!ELEMENT structure_constraint ((constraint, (or)?)+)> <!--The structure constraint part is a list of constraints-->

<!--A constraint is made as such : -->

<!ELEMENT constraint (((decoration|predicate|assignment|constraint), ((or|and), (decoration|predicate|assignment|constraint))?)+)> <!--A constraint is a list of information about the decorations of the node. They can be articulated using logical operators. Predicate can also be used instead of decoration. Element contraint allows to write complex expressions that imply using logical operator and parenthesis -->

<!ELEMENT decoration EMPTY>

<!ATTLIST decoration name CDATA #REQUIRED node CDATA #REQUIRED value CDATA #REQUIRED> <!--The element decoration is only made of three attributes : one for the type of decoration described, one for its value, and the node attribute to say what element the constraint is linked to-->

<!ELEMENT assignment EMPTY> <!-- An assignment works the same as a decoration, except that the value given to the node is the value of another node. So no attribute value and the value is expressed by a <decoration> element. -->

<!ATTLIST assignment name CDATA #REQUIRED node CDATA #REQUIRED name_value CDATA #REQUIRED node_value CDATA #REQUIRED>

<!ELEMENT predicate (applied_to+)> <!-- Contains one or more elements <applied_to> with the node the predicate applies to -->

<!ATTLIST predicate name CDATA #IMPLIED node CDATA #IMPLIED value CDATA #REQUIRED> <!--A predicate can have an attribute when it also works as a decoration, that is when the value returned by the predicate is the value of a decoration. Required attribute value contains the predicate-->

<!ELEMENT applied_to EMPTY>

<!ATTLIST applied_to node CDATA #REQUIRED> <!-- Attribute allows to say what node the predicate applies to -->

<!ELEMENT or EMPTY> <!--Simple logical operator-->

<!ELEMENT and EMPTY> <!--Simple logical operator-->

Annexe 10

Exemple de planche stockée en XML : R-NPs6

```
<?xml version="1.0" ?>
<!DOCTYPE board SYSTEM "board.dtd">
<board>
	<meta>
		<board_name>R-NPs6</board_name>
		<type> NPs simple noun phrase</type>
		<handled_cases> Simple noun phrase governed by a common noun or proper noun with possibility of determiner adjectives, NIMP, adjectives and prepositions. </handled_cases>
		<references>
			<reference node="nump"><ref>R-NUMP1</ref></reference>
			<reference node="ap1"><ref>R-AP1</ref></reference>
			<reference node="ap2"><ref>R-AP1</ref></reference>
		</references>
		<corresponding_rules>
			<grammar name="COMPLN" phase="analysis"><robra_rule>CARDNP</robra_rule><robra_rule>DNP</robra_rule></grammar>
			<grammar name="CALL" phase="analysis"><robra_rule>NPQTF</robra_rule><robra_rule>QNPM</robra_rule></grammar>
			<grammar name="PREPN" phase="analysis"><robra_rule>PREPNP</robra_rule><robra_rule>NPMD</robra_rule></grammar>
		</corresponding_rules>
		<examples>
			crève-canot pneumatique
			porte bouteille (pb avec porte bouteille d'azote)
			radio-altimètre

			ultra-son
			contre-fiche
			demi-empennage

			auto-maintien (freineur)
			turbo-compresseur (régulateur, réfrigérateur)
			servo-commande (moteur)
			radio-altimètre (navigation)
		</examples>
		<comments/>
	</meta>
	<rule>
		<main_corr>
			<root type="NP">
				<node name="npmd" occurrence="0or1"/>
				<node name="sub" occurrence="0or1"/>
				<node name="qnpm" occurrence="0or1"/>
				<node name="npqtf" occurrence="0or1"/>
				<node name="det" occurrence="0or1"/>
				<node name="adj" occurrence="0or1"/>
```

```
                        <node name="nump" occurrence="0or1"/>
                        <node name="ap1*" occurrence="0orMore"/>
                        <node name="noun"/>
                        <node name="ap2" occurrence="0or1"/>
                    </root>
                    <string>npmd?.sub?.qnpm?.npqtf?.det?.adj?.#nump?.#ap1*.noun.#ap2?
</string>
                    <existence_conditions>
                        <existence_condition if_node="adj">
                            <implication exists="no" node="qnpm"/>
                            <implication exists="yes" node="det"/>
                            <implication exists="yes" node="nump"/>
                        </existence_condition>
                    </existence_conditions>
                </main_corr>
                <sub_corr>
                    <sub>
                        <sub-root name="noun">
                            <tag>
                                <constraint><decoration name="CAT" node="noun"
value="N"/></constraint>
                                <constraint>
                                    <constraint><decoration name="SUBCAT"
node="noun" value="CN"/></constraint>
                                    <or/>
                                    <constraint><decoration name="SUBCAT"
node="noun" value="PN"/></constraint>
                                </constraint>
                            </tag>
                        </sub-root>
                        <string>noun</string>
                    </sub>
                    <sub>
                        <sub-root name="npmd">
                            <tag>
                                <constraint><decoration name="CAT" node="npmd"
value="A"/></constraint>
                                <constraint><decoration name="SUBCAT"
node="npmd" value="NPMD"/></constraint>
                                <constraint><decoration name="APOS"
node="npmd" value="PRE"/></constraint>
                            </tag>
                        </sub-root>
                        <string>npmd</string>
                    </sub>
                    <sub>
                        <sub-root name="sub">
                            <tag>
                                <constraint><decoration name="CAT" node="sub"
value="S"/></constraint>
                                <constraint><decoration name="KREG" node="sub"
value="N"/></constraint>
                            </tag>
```

```
                    </sub-root>
                    <string>sub</string>
                </sub>
                <sub>
                    <sub-root name="qnpm">
                        <tag>
                            <constraint><decoration name="CAT" node="qnpm"
value="A"/></constraint>
                            <constraint><decoration name="SUBCAT"
node="qnpm" value="QNPM"/></constraint>
                            <constraint><decoration name="POTRS"
node="qnpm" value="PROX"/></constraint>
                        </tag>
                    </sub-root>
                    <string>qnpm</string>
                </sub>
                <sub>
                    <sub-root name="npqtf">
                        <tag>
                            <constraint><decoration name="CAT" node="npqtf"
value="A"/></constraint>
                            <constraint><decoration name="SUBCAT"
node="npqtf" value="NPQTF"/></constraint>
                            <constraint><decoration name="POTRS"
node="npqtf" value="QFIER"/></constraint>
                        </tag>
                    </sub-root>
                    <string>npqtf</string>
                </sub>
                <sub>
                    <sub-root name="adj">
                        <tag>
                            <constraint><decoration name="CAT" node="adj"
value="A"/></constraint>
                            <constraint><decoration name="SUBCAT"
node="adj" value="ADJ"/></constraint>
                            <constraint><decoration name="UL" node="adj"
value="next|last|previous|following"/></constraint>
                        </tag>
                    </sub-root>
                    <string>adj</string>
                </sub>
                <sub>
                    <sub-root name="det">
                        <tag>
                            <constraint><decoration name="CAT" node="det"
value="D"/></constraint>
                        </tag>
                    </sub-root>
                    <string>det</string>
                </sub>
                <sub>
                    <sub-root name="nump" reference="R-NUMP1">
```

<tag>
<constraint><decoration name="K" node="nump" value="NUMP"/></constraint>
<constraint><decoration name="CAT" node="nump" value="A"/></constraint>
<constraint>
<constraint><decoration name="SUBCAT" node="nump" value="ORD"/></constraint>
<or/>
<constraint><decoration name="SUBCAT" node="nump" value="CARD"/></constraint>
</constraint>
</tag>
</sub-root>
<string>nump</string>
</sub>
<sub>
<sub-root name="ap1*" reference="R-AP1">
<tag>
<constraint><decoration name="K" node="ap1" value="AP"/></constraint>
</tag>
</sub-root>
<string>ap1*</string>
</sub>
<sub>
<sub-root name="ap2" reference="R-AP1">
<tag>
<constraint><decoration name="K" node="ap2" value="AP"/></constraint>
<constraint>
<constraint>
<constraint><decoration name="SUBCAT" node="ap2" value="ORD"/></constraint>
<and/>
<constraint><decoration name="APOS" node="ap2" value="POST"/></constraint>
</constraint>
<or/>
<constraint><decoration name="SUBCAT" node="ap2" value="PAP"/></constraint>
</constraint>
</tag>
</sub-root>
<string>ap2</string>
</sub>
</sub_corr>
<zoneIII>
<internodal_constraint><decoration name="K" node="NP" value="NP"/></internodal_constraint>
<internodal_constraint><predicate value="egal.cat.subcat.drv.val1.val2.sem1.sem2.semn.semnc.gnr"><applied_to node="NP"/><applied_to node="noun"/></predicate></internodal_constraint>

```
                        <internodal_constraint><predicate name="NB" node="NP"
value="$int"><applied_to node="npqtf"/><applied_to node="adj"/><applied_to
node="nump"/><applied_to node="noun"/></predicate></internodal_constraint>
                        <internodal_constraint> <predicate name="NCOUNT" node="NP"
value="$int"><applied_to node="npqtf"/><applied_to node="adj"/><applied_to
node="nump"/><applied_to node="noun"/></predicate></internodal_constraint>
                        <if>
                                <string_constraint>
                                        <decoration name="CAT" node="det" value="D"/>
                                        <and/>
                                        <decoration name="UL" node="det" value="my"/>
                                        <and/>
                                        <decoration name="DRV" node="noun" value="VN"/>
                                        <and/>
                                        <decoration name="POTVALE" node="noun" value="INC-
N"/>
                                </string_constraint>
                                <structure_constraint>
                                        <constraint><decoration name="VAL1" node="NP"
value="N"/></constraint>
                                        <constraint><assignment name="SEM1" node="NP"
name_value="SEM1" node_value="noun"/></constraint>
                                        <constraint><assignment name="SEMN" node="det"
name_value="SEM1" node_value="noun"/></constraint>
                                </structure_constraint>
                        </if>
                        <else_if>
                                <string_constraint>
                                        <decoration name="CAT" node="det" value="D"/>
                                        <and/>
                                        <decoration name="UL" node="det" value="my"/>
                                        <and/>
                                        <decoration name="DRV" node="noun" value="VN"/>
                                        <and/>
                                        <decoration name="POTVALE" node="noun" value="NINC-
N"/>
                                </string_constraint>
                                <structure_constraint>
                                        <constraint><assignment name="VAL1" node="NP"
name_value="POTVAL" node_value="NP"/></constraint>
                                        <constraint><assignment name="SEM1" node="NP"
name_value="SEM1" node_value="noun"/></constraint>
                                        <constraint><assignment name="SEMN" node="det"
name_value="SEM1" node_value="noun"/></constraint>
                                </structure_constraint>
                        </else_if>
                        <if>
                                <string_constraint>
                                        <decoration name="DEG" node="det" value="NOT-DEG0"/>
                                </string_constraint>
                                <structure_constraint>
                                        <constraint><assignment name="DEG" node="NP"
name_value="DEG" node_value="det"/></constraint>
```

```
                    </structure_constraint>
                </if>
                <internodal_constraint>
                    <decoration name="SF" node="npmd" value="GRCP"/>
                    <and/>
                    <decoration name="SR" node="npmd" value="QUAL"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="sub" value="RUL"/>
                    <and/>
                    <decoration name="SR" node="sub" value="INT"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="qnpm" value="GOVCP"/>
                    <and/>
                    <decoration name="SR" node="qnpm" value="PROX"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="npqtf" value="GOVCP"/>
                    <and/>
                    <decoration name="SR" node="npqtf" value="QFIER"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="det" value="DES"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="nump" value="GOVCP"/>
                    <and/>
                    <decoration name="SR" node="nump" value="QFIER"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="ap2" value="GOVCP"/>
                </internodal_constraint>
                <internodal_constraint>
                    <decoration name="SF" node="ap1*" value="GOVCP"/>
                    <and/>
                    <decoration name="SR" node="ap1*" value="GRA0"/>
                </internodal_constraint>
                <if>
                    <string_constraint>
                        <decoration name="POTRS" node="det" value="QFIER"/>
                    </string_constraint>
                    <structure_constraint>
                        <constraint><decoration name="SR" node="det"
value="QFIER"/></constraint>
                    </structure_constraint>
                </if>
                <else>
                    <structure_constraint>
                        <constraint><decoration name="SR" node="det"
value="DET"/></constraint>
                    </structure_constraint>
                </else>
```

```
                        <if>
                                <string_constraint>
                                        <decoration name="SUBV" node="ap2" value="PAP"/>
                                        <and/>
                                        <predicate value="$agr.semn/sem1"><applied_to
node="noun"/><applied_to node="ap2"/></predicate>
                                </string_constraint>
                                <structure_constraint>
                                        <constraint><predicate name="SEM1" node="ap2"
value="$inter.semn/sem1"><applied_to node="noun"/><applied_to
node="ap2"/></predicate></constraint>
                                        <constraint><predicate name="SEMN" node="NP"
value="$inter.semn/sem1"><applied_to node="noun"/><applied_to
node="ap2"/></predicate></constraint>
                                        <constraint><decoration name="LR" node="noun"
value="GRA1"/></constraint>
                                </structure_constraint>
                        </if>
                        <else>
                                <structure_constraint>
                                        <constraint><decoration name="SR" node="ap2"
value="GRA0"/></constraint>
                                </structure_constraint>
                        </else>
                </zoneIII>
        </rule>
</board>
```