Université Paris 7-Denis Diderot

Serge Fleury

POLAS FRITAS
Prototypes Oriented Language Has Freed Us

La Programmation à Prototypes, un outil pour une linguistique expérimentale: mise en oeuvre de représentations évolutives des connaissances pour le traitement automatique du langage naturel

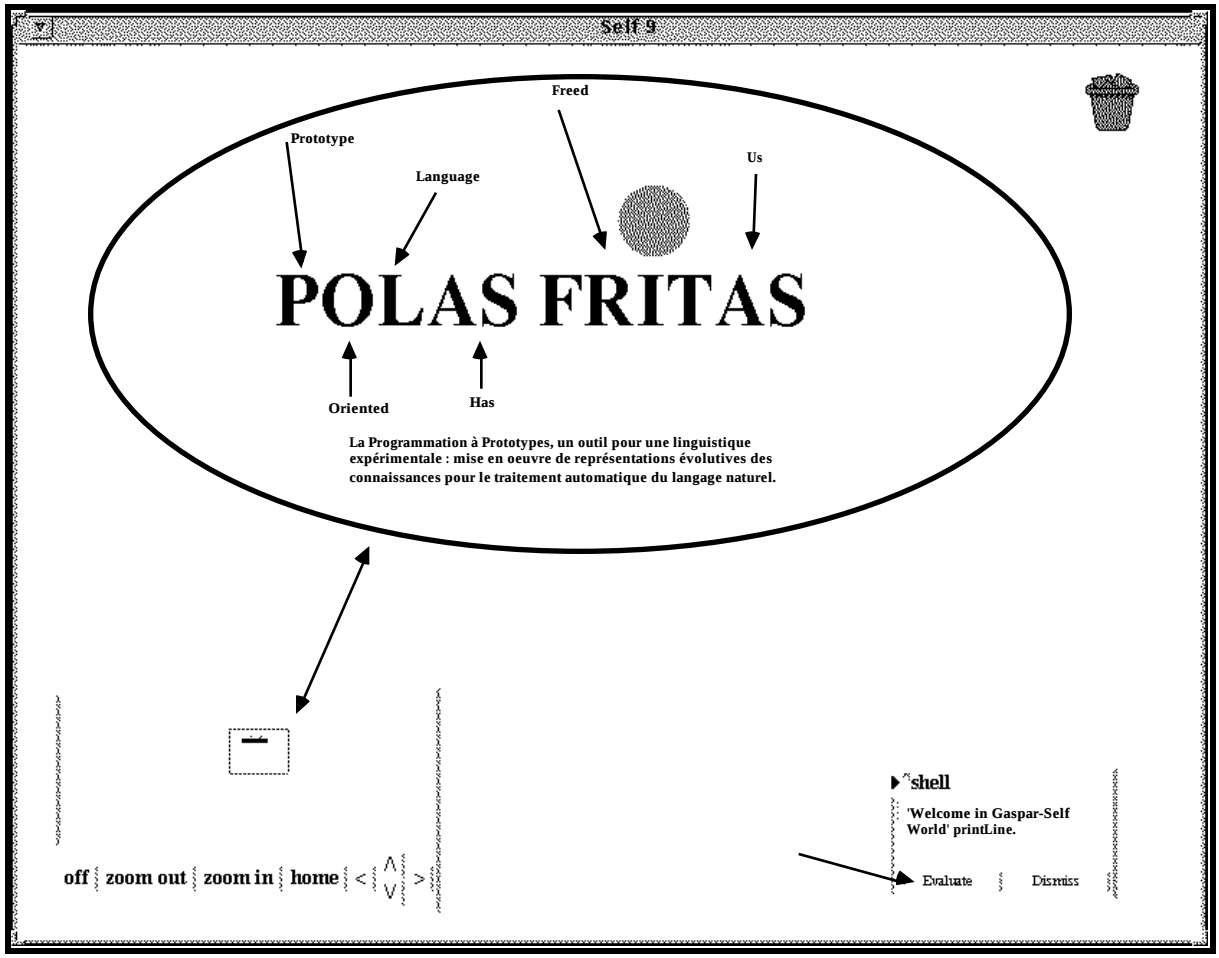
Thèse

présentée pour l'obtention
du titre de Docteur d'Université
Spécialité: Linguistique

Volumes 1 & 2

Soutenue devant la Commission d'Examen
composée de

MM.	**Bernard Victorri**	**(Rapporteur)**
	Pierre Cadiot	**(Rapporteur)**
	François-Xavier Testard-Vaillant	**(Directeur)**
	Benoît Habert	
	Laurence Danlos	
	Violaine Prince	


Self 9
Freed
Prototype
Us
Language
POLAS FRITAS
Oriented
Has
La Programmation à Prototypes, un outil pour une linguistique expérimentale : mise en oeuvre de représentations évolutives des connaissances pour le traitement automatique du langage naturel.
shell
'Welcome in Gaspar-Self World' printLine.
Evaluate
Dismiss
off
zoom out
zoom in
home

Remerciements

Ce document est le résultat de plusieurs années de travail avec Benoît Habert (ELI - ENS de Fontenay-St Cloud) à l'E.N.S de Fontenay aux Roses. Ce travail est donc aussi le "récit" du parcours de nos interrogations partagées et échangées.

Je tiens à remercier Bernard Victorri, Pierre Cadiot, Laurence Danlos et Violaine Prince d'avoir accepté de participer à mon jury de thèse et de m'avoir ainsi témoigné leur confiance.

Je voudrais exprimer ma gratitude à François-Xavier Testard-Vaillant pour avoir accepté de diriger cette thèse.

Je remercie Adeline Nazarenko (ELI - ENS de Fontenay-St Cloud) pour ses précieuses remarques sur mon travail. Je remercie également Didier Bourigault (LLI-CNRS), Benoît Habert et Adeline Nazarenko pour m'avoir donné accès aux résultats de Lexter et de Cyclade.

Je voudrais remercier chaleureusement Josiane Bernillon, Serge Heiden (ENS de Fontenay-St Cloud) qui m'ont accueilli à l'E.N.S de Fontenay aux Roses. Je remercie aussi les membres de l'ELI à l'E.N.S de Fontenay aux Roses pour leur soutien et leurs encouragements permanents.

Je n'oublie bien évidemment pas les membres du groupe Self à Stanford et à Sun Microsystems qui ont développé le langage Self et qui ont toujours répondu avec bienveillance à mes sollicitations.

J'exprime ma profonde reconnaissance à tous ceux qui m'ont aidé dans ce travail.

A Marie, tout ce qui ne peut être dit ici.

Prologue

Quand nous instaurons un classement réfléchi, quand nous disons que le chat et le chien se ressemblent moins que deux lévriers, même s'ils sont l'un et l'autre apprivoisés ou embaumés, même s'ils courent tous deux comme des fous et même s'ils viennent tous les deux de casser la cruche, quel est donc le sol à partir de quoi nous pouvons l'établir en toute certitude? Sur quelle "table", selon quel espace d'identités, de similitudes, d'analogies, avons-nous pris l'habitude de distribuer tant de choses différentes et pareilles? Quelle est cette cohérence - dont on voit bien tout de suite qu'elle n'est ni déterminée par un enchaînement a priori et nécessaire, ni imposée par des contenus immédiatement sensibles? Car il ne s'agit pas de lier des conséquences, mais de rapprocher et d'isoler, d'analyser; d'ajuster et d'emboiter des contenus concrets; rien de plus tâtonnant, rien de plus empirique (au moins en apparence) que l'instauration d'un ordre parmi les choses; rien qui n'exige un oeil plus ouvert, un langage plus fidèle et mieux modulé; rien qui ne demande avec plus d'insistance qu'on se laisse porter par la prolifération des qualités et des formes. Et pourtant un regard qui ne serait pas armé pourrait bien rapprocher quelques figures semblables et en distinguer d'autres à raison de telle ou telle différence: en fait, il n'y a, même pour l'expérience la plus naïve, aucune similitude, aucune distinction qui ne résulte d'une opération précise et de l'application d'un critère préalable. Un "système des éléments" - une définition des segments sur lesquels pourront apparaître les ressemblances et les différences, les types de variation dont ces segments pourront être affectés, le seuil enfin au-dessus duquel il y aura une différence et au-dessous duquel il y aura similitude - est indispensable pour l'établissement de l'ordre le plus simple. L'ordre, c'est à la fois ce qui se donne dans les choses comme leur loi intérieure, le réseau secret selon lequel elles se regardent en quelque sorte les unes les autres et qui n'existe qu'à travers la grille d'un regard, d'une attention, d'un langage; et c'est seulement dans les cases blanches de ce quadrillage qu'il se manifeste comme déjà là, attendant en silence le moment d'être énoncé.

Michel Foucault *Les Mots et Les Choses* (préface).

Résumé

Le mythe des industries de la langue.

J.C Milner (Milner 1989) caractérise une science par *"la mathématisation de l'empirique"* et par *"la constitution d'une technique, telle que la technique se définisse comme l'application pratique de la science (d'où le terme de science appliquée) et que la science se définisse comme la théorie de la pratique (d'où le terme de science fondamentale)"*.

Une industrie suppose préalablement une science achevée (provisoirement). Il ne saurait en être ainsi pour les langues naturelles : les descriptions sont toujours partielles, partiales et destinées à le rester. Le Traitement Automatique du Langage Naturel (TALN) doit placer au centre de ses préoccupations la nécessité d'outils, de dispositifs permettant de mettre au point les connaissances à utiliser, de tenir compte de leur caractère avant tout provisoire, daté et évolutif. L'objectif de la thèse est de donner l'intuition de ce type de dispositif, (en donnant à voir) par des morceaux de réalisation, en s'appuyant sur deux approches informatiques traitant différemment de la classification : la PàO (Programmation à Objets) et la PàP (Programmation à Prototypes), et en utilisant le paradigme de la réflexivité. Le point d'ancrage langagier est tout d'abord le traitement d'une série d'expressions figées puis le traitement de savoirs lexicaux extraits à partir de corpus. On traitera de l'organisation des connaissances (ici lexicales au sens d'entrées lexicales dans toutes leurs dimensions : morphologie, sémantique, pragmatique), et de l'utilisation des connaissances (règles de combinaison et de structuration).

Notre travail s'inscrit plus particulièrement dans une approche informatique manipulant des prototypes. Cette approche de représentation ne préprogramme pas de manière figée ni toutes les opérations pour un traitement automatique de la construction du sens ni les processus de représentation des unités lexicales manipulées. Il faut faire en sorte que les programmes construits puissent évoluer aussi bien dans les opérations que ces derniers permettent que dans les représentations du domaine construites. La PàP (Programmation à Prototypes) conduit à penser différemment pour construire une représentation informatique d'un certain domaine de connaissances. Il ne s'agit pas de construire une représentation informatique d'un domaine donné à partir d'une somme de connaissances figées et connues par avance. Ce modèle de représentation permet de construire progressivement les entités informatiques suivant les connaissances dont on dispose sur le domaine visé : si de nouvelles connaissances sur le domaine sont mises à jour, on peut affiner le processus de représentation déjà amorcé en tenant compte de ces nouvelles informations sans avoir à reconstruire entièrement de nouvelles structures. Le processus de représentation lié à la PàP ne nécessite donc pas une définition exacte des concepts à représenter. La PàP permet d'introduire une dimension temporelle dans le processus de la représentation : ce dernier peut évoluer ou faire évoluer les résultats construits de manière continue.

REMERCIEMENTS **3**

TABLE DES FIGURES **16**

INTRODUCTION GÉNÉRALE **21**

INTRODUCTION **22**

EBAUCHE POUR UN TRAITEMENT AUTOMATIQUE DE LA CONSTRUCTION DU SENS 22
Au commencement, une histoire de traces *22*
Le Traitement Automatique du Langage Naturel (TALN) : du dur désir de représenter *23*
De la réalité des représentations : du renoncement à tout prévoir à l'expérimentation linguistique. *23*
Quels outils informatiques pour la représentation des connaissances en TALN? *23*
Quels objets pour représenter la mouvance ? *24*
DESCRIPTION DU TRAVAIL 25
Plan de la thèse *25*

ORIGINES ET CHOIX **28**

1. POURQUOI DES CLASSIFICATIONS MOUVANTES ? 29
2. MODÈLES POUR LA CONSTRUCTION DU SENS : CRITIQUES ET DISCUSSION 30
2.1. Savoirs linguistiques et heuristiques : comment représenter la mouvance ? *31*
2.2. Ce que ces modèles peuvent (ne peuvent pas) faire ? *31*
2.2.1. Kayser et la profondeur variable 31
2.2.2. Le lexique génératif de Pustejowsky 31
2.2.3. Les approches Objet pour le TALN 32
2.2.4. Rastier et la sémantique différentielle 33
2.2.5. Continuité et TALN 34
2.3. Problèmes posés par ce type d'approche *35*
2.3.1. Représenter, c'est figer un peu. 35
2.3.2. "Ajuster les processus d'ajustements" : approches de la réflexivité 35
2.4. PàP : une nouvelle approche pour représenter la mouvance *35*
2.4.1. Choix et perspectives 35
2.4.2. Que fait la PàP ? 36
2.4.3. PàP et TALN : un certain point de vue 36
3. OBJECTIFS 37
3.1. Travail sur `moulin` *37*
3.2. Un dispositif pour la représentation et le classement automatiques des mots et de leurs comportements *37*
3.3. Un méta-niveau pour le Traitement Automatique du Langage Naturel *38*
3.4. Cadre choisi : la PàP (Programmation à Prototypes) *38*
4. CE QUI A ÉTÉ RÉALISÉ 38
4.1. Travail sur moulin *39*
4.2. Un dispositif pour la représentation et le classement automatiques des mots et de leurs comportements *39*
4.3. Un méta-niveau pour l'analyse *40*

PARTIE 1 : DU RÉEL **42**

ENJEUX LINGUISTIQUES ET INFORMATIQUES DU TRAITEMENT AUTOMATIQUE DE LA CONSTRUCTION DU SENS **43**

1.1. COMMENT REPRÉSENTER LA MOUVANCE ? 43
1.1.1. Les noms composés : un micro-phénomène pour l'étude de la construction du sens *43*
1.1.1.1. Préambule 43
1.1.1.2. Une première tâche : étudier les formes nominales `moulin à N2` 44
1.1.1.3. Examen des évolutions de `moulin` sur corpus 44
1.1.1.4. `Moulin` : Reconstruire une Représentation Sémantique 46
1.1.1.3. De l'étude de certains faits empiriques à la construction d'un dispositif : problèmes de représentation 48
1.1.2. `Moulin à N2` : une approche hiérarchique de la construction du sens (Habert & Fabre 1993) *48*
1.1.2.1. Objectif : un modèle des règles de production de Dénomination Complexe 49
1.1.2.2. Les règles à l'oeuvre dans le patron `moulin à N2` 49
Un premier classement 49
Réexamen de la classe résiduelle pour une analyse unifiante du sémantisme de `moulin` 50
Construction du sens des séquences `moulin à N2` 52

Affiner le modèle sémantique proposé 54
1.1.3. Changements Réguliers de sens et organisation des connaissances *54*
1.1.3.1. Limites du filtrage des arguments acceptables 54
1.1.3.2. Structurer les connaissances syntaxiques 55
1.2. QUELS SAVOIRS POUR LES MOTS ? 56
1.3. APPROCHE D'UNE GRAMMAIRE DE SOUS-LANGAGE 58
1.3.1. Notion de sous-langage *58*
1.3.2. L'analyse des sous-langages : la démarche harrissienne *58*
Propriétés des sous-langages 59
Obtention de classes d'opérateurs et d'arguments à partir de phrases élémentaires 59
1.4. CONCLUSION 60

LE LANGAGE : "SI LOIN, SI PROCHE" 61

2.1. DE L'ÉTUDE DE FAITS LINGUISTIQUES 61
2.1.1. Les faits linguistiques sont empiriques *61*
2.1.1.1. Si la linguistique est une science, elle est une science empirique 61
2.1.1.2. Le langage est à la fois, support, instrument et fondation de nos activités de représentation 61
2.1.1.3. Du travail sur les faits de langue 62
2.1.2. De la finesse des questions linguistiques *62*
2.1.3. De l'interaction de phénomènes divers dans les questions linguistiques *62*
2.1.4. Du non-achèvement des résultats dans la science linguistique *62*
2.1.5. La science linguistique résiste au formalisme *63*
2.1.5.1. De la formalisation du langage naturel 63
2.1.5.2. De l'excès dans la langue à la formalisation 63
2.1.5.3. Le TALN ou le dur désir de formaliser les faits de langue 64
2.1.5.4. Formalisation et réalité linguistique 64
2.1.6. La science linguistique et le dispositif *64*
2.1.7. Des industries de la langue ? *65*
2.1.8. La langue comme modèle de système de représentation des connaissances *66*
2.2. DE LA CONSTRUCTION DU SENS 67
2.2.1. Représentation(s) du sens *67*
2.2.2. De la construction du sens comme un travail *69*
2.2.2.1. L'interprétation : une mise en action 69
2.2.2.2. La construction du sens : un télescopage de savoirs 70
2.3. MODÈLES POUR LA CONSTRUCTION DU SENS 71
2.3.1. Connexionnisme et Composés *71*
2.3.1.1. Apprentissage et réseaux connexionnistes 71
2.3.1.2. Réseau de neurones et TALN 71
2.3.1.3. Réseau de neurones et "noms composés" 72
2.3.2. Représentation hiérarchisées : mots ou choses *72*
2.3.2.1. Hiérarchies 72
Ontologie vs. Classifications linguistiques 72
Savoir sur le monde et savoir sur la langue 73
a. Articulation Langagier-Concepts 73
1. Inutilité de définir le sens d'un mot 73
2. Utilité de définir le sens d'un mot 74
b. Représentation des concepts 74
c. Le Concept : un point d'entrée pour des inférences 75
d. `Moulin` et la profondeur variable 76
2.3.2.2. Accès à des scénarios (Pustejovsky & Boguraev 1991) 76
Deux approches pour traiter la polysémie 76
a. Approche lexicographique traditionnelle 76
b. Approche semi-polymorphique du lexique génératif de Pustejovsky 77
Repérer les régularités lexicales 77
L'opération de coercion 78
L'exemple de `"fast"` 78
a. Analyse de `"fast"` comme s'appliquant au rôle télique du nom qu'il modifie 79
Représentation de la polysémie nominale 80
Evaluation paresseuse par nécessité (Copestake & Briscoe 1994) 80
a. Traitements de la polysémie 81
b. Les règles lexicales 81
Telle entrée lexicale : tels scénarios? 82
a. Tous les noms sont-ils descriptibles par une structure prédicative ? Tous les noms ont-ils des rôles téliques ?. 82
b. Aspect fini de la représentation : on fixe le nombre de scénarios 82
2.3.2.3. Les classes de mots 82
2.3.2.4. Types et distributions (Godard & Jayez 1995) 84

2.3.3. Polysémie et changements de sens *85*
2.3.3.1. Changements réguliers de sens (CRS) et hiérarchies 85
Multitypage et choix en contexte : (Godard & Jayez 1995) 85
Les règles lexicales (Briscoe & Corperstake 1993) 85
Coercion de type (Pustejovsky 1991) 85
2.3.3.2. Contraintes 86
Continuité du passage d'un type à un autre 86
a. Un noyau de sens pour conduire l'interprétation 86
1. Du discret au continu 86
1.1. Modèle mathématique de la polysémie 86
1.2. Hypothèse méthodologique pour le classement des significations 88
1.3. Accès aux valeurs sémantiques 88
2. Le noyau de sens 88
3. Définir un noyau de sens comme un opérateur abstrait auquel on peut faire correspondre une opération cognitive 88
b. Interpréter, c'est opérer une confrontation frontale de tous les indices sémantiques présents dans un énoncé .. 89
Modèle discret vs. continu : notre problème est ailleurs 89
2.3.3.3. Statut des CRS 90
Mécanismes pragmatiques très généraux (Kleiber & al. 1991) 90
Liés à une hiérarchie de concepts (Kayser 1992), (Pustejovsky 1991) 90
2.3.4. Evolution des classifications *91*
2.3.4.1. Hiérarchies rigides 91
2.3.4.2. Changer la hiérarchie 91
2.4. CONCLUSION 92
2.4.1. Représenter, c'est figer un peu. *92*
2.4.2. Ajuster les processus d'ajustements : approches de la réflexivité *92*

MODÈLE DE CLASSIFICATION POUR LE LANGAGE NATUREL 93

3.1. CLASSIFICATION 93
3.1.1. Les systèmes à héritage *93*
3.1.1.1. Les langages de classes 94
3.1.1.2. Les langages à prototypes 94
3.1.1.3. Les langages hybrides 94
3.1.2. Le Filtrage *94*
3.1.3. La classification dans les systèmes à héritage *94*
3.1.3.1. Les langages de classes 95
3.1.3.2. Les langages à prototypes 95
3.1.3.3. Savoir harmoniser certaines complémentarités des formalismes sous-jacents aux modèles de représentation utilisés 95
3.2. LE TALN ET LA CLASSIFICATION 96
3.2.1. De la connaissance et du TLN. *96*
3.2.1.1. Modularité ou intégration 97
3.2.2. Manières de concevoir les connaissances *98*
Connaissance et Utilisation des connaissances 99
3.2.3. Paradigmes récents en TLN : vers plus de déclarativité *99*
3.2.3.1. Les ATN 100
3.2.3.2. Les Grammaires d'unification 101
Limites des grammaires hors contexte 101
Les grammaires indexées 102
Grammaires basées sur l'unification (-> incise 1) 103
3.3. CLASSIFIER EN TERRAIN CONNU : TAXONOMIES BIEN IDENTIFIÉES ET LANGAGES À TAXONOMIES DE CLASSES 104
3.3.1. La PàO *104*
3.3.1.1 Présentation de la programmation Objet avec CLOS 104
Objets et classe 104
Définir une classe 105
Créer des instances 106
Graphe d'héritage et masquage 107
Fonction générique et les méthodes associées 107
Documentation 108
3.3.2. Représentation de lexique et Objets *108*
3.3.2.1. DATR : un outil de représentation hiérarchisée 108
3.3.3. Représentation de règles et Objets *108*
3.3.3.1. `Moulin à N2` et DATR 108
3.3.3.2. La hiérarchie au service de la sous-détermination 111
3.3.4. Limites : comment prendre en compte l'évolution des connaissances *111*
3.3.4.1. Les savoirs lexicaux résistent aux classements hiérarchisés 111
3.3.4.2. Représentation de `moulin` et construction du sens des séquences `moulin à N2` 112

3.3.4.3. Représenter en hiérarchisant pas à pas les interprétations 112
3.4. CLASSIFIER EN TERRAIN MOUVANT : LA PÀP....113
3.4.1. Prototypie et représentation des connaissances 113
3.4.1.1. Prototypie et représentation du savoir : une présentation 113
3.4.1.2. La PàP : les langages à prototypes, un outil qui permet l'hétéro-modification 113
PàO vs. PàP 113
Illustration de la représentation prototypique avec la catégorie grammaticale des noms....114
Un nom = un prototype....114
Représentation d'un autre nom par clonage et différenciation 114
Partage des attributs communs par délégation 115
3.4.1.3. Le langage Self (Ungar & al 1987) 115
Présentation 116
Concept de Bases (Cointe & al 1991) 117
Objets et attributs Self....118
Héritage et Self : la délégation, une vision dynamique de l'héritage 118
Permettre l'hétéromodification 119
Les langages à taxonomie de classes 119
Les langages à prototypes....120
Changer dynamiquement les comportements....121
3.4.2. Prototype et psychologie cognitive....122
3.4.2.1. Prototype : une version standard....122
Le prototype comme meilleur exemplaire....122
Prototype et catégorisation....123
Le niveau de base....124
3.4.2.2. Prototype : une version étendue....124
3.4.2.3. PàP vs. Prototypie cognitiviste 125
INCISE 1. GRAMMAIRE BASÉE SUR L'UNIFICATION....127
Structure d'information....127
Unification 129
Règles de réécriture et contraintes additionnelles : DAGs 131
Unification et analyse automatique 132
Unification et représentation des connaissances....133
Héritage simple....133
Héritage par défaut....133
INCISE 2. LES GRAMMAIRES D'ARBRES ADJOINTS (TAG)....135

UN DISPOSITIF OU LE DUR DÉSIR DE REPRÉSENTER.... 137

4.1. PRÉAMBULE : LE DISPOSITIF EST AU BORD D'UN GOUFFRE....137
4.2. UN DISPOSITIF QUI CONSTRUIT, UTILISE ET MODIFIE DES HIÉRARCHIES 139
4.2.1. De la complexité de l'objet à représenter 139
4.2.2. Organiser le savoir lexical....139
4.2.3. Du possible dans la représentation....139
4.2.4. Attacher des scénarios aux entrées lexicales : prédire les possibles d'interprétation....140
4.2.5. Difficulté de l'affinement d'une hiérarchie lexicale en fonction des observations faites lors d'une analyse140
4.2.5.1. Les faits de langue sont difficilement réductibles en termes d'objets figés 140
4.2.5.2. "Donner du mouvement" au jeu de la représentation....140
4.2.5.3. Articuler les représentations....141
4.2.5.4. Permettre un affinement dans les représentations 141
4.2.6. Inscrire des mécanismes inférentiels au sein des représentations....141
4.3. UN DISPOSITIF QUI MANIPULE DES PROTOTYPES....142
4.3.1. Construire une analyse automatique guidée par des prototypes....142
4.3.2. Représentation des connaissances et Objet....142
4.3.2.1. Des prototypes pour coder la construction du sens de certains composés nominaux 143
4.3.2.2. Quels objets linguistiques devons-nous admettre dans notre propre manière de rendre compte des faits ?....143
4.3.3. Etablir une évaluation critique des pouvoirs expressifs des modèles de représentation 144
4.3.4. Représenter les mécanismes de construction du sens....144
4.3.4.1. Représenter en tenant compte des absences du représentable 144
4.3.4.2. Repérer les mécanismes d'extension de sens et (re)construire ou affiner les représentations 144
4.3.5. Privilégier une représentation qui permette la spécialisation 144
4.3.5.1. Construction des composés : une hiérarchie implicite 144
4.3.5.2. Des représentations au "jeu" sur les représentations....145
4.3.6. Développer des outils de contrôle pour l'analyse....145
4.4. PROTOTYPES, REPRÉSENTATION DES CONNAISSANCES ET MODÈLES DU SENS....146
4.4.1. Noyau de sens et prototypes....146
4.4.2. Adaptation d'une représentation prototypique et d'une approche à la Pustejovsky ?....146

4.4.2.1. Inscrire des processus d'interprétation dans la représentation prototypique 146
4.4.2.2. De l'unité dans la représentation 151
4.5. APPROCHE POUR UNE FORMALISATION PROTOTYPIQUE DE LA CONSTRUCTION PROGRESSIVE DU SENS DES FORMES MOULIN À N2 152
4.5.1. Une première approche guidée par les hypothèses sémantiques développées par (Habert & Fabre 1993)152
4.5.1.1. Un calcul sémantique à base de traits 152
Calcul sémantique de moulin à x 153
4.5.1.2. Limites de ce type de représentation sémantique en termes de traits 153
4.5.2. Moulin : (Re)construire une Représentation Sémantique Prototypique 155
4.5.2.1. Moulin emballe la machine prototypique 156
4.5.2.2. Des moulins aux moulins à N2 158
4.5.2.3. Des moulins à prières aux moulins à paroles 160
4.5.2.4. Moulin et les déverbaux 163
4.5.2.5. Articuler les niveaux de représentation 166
4.5.2.6. Préférences d'interprétation et prototypie 170
4.5.2.7. Moulin et les noms d'instruments 172
4.5.2.8. Construire un espace de concepts 173
4.6. COMMENT NOMME-T-ON LES CHOSES ? 174

PARTIE 2 : DU POSSIBLE 175

CLASSIFIER EN TERRAIN MOUVANT : APPROCHE PROTOTYPIQUE POUR LA CONSTRUCTION DU SENS DE MOULIN À N2 176

5.1. "PARTI DE RIEN, VIRGULE..." 176
5.2. PROTOTYPES ET ANALYSE AUTOMATIQUE 177
5.2.1. Préliminaires 177
5.2.2. Cadre linguistique 177
5.2.3. Approches pour une Analyse automatique prototypique avec Self 177
5.2.3.1. Phase0 : Filtrage Catégoriel 177
Existe-t-il des éléments prototypiques dans la langue ? 178
Prototype et analyse de mots 178
Représentation du lexique et des catégories grammaticales 178
Contrôle et analyse dans un monde de prototypes 178
5.2.3.2. Phase1 : Affinement du filtrage 180
Analyse et reconnaissance/construction des prototypes N1 à N2 181
5.2.3.3. Phase2 : Vers une représentation prototypique des connaissances 182
5.2.3.4. Phase 0-2 : Premiers résultats acquis, perspectives 187
Les prototypes construits peuvent être ajustés automatiquement (i.e. recatégorisés) 187
Elargir la couverture sémantique du modèle 187
5.2.4. Analyse automatique et construction des prototypes N1 à N2 188
5.2.4.1. Préliminaires 188
5.2.4.2. Phase3.0 : Filtrage pour la construction du prototype moulin à N2 188
5.2.4.3. Phase3.1 : Filtrage pour la construction du prototype N1 à N2 191
5.2.4.4. Phase3.2 : Filtrage pour une construction spécialisée des prototypes N1 à N2 197
5.2.5. Une Analyse automatique prototypique étendue 197
5.2.5.1. Analyse par un réseau de prototypes 197
Mise en place d'un réseau de prototypes 197
Déroulement de l'analyse 198
5.2.5.2. L'utilisateur peut contrôler la construction du sens des séquences en moulin à N2 202
5.3. APPROCHE PROTOTYPIQUE DE LA CONSTRUCTION DU SENS DES SÉQUENCES MOULIN À N2 : RÉSUMÉS ET PERSPECTIVES 204
5.3.1. Lexique et prototypes 204
5.3.2. Syntaxe et prototypes 206
5.3.3. Sémantique et prototypes 206
5.3.4. Ajuster les mécanismes statiques/dynamiques de re/ classification en/pour ajustant/ajuster les représentations initiales 207
5.3.4.1. Ajuster les mécanismes de classification en ajustant les représentations initiales 207
Ne pas maintenir des représentations contraignantes et non éclairantes 208
Isoler les mécanismes qui participent à la construction du sens et les articuler hiérarchiquement 208
5.3.4.2. Ajuster les mécanismes de classification pour ajuster les représentations initiales 209
5.3.4.3. Un analyseur qui manipule des mots experts : WEP 209
Accès d'un expert aux informations 210
Activation d'un expert 210
Représentation de la compréhension de la phrase 210
Comportement de l'expert 210

Mise en oeuvre des actions d'un expert 211
WEP et PàP : confrontations....211
INCISE 1. ENVIRONNEMENT DE PROGRAMMATION AVEC SELF 212
I.1.1. Inspecter les prototypes....214
I.1.2. Ajouter, supprimer des attributs....214
I.1.3. Définir des prototypes 214
I.1.4. Créer des prototypes....216
I.1.5. Accès aux attributs des prototypes....217
I.1.6. Documenter le code 218
INCISE 2. ILLUSTRATION PAR LA MODÉLISATION DES POINTS AVEC SELF (UNGAR & AL 1987)....219
I.2.1. Définition d'un objet attaché aux comportements des objets 219
I.2.2. Définition d'un objet attaché aux prototypes....219
I.2.3. Définition des comportements attachés aux points 219
I.2.4. Définition d'un prototype de point....219
INCISE 3. ILLUSTRATION PAR LA MODÉLISATION DES NOMS AVEC SELF 221
I.3.1. Définitions des objets attachés aux comportements des prototypes 221
I.3.2. Définitions des prototypes....222

LE PROTOTYPE VEUT ENFIN REPRÉSENTER : UN NOUVEAU TYPE DE `MOULIN`..... 224

6.1. REPRÉSENTATION DYNAMIQUE DE LA CONSTRUCTION DU SENS DES SÉQUENCES EN `MOULIN À N2` 224
6.1.1. `"Moulin"` est avant tout une machine 224
6.1.2. `"Moulin"` fonctionne aussi comme un opérateur syntaxique qui prend en arguments une préposition et un nom 224
6.1.3. `Moulin` et les prototypes....225
6.1.4. `Moulin à paroles` peut être vu comme une spécialisation 226
6.1.5. `Moulin à paroles` peut être vu comme une généralisation : un nouveau type de moulin 227
6.2. EXTENSION DU DOMAINE DE L'ANALYSE....230
6.2.1. Du côté de l'analyse 230
Déroulement de l'analyse 230
6.2.2. Du côté des "moulin à N2"....237
6.3. TRAITEMENT AUTOMATIQUE DU LANGAGE NATUREL ET PROTOTYPES....247
6.3.1. Représentation prototypique et stabilisation des savoirs....247
6.3.1.1. Du méta-regard sur l'analyse à l'automodification des représentations 247
6.3.1.2. Comment définir des "démons" capables de redéfinir les comportements généraux à factoriser....248
6.3.2. Quels outils pour une évolution dynamique des représentations?....248
6.4. DE LA LINGUISTIQUE À L'INFORMATIQUE : RETOUR SUR UN EXAMEN DES BESOINS PROPRES À LA REPRÉSENTATION DES CONNAISSANCES POUR LE TALN....250
6.4.1. Paradigmes courants en TALN et environnements informatiques....250
6.4.1.1. Posés et présupposés dans les travaux courants actuels en TALN....250
6.4.1.2. Représentation par Objets et TALN : la PàO et la PàP....250
6.4.2. La sémantique différentielle bouscule les paradigmes courants du TALN : problèmes théoriques et implications pour une mise en oeuvre informatique 250
6.4.2.1. Posés et présupposés de la sémantique différentielle....251
6.4.2.2. Quels outils informatiques pour la sémantique différentielle ? 253
6.4.3. Prototypie (au sens de la PàP) et sémantique différentielle 253
6.5. CONCLUSION ET PERSPECTIVES 255

DES ARBRES AUX PROTOTYPES : VERS UNE CLASSIFICATION DYNAMIQUE..... 256

7.1. ACQUISITION DE CONNAISSANCES À PARTIR D'UN CORPUS....256
7.1.1. Un Problème 256
7.1.2. Un Corpus....257
7.1.3. Un Outil Informatique 257
7.2. SITUATION EFFECTIVE PAR RAPPORT AU TRAVAIL DE HARRIS 258
Mise en évidence des comportements des mots d'un corpus pour une automatisation de leur représentation et de leur classement dans un programme de TALN manipulant des prototypes....258
7.3. EXTRACTION DES FONCTIONNEMENTS LEXICAUX : LEXTER ET ZELLIG....259
7.3.1. Buts : retrouver les fonctionnements lexicaux....259
7.3.2. LEXTER (Bourigault 1993)....259
7.3.3. ZELLIG (Habert & al. 1996)....260
7.3.3.1. Buts....260
7.3.3.2. Simplification d'arbres d'analyse....260
7.3.4. Difficultés pour extraire des arbres "élémentaires"....261

7.3.4.1. Les arbres d'analyse produisent des arbres "élémentaires" non pertinents....261
7.3.4.2. Imprévisibilité des contraintes sur les combinaisons d'arbres....262
7.3.4.3. Imprévisibilité des comportements lexicaux....262
7.4. DU LEXIQUE JAILLIT UN RÉSEAU DE PROTOTYPES....265
7.4.1. Définitions préliminaires....265
7.4.2. Dégager les comportements syntaxiques d'une entrée lexicale....265
7.4.3. Etendre la représentation en ordonnant les comportements....267
7.5. ESQUISSE D'UN RÉSEAU DE COMPORTEMENTS SYNTAXICO-SÉMANTIQUES HIÉRARCHISÉS....269
7.5.1. Une première phase de représentation des informations recueillies sur un corpus donné....269
7.5.2. Construction d'une hiérarchie approximative....270
7.5.3. Illustration avec Self....271
7.5.4. Hiérarchie mouvante et apprentissage....271
7.6. AUTOMATISATION DES TRAITEMENTS DE REPRÉSENTATION DES MOTS ET DES ARBRES....273
7.6.1. Construire les représentations à la demande....273
7.6.2. Génération automatique de prototypes....273
7.6.2.1. Génération par création explicite d'un nouvel objet....273
7.6.2.2. Génération par clonage et ajustements....277
7.6.3. Une construction inductive des savoirs et une classification dynamique de ces savoirs....277
7.6.3.1. Construire un prototype de grammaire lexicalisée....277
7.6.3.2. Une amorce de classement guidée par la syntaxe....279
7.6.3.3. Créer des pôles multiples de regroupements de comportements partagés, les abstraire et les organiser....281
Difficultés techniques....282
Difficultés linguistiques....282
7.6.3.4. Intervenir manuellement pour interpréter les regroupements....283
7.7. VERS UNE CLASSIFICATION DYNAMIQUE : BILANS ET PERSPECTIVES....284
7.7.1. La PàP : une nouvelle approche pour le TALN?....284
7.7.2. Un dispositif expérimental guidé par le linguiste....284
INCISE 1. SELF ET L'HÉRITAGE....286
I.1.1. Héritage non-ordonné....286
I.1.2. Héritage multiple....286
I.1.3. Règles de précédence dans l'héritage....287
I.1.4. Renvoi (dirigés) de messages....288
I.1.4.1. Receveur implicite....288
I.1.4.2. Receveur explicite....289
INCISE 2. POSITIONNEMENT PAR RAPPORT À L'APPROCHE PROBABILISTE....290
I.2.1. Probabilisation des modèles syntaxiques (Rajman 1995)....290
I.2.1.1. Approche probabiliste pour l'analyse syntaxique....290
I.2.1.2. Schéma Global de probabilisation d'un modèle syntaxique....290
I.2.1.3. Prototypes et Comportements vs. Approche Probabiliste....291

REPRÉSENTATION ET CLASSIFICATION DYNAMIQUES.... 292

8.1. UN DISPOSITIF POUR LA REPRÉSENTATION ET POUR LE CLASSEMENT DES MOTS ET DE LEURS COMPORTEMENTS....292
8.1.1. Un dispositif pour l'expérimentation linguistique....292
8.1.2. Description des informations à représenter....293
8.1.3. Mise en forme des connaissances....297
8.1.4. Une construction inductive des savoirs....297
8.1.4.1. Génération dynamique des prototypes lexicaux et des prototypes d'arbres....297
8.1.4.2. Génération du dictionnaire....299
8.1.4.3. Génération d'un prototype de grammaire....299
8.1.4.4. Ajustement des bases de savoirs....300
8.1.5. Le réseau de prototypes dévoile une hiérarchie évolutive....300
8.1.5.1. Repérer les comportements partagés par les prototypes lexicaux....300
8.1.5.2. Rôle et utilisations des arbres élémentaires....304
Les arbres élémentaires : des amorces syntaxiques pour le classement des mots....304
Les arbres élémentaires ont une histoire syntaxique....304
Classement des arbres élémentaires....304
Génération automatique d'arbres (au cours d'une analyse)....305
Construction des arbres de substitution....306
Construction des arbres auxiliaires pour l'adjonction....306
8.2. EXPÉRIMENTATIONS SUR CORPUS....308
8.2.1. Résultats construits sur un corpus de test....308
8.2.2. Résultats construits sur des gros corpus....314
8.2.2.1. Etude de la relation NAdj....314

Remarques préliminaires sur les corpus étudiés 315
Résultats Lexter et Cyclade : Caractérisation des relations pour les adjectifs (Habert & al. 1996)....315
8.2.2.2. Travail sur le corpus fourni par Lexter 317
Données traitées 317
8.2.2.3. Travail sur le corpus fourni par AlethIP....317
Données traitées 317
8.2.2.4. Le travail sur les gros corpus et les limites de Self 317
8.3. REPRÉSENTATION ET CLASSIFICATION DYNAMIQUES : BILANS ET PERSPECTIVES 319
8.3.1. Une démarche interprétative contrôlée et en spirale....319
8.3.2. Des amorçages d'interprétation : le retour du linguiste 319
8.3.3. Un système qui module les flux d'informations....320
8.3.4. Prototypie et représentation des savoirs pour le TALN....322
8.3.4.1. Processus de représentation et TALN....322
Les mots ont leur(s) histoire(s) 322
Retrouver/rechercher les connaissances liées à chaque mot 322
Processus de représentation évolutifs....322
Importance du travail manuel sur le corpus initial, puis sur les représentations construites. 322
8.3.4.2. Affinements des représentations....323
Clonage et ajustements....323
Ajouts/Retraits d'attributs et effets de bord 323
8.3.4.3. Retour sur le stochastique 323
Affecter des poids aux arbres associés aux mots 324
Apprentissage et mot nouveau 324
Utilisation des savoirs construits 324
8.4. CONCLUSION 326
INCISE 1. SELF 4.0, UN ENVIRONNEMENT INFORMATIQUE QUI DYNAMISE LES OBJETS 327
I.1.1. Self-4.0 (Smith, Maloney, Ungar 1995)....327
Environnement de programmation 327
Démarrer Self....327
I.1.2. Interfaces, outils, opérations et menus dans le monde Self....328
Menus des opérations sur le monde Self 328
Menus des opérations spécifiques sur les objets Self....329
Menu des opérations spécifiques sur un objet 329
Opérations globales....330
"Expanders"....331
"Evaluator"....331
INCISE 2. MANIPULATION ET INTERVENTION DIRECTES SUR LES OBJETS AVEC SELF....334
INCISE 3. GASPAR : UN ANALYSEUR À MOTS IMPLÉMENTÉ AVEC SELF-4.0 339
I.3.1. Catégorisation des outils pour l'analyse....339
I.3.2. Interfaces disponibles....340
INCISE 4. LA CLASSIFICATION ÉVOLUTIVE EST EN MARCHE 343
I.4. Des regards multiples sur les mots 343
I.4.1. Descriptif des opérations de classement réalisées 343
I.4.2. Mise en place de réseaux de prototypes : autant de réseaux à interpréter....346
INCISE 5. APPROCHE D'UN MÉTA-PROTOCOLE....348
I.5. Une approche expérimentale guidée par le méta-regard du linguiste....348
I.5.1. Du méta-regard à l'automodification....348
I.5.2. Limites actuelles des méta-processus....349

PARTIE 3 : DE LA SCIENCE FICTION 350

LA RÉFLEXIVITÉ POUR UNE ÉVOLUTION DES REPRÉSENTATIONS DANS LE TRAITEMENT AUTOMATIQUE DU LANGAGE NATUREL 351

9.1. RÉFLEXIVITÉ, UNE INTRODUCTION....351
9.2. CONCEPTS FONDAMENTAUX DE LA RÉFLEXIVITÉ : L'APPORT DE PATTIE MAES (MAES 1987) 353
9.2.1. (Méta) Système (Réflexif)....353
9.2.2. Langages de programmation informatique et réflexivité....354
9.2.3. Architecture réflexive 355
9.2.4. Développement d'une architecture réflexive 355
9.2.4.1. Comment se fait l'auto-représentation du système ? 355
9.2.4.2. Comment programmer la partie réflexive ?....356
9.2.4.3. Quel lien existe-t-il entre le système et la partie réflexive ?....356
9.2.5. Utilisation de la réflexivité 356
9.2.6. Dangers de la réflexivité....356
9.2.7. Comment programmer la partie réflexive?....357

9.2.7.1. Réflexivité procédurale 357
9.2.7.2. Réflexivité déclarative 357
9.2.8. Vers une mise en place de processus réflexifs dans une analyse automatique 358
9.3. QU'EST-CE QU'ON NE PEUT PAS FAIRE SANS LA RÉFLEXIVITÉ ? 359
9.4. CONNAISSANCE ET RÉFLEXIVITÉ EN IA 360
9.4.1. De la méta-connaissance 360
9.4.2. Faciliter la production de méta-connaissances sur les entités linguistiques et l'analyse : Illustration avec OLMES (1993) 360
9.4.3. Intelligence artificielle, formalismes d'unification, langages de programmation : des réflexivités distinctes ? 362
Domaines de la réflexivité. 363
Constituer et articuler des points de vue. 363
9.4.3.1. IA : la réflexivité pour la modularisation et l'articulation des savoirs 363
9.4.3.2. Les formalismes d'unification : peu de réflexivité 363
9.4.3.3. Réflexions sur la réflexivité 364
Réflexivité approchée / implicite / explicite. 364
Réflexivité naturelle et réflexivité artificielle 365
9.5. TRADITION DE LA PROGRAMMATION RÉFLEXIVE 366
9.5.1. Réflexivité et Méta-niveau dans les langages orientés Objet 366
9.5.2. Clos et la Réflexivité. 366
9.5.2.1. Méta-Object. 367
9.5.2.2. Le MOP de CLOS 367
9.5.2.3. Les limites de `change-class` 368
9.5.3. Self et la réflexivité 368
9.5.3.1. Réflexion de structure en Self. 368
9.5.3.2. Réflexion de comportement 369
9.5.3.3. Moostrap : un noyau réflexif pour un langage à prototypes 370
9.6. OUTILS DISPONIBLES POUR LE TALN 371
9.6.1. Un lien Méta 371
9.6.2. Analyse Automatique et Méta-niveau. 371
9.7. RÉFLEXIVITÉ : MODIFICATION DES PROGRAMMES 374
9.7.1. Hétéro-modifications 374
9.7.1.1. Savoir ce qui se passe, vers des traces pour linguistes 374
Etape de mise au point de traceurs. 374
Les traceurs. 374
Un méta-niveau dans OLMES. 374
9.7.1.2. De la méta-connaissance à la réflexivité 377
9.7.2. Auto-modification 378
9.7.2.1. `Gaspar` : Un analyseur à mots non entièrement définis 378
Comment définir des "démons" capables de redéfinir les comportements généraux à factoriser. 378
Du méta-regard sur l'analyse à l'automodification des représentations 379
Un méta-niveau pour l'analyse avec `Gaspar` 379
Utilisation de la réflexion de structure en Self 379
Utilisations de méta-savoirs avec Gaspar. 380
1. Un méta-regard sur les objets 380
2. Sélection des savoirs à manipuler 380
3. Ajouter de la délégation. 381
4. Supprimer la délégation 381
5. Un Méta-traçage. 381
6. Un Méta-regard pour maintenir la cohérence des objets construits. 382
Classification automatique et processus réflexifs 382
Mettre en place la génération de nouveaux savoirs. 382
Affiner les processus autodescriptifs des objets 383
Création de pôles de comportements 383
Création automatique de prototypes de mot au cours de l'analyse 383
Des règles pour la génération des nouveaux objets 383
Créer un dictionnaire d'erreurs. 385
9.9. BILAN DES ACTIVITÉS GASPARIENNES ET MÉTA-PERSPECTIVES 386
9.9.1. Activités gaspariennes 386
9.9.2. Perspectives 389

GLOSSAIRE 391

RÉFÉRENCES BIBLIOGRAPHIQUES 401

ANNEXES 421

ANNEXES PARTIE 2 CHAPITRE 5 422

A-2-5.1. PHASE0 : FILTRAGE CATÉGORIEL 422
A-2-5.2. PHASE1 : AFFINEMENT DU FILTRAGE 427
A-2-5.3. PHASE3 : ANALYSE PAR UN RÉSEAU DE PROTOTYPES 428
A-2-5.4. MISE EN PLACE D'UN AFFINAGE PROGRESSIF POUR LA CONSTRUCTION DU SENS DES SÉQUENCES MOULIN À N2 434

ANNEXES PARTIE 2 CHAPITRE 7 438

A-2-7.1. ILLUSTRATION DE L'HÉRITAGE AVEC SELF SUR UN EXEMPLE 438
A-2-7.2. GÉNÉRATION DYNAMIQUE DE PROTOTYPES LEXICAUX (1) 441
A-2-7.3. GÉNÉRATION DYNAMIQUE DE PROTOTYPES LEXICAUX (2) 445
A-2-7.4. GÉNÉRATION DYNAMIQUE DE PROTOTYPES LEXICAUX (3) 447
A-2-7.5. DÉTERMINATION DES COMPORTEMENTS PARTAGÉS À UNE CLASSE CATÉGORIELLE DONNÉE 451

ANNEXES PARTIE 2 CHAPITRE 8 453

A-2-8.1. `GASPAR` : UN ANALYSEUR À MOTS IMPLÉMENTÉ AVEC SELF-4.0 453
A-2-8.2. CHARGEMENT DU CORPUS ET GÉNÉRATION DYNAMIQUE DE PROTOTYPES LEXICAUX ET DE LEURS CONTRAINTES (SUR CORPUS DE TEST) 454
Trace de la génération automatique des prototypes 454
A-2-8.3. CLASSEMENT AUTOMATIQUE DES PROTOTYPES LEXICAUX (SUR CORPUS DE TEST) 464
A-2-8.4. LA CLASSIFICATION ÉVOLUTIVE EST EN MARCHE 469
A-2-8.4.1. Recherche des comportements partagés et individuels par deux prototypes de mot sélectionnés 469
A-2-8.5.2. Construction d'un réseau de comportements partagés sur les prototypes d'arbre élémentaire 474
A-3-9.2.7. Analyse avec `Gaspar` 491
Parseur `Gaspar` 491
Analyse sur `"une petite brise la glace"` 492
Analyse sur `"analyseur à prototypes"` avec construction automatique de nouveaux prototypes de mot 504

Table des figures

Figure 1.1 : moulin à N2 : un premier classement. 49
Figure 1.2 : Classification sémantique. 50
Figure 1.3 : Nouvelle répartition sémantique des moulin à N2. 50
Figure 1.4 : Une vision unifiante de la construction du sens des séquences moulin à N2. (1) 51
Figure 1.4 : Une vision unifiante de la construction du sens des séquences moulin à N2. (2) 52
Figure 1.5 : Représentation scénarisée de moulin. 53
Figure 1.6 : Répartition lexicale dans le modèle sémantique construit. 54
Figure 2.1 : Fragment de la hiérarchie de types (Briscoe & Copestake 1993) 80
Figure 2.2 : Classement des compléments nominaux du verbe lire. 83
Figure 2.3 : Multitypage des entrées lexicales. 85
Figure 2.4 : Coercion de type sur livre. 86
Figure 2.5 : Dynamique associée à un énoncé. 87
Figure 2.6 : Interprétation avec ambiguïté-alternative. 87
Figure 2.7 : Interprétation univoque. 87
Figure 2.8 : Interprétation avec indétermination. 87
Figure 3.1 : Connaissances et TALN. 97
Figure 3.2 : Connaissance et utilisation des connaissances. 99
Figure 3.3 : ATN et contruction d'un groupe nominal. 101
Figure 3.4 : Grammaire Hors Contexte : Une Règle trop générale. 101
Figure 3.5 : Grammaire Hors Contexte : rajout de traits de nombre. 102
Figure 3.6 : Grammaire Hors Contexte : rajout de traits de genre. 102
Figure 3.7 : Grammaire indexée : dérivation pour $a^n b^n c^n$. 103
Figure 3.18 : Classe mot. 105
Figure 3.19 : Définition d'une classe. 106
Figure 3.20 : Créer des instances. 106
Figure 3.21 : Appel nu à make instance. 107
Figure 3.24 : Représentation morpho-sémantique de N1 à N2. 109
Figure 3.25 : Moulin à N2, représentation hiérarchisée des connaissances et construction du sens. 110
Figure 3.27 : De la description individuelle au prototype Nom. 114
Figure 3.28 : Clonage des prototypes Noms. 115
Figure 3.29 : Prototype et délégation chez les Noms. 115
Figure 3.30 : Traits et prototypes. 117
Figure 3.31 : Objet Self. 118
Figure 3.32 : Prototype et délégation en Self. 119
Figure 3.33 : Utilisation de classes abstraites pour prendre en compte des modifications. 120
Figure 3.34 : Prototype polygone et modification de comportement. 121
Figure 3.35 : Modification dynamique de comportement. 122
Figure 3.8 : FémSing 127
Figure 3.9 : NomFémSing 128
Figure 3.10 : Partage de valeurs entre DAGs 128
Figure 3.11 : Unification possible sur le trait accord 129

Figure 3.12 : Unification impossible sur le trait `accord` 130
Figure 3.13 : Unification ajoutant de l'information 130
Figure 3.14 : Précision du trait `accord` *de* `voile` *grâce à l'unification* 130
Figure 3.16 : Héritage et redéfinition. 133
Figure 3.17 : Héritage et ambiguïté. 134
Figure 4.1 : Telle entrée lexicale, tels comportements ? 147
Figure 4.2 : "Fast car" 147
Figure 4.3 : Des prototypes comme moules définitoires pour les entrées lexicales. 148
Figure 4.4 : De la multiplication des entités scénarisées. 148
Figure 4.5 : Des scénarios adaptés à la réalité prototypique. 148
Figure 4.6 : Tout est "prototypisable". 149
Figure 4.7 : Mise en ordre et prototypie. 149
Figure 4.8 : Recouvrement de comportements sémantiques. 150
Figure 4.9 (1) : `Moulin` *est à l'origine de la création du monde des noms : première étape, la création.* 155
Figure 4.10 (2) : `Moulin` *est à l'origine de la création du monde des noms : deuxième étape, la reproduction.* 155
Figure 4.11 : `Moulin`*, quelle représentation sémantique?* 156
Figure 4.12 : Un prototype `moulin` *qui capte les régularités lexicales et sémantiques.* 157
Figure 4.13 : `Moulin`*, un noyau de sens ?* 157
Figure 4.14 : `Moulin`*, un noyau de sens qui articule les transformations de sens observées* 158
Figure 4.15 : Certaines extensions peuvent s'appliquer aux noms d'instruments. 158
Figure 4.16 : Des `moulins` *aux* `moulins à N2`*.* 159
Figure 4.17 : Un prédicat générique pour `moulin.` 159
Figure 4.18 : Une représentation prédicative sous-spécifiée pour `moulin.` 160
Figure 4.19 : Représentation Sémantique Prototypique. 162
Figure 4.20 : Représentation Sémantique Prototypique pour `moulin Nhumain`*.* 163
Figure 4.21 : La co-compositionalité pour l'analyse des `N à Ni`*.* 164
Figure 4.22 : Représentation Sémantique Prototypique affinée. 165
Figure 4.24 : `Moulin`*, des pistes de sens.* 168
Figure 4.24 : `Moulin`*, des préférences d'interprétation.* 171
Figure 4.25 : `Moulin à bras` *et préférences d'interprétation.* 172
Figure 4.26 : `Moulin` *et noms d'instrument, des comportements sémantiques partagés.* 173
Figure 5.1 : Analyse prototypique Phase0. 180
Figure 5.2 : Prototypes et traits pour analyse prototypique Phase1. 181
Figure 5.3 : Analyse prototypique Phase1. 182
Figure 5.4 : Classification sémantique. 183
Figure 5.5 : Modèle sémantique pour les séquences `N1 à N2`*.* 183
Figure 5.6 : Sémantiques `N2` *initial et* `N1 à N2` *résultant.* 184
Figure 5.7 : Traits et prototypes pour analyse prototypique Phase2. 184
Figure 5.8 : Analyse prototypique Phase2. 186
Figure 5.9 : Répartition lexicale suivant le modèle sémantique retenu. 188
Figure 5.10 : Analyse prototypique Phase3.0. 190
Figure 5.11 : Analyse prototypique Phase3.1(A). 193
Figure 5.12 : Analyse prototypique Phase3.1(B). 194
Figure 5.13 : Analyse Phase3.1. 196
Figure 5.14 : Analyse prototypique et construction progressive du sens des séquences `N1 à N2`*.* 197
Figure 5.15 : Analyse par un réseau de prototypes. 199
Figure 5.16 : Mise en place du réseau de prototypes pour l'analyse. 200
Figure 5.17 : Une petite brise la glace (1) 201
Figure 5.17 : Une petite brise la glace (2) 202
Figure 5.18 : Prototypes et représentation des composés `N1 à X.` 205
Figure 6.1 : `Moulin` *est avant tout une machine.* 224
Figure 6.2 : Sous-spécification du prédicat associé à `moulin`*.* 225
Figure 6.3 : Un prototype pour `moulin à blé` 226
Figure 6.4 : Un prototype pour `moulin à farine` 226
Figure 6.5 : Un prototype pour `moulin à paroles` 227
Figure 6.6 : Un nouveau type de `moulin`*.* 229
Figure 6.7 : Construction ascendante de prototypes pour l'analyse. 232
Figure 6.9 : Clonage d'un prototype à construire. 234
Figure 6.10 : Déroulement d'une analyse sur : "Une petite brise la glace" (1) 235

Figure 6.10 : Résultat d'une analyse sur : "Une petite brise la glace" (2) 236
Figure 6.11 : Construction du sens des composés en moulin à N2 (Phase10). 238
Figure 6.12 : Clonage et ajustement de prototype `moulin à N2` *au cours de l'analyse.* 239
Figure 6.13 : Un réseau de prototypes pour `moulin à blé` 240
Figure 6.14 : Un réseau de prototypes pour `moulin à farine` 241
Figure 6.15 : Un réseau de prototypes pour `moulin à paroles` 242
Figure 6.16 : Un réseau de prototypes pour `moulin à prieres` 243
Figure 6.17 : Un réseau de prototypes pour `moulin à vent` 244
Figure 6.18 : Un réseau de prototypes pour `moulin à ventilateur` 245
Figure 6.19 : Des prototypes pour `le manager analyse un moulin à paroles` 246
Figure 7.1 : "stenose severe de la tronc commun gauche". 259
Figure 7.2 : Un arbre syntagmatique. 260
Figure 7.3 : Cyclade : simplifications des arbres d'analyse. 261
Figure 7.4 : [occlusion coronaire] droite 261
Figure 7.5 : occlusion droite 261
Figure 7.6 : occlusion [coronaire droite] 262
Figure 7.7 : coronaire-coronarien 262
Figure 7.8 : Les comportements des mots sont imprévisibles 263
Figure 7.9 : Un prototype pour `lesion`*.* 267
Figure 7.10 : Des Comportements et des Prototypes. 268
Figure 7.11 : Les prototypes apprennent peu à peu. 270
Figure 7.12 : Les prototypes s'organisent peu à peu. 271
Figure 7.13 : Les représentations se développent par apprentissage et organisation des savoirs. 272
Figure 7.14 : Génération à la volée de prototype catégoriel. 274
Figure 7.15 : Génération à la volée de prototypes d'arbre élémentaire. 275
Figure 7.16 : Génération à la volée de prototype d'arbre d'analyse. 276
Figure 7.17 : Génération à la volée de prototype par clonage et ajustement. 277
Figure 7.18 : Similitude formelle de syntagmes 279
Figure 7.19 : Affiner les savoirs initiaux... 280
Figure 7.20 : Pour affiner les arbres "élémentaires" à construire. 281
Figure 7.21 : Un réseau de comportements partagés. 282
Figure 7.22 : Un méta-contrôle humain pour nommer les choses 283
Figure 7.23 : absence-ensemble 283
Figure 7.24 : Receveur implicite. 288
Figure 7.25 : Receveur explicite. 289
Figure 8.1 : Représentation des mots et "représentactions" sur les mots 293
Figure 8.2 : `pontage` *et arbres associés* 295
Figure 8.3 : Description des informations associées à `pontage` 296
Figure 8.4 : Description des informations en entrée 297
Figure 8.5 : Déroulement de la génération 298
Figure 8.6 : Génération d'un prototype lexical et des prototypes d'arbres associés 301
Figure 8.7 : Un mini-réseau de hiérarchies locales 302
Figure 8.9 : Opération d'ajonction sur arbres 307
Figure 8.10 : Génération des prototypes : trace graphique (1) 309
Figure 8.11 : Génération des prototypes : trace graphique (2) 310
Figure 8.12 : Classement des prototypes : trace graphique (1) 311
Figure 8.13 : Classement des prototypes : trace graphique (2) 312
Figure 8.14 : Une démarche en spirale (1) 320
Figure 8.15 : Une démarche en spirale (2) 321
Figure 8.16 : Une démarche en spirale (3) 321
Figure 8.26 : Manipulation et modification directes d'objet (1) 334
Figure 8.27 : Manipulation et modification directes d'objet (2) 335
Figure 8.28 : Manipulation et modification directes d'objet (3) 336
Figure 8.29 : Manipulation et modification directes d'objet (4) 337
Figure 8.30 : Manipulation et modification directes d'objet (5) 338
Figure 8.33 : Classement des mots deux à deux 343
Figure 8.34 : Comportements partagés par deux mots 344
Figure 8.35 : Classement des mots en sous-famille 345
Figure 8.36 : Classement des arbres élémentaires 346
Figure 8.37 : La classification évolutive est en marche 347
Figure 9.1 : (Méta) Système (Réflexif). 353

Figure 9.2 : Le sous-graphe de classes représentant `GN -> DET N ADJ`. _____ *361*
Figure 9.3 : Analyse par les instances d'un réseau de classes. _____ *361*
Figure 9.4 : Méta-grammaire et règles. _____ *362*
Figure 9.5 : Une tour réflexive. _____ *366*
Figure 9.6 : Classes MetaObject en CLOS. _____ *367*
Figure 9.7 : Exécution de message en Self. _____ *369*
Figure 9.8 : Un lien Méta. _____ *371*
Figure 9.9 : DINOSOR 92 : un premier méta-niveau pour OLMES. _____ *375*
Figure 9.10 : OLMES 93 : une méta-grammaire pour OLMES. _____ *376*
Figure 9.11 : OLMES 94 : un méta-niveau pour l'analyse avec OLMES. _____ *377*
Figure 9.12 : Analyse par les instances dans un réseau de classe. _____ *378*
Figure 9.13 : Mémorisation des essais et des succès dans les classes d'une règle. _____ *378*
Figure 9.14 : `Moulin` *et noms d'instrument, des comportements sémantiques partagés.* _____ *379*
Figure 9.15 : Schéma global des activités gaspariennes. _____ *386*
Figure 9.16 : Activités gaspariennes pour l'analyse. _____ *388*
Figure 9.17 : Méta-perspective. _____ *389*

Annexe Partie 2

Chapitre 5

a-2-5.1 Un réseau de prototypes pour représenter le lexique
a-2-5.2 Un prototype de parser
a-2-5.3 Résultat d'analyse sur "un moulin à paroles de vipères"
a-2-5.4 Un réseau de prototypes pour "une petite brise la glace"

Chapitre 7

a-2-7.1 L'héritage et Self
a-2-7.2 Génération du prototype `pontage`
a-2-7.3 Génération du prototype `effort`
a-2-7.4 Pas de partage de comportement après génération
a-2-7.5 Le classement met en place un partage de comportement

Chapitre 8

a-2-8.1 "Welcome to Self-4.0"
a-2-8.2 Gaspar et l'interface graphique de Self : prototypes construits (1)
a-2-8.3 Gaspar et l'interface graphique de Self : prototypes construits (2)
a-2-8.4 Gaspar et l'interface graphique de Self : prototypes classés (1)
a-2-8.5 Gaspar et l'interface graphique de Self : prototypes classés (2)

Annexe Partie 3

Chapitre 9

a-2-9.1 Analyse avec Gaspar

Conventions

Polices de caractères

Corps du texte

Citation

`Code`

`Exemple`

Protocoles de lecture

(-> renvoi vers texte infra)

(<- renvoi vers texte supra)

Introduction Générale

Introduction

"Il y a en même temps une mauvaise adaptation du monde au langage, et de l'être pensant au monde."
Jean Tardieu

Ebauche pour un traitement automatique de la construction du sens

Le travail présenté ici s'inscrit dans le cadre du développement de dispositifs pour le TALN qui permettent une analyse critique des conjectures dont les analyseurs automatiques étudiés donnent une image.

Au commencement, une histoire de traces

L'origine de ce travail est né à la suite de l'utilisation d'analyseurs automatiques au cours de mon année de DEA : XTAG (Paroubek & al. 1992) et LN-2-3 (Zweigenbaum 1992). L'utilisation de ces analyseurs dans le cadre d'un séminaire sur les formalismes grammaticaux et certains outils informatiques associés a fait apparaître quelques insatisfactions. Ces logiciels ne fournissent pas d'interface satisfaisante entre l'utilisateur et le processus d'analyse en cours. L'utilisateur n'a aucun moyen d'intervenir au cours de l'analyse. Quand l'analyse réussit, l'analyseur propose une représentation graphique de la dérivation construite. Par contre quand elle échoue, la réponse rendue par l'analyseur est loin d'être satisfaisante. Dans le cas de XTAG, cette réponse peut d'ailleurs résulter soit de la donnée d'une phrase mal construite, soit de la donnée d'une phrase contenant des erreurs orthographiques ou un item lexical inconnu par l'analyseur. GASPAR (Fleury 1992) constitue une approche pour l'étude des problèmes de représentation des connaissances dans un analyseur et pour le développement de processus capables de renseigner l'utilisateur sur l'activité de l'analyseur au cours d'une analyse. Une partie de ce travail a permis de proposer une représentation possible des connaissances syntaxiques reposant sur les cadres offerts par la programmation à objets. Ce travail a aussi permis de plus de mettre en valeur le problème crucial mais non trivial qui est celui de construire un méta-langage cohérent pour exploiter les ressources mises en jeu lors d'une analyse et qui réponde aux besoins de l'utilisateur.

Dans le cadre du travail présenté ici, notre tâche a consisté à poursuivre l'entreprise permettant de rendre "lisible" les activités d'une analyse : on verra que cet objectif est atteint dans le dispositif `Gaspar` présenté ici. Cette lisibilité doit permettre aux utilisateurs des outils de l'analyse de visualiser/manipuler les entités présentes au cours d'une analyse sous la forme "linguistique" qu'ils connaissent avant d'avoir recours au dispositif qu'ils vont utiliser. On doit donc disposer d'outils qui permettent une identification des dispositifs techniques mis en place pour l'analyse et les phénomènes linguistiques représentés.

Le Traitement Automatique du Langage Naturel (TALN) : du dur désir de représenter

Le travail du linguiste est une recherche d'équilibre entre différentes contraintes et il **est condamné à rester provisoire** : les résultats sont toujours remis en question par la prise en compte de nouvelles informations (Milner 1989). Le TALN est donc une entreprise dont les résultats ne peuvent être que progressifs. On est encore loin d'atteindre un Traitement Automatique en Langage Naturel intégral. Les travaux existants ou à venir constituent, semble-t-il, une infime partie du parcours à accomplir pour atteindre ce but. Il n'est d'ailleurs pas certain que ce but ultime soit envisageable en raison justement de ce que *"la donnée minimale de langue, qui est une phrase, est toujours trop complexe par rapport à la proposition minimale de linguistique"* (Milner 1989). La difficulté principale consiste à cerner l'objet de la science linguistique et a fortiori du TALN. L'incertitude quant à la réussite de cette tâche ne permet pas d'entrevoir une issue définitive et proche pour la mise en place d'un traitement automatique sur un objet parfaitement déterminé. Il semble bien d'ailleurs qu'il soit impossible de résoudre les problèmes liés au traitement automatique en ne restreignant pas son champ d'application.

De la réalité des représentations : du renoncement à tout prévoir à l'expérimentation linguistique.

> *"Il se voyait clairement en artisan, en fabriquant, en 'fabbro' des déductions, en menuisier des propositions, des corrolaires, des 'scollies'..."*.
> Jacques Roubaud, Mathématique:, Editions du Seuil, 1997.

Vouloir entreprendre par exemple de résoudre les problèmes liés à une analyse automatique intégrale du langage naturel sur un objet aussi hétérogène et complexe semble quelque peu déraisonnable dans l'état actuel de nos connaissances sur l'objet de la science linguistique, à moins de construire des dispositifs incomplets et qui ne couvrent pas l'intégralité des phénomènes linguistiques du langage naturel. Une couverture raisonnable de la syntaxe et du lexique, pour une analyse automatique, demande des moyens considérables. De plus cela nécessite un grain très fin de représentation (pour les modèles correspondants) et parallèlement des moyens considérables pour stocker et traiter des informations extrêmement complexes (Pitrat 1992).
Un dispositif propose une représentation matérielle des phénomènes traités. Les remarques faites précédemment permettent tout de suite de préciser qu'il s'agit d'étudier des dispositifs existants ou à venir qui ne constituent pas des monstres de technicité non manipulables / gérables par des individus qui ne sont pas de purs techniciens (Marandin 1993). Dans notre cadre de travail, les dispositifs mis en place ou étudiés sont constitués par des outils informatiques développés dans le cadre de la programmation à objets. Il s'agira d'établir qu'on a bien une adéquation forte entre les opérations de connaissance que ces outils permettent et les représentations de la langue. Et dans tous les cas, on sera amené à entreprendre une analyse critique des techniques utilisées pour la mise au point du dispositif (Marandin 1993) puisque le recours à une technique est déterminé par le choix de la meilleure référence à un moment donné capable de fournir le dispositif le plus clair et le plus distinct pour la représentation. Ce travail est indispensable dans le cadre de la construction de dispositif informatique développé à l'aide de langages de haut niveau si l'on veut satisfaire une des conditions du statut du dispositif choisi à savoir : fournir le dispositif le plus clair et le plus distinct pour la représentation et ainsi permettre d'évaluer la cohérence entre le dispositif et les hypothèses théoriques suivies. On reprendra pour cela le travail présenté dans GASPAR (Fleury 1992) en tentant de résoudre ou d'améliorer les (pré-)résultats qui y étaient annoncés.

Quels outils informatiques pour la représentation des connaissances en TALN?

Ce travail vise à une réflexion sur la mise au point de la connaissance linguistique à utiliser pour l'analyse automatique. Celle-ci implique une adaptation, un affinage des connaissances linquistiques à utiliser lors de l'analyse. Le TALN pose aussi le problèmes du type d'architectures, du type de paradigmes computationnels, à utiliser ou à mettre en place pour définir un dispositif digne de produire la meilleure réponse au problème de représentation matérielle d'une activité linguistique. En particulier, le choix d'une représentation modulaire ou non des connaissances linguistiques est fondamentale pour le TALN. D'autant plus que l'utilisation de langages informatiques de haut niveau à architecture réflexive permet dans le cas d'une présentation modulaire des connaissances linguistiques d'envisager de résoudre des problèmes fondamentaux pour l'analyse automatique. A l'inverse, la non-modularité peut se révéler être une entrave à la progression de dispositifs cohérents et puissants (Habert & Fleury 1993a).
Les travaux menés en TALN depuis une quinzaine d'années ont longtemps privilégié les modèles de représentation fondées sur les taxonomies de classes. Depuis la fin des années 1980, les représentations hiérarchisées des connaissances se sont généralisées (Habert 1995):

- 1. La place croissante donnée au lexique dans les formalismes d'unification a nécessité l'utilisation d'outils pour gérer les redondances (DATR (Gazdar & Evans 1990), TAGs (Schaber & Vijay-Shanker 1992), etc.).

- 2. Les développements continus et parallèles des applications utilisant les langages à objets et des réseaux sémantiques ont convergé vers une représentation hiérarchisée des connaissances.

- 3. Enfin, les types de hiérarchie utilisés par les langages à objets et ceux utilisés en linguistique se sont rejoints sur un modèle de raisonnement non-monotone.

Dans le même temps, de nombreuses recherches sur la catégorisation (Rosch 1975-76-77-78) ont été menées en termes de prototypes. Les développements linguistiques de cette approche prototypique sont restés au niveau programmatique et n'ont pas débouché sur des projets descriptifs importants. La prédominance des approches fondées sur les taxonomies de classes restent de mise pour le TALN. Cela s'explique en partie par le fait que les classements utilisés en TALN reposent sur des années de tradition. De plus, le TALN se concentre principalement sur des domaines de savoir pour lesquels il y a le plus de recouvrement. Les domaines de savoir où la catégorisation est mouvante (non figée) se confrontent à deux types d'organisation : un mode apriorique (les taxonomies de classes), et un mode inductif (celui des prototypes). La volonté de modéliser les connaissances utilisées par le locuteur dans l'interprétation de séquences énonciatives nouvelles pose le problème crucial du mode de catégorisation le plus adéquat.

Quels objets pour représenter la mouvance ?

Pour illustrer ces phénomènes, on étudie dans un premier temps une partie privilégiée des formes dénominatives complexes, les `N1àN2` et plus particulièrement celles où `N1 = moulin` dans deux cadres de représentation:

- On présente tout d'abord un cadre de représentation (Habert & Fabre 1993) où l'on utilise des catégories prédéfinies et des modèles d'interprétation. Pour cela, une approche lexicale hiérarchisée est mise en place pour représenter les arguments possibles d'un prédicat sous-jacent. L'utilisation d'une hiérarchie de règles sert dans ce cadre à modéliser l'affinement progressif d'une interprétation.

- On utilise ensuite un cadre privilégiant une approche prototypique qui permet une classification évolutive des séquences visées.

Dans un deuxième temps, on montre comment la Programmation à Prototypes (i.e. PàP) (Liebermann 1986, Ungar & al. 1988, Self Group 1993-1996) permet de rendre compte d'une représentation dynamique des savoirs linguistiques et d'un classement automatique de ces savoirs (en utilisant le mécanisme de la délégation[1]). La PàP permet de représenter dynamiquement par apprentissage et par affinements successifs

[1] Comportements partagés "localement" par un ensemble de prototypes (-> partie 2 chapitre 3) (Liebermann 1986).

des unités lexicales en utilisant des informations recueillies sur corpus et elle permet aussi d'amorcer un début de classement de ces unités lexicales sur la base des contraintes syntaxiques attachées aux unités lexicales en construisant des hiérarchies locales de comportements partagés.

Description du travail

Plan de la thèse

Cette thèse comprend trois parties.

Partie 1 : De la réalité

Chapitre 1

Ce chapitre introduit la nécessité de construire des représentations évolutives pour modéliser les faits de langue. Les faits de langue sont évolutifs et non figés. Les processus de représentation de ces faits de langue doivent donc tenir compte de ces évolutions potentielles. Le travail de représentation doit construire des ébauches. Il faut ensuite ajuster les représentations initialement définies en tenant compte des informations que l'analyse dévoile. Pour la construction du sens, il s'agit de décrire au mieux les mécanismes qui opèrent pour réaliser cette mise en action de mécanismes complexes si on veut s'approcher au plus près de ces mécanismes.

Chapitre 2

Ce chapitre présente certains modèles qui traitent de la construction du sens et qui sont globalement cohérents par rapport à nos hypothèses de travail. Il vise surtout à mettre en lumière les difficultés rencontrées pour cette modélisation dans un domaine de savoirs mouvants. La construction du sens s'effectue par touches successives. De nombreux savoirs réalisent cette construction du sens, et ils interagissent de manière complexe. Les mécanismes qui participent à l'interprétation agissent sur les savoirs présents dans le dit, mais ils sont aussi capables de reconstruire des savoirs matériellement absents de cette chaîne du dit.

Chapitre 3

Ce chapitre présente deux types d'outils disponibles pour mettre en oeuvre des processus de représentation (la Programmation à Objet dans une taxonomie de classes et la Programmation à Prototypes) et vise à établir une comparaison critique du pouvoir expressif de ces deux cadres de représentation. Dans un cas, il est plus facile de représenter des savoirs établis donc de fixer ces savoirs dans des hiérarchies. Dans l'autre cas, le processus de représentation est contraint par des savoirs non connus à l'avance : il est donc plus difficile de présumer de ces savoirs et de leur mode de structuration et d'organisation. Il s'agit donc d'évaluer les apports pertinents de chacun de ces cadres pour la représentation du langage naturel, ce dernier articulant très bien ces deux aspects de la manipulation des connaissances.

Chapitre 4

Ce chapitre insiste tout d'abord sur le fait que le TALN est confronté à des problèmes non triviaux quand il s'agit de donner une représentation matérielle de certains faits de langue en particulier quand il s'agit de construire un dispositif informatique qui tente de résoudre ou d'illustrer les problèmes liés aux traitements automatiques de ces faits de langue. Le traitement automatique doit passer par une phase de représentation des éléments linguistiques manipulés. Cette phase de représentation est contrainte de figer les savoirs à modéliser, de présumer de ce que les savoirs représentés peuvent réaliser.

Ce chapitre introduit ensuite la démarche de représentation qui sera suivie dans ce travail via la programmation à prototypes. Il présente enfin les premiers choix faits pour une représentation de la construction du sens des séquences nominales du type `moulin à N2`

Partie 2 : Du possible

Un cadre particulier de représentation : les prototypes.

Le choix d'une représentation à l'aide de prototypes vise à illustrer le problème de l'évolution des représentations que le traitement automatique de la construction du sens requiert. Une telle approche de représentation privilégie la définition des éléments représentés de manière contextuelle puis leur spécialisation en tenant compte des évolutions contextuelles rencontrées. En termes de représentation, les prototypes permettent donc d'envisager une définition minimale du savoir linguistique qu'il est ensuite possible d'affiner. Il conviendra ensuite de mettre en lumière les mécanismes qui permettent d'affiner ces savoirs initialement représentés.

Chapitre 5

Ce chapitre est une introduction pour la mise en place d'une représentation de savoirs linguistiques avec des prototypes dans la PàP et en particulier pour la mise au point d'outils pour la construction du sens des séquences `moulin à N2`. On prend appui ici sur un modèle semantique particulier présenté dans le chapitre 1.

Chapitre 6

Ce chapitre poursuit le travail de mise au point du dispositif qui traite de la construction du sens des séquences `moulin à N2`. La démarche de représentation suivie amène à reformuler le problème traité et à proposer une nouvelle solution pour la représentation des savoirs manipulés.

Chapitre 7

Ce chapitre étend le processus de représentation de savoirs linguistiques. Il ne s'agit plus de restreindre le processus de représentation à certains types de séquences nominales mais de prendre appui sur des corpus (de taille importante) pour représenter les savoirs linguistiques à l'oeuvre sur les mots de ces corpus. Cette nouvelle approche utilise des savoirs extraits sur corpus pour construire automatiquement des structures décrivant ces savoirs puis classe les savoirs ainsi representés. Ce travail de représentation est conditionné par une phase d'extraction de savoirs sur corpus realisé par des analyseurs syntaxiques traditionnels.

Chapitre 8

Ce chapitre présente les différents outils mis en place pour réaliser les objectifs établis dans le chapitre précédent. Il met aussi en avant les limites des processus mis en place et le travail qu'il reste à definir pour affiner les résultats construits.

Partie 3 : De la "Science Fiction"

Ce qu'il reste à faire : de la méta-connaissance à la réflexivité.

Chapitre 9

Les résultats obtenus dans les chapitres précédents ont mis en avant la nécessité de disposer de processus qui réalisent des méta-contrôles sur les opérations mises en oeuvre dans les dispositifs construits. Tout d'abord, ces processus doivent permettre de tracer les mécanismes fins et complexes mis en oeuvre. Ensuite, les problèmes posés mettent en avant un point fondamental pour la résolution d'un traitement automatique des faits de langue (aussi bien pour la construction du sens que pour le classement des savoirs linguistiques). Un programme de TALN doit pouvoir travailler sur de nombreuses connaissances, sur des méta-connaissances de ces dernières tout en reconstruisant dynamiquement de nouvelles connaissances. Ce dernier point n'est évidemment pas résolu ici mais est illustré dans notre cadre linguistique. Il n'est d'ailleurs pas encore accessible. On n'en sait pas encore assez sur les mécanismes qui permettent de reconstruire des connaissances. Il convient toutefois de poursuivre le travail de recherche sur l'utilisation des méta-connaissances pour le traitement automatique du langage naturel, quitte à limiter le champ d'étude sur des phénomènes restreints qui permettent de préciser les tâches encore à effectuer.

Origines et choix

"C'est un fait que ce que nous exprimons en paroles, couchons sur le papier, est dix fois plus bête que ce que nous pensons, et cependant nous acceptons, comme les grands écrivains, de passer pour beaucoup plus bêtes que nous ne sommes et commettons ce non-sens de dire quelque chose, de le coucher sur le papier, d'exprimer une opinion, de défendre une orientation, de prendre partie pour une idée,"
Thomas Bernhard, *Dans les hauteurs.*

Objectif

Affiner les connaissances à utiliser lors de l'analyse automatique : architectures d'analyseurs et paradigmes computationnels.

Le travail présenté ici s'inscrit dans le cadre du développement de dispositifs pour le TALN qui permettent une analyse critique des conjectures dont les analyseurs automatiques étudiés donnent une image. Il ne s'agit en aucun cas ici de construire une théorie. En particulier, puisque nous allons beaucoup parler de prototype, notre tâche n'est pas de prendre appui sur la théorie des prototypes des cognitivistes pour théoriser des faits de langue. Nous souhaitons au contraire confronter certains paradigmes informatiques aux problèmes que pose le traitement automatique de la construction du sens puis construire des outils en réponse aux problèmes rencontrés. On utilise donc le prototype (dans le cadre de la programmation à prototypes) comme un outil de représentation de certains faits linguistiques dans la mesure où nous pensons qu'il peut répondre à certains problèmes que posent ces faits de langue. Cet outil de représentation conduit à construire des structures de représentation simples et ajustables pour rendre compte justement des problèmes d'ajustements qui sont à l'oeuvre dans la construction du sens dans le langage naturel.

1. Pourquoi des classifications mouvantes ?

"Tous les objets du monde ont ceci en commun d'être et de n'être que la permanence provisoire de certains changements."
Jacques Roubaud, "La boucle", Editions du Seuil 1993.

Le traitement automatique du langage naturel doit passer par une phase de représentation des éléments linguistiques contrainte à figer les savoirs à modéliser, de présumer de ce que les savoirs représentés peuvent réaliser. Un examen des comportements des mots révèle des variations qu'il semble difficile de fixer dans des structures de représentations statiques. En langue, ce qui est considéré comme un objet n'est pas stable et la manière d'être d'un objet ne l'est pas non plus (Bachimont 1995).

Prenons trois exemples:

- L'exemple de `moulin`.

Dans le Petit Robert, on trouve associées à `moulin` les définitions suivantes:

Appareil servant à broyer, à moudre le grain des céréales.
Par extension, établissement qui utilise cet appareil.
Bâtiment où ce type de machine est installé.
Moulin servant à battre, à piler, à extraire le suc par pression.
Sens figuré : moulin à paroles.
Moulin à prières : cylindre renfermant des bandes de papier recouvertes de formules sacrées pour acquérir les mérites attachés à la répétition de cette formule.
Sens familier actuel, désigne un moteur de voiture ou d'avion, `"faire tourner son moulin"`, `"emballer son moulin"`.

Cette simple énumération ne rend pas compte des nuances de sens que l'on rencontre effectivement sur notre corpus (-> annexe corpus moulin, -> partie 1 chapitre 1). Les emplois métaphoriques étendent l'interprétation de `moulin` dans des séquences comme : `moulin à paroles, moulin à ennui` ou `moulin à thèmes anglais`. De plus les énumérations précédentes ne sont ni exhaustives ni stabilisées et ne sauraient l'être : un domaine notionel est toujours ouvert, soumis à des variations ou à des créations, et ne peut donc pas s'assimiler à un inventaire fini (Franckel & Lebaud 1992).

- L'exemple de `livre`.

Un livre peut être considéré comme un `Objet-Physique`. Un livre peut aussi être appréhendé via le contenu qui le compose. Il est aussi un élément constitutif du processus de publication ou de commercialisation attaché à la réalisation de ce livre. La "polémique" entre D. Kayser et M. Kleiber autour de `livre` met d'ailleurs en évidence les difficultés inhérentes à la description sémantique des mots (Kayser 1987,1989a, Kleiber & Riegel 1989,1991).

- L'exemple de `antialcoolique`.

Danielle Corbin (Corbin 1990) montre que les règles dérivationnelles du français permettent de construire quatorze mots `antialcoolique`. Et grâce au principe d'associativité[2], chacun des mots construits possède une structure morphologique et donc une interprétation sémantique (celle-ci étant différente des autres mots). Même si les mots construits possibles ne sont pas tous attestés, *"ils ne témoignent pas seulement d'un goût immodéré de l'auteur pour les jeux de langue, ils ont surtout une valeur méthodologique et théorique. D'une part ils permettent de mesurer en grandeur réelle la différence entre le lexique conventionnel et le*

---

[2] Principe d'associativité : à une pluralité de structures morphologiques correspond biunivoquement une pluralité d'interprétations sémantiques (Corbin 1990).

lexique dérivationnel, dans lequel tout locuteur peut puiser, pour peu qu'il en ait besoin et/ou envie, d'autre part ils sont l'outil dont dispose le linguiste pour trouver et faire apparaitre les régularités qui sous-tendent le lexique construit" (Corbin 1990). (Corbin & al. 1993) montrent d'ailleurs que les sens a priori figurés apparaissant dans les mots construits peuvent avoir des origines diverses. Ces sens résultent soit de règles sémantiques appliquées à différentes strates de la construction du mot, soit d'opérations sémantiques associées à un suffixe particulier ou encore le produit nécessaire d'une combinaison de contraintes culturelles et linguistiques (Corbin & al. 1993).

Les mots indiquent des accès

(Cadiot & Nemo 1997a) montrent qu'il est illusoire de vouloir caractériser les mots par leurs propriétés intrinsèques : il est donc illusoire de classer un objet par ses propriétés physiques immédiates . Ce n'est pas tant les propriétés physiques d'un objet qui permet de le définir mais plutôt les liens ou rapports que l'on établit avec ces objets. De même, il est illusoire de vouloir lister tous les sens possibles qu'il est possible d'attacher à un mot. Hors contexte, l'expression linguistique, ne portant pas en elle-même les prescriptions interprétatives permettant de lui associer un sens unique et non ambigü, perd les prescriptions interprétatives du contexte et devient donc équivoque. La pluralité des déterminations en contexte ne se généralise pas en une signification hors contexte univoque (Bachimont 1995).

Donner du jeu aux représentations des descriptions linguistiques

La langue évolue en permanence et les résultats acquis par la description linguistique sont toujours remis en question par la prise en compte de nouvelles informations. Il en va de même pour les comportements lexicaux : les savoirs associés aux mots ne sont pas donnés une fois pour toute. Ces derniers peuvent bouger et remettre en cause des représentations construites à un moment donné. En suivant ce constat, notre travail vise à décrire puis à représenter des conditions nécessaires pour des sens possibles et non des conditions nécessaires pour des sens effectifs. Puisqu'il n'est pas satisfaisant de se contenter d'un modèle apriorique pour construire une représentation des comportements des unités lexicales, notre travail de représentation s'inscrit dans une approche expérimentale qui ne présume pas complètement des choses à représenter. L'hypothèse suivie dans ce travail pour la représentation des informations linguistiques consiste à ne pas prédéterminer de manière figée ni les structures définies pour cette représentation ni leurs classements. Nous convenons avec (Haton & al. 1991) que *"la construction d'une hiérarchie est un processus incrémental"* et qu'une hiérarchie *"évolue et s'améliore en fonction des résultats obtenus jusqu'à ce qu'une certaine forme de stabilité soit atteinte"* . Notre démarche met en place un processus de représentation évolutif : les structures de représentation construites devront pouvoir être affinées dès que de nouveaux savoirs seront mis à jour. On peut en effet considérer que les comportements des mots ne sont pas tous prédéfinis mais que ceux-ci "émergent" dans le contexte dans lequel ces mots "agissent". Il n'est donc pas raisonnable de les considérer comme acquis par définition. Il s'agit au contraire de les mettre en lumière ainsi que les corrélations multiples qui existent entre les mots dans un flot continu de discours. Notre démarche consiste en quelque sorte à "faire émerger" les comportements des mots puis à les représenter ou à affiner les représentations existantes et enfin à classer les structures de représentation construites.

2. Modèles pour la construction du sens : critiques et discussion

> *"Le sens d'un mot en langue naturelle est peut-être le meilleur exemple de tous les éléments qui peuplent notre monde naturel : il faut bien connaître une langue pour appréhender les sens multiples d'un mot, et le même mot contribue par ailleurs à définir le sens de tous les autres mots. Aucun aspect de notre monde naturel et vivant ne peut être classifié à partir de délimitations nettes : on ne peut en faire un domaine dont on tracerait la carte."*
> (Varela 1986)

2.1. Savoirs linguistiques et heuristiques : comment représenter la mouvance ?

L'étude de certains modèles sémantiques (Fuchs et Victorri 1990, Pustejovsky 1991, Briscoe et Copestake 1994, Kayser 1987-1992, Rastier 1987-1992) nous oriente vers une représentation de la construction du sens qui se réalise sous forme de processus qui agissent en interaction. Notre travail s'inscrit fondamentalement dans la lignée de ces différentes approches, même si nous nous montrons critique sur certains aspects de ces modèles. Ces modèles visent tous à concevoir la construction du sens comme la mise en action de mécanismes. Ces approches s'écartent d'un appariement de termes et de sens. Elles ne consistent pas non plus à recourir uniquement à de l'induction ou à du raisonnement. Il s'agit plutôt d'établir un compromis entre des informations figées et des heuristiques qui peuvent enrichir ces informations. Interpréter c'est donc parcourir les chemins possibles à partir d'un savoir minimum que des pistes de sens permettent d'affiner. De même, dans les situations de communication, les savoirs linguistiques à l'oeuvre s'ajustent en tenant compte des incidences multiples que ces situations induisent. Ce qui est en jeu, c'est la confrontation entre la représentation de savoirs minimaux et les mécanismes qui permettent d'étendre ces savoirs en fonction des réalités contextuelles. Il s'agit pour certains d'organiser les savoirs de manière hiérarchique de telle sorte que des processus attachés à ce type de représentation permettent de reconstruire le sens. Pour les autres, il s'agit au contraire de décrire plus précisément les mécanismes qui réalisent la construction du sens. Dans le domaine particulier des séquences `moulin à N2`, la construction d'une telle hiérarchie de savoirs n'est pas triviale et pousse à réfléchir sur l'organisation des connaissances dans un tel cadre. Il est important de souligner que ce n'est pas tant le problème de construire une hiérarchie qui importe le plus[3], mais bien de donner les moyens de faire évoluer cette hiérarchie. Plus précisément, dans le domaine restreint étudié, l'interprétation du patron `moulin à N2` met en lumière les difficultés à organiser les savoirs sur des formes qui autorisent des extensions de sens : faut-il pour cela considérer que ces extensions de sens doivent être prises en compte dans une hiérarchie de concepts, ou bien dans une hiérarchie lexicale? De plus quel est le lien de dépendance entre ces types de hiérarchies? Les entrées lexicales se réduisent-elles à l'addition d'informations héritées d'une hiérarchie conceptuelle?

2.2. Ce que ces modèles peuvent (ne peuvent pas) faire ?

2.2.1. Kayser et la profondeur variable

L'approche suivie par D. Kayser vise à établir un équilibre entre des savoirs établis (non modifiables par inférence) et du raisonnement (les processus qui affinent par inférence les savoirs établis). La profondeur variable suppose en effet une hiérarchie de concepts aux noeuds de laquelle sont attachées des connaissances, des rôles qui permettent des inférences. Cela structure le savoir sur le monde effectivement mis en oeuvre. Cette hiérarchie est le support de l'inférence. Le savoir sur le monde est structuré et cette organisation guide le travail inférentiel. Ce type d'approche permet tous les remodelages possibles et met clairement en avant la difficulté qu'il y a à articuler les différents niveaux de savoirs manipulés. Le problème étant bien entendu de rendre compte dans la description des savoirs linguitiques de leurs interactions multiples et de leurs potentielles évolutions respectives. La mise en place d'articulations dynamiques entre les savoirs décrits n'est pas un problème qui se laisse résoudre facilement surtout si les structures de représentation utilisées ne permettent pas de tenir compte de tels ajustements toujours possibles. De plus une approche conceptuelle à la Kayser est difficilement compatible avec le type d'outils de représentation que nous utilisons. Il semble en effet difficile de penser et de représenter une articulation cohérente des connaissances conceptuelles et des connaissances lexicales manipulées dans le modèle de représentation du langage naturel proposé ici. De bons outils réalisant cette articulation ne sont pas encore disponibles en raison justement de la difficulté à la mettre en place.

2.2.2. Le lexique génératif de Pustejowsky

---

[3] Ce n'est d'ailleurs pas le plus difficile, avec la programmation à objets tout est objet, i.e. on peut tout représenter dans une taxinomie de classes hiérarchisées.

De même, le lexique génératif de Pustejovsky suppose une hiérarchie aux noeuds de laquelle sont attachés des connaissances, des scénarios qui permettent des inférences. Là encore ce type de représentation[4] reste en deçà des possibles de réalisation d'une entité donnée. Puisque les faits de langue sont soumis à des évolutions permanentes, leur représentation se trouve en permanence remise en question et il est difficile de prévoir tous les scénarios possibles à attacher aux mots.

2.2.3. Les approches Objet pour le TALN

Si on choisit de représenter les savoirs linguistiques dans une taxonomie de classes, on se heurte de front au modèle de représentation sous forme d'objets dans lequel "un objet est (principalement) ce qu'il est". Les outils pour la représentation dans une telle approche ont tendance à figer les choses. Quand il s'agit de représenter une matière organique, les problèmes surgissent rapidement. Si on accepte de penser en termes d'objet, c'est-à-dire d'admettre que tout est réductible en un ensemble de propriétés définitoires stables, une telle approche se justifie. Malheureusement, on n'en saura jamais assez sur les faits de langue. Le langage crée en permanence. Un fait de langue peut prendre le contre-pied de résultats établis, bousculer les règles définitoires. On est donc contraint à des esquisses de représentation.

- Les classes de mots de Gross

G.Gross (Gross 1994, Le Pesant 1994) travaille dans le cadre théorique d'un lexique composé de phrases et non de mots isolés et propose un modèle pour le traitement informatisé du lexique à des fins notamment de traduction automatique. Il construit pour cela des classes de noms clairement spécifiées. Elles sont définies non pas "dans l'absolu", mais toujours en fonction d'un opérateur (plus précisément en fonction d'une construction syntaxique). Notre travail de classification vise lui aussi à classer les mots en prenant appui au départ sur des savoirs syntaxiques associés aux mots. Il ne s'agit pas de préconstruire une classification générale en dehors de toutes réalités textuelles mais de reconstruire automatiquement cette classification des mots en tenant compte justement de réalisations textuelles dans lesquelles le processus de classification doit tout d'abord retrouver les comportements lexicaux puis les classer : les classes de mots ne sont pas données de manière définitive mais construites sur corpus, dans la mesure où ces comportements sont soumis à des variations quand on passe d'un corpus[5] à un autre. Le travail sur les sous-langages révèle que les classements généraux[6] que l'on peut établir sur la langue ne sont pas toujours pertinents[7]. L'utilisation des mots dans des domaines de spécialités particuliers peut induire des comportements linguistiques nouveaux qui mettent justement en lumière un nouveau type de rapport avec l'objet visé dans ce domaine particulier (Cadiot & Nemo 1997). Si notre travail s'éloigne sur ce point de l'approche suivie par Gross, il en retient la reprise de l'approche harrissienne (-> partie 1 chapitre 1) (Daladier 1990, Dachelet 1994, Sager & al. 1987, Habert & Nazarenko 1996) : en fait la détermination de classes d'opérateurs et d'opérandes par le

---

4 Pustejowsky manipule un treillis de types figés.

5 *"Two findings repeatedly come out of corpus-based studies: first, individual linguistic features are distributed differently across registers, and second, the same (or simular) linguistic features can have different functions in different registers"* (Biber 1993).

6 *"Corpus-based research shows that our intuitions about lexicals patterns for grammatical structures, for many words there is no general pattern of use that holds across the whole language; rather, different word senses and collocational patterns are strongly preferred in different registers"* (Biber 1993).

7 *"The collocational patterns (confirmed by concordance listings) identify two surprising fact about (these) words : 1) 'certain' is commonly used to mark uncertainty rather than certainty; and, 2) certainty is rarely expressed in social science at all. Thus, the most common collocations for 'certain' in social science reflect a kind of vagueness, marking a referent as possibly being known (by someone) but not specified in the text (e.g., a certain kind of..., in certain cases..., there certain indications that...). These collocations are relatively rare in fiction. In contrast, those collocational patterns for 'certain' that directly state that someone or something is certain - he/she/they/you/it + BE + certain and i/we + BE + certain - are extremely rare in social science but relatively common in fiction"* (Biber 1993).

fonctionnement linguistique. Il ne s'agit donc pas d'une sémantique ontologique reposant sur des existants hors de toute réalisation linguistique.

- Hiérarchies rigides pour le TALN

Même si un choix de représentation qui manipule la notion d'objet dans une taxonomie de classes est adéquat pour fixer un domaine de savoir établi dans une structure hiérarchisée, il s'avère parfois délicat de construire puis de prendre en compte les évolutions de telles représentations hiérarchiques quand celles-ci décrivent des domaines de savoir non stabilisés. Dans une telle approche, la mise en contexte des instances de classes peut révéler la nécessité d'une mise à jour de la structure initialement adoptée pour la définition des classes. Dans le cas de la construction d'une hiérarchie lexicale pour laquelle on souhaite inscrire dans la définition des entrées lexicales une articulation forme-sens, une telle représentation doit affronter le problème complexe de la confrontation d'un savoir établi et d'un savoir à venir. La prise en compte de nouvelles informations, de résultats issus de l'analyse, de configurations énonciatives particulières... peut conduire à modifier, à affiner la représentation initiale. Dans le cadre de notre travail sur la modélisation des configurations nominales du type `moulin à N2`, une telle évolution sur le savoir initial est à l'oeuvre sur des formes du type `moulin à paroles` pour laquelle `moulin` devient `+humain`. Et c'est justement le travail de mise à jour d'une hiérarchie qui pose problème. Plus on structure un domaine de savoir, plus les liens de dépendance entre les noeuds du graphe d'héritage se complexifient, et il devient ainsi délicat de modifier cette structure hiérarchique. (Godard & Jayez 1994) propose le multitypage des entrées lexicales (`livre : objet matériel & objet informationnel`) pour prendre en compte les différentes réalisations sémantiques d'une entrée lexicale. On est confronté ici à une difficulté liée à la mise en place de hiérarchies rigides qu'il est souvent difficile de faire évoluer : le multitypage (de même que la coercion de type à la Pustejowsky ou les règles lexicales de Briscoe) s'accommodent d'un graphe des types figé. Vouloir classer les choses de manière figée pose donc des problèmes difficilement compatibles avec une nécessaire évolution dans les représentations construites des savoirs linguistiques à un moment donné. Il ne s'agit pas ici de dire que la représentation sous forme hiérarchique est inconcevable mais d'adapter les structures hiérarchiques aux processus qui sont à l'oeuvre dans la construction du sens des faits linguistiques. Ce travail vise aussi à établir une comparaison critique du pouvoir expressif de deux cadres de représentation : la Programmation à Objet dans une taxonomie de classes et la Programmation à Prototypes. Dans un cas, il est plus facile de représenter des savoirs établis donc de fixer ces savoirs dans des hiérarchies. Dans l'autre cas, le processus de représentation est contraint par des savoirs non connus à l'avance : il est donc plus difficile de présumer de ces savoirs et de leur mode de structuration et d'organisation. Il s'agira donc aussi d'évaluer les apports pertinents de chacun de ces cadres pour la représentation du langage naturel, ce dernier articulant très bien ces deux aspects de la manipulation des connaissances.

2.2.4. Rastier et la sémantique différentielle

Dans l'approche sémantique suivie par Rastier, c'est le global (i.e. les textes) qui détermine le local. Un mot ne se définit pas par rapport à des états de choses ou à des états mentaux mais par rapport à ses contextes : un mot (occurrence) ne se définit que dans et par un contexte et reçoit des déterminations par le texte. Le sens d'un mot ne lui est donc pas permanent, il est toujours le produit d'une interprétation fondée ou non sur des inférences et si le sens résulte d'une interprétation, il convient pour en rendre compte de décrire au mieux les parcours interprétatifs et les contraintes linguistiques sur ces parcours. De fait, placer les mots sous l'autorité d'un type revient à les décontextualiser et à les détextualiser et le rapport du type à l'occurrence s'éclaire si l'on considère que les types ne préexistent pas aux occurrences mais sont reconstruits à partir d'elles (Rastier 1995). Nous verrons (-> partie 2 chapitres 5, 6) que notre travail est contraint par une décontextualisation des faits de langue décrits, ce qui a priori nous éloigne de ces positions de fond. Il ne s'agit pas d'un choix théorique mais d'une condition de travail initiale minimale pour illustrer le traitement de certaines séquences nominales. En fait, dans l'état actuel de notre travail, les déterminations globales que les mots peuvent recevoir des textes dans lesquels ils s'inscrivent ne sont pas accessibles.

- Dans la première partie de ce travail (-> partie 2 chapitres 5, 6), on étudie des séquences nominales du type `moulin à N2` dont l'interprétation est principalement déterminée par le local (i.e. les

composants de ces séquences) : ce sont les valeurs locales des composants qui déterminent les valeurs globables des composés étudiés. Cependant dans le cadre particulier étudié ici, il semble possible de faire converger une détermination du local sur le global et une détermination du global sur le local. L'absence (provisoire) de telles contraintes globales sur la production des séquences analysées force pour le moment l'analyse à donner plus de poids aux informations sémantiques locales (en particulier celles attachées au N2 dans le composé) et donc à l'orienter dans une version compositionnelle : ce n'est pas un choix théorique de fait. Les savoirs lexicaux représentés étant ajustables dynamiquement, il reste possible d'affiner ces interprétations si des processus décrivant le rôle des textes sur ce type de séquence sont disponibles : soit en récupérant des contraintes sémantiques liées au développement de l'analyse, soit en tenant compte des contraintes générales que les séquences analysées révèlent. Ce dernier point n'est pas traité dans cette première phase de notre travail mais la question reste ouverte. (Fabre 1996)[8] montre d'ailleurs qu'il est possible de combiner des informations générales et des informations spécifiques pour une interprétation des noms composés. Nous souhaitons à terme pouvoir confronter les informations sémantiques disponibles, aussi bien contraindre les opérations que le global impose sur le local et réciproquement. Cette dernière exigence pose d'ailleurs le problème de savoir de quelle manière représenter les contraintes globales sur un texte dans un formalisme qui manipule des prototypes.

- Dans la deuxième partie de ce travail (-> partie 2 chapitres 7, 8), on travaille sur des savoirs extraits à partir de corpus. Là encore, l'impact des textes scrutés sur les éléments constitutifs de ces textes n'est pas complètement formalisé. En particulier, on ne tient compte pour le classement des mots représentés que des descriptions syntaxico-sémantiques qui leur sont associées. Si les savoirs syntaxiques correspondent à des réalisations avérées sur les corpus étudiés, certaines valeurs sémantiques utilisées sont quant à elles des prédéfinitions non issues du travail d'extraction de savoirs sur ces corpus. Là encore si on dispose de processus capables de décrire les liens entre les types ou les genres des textes sur les occurrences des mots rencontrés, il reste possible d'ajuster les mécanismes de représentation définis pour en tenir compte.

2.2.5. Continuité et TALN

La notion de continuité[9] pour décrire les processus sémantiques n'est pas une nécessité absolue : (Kayser 1995) montre qu'il est possible avec des modèles discrets de représenter une continuité conceptuelle. Mais l'utilisation de la continuité pour la modélisation des processus sémantiques (Fuchs & Victorri 1994) met surtout l'accent sur les difficultés rencontrées pour mettre en oeuvre (dans un dispositif) les phénomènes linguistiques ainsi décrits. En effet le point important soulevé par ce type d'approche est clairement énoncé dans (Victorri 1994) : *"/./ representing continuity on a machine is all but a simple problem"*, et cette question ne se laisse pas résoudre facilement. La mise en oeuvre d'un dispositif informatique pour représenter ce type de phénomène rejoint d'ailleurs un point important de notre travail : la notion de **contrôle** et la **méta-représentation** (-> partie 3 chapitre 9). Si le connexionnisme par exemple est capable de rendre compte des phénomènes de la continuité, *"it has a drawback that prevents it from playing in continuous modeling the same role as classical IA tools play in discrete modeling. This flaw is related to an important notion developped in IA : the notion of control. A connectionist network remains 'a black box' which does not allow much reasoning about its functioning"*. Or, notre travail vise aussi à mettre en avant la nécessité de disposer de processus de contrôle pour le TALN. Si les descriptions/processus linguistiques à représenter (de manière discrète ou continue) sont potentiellement évolutifs, il est impératif de pouvoir évaluer les évolutions réalisées et donc de disposer d'outils de contrôle. De plus, si les processus sémantiques à l'oeuvre au cours de l'interprétation sont capables de créer de nouvelles connaissances, les

---

8 *"The aim of our research is to define as precisely as possible the border line between what can be regularly described with general linguistic mechanisms, and what has to do with subregular or irregular phenomena which depend on corpus charactericts." /.../ "This work describes a domain-independent model for the interpretation of nominal compounds; it shows how general knowledge and domain-specific information may be combined for the interpretation of nominal compounds"* (Fabre 1996).

9 Cette notion vise à *"représenter la signification en termes non plus en termes de valeurs discrètes et disjointes mais en termes d'un continuum de valeurs"* (Fuchs & Le Goffic 1985). La formalisation mise en place pour représenter ce continuum de valeurs se situe dans le cadre de la géométrie différentielle.

dispositifs à mettre en place sont contraints non seulement de travailler sur de nombreuses connaissances mais aussi d'en reconstruire. On sent bien d'ailleurs que dans certaines phases d'une analyse automatique par exemple, si on pouvait disposer d'un méta-regard sur le déroulement de l'analyse et fournir des informations supplémentaires aux moments critiques de celle-ci, il serait possible d'infléchir le cours de l'analyse. Les dispositifs informatiques doivent donc impérativement prendre en compte ces phénomènes à travers les systèmes de représentation qu'ils développent. C'est en ce sens qu'il convient de les interroger pour que les modèles qu'ils représentent puissent être considérés comme des représentations empiriquement correctes.

2.3. Problèmes posés par ce type d'approche

2.3.1. Représenter, c'est figer un peu.

- Comment représenter la base de savoirs initiaux pour tenir compte de ce que l'on a affaire à des informations évolutives et non figées ? Comment enrichir ces savoirs établis quand de nouveaux types de descriptions linguistiques révèlent une certaine stabilité ?

Si les descriptions linguistiques se bornent à refléter un savoir partagé sur le monde, en particulier si le lexique est organisé (de manière hiérarchique ou non) sur la base de connaissances (le monde objectif), on perd une grande partie des mécanismes à l'oeuvre dans le langage naturel. L'interprétation permet en effet la mise en oeuvre de processus sémantiques qui créent de nouvelles acceptations et de nouveaux emplois non dérivables de la connaissance encyclopédique. Dans ce cas les parcours (arborescents ou non) n'apprennent rien que l'on ne sache déjà (Rastier 1995). Figer les savoirs linguistiques dans des taxonomies figées et non évolutives ne convient guère à la structure sémantique des langues et assez mal à celle des discours de spécialité (Biber 1993) : il n'est pas certain que les différents niveaux de l'arbre appartiennent à une même pratique (Rastier 1995). Un travail de représentation des faits de langue doit donc s'attacher à rendre les savoirs de base évolutifs en sachant qu'un ajustement est toujours potentiellement à venir sur ces savoirs.

2.3.2. "Ajuster les processus d'ajustements" : approches de la réflexivité

L'utilisation d'heuristiques pour étendre les savoirs initiaux représentés dans des configurations particulières pose quant à elle au moins deux types de problèmes.

- Comment représenter le fonctionnement des heuristiques pour rendre compte à la fois de leurs évolutions potentielles et de leurs applications sélectives (éviter de surgénérer) ?

Un dispositif pour le TALN est contraint d'enrichir ou d'affiner les processus inductifs mis en place (il en va de même pour les savoirs représentés). Dans la mesure où il est impossible de lister et de prévoir tous les scénarios possibles à attacher aux mots, il est nécessaire de pouvoir ajuster les processus définis au cas où telle heuristique doit justement être remodelée. On est donc confronté au problème fondamental qui consiste à travailler sur un nombre important de connaissances interconnectées, toujours en mouvement, remodelées par des processus évolutifs et capables de s'enrichir.

2.4. PàP : une nouvelle approche pour représenter la mouvance

2.4.1. Choix et perspectives

Notre travail vise à ne pas partir de représentations préconstruites et classées hiérarchiquement mais à reconstruire automatiquement des représentations de ces savoirs associés aux mots, en partant de savoirs peu déterminés qu'il s'agira ensuite d'affiner en tenant compte des informations disponibles. Il convient ici d'insister sur le point suivant. Il ne s'agit pas pour nous de nier en bloc toute forme de savoirs stabilisés dans ces processus de représentation automatiques des descriptions linguistiques. Les savoirs généraux en langue

ne manquent pas. Les dictionnaires constituent d'ailleurs une trace de la fixation sur le moyen terme de ces savoirs. Il ne s'agit donc pas de tout réinventer. Notre travail vise au contraire à prendre appui sur des connaissances très générales (on utilise au départ par exemple un cadre de sous catégorisation approximatif) qui sont affinées voire remodelées et changées au gré des observations rencontrées de la même manière que les savoirs linguistiques évoluent (les dictionnaires sont mis à jour périodiquement...). Il s'agit ainsi pour nous de tendre vers un modèle cohérent qui ajuste les descriptions linguistiques dès qu'une certaine stabilité apparait. De même, dès qu'un certain type de comportement se révèle pertinent, on ajuste les savoirs définis pour en tenir compte. Notre objectif est double:

- 1. Extraire ces savoirs sur des corpus donnés et construire dynamiquement des structures de représentations évolutives.
- 2. Repérer les similitudes de comportements entre les unités lexicales rencontrées sur les corpus visés. On amorce ensuite une classification évolutive des unités de langue représentées.

2.4.2. Que fait la PàP ?

La représentation prototypique est un représentation progressive, en devenir.

La PàP (Liebermann 1986, Self Group 1987-1996) peut être considéré comme un outil privilégié pour la représentation de domaines de savoirs pour lesquels on ne dispose pas de catégories bien précises. La PàP conduit à penser différemment de la PàO pour construire une représentation informatique d'un certain domaine de connaissances. Il ne s'agit pas de partir d'une somme de connaissances figées et connues par avance mais de construire progressivement les entités informatiques suivant les connaissances dont on dispose sur le domaine visé. Si les savoirs à représenter ne sont pas connus de manière définitive, il est possible de commencer le processus de représentation en utilisant les savoirs déjà recensés puis d'affiner dynamiquement les objets construits dès que de nouveaux savoirs sont disponibles. A la limite, cette démarche de représentation n'a pas à se préoccuper de la forme terminale des objets à construire. Elle peut démarrer à partir d'une structure sous-déterminée qui sera mise à jour dès que des informations sont disponibles. **Cette flexibilité dans la PàP rend possible une mise en oeuvre du processus de représentation sur des domaines de connaissances évolutives**. Le choix des prototypes semble cohérent avec la volonté de représenter des opérations dynamiques d'interprétation. Les prototypes sont malléables : ils se construisent contextuellement et leur spécialisation se définit suivant les évolutions contextuelles. Ils doivent commencer par fixer un savoir minimal - qu'il est possible d'attacher à une entrée lexicale - puis finir par étendre ce noyau de sens dans les directions permises par les configurations interprétatives rencontrées. La PàP met en place une représentation de savoirs que l'on peut considérer comme peu organisée au départ. Les savoirs représentés sont répartis sur une myriade de structures peu connectées entre elles à priori. **La PàP ne rejette pas pour autant toute idée de classification, bien au contraire**. Elle met d'ailleurs en place un système d'héritage simple basé sur la délégation qui permet de factoriser localement des comportements[10] partagés. Il reste donc possible d'affiner la représentation du domaine de savoirs étudié en reclassant les structures construites sur la base des "généralisations" potentielles que ces structures peuvent induire.

2.4.3. PàP et TALN : un certain point de vue

Notre utilisation de la PàP n'est pas celle qui est habituellement de mise avec ce type de langage. Notre approche met en avant la représentation des connaissances associées aux faits de langue avec la PàP alors que celle-ci a d'abord été conçue comme une boite à outils pour la réalisation d'interfaces graphiques. De

---

[10] Utiliser un langage de programmation pour modéliser des objets du monde réel conduit à faire référence à deux types de comportements. D'une part les comportements de l'objet réel décrit, dans notre cadre de travail, cela concerne par exemple les comportements syntaxiques ou sémantiques des mots. D'autre part les comportements de l'objet informatique qui sert de support à l'objet décrit, par exemple des opérations qui permettent d'afficher ou d'imprimer l'objet informatique visé (Lemaire & Rechenmann 1995). La factorisation des comportements partagés possible avec la PàP peut concerner ces deux niveaux de description.

plus notre travail ne s'appuie pas sur une conception cognitiviste du prototype. Il s'agit plutôt de le considérer comme un mode, un moyen de représentation adéquat pour décrire certains phénomènes du domaine visé. Ce qui est en jeu, c'est la confrontation dans la représentation entre une généralisation minimale et une évolution potentielle de ces représentations.

3. Objectifs

Notre travail se développe autour des thèmes suivants:

3.1. Travail sur moulin

Dans cette première partie de notre travail, on cherche à représenter la construction du sens des configurations nominales du type moulin à N2. Si le sens n'est pas donné, sa représentation ne peut pas être le résultat d'un simple codage. La représentation des phénomènes de la construction du sens met en avant la difficulté d'organiser les savoirs à l'oeuvre dans les énoncés. Pour réaliser ce premier objectif, on veut mettre en place une représentation matérielle de la construction du sens de ces configurations qui permette la mise en action de processus travaillant en interaction. Pour cela, on construit un dispositif informatique qui manipule des prototypes. Ce dernier doit identifier, classer et décrire ces formes pour construire une taxonomie ouverte et évolutive de cette classe particulière de composés.

3.2. Un dispositif pour la représentation et le classement automatiques des mots et de leurs comportements

Représentation des connaissances à partir de savoirs extraits sur corpus

Notre démarche vise ensuite à mettre en lumière dans des domaines restreints les comportements des mots ainsi que les corrélations multiples qui existent entre les mots dans un flot continu de discours. Elle consiste en quelque sorte à "faire-émerger" ces comportements des mots puis à les représenter ou affiner les représentations existantes. L'hypothèse suivie ici est qu'il y a peu de sens à vouloir faire de l'acquisition sémantique en dehors d'un sous-langage. L'apprentissage sur corpus vise à repérer les arbres élémentaires de dépendance entre mots (relations opérateur-opérandes) et les contraintes sur les combinaisons de ces arbres. Nous envisageons de construire des représentations informatiques des mots et d'attacher à ces objets leurs comportements syntaxiques et sémantiques en ne prédéterminant pas complètement les savoirs à représenter. Notre travail s'inscrit dans cette reprise de l'approche harrissienne (Sager & al. 1987, Habert & Nazarenko 1996) : en fait la détermination de classes d'opérateurs et d'opérandes par le fonctionnement linguistique. Pour cette représentation nous utilisons le corpus MENELAS (Zweigenbaum 1994) à partir duquel sont extraits des savoirs pour construire des structures de représentations lexicales sous la forme de prototypes. Les savoirs extraits sont des arbres d'analyse fournis par un outil d'extraction terminologique (Bourigault 1993), ces arbres étant ensuite simplifiés dans le but de déterminer les arbres minimaux qu'il est possible d'associer aux entrées lexicales (Habert & Nazarenko 1996).

Du lexique jaillit un réseau de prototypes

Notre tâche vise à s'élever du particulier au général et à procéder à des regroupements pour y découvrir les faces cachées derrière les détails. Puisque la hiérarchie est un artifice, utile d'ailleurs pour classer les choses, nous ne nous imposons pas un classement prédéfini des savoirs à représenter, mais nous choisissons une démarche qui tente de reconstruire ce classement en tenant compte des informations délivrées par le travail d'extraction de savoirs à partir de corpus. Ce classement sur les entités représentées vise à la contruction de réseaux entre les prototypes définis à la manière d'un parcours entre ces prototypes pour y trouver le chemin interprétatif adéquat. Le réseau mis en place ne construit pas pour autant une représentation du sens attaché aux unités lexicales ou au structures syntaxiques représentées, il doit proposer des chemins interprétatifs qui ne constituent que des amorces d'interprétation devant être affinées

par un travail d'interprétation plus fin. Notre approche vise donc à définir une démarche en spirale : construire des représentations par affinements successifs avec projections de savoirs soit constitués soit établis par ailleurs; puis ajustements et affinements des représentations à chaque nouvelle étape.

3.3. Un méta-niveau pour le Traitement Automatique du Langage Naturel

Puisque notre approche se situe dans un cadre de modélisation des connaissances par la construction de représentations évolutives, il est nécessaire de disposer d'éléments "dégagés" de ces représentations et qui permettent d'évaluer ce qui est en jeu pour ensuite permettre une prise en compte statique/dynamique des éléments pertinents susceptibles de faire évoluer les représentations initiales. Notre activité de production de faits de langue dispose de mécanismes de contrôle sur ce qui est produit : leur but principal est d'assurer une cohérence linguistique sur ce qui est produit (et notamment de produire une articulation forme-sens adéquate pour la compréhension). La production linguistique est faite de corrections, de reprises, de commentaires; les signes émis ne sont généralement pas de simples médiateurs physiques entre le "vouloir-dire" et le "dire construit" mais peuvent aussi matérialiser des ruptures, des précisions, des raffinements,... qui s'interposent dans le trajet de cette production (Authiez-Revuz 1989). On est ainsi confronté à des processus qui agissent à un méta-niveau. Ils accompagnent la formation du dire et les mécanismes à l'oeuvre sont "auto-contrôlés" dans le déroulement de cette construction. Si de tels mécanismes existent dans le langage, une représentation des connaissances de la langue ne doit pas les éviter. On sent bien d'ailleurs que dans certaines phases d'une analyse automatique par exemple, si on pouvait disposer d'un méta-regard sur le déroulement de l'analyse, et si on pouvait fournir des informations supplémentaires aux moments critiques de celle-ci, il serait possible d'infléchir le cours de l'analyse. Mais il ne faut pas se faire d'illusions hâtives sur une utilisation globale de tels mécanismes. Notre difficulté à isoler les interactions complexes des savoirs à l'oeuvre dans la production d'énoncé accentue la représentation et la construction de tels mécanismes. Il semble possible d'y voir plus clair sur une utilisation de processus de contrôle à un méta-niveau si l'on restreint leur étude à des niveaux de complexité moindre. Notre réflexion portera aussi sur cet aspect fondamental dans la représentation des connaissances pour un dispositif informatique. D'autant plus que les langages de haut niveau que nous utilisons mettent en place des mécanismes qui favorisent les approches réflexives. Celles-ci peuvent éclairer une étude sur la construction de mécanismes de contrôle pour l'analyse et sur la définition même de ces mécanismes : peut-on construire de tels mécanismes capables de restructurer les représentations initiales, de modifier les opérations pré-définies ? C'est en tout cas ce que nous essaierons de présenter dans notre cadre linguistique choisi.

3.4. Cadre choisi : la PàP (Programmation à Prototypes)

Self[11] est le langage qui est utilisé ici pour représenter les mots et leurs comportements. Il a été conçu en 1986 par David Ungar & Randall Smith (Ungar & al. 1987). La première implémentation a été réalisée à Stanford en 1987, la dernière version (Self-4.0) est disponible depuis juillet 1995. Ce langage est désormais développé par Sun Microsystems avec beaucoup de moyens, ce qui semble indiquer l'importance de ce type de représentation dans le développement de la Programmation à Objets. Self est un langage qui permet l'héritage multiple et l'héritage dynamique via la délégation de comportements partagés par un ensemble d'objets. Un prototype ou un objet en Self est une entité composée d'attributs. Ces attributs peuvent porter des données, des méthodes ou pointer sur d'autres objets. Les objets définis dialoguent entre eux via un mécanisme d'envoi de messages.

4. Ce qui a été réalisé

---

11 *"Self is more than a language. It is the combination of a language, a set of reusable objects, a programming environment, a user interface, and a virtual machine. /./ Self is based on prototypes and has no classes or meta-classes. Simplicity is ensured through uniformity: all data are objects, all expressions are message sends, and computation is performed solely by sending messages"* (Agesen 1996).

4.1. Travail sur moulin

Moulin existe seul pour désigner un moteur, une machine ou un bâtiment où un certain type de moulin est installé ou l'a été. Moulin fonctionne aussi comme un opérateur syntaxique qui prend en arguments une préposition et un nom moulin à blé, moulin à farine... Moulin est donc un opérateur : notre prototype moulin initial peut déclencher la construction d'un prototype moulin à N2 qui sera construit s'il est possible de lui adjoindre un prototype Prep de forme "à" et un prototype de catégorie Nom en position N2. On associe à moulin un prédicat dont la valeur prédicative dépend du N2. Ce choix permet tout d'abord de maintenir une représentation qui s'accorde avec bon nombre de noms d'instrument. Ensuite, la sous-spécification du prédicat peut être vue soit comme un problème de spécialisation lexicale, soit comme un phénomène de co-compositionalité. Un examen du fonctionnement des emplois métaphoriques comme moulin à paroles nous amène ensuite à proposer un modèle de représentation qui se construit par spécialisation. Moulin à paroles peut être associé au prédicat créer (création d'un excès de paroles) et désigne une entité de type +humain (une personne qui telle une machine produit une grande quantité de paroles). Cependant les emplois métaphoriques permettent aussi d'élargir la prise en compte des N2 de type Nparole : moulin à coups, moulin à systèmes, moulin à thèmes anglais, moulin à images, moulin à ennui... On a dans ce cas un changement d'état du processus de création de l'argument donné : production massive, importante... Nous choisissons donc une reconfiguration de la représentation initiale et la définition d'un nouveau type de moulin. La construction sémantique des composés en moulin à N2 propose une interprétation privilégiée : un moulin est avant tout un nom de la classe instrument, la sémantique "de base" pour moulin porte la valeur prédicative Faire-Changer-D'état. C'est le "poids" de moulin (du N1 dans le composé en N1 à N2) qui déclenche la construction sémantique du composé visé; celle-ci peut à son tour être modifiée en tenant compte des informations co-textuelles rencontrées : en l'occurrence, le "poids" sémantique du N2 peut sous-spécifier cette sémantique initiale soit la respécifier complètement. Cette solution permet de préserver une évolution potentielle pour l'interprétation des configurations en moulin à N2. Si l'on rencontre une séquence nouvelle, il reste possible à partir des interprétations existantes, de construire une nouvelle interprétation. Ces ajustements peuvent être réalisés "à la main" : mise à jour dynamique des attributs des prototypes visés. Si l'on dispose de méta-connaissances qui permettent de mettre en lumière les informations pertinentes qui éclairent une nouvelle interprétation, ces ajustements peuvent être réalisés via l'utilisation de méthodes idoines : mise en place de comportements généraux ou de comportements spécifiques au niveau des prototypes rencontrés.

4.2. Un dispositif pour la représentation et le classement automatiques des mots et de leurs comportements

Nous montrons tout d'abord comment la PàP permet la génération automatique des prototypes lexicaux et des arbres syntaxiques associés sur la base de savoirs extraits d'un corpus arboré. Les savoirs extraits sont des arbres d'analyse fournis par un outil d'extraction terminologique (Lexter[12], Bourigault 1993), ces arbres étant ensuite simplifiés dans le but de déterminer les arbres élémentaires de dépendance qu'il est possible d'associer aux entrées lexicales (Cyclade[13], Habert & Nazarenko 1996). La PàP permet ensuite d'amorcer un début de classement des unités lexicales représentées sur la base de leurs contraintes syntaxiques en construisant des hiérarchies locales de comportements[14] partagés. On classe les prototypes lexicaux définis

---

[12] Lexter est un outil d'extraction terminologique conçu et réalisé dans un environnement industriel: la Direction des Etudes et Recherches de EDF, pour aider à la mise au point de terminologie. Il prend en entrée des textes arbitrairement longs et produit des arbres d'analyses de séquences nominales en décomposant ces séquences en Tête et Expansion, de manière récursive

[13] Le programme Cyclade (Habert & al. 1995) a pour tâche de simplifier les arbres d'analyse fournis par une analyse, ici par Lexter, et de mettre en évidence les arbres qui présentent des relations de dépendance entre mots pleins (vs. mots vides : préposition, déterminant, conjonction...)

[14] On classe donc les objets représentant des mots sur la base des comportements réels associés aux mots ainsi modélisés. Les comportements informatiques qui ne décrivent les mots dans leur activité langagière ne sont pas pertinents pour ce classement.

sur la base des (prototypes d') arbres qui leurs sont associés. Les savoirs attachés aux prototypes lexicaux se composent en effet d'une liste d'arbres élémentaires (avec des contraintes possibles sur les noeuds de l'arbre) et pour chacun de ces arbres élémentaires d'une éventuelle liste d'arbres d'analyses (avec là aussi des contraintes possibles). On recherche donc les arbres élémentaires communs à un ensemble de prototypes lexicaux. Si on trouve des arbres élémentaires communs à un sous-ensemble de prototypes, on construit un pôle de comportements partagés qui va porter les arbres élémentaires communs. On établit ensuite un lien de délégation entre ce pôle et les prototypes concernés. On met donc en place, automatiquement, un réseau de pôles de comportements partagés en définissant des hiérarchies locales sur des sous-ensembles de prototypes. L'évaluation de connaissances définies préalablement comme prototypiques permet d'envisager une classification hiérarchisée des savoirs représentés, même si celle-ci reste évolutive. Notre objectif est de tendre vers la détermination de classes sémantiques, de manière inductive, dans une approche symbolique[15], celle des sous-langages. Il convient de souligner que le classement opéré s'appuie principalement sur des contraintes syntaxiques. Ce premier classement ne dit rien de plus sur les agrégats de comportements partagés construits. La syntaxe est utilisée pour dégrossir le classement. A l'inverse des approches harrissiennes[16] et statistiques[17], notre approche ne conduit pas à la détermination de classes sémantiques satisfaisantes mais elle constitue une méthode d'amorçage pour l'élaboration de l'ontologie du domaine, nous suivons sur ce point la démarche suivie par (Habert & Nazarenko 1996) : la construction de l'ontologie du domaine étudié nécessite un part d'interprétation, *"il y a, entre le flou notionnel inhérent aux langues naturelles (Kayser 1992), y compris aux langues de spécialités (Dachelet 1994), et la stabilité conceptuelle qui est requise dans les ontologies construites, un seuil qui ne peut être franchi automatiquement"* (Habert & Nazarenko 1996).

4.3. Un méta-niveau pour l'analyse

Le travail de représentation mis en place ne construit que des ébauches. Il s'agit ensuite d'ajuster les représentations initialement définies en tenant compte des informations que l'analyse dévoile. Pour la construction du sens, il convient de décrire au mieux les mécanismes qui opèrent pour réaliser cette mise en actions de mécanismes complexes et de s'approcher au plus près de ces mécanismes. On ne les touchera de près que lorsque nous serons capables de travailler sur de nombreuses connaissances et de produire de nouvelles connaissances à partir de savoirs initiaux. Le traitement automatique de la construction du sens est en effet confronté au problème de travailler sur de nombreuses connaissances et sur la nécessité d'en reconstruire de nouvelles. Cet impératif n'est pas encore accessible[18]. On n'en sait pas encore assez sur les

---

15 (Resnik 1993) présente une approche de construction de classes sémantiques de manière inductive dans une optique statistique.

16 Dans l'approche suivie par N.Sager, les classes de mots et les formules syntaxiques d'une grammaire de sous-langage sont en correspondance étroite avec les classes d'objets du monde réel et à leurs relations, ce qui conduit à sous-estimer le travail d'interprétation des résultats construits.

17 Dans les approches statistiques (Church & Hanks 1990, Hindle 1990), les classes de mots sémantiquement pertinentes sont obtenues statistiquement et induites du corpus à partir des seules mesures de similarités entre mots : *"En réalité, les listes de mots obtenues ne constituent pas de véritables classes de mots cohérentes et homogènes. Il est souvent difficile de leur attribuer une étiquette globale, et ce d'autant plus que les mesures de similarité constituent des résumés statistiques bruts dont les critères de construction sont effacés"* (Habert & Nazarenko 1996).

18 *"Atteindre le même niveau de performance que les êtres humains dans leur traitement des langues naturelles est un problème extraordinairement difficile que beaucoup croient utopique. Je doute que nous puissions y ariver sans résoudre au préalable les deux problèmes clés de l'IA : utiliser efficacement une grande quantité de connaissances déclaratives et découvrir automatiquement de nouvelles connaissances. Et je ne pense pas que les langues naturelles soient un bon support pour étudier ces deux problèmes, parce que ce domaine est beaucoup trop complexe; il vaut mieux faire progresser ces deux domaines de recherche en étudiant des domaines qui n'exigent pas une si grande quantité de connaissances. Travailler sur l'utilsation de métaconnaissances est un détour qui aura des conséquences pour les systèmes manipulant les langues naturelles, mais nous ne devons pas en attendre des résultats spectaculaires à court terme. Vu l'importance du traitement des langues naturelles, ce domaine doit être attaqué de tous les angles possibles. Aussi est-il*

mécanismes qui permettent de reconstruire des connaissances. Il convient toutefois de poursuivre le travail de recherche sur l'utilisation des méta-connaissances pour le traitement automatique du langage naturel, quitte à limiter le champ d'étude sur des phénomènes restreints qui permettent de préciser les tâches encore à effectuer. Ce travail sur des processus réflexifs pour un traitement automatique passe impérativement par une phase de développement de processus de traçage des activités d'un programme de traitement automatique. Dans notre travail, les processus réflexifs disponibles avec le cadre de représentation choisi ont permis de développer des outils de suivi ou de mise au point de l'analyse. Ces outils présentent, de manière raisonnée, les différents états pertinents des traitements réalisés et des résultats construits à chaque étape.

---

indispensable de poursuivre simulatément des travaux dès maintenant utilisables dans le domaine des langues naturelles, mais il est douteux que cela permettra d'atteindre une qualité de traitement comparable à la nôtre même à moyen terme" (Pitrat 1992).

Partie 1 : Du réel

Chapitre 1

Enjeux linguistiques et informatiques du traitement automatique de la construction du sens

1.1. Comment représenter la mouvance ?

1.1.1. Les noms composés : un micro-phénomène pour l'étude de la construction du sens

"Il est illusoire de songer à un traitement automatique avant de disposer d'un taux raisonnable de couverture des structures composées."
(Gross 1988)

"Les noms composés sont répartis sur un continuum qui va de l'opacité totale à la compositionalité limpide. La question est alors de répartir de manière cohérente les séquences considérées entre les deux pôles. L'autre difficulté étant, pour les séquences analysées de manière compositionnelle, de faire la part dans leur interprétation entre le savoir proprement linguistique et les connaissances pragmatiques."
(Habert & Jacquemin 1993)

1.1.1.1. Préambule

Il s'agit ici d'introduire[19] ce qui sera notre premier point d'ancrage linguistique, à savoir les problèmes de reconnaissance et de la construction du sens de certaines structures nominales au travers des modèles de représentation que l'on met en place pour le TALN. L'étude de la composition nominale est significative de la complexité du travail à accomplir pour modéliser les phénomènes qu'elle met en jeu dans un traitement automatique. La composition nominale illustre de manière fondamentale la difficulté à intégrer dans une modélisation les aspects lexicaux, syntaxiques, sémantiques, idiosyncratiques ou pragmatiques qui sont à l'oeuvre dans cette opération. Si cette opération met en jeu des mécanismes linguistiques réguliers pour la

[19] Il ne s'agit pas ici de proposer une étude détaillée sur les problèmes des composés (le travail de (Habert & Jacquemin 1993) propose une réflexion complète sur les enjeux de cette étude).

construction de la forme et du sens, elle s'inscrit aussi dans un cadre plus large qui nécessite la prise en compte de savoirs pragmatiques dans les raisonnements qui complètent ou affinent la représentation linguistique.

1.1.1.2. Une première tâche : étudier les formes nominales moulin à N2

Notre travail vise à repérer puis à classer les régularités qui se dessinent dans la construction du sens des configurations nominales du type moulin à N2. Il s'agira ensuite d'évaluer les affinements qu'il est possible d'apporter aux modèles de représentations adoptés. Outre les problèmes de classification que posent ces configurations nominales, une réflexion sur la construction du sens de ces configurations met en lumière la complexité des interactions entre les savoirs qui participent à cette construction. La formalisation puis la représentation "matérielle" de ces phénomènes entraîne une réflexion générale sur la mise en place des différents types de savoirs qui permette d'organiser au mieux les dépendances que ces savoirs établissent dans l'interprétation des faits particuliers de langue. Il s'agit donc d'étudier les problèmes de représentation de plusieurs de domaines de savoir qui participent ensemble à la résolution du codage ou du décodage de certains faits de langue. Et parallèlement, il faut analyser les mécanismes qui réalisent cette construction du sens en confrontant ces domaines de savoir.

1.1.1.3. Examen des évolutions de moulin sur corpus

Le corpus que nous avons utilisé pour l'étude des configurations nominales en moulin a été constitué en exécutant une requête sur moulin dans FRANTEXT. Les numéros donnés sont les références à ce corpus (-> annexe corpus moulin : extraits du corpus constitué). Dans ce corpus, on trouve moulin dans des configurations avec ou sans structure argumentale. Examinons tout d'abord les configurations avec structure argumentale.

- Moulin à Vinf[20]

moulin à scier: 71
moulin à frapper (les monnaies): 135
moulin à s'agiter (sans résultats):104
moulin à broyer: 167
moulin à battre: 227
moulin à distribuer (les cartes à jouer): 284

- Moulin à N2

Emplois non métaphoriques

moulin à vent: 1, 3, 4, 5, 6, 7, 9, 14, 15, 21, 22, 23, 33, 34, 35, 40, 42, 46, 47, 49, 50, 51, 54, 55, 58, 59, 60, 65, 68, 70, 91, 92, 95, 96, 97, 98, 103, 105, 106, 107, 113, 114, 121, 124, 128, 133, 139, 144, 145, 146, 149, 150, 153, 154, 155, 156, 157, 159, 160, 161, 163, 171, 172, 174, 175, 178, 179, 181, 184, 186, 188, 190, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 213, 218, 221, 226, 228 (cité par Saussure dans le CLG!), 232, 233, 238, 239, 246, 247, 248, 251, 252, 254, 259, 260, 262, 263, 265, 266, 269, 270, 271, 272, 273, 275, 277, 278, 283, 289, 291, 296, 298, 304, 307, 309, 316, 318, 322, 326, 331, 332, 333, 338, 339, 345
moulin à tan: 2, 118, 119, 120, 191
moulin à papier:8, 10, 12 (?), 13, 30, 32, 39, 43, 45, 212, 250, 286
moulin à bled: 28, 31, 32, 72, 77, 88, 90
moulin à blé:69, 93, 116, 191, 315
moulin à l'huile: 29 (archaisme ?)
moulin à huile:32, 90, 258, 287
moulin à eau: 35, 38, 42, 47, 48, 65, 91, 94, 112, 114, 122, 123, 126, 129, 130, 131, 132, 134, 136, 138, 141, 154, 183, 185, 187, 229, 231, 261, 308, 312, 313, 314, 322, 323
moulin à bras: 36, 37, 41, 61, 62, 63, 64, 101, 140, 169, 268, 297, 305, 334
moulin à poudre: 44, 45, 52, 66, 143, 170
moulin à sucre: 45, 109, 244
moulin à soie: 45
moulin à scie: 52, 72, 73, 74, 75, 76, 78, 80, 82, 83, 84, 85, 86, 87, 90, 115
moulin à coton:53
moulin à farine: 56, 79, 89, 240, 241, 242, 321
moulin à écorce : 56
moulin à ventilateur: 81
moulin à charbon: 125

---

[20] La rareté des Vinf serait-elle un indice de la nature prédicative de "moulin" ?

moulin à café:147, 148, 151, 152, 173, 189, 196, 219, 220, 224, 225, , 243, 245, 255, 274, 279, 282, 303, 311, 317, 324, 327, 328, 329, 330, 335, 336, 340, 341, 346, 347, 349, 350
moulin à foulon: 90, 176, 177, 180, 235
moulin à poivre:192, 299, 300, 325, 342
moulin à vapeur: 194, 195, 298
moulin à sel : 214, 344
moulin à lin : 267
moulin à cacao : 295
moulin à marée : 301, 302
moulin à grains : 52
moulin à scies : 90
moulin à prières: 99, 236, 256, 264, 280, 320
moulin à planches : 142
moulin à barres: 276
moulin à légumes:310
moulin à bactéries : 319

Emplois métaphoriques[21]

moulin à paroles: 19, 20, 25, 162, 193, 290
moulin à papier:12 (des copistes?)
moulin à sophismes: 57
moulin à thèmes anglais: 102
moulin à systèmes: 108
moulin à ennui: 165
moulin à café à vent:217
moulin à café à monnaie: 253
moulin à images: 257
moulin à coups: 285
moulin à lunettes: 288
moulin à soleil: 292
moulin à galette: 337

Examinons maintenant les configurations sans structure argumentale :

- Le moulin comme moteur, machine, ou instrument

un cheval de moulin à faire un certains nombres de tours:67
les roues des moulins à cinq cents pas au dessous: 168
rendit le moulin à Marguerite:343
le moulin à mi-régime:348
le petit moulin : 353

- Le moulin comme bâtiment

deux moulins à chaque lieue quarrée: 27
fermé à l'un de ses pignons, moulin à l'autre: 137
le garçon de moulin à la balafre: 166
le moulin à qui les arbres faisaient place en s'écartant: 222
pas de moulin à l'horizon: 230
moulins à faible distance: 234
moulin à notre droite: 250
maître de moulins à Manet et à Cézanne:281
moulin à la même place: 349
les ruines d'un moulin : 351

---

21 *"La métaphore réside en un transfert de dénomination fondé sur la représentation du partage d'une ou de plusieurs propriétés par les occurrences de deux classes référentielles. Une autre facon d'aborder la métaphore, non nécessairement contradictoire avec la première, consiste à poser qu'il s'agit en fait d'une restriction d'intension: le mot x servant à dénommer une categorie R voit son intension réduite à une ou plusieurs propriétés P. Une seconde catégorie R1 pourra être dénommée x si ces occurrences sont vues comme présentant P de façon saillante. Que l'on définisse la métaphore comme fondée sur la représentation d'un partage de propriétés(s) ou comme une identification simulée de deux classes, on laisse entendre que la focale peut être mise indifferemment sur toute propriété. Or il semblerait que certaines soient pointables de façon privilegiée: 1. Nombre de métaphores trouvent leur motivation dans des proprietés expérientielles /./ encore faut-il que la propriété focalisée soit susceptible de se réaliser dans divers particuliers, et que la catégorie-source de la metaphore présente un haut degré de typicité au regard de cette propriété /./. 2. /./ Les propriétés stéréotypiques sont d'autres propriétés privilegiés par la métaphore. La prégnance de la lune dans la vie quotidienne ou dans l'imaginaire des humains expliquerait ainsi cognitivement la multiplicité des classes référentielles dénommables à lunette(s)"* (Corbin & al. 1993).

qui traînent d'un moulin à l'autre : 352

- Moulin comme nom propre

Moulins ou Moulin: 11, 100, 110, 111, 215, 223

1.1.1.4. Moulin : Reconstruire une Représentation Sémantique

Les différentes réalisations du corpus permettent de tracer une première piste de description pour moulin. Une forme initiale de description de moulin adoptée peut être la suivante :

moulin : nom masculin singulier

Dans le Petit Robert, on trouve associées à moulin les définitions suivantes :

Appareil servant à broyer, à moudre le grain des céréales. Par extension, établissement qui utilise cet appareil. Bâtiment où ce type de machine est installé. Moulin servant à battre, à piler, à extraire le suc par pression. Sens figuré : moulin à paroles. Moulin à prières : cylindre renfermant des bandes de papier recouvertes de formules sacrées pour acquérir les mérites attachés à la répétition de cette formule. Sens familier actuel, désigne un moteur de voiture ou d'avion, "faire tourner son moulin", "emballer son moulin".

- Moulin : le moteur avant le moteur

Moulin est donc principalement une machine. Si originellement moulin peut être considéré comme une machine destinée à moudre les céréales, on peut tout de suite remarquer que moulin n'existe seul que pour désigner un moteur, une machine ou un bâtiment où un certain type de moulin est installé ou l'a été.

"Un lecteur ouvre un volume de planches, il apperçoit une machine qui pique sa curiosité : c'est, si l'on veut, un moulin à poudre" (D'Alembert, Disc. Prélim. de l'Enclyclop. 1751)

"Qu'un homme /./ invente une machine, telle qu'un moulin à vent, ces productions de son esprit peuvent en faire un bienfaiteur du monde." (Helvetius, De l'esprit, 1758)

"Le célèbre mathématicien De la Hire ne passait jamais devant un moulin à vent sans ôter son chapeau, par respect, disait-il, pour la mémoire de celui qui l'avait inventé." (Bernardin de St Pierre, Harmonies de la nature, 1814)

"Le moulin à vent, père de tant de machines nouvelles et à venir." (Alain, Système des beaux-arts, 1920)

"Le monastère et le chapelet son universels comme l'arc et le moulin à vent..." (Alain, Propos, 1936 p1080)

Le "moulin" est le prototype même de la machine. Il sert d'ailleurs de référence pour désigner d'autres types de machines :

"Telles sont, dans la période de transition que traverse le langage des images parlantes et sonores, les relations commerciales, intellectuelles et morales qui se sont établies entre nos moulins à images et la collectivité." (Arts et littérature dans la société contemporaine, 1935 p7808)

Son empreinte sur la société évolue en même temps que celle-ci :

"Le moulin à bras nous donnera la société avec le suzerain, le moulin à vapeur la société avec le capitalisme industriel." (Vedel G., Manuel élémentaire de droit constitutionnel 1949 p204)

"La société est donc l'expression d'un certain type de technique : suivant une formule connue, "le moulin à eau a produit la société féodale, la machine à vapeur a donné naissance à la société bourgeoise." (Lesourd-Gérard, Histoire économique 19ème et 20ème siècle, 1968 p164)

Moulin fonctionne aussi comme un opérateur syntaxique qui prend en arguments une préposition et un nom. Une trace de l'emploi originel de moulin est peut-être à l'origine de la prédominance[22] de moulin à blé sur moulin à farine, le produit naturel pouvant potentiellement donner lieu à plusieurs transformations. Dans ce cas moulin traduit l'opération qui consiste à transformer un produit naturel :

un moulin à blé transforme du blé en farine
un moulin à café transforme du café en grains en café en poudre

On trouve aussi des réalisations de moulin à N2 qui traduisent le point de vue inverse. Un moulin est alors le créateur de produit issu d'une transformation :

un moulin à farine est un moulin qui crée de la farine

on a aussi :

un moulin à beurre est une machine qui crée du beurre
un moulin à papier est une machine qui crée du papier

Si les moulins à vent n'existent pratiquement plus en tant qu'instrument de production, la séquence "moulin à vent" est à l'origine des nombreux emplois métaphoriques :

"Mais cette dernière, profitant de la mise en accusation, se réforme, emprunte à la sociologie ses méthodes, de sorte que celle-ci ne se bat en réalité que contre des moulins à vent, contre des attitudes de pensées dépassées par la psychologie elle-même." (Vuillemin J., L'être et le travail, 1949 p115)

- Des moulins à paroles et des emplois métaphoriques

"Il se mit à détester cette prose de moulin à vent, avec son éloquence gesticulaire." (Sabatié R., Les chinois d'Afrique, 1966 p253)

Une "prose de moulin à vent" peut donc être assumée par un "moulin à paroles", qui outre la quantité de "bruit" générée, ajoute à cette production une gesticulation physique digne d'un moulin à vent. Dans la série des "moulins à paroles" rencontrés, l'expression désigne tantôt la langue en tant que telle, tantôt la personne bavarde. Le "moulin à vent" et le "moulin à café" sont les principaux fournisseurs de métaphores, l'un peut-être par sa forme caractéristique et par sa diffusion, sans compter l'emprunt net à Don Quichotte, l'autre peut-être comme premier instrument ménager. L'assimilation du mouvement de l'instrument "moulin" à un comportement physique dénotant une gesticulation est souvent de mise:

92 : "Il se moque de ces bras en moulin à vent..."
181 : "On dirait que ces bras sont des ailes de moulin à vent..."
269 : "Un jeune type agité, les bras en moulin à vent..."
283 : "/./ sa coiffe noire, vue de dos, se silhouettait comme un étrange moulin à vent."
338 : "On dirait un moulin à vent quand il se fraie un chemin parmi les infirmiers..."

De même pour le bruit provoqué par l'instrument "moulin" :

327 : "Un moulin à café ronronnait dans l'appartement voisin."

puis

328 : "Le moulin à café s'arrête et mon père ronronne à son tour..."

Un être humain peut aussi être assimilé à un automate qui se comporte comme un cheval de moulin :

67 : "Au nombre des meilleurs sont Laplace, Monge et Cousin, espèces d'automates, habitués à suivre certaines formules, et à les appliquer à l'aveugle, comme un cheval de moulin à faire un certain nombre de tours avant de s'arrêter."

Les emplois métaphoriques permettent d'élargir la prise en compte des N2 de type Nparole. En effet, on rencontre les séquences telles moulin à coups, moulin à systèmes, moulin à thèmes anglais, moulin à images, où N2 est un pluriel sauf pour "moulins à papier" dans 12;

---

[22] Dans le corpus donné en annexe, on trouve 7 réalisations en "moulin à bled" et 5 en "moulin à blé" contre sept réalisations en "moulin à farine".

on trouve aussi moulin à ennui où N2 est massif singulier. Dans tous les cas, il y a production[23] massive.

- Moulin et les déverbaux

Certains exemplaires de moulin à N2 ne décrivent pas ce que produit le moulin ni ce qu'il transforme, mais ils représentent un instrument constitué par le moulin. Dans moulin à scie, moulin à foulon, moulin à ventilateur, l'énergie du moulin sert respectivement à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation : instrument dont l'énergie est apportée par le moulin.

La description initiale de moulin et l'examen de ses emplois mettent en évidence les mécanismes évolutifs à l'oeuvre dans les faits de langue. Un traitement automatique de ces phénomènes est contraint par la prise en compte impérative de cette mouvance illustrée sur moulin : **les emplois précités ne peuvent être considérés comme figés et non évolutifs**. Leurs représentations dans un dispositif de traitement automatique doivent donc pouvoir être ajustées voire remodelées pour rendre compte de ces évolutions.

1.1.1.3. De l'étude de certains faits empiriques à la construction d'un dispositif : problèmes de représentation

On perçoit déjà la grande difficulté à identifier dans les représentations à adopter ce qui relève, pour simplifier, du général (le "conceptuel") et du particulier (le "lexical"). Dans quelles mesures, les informations sémantiques doivent-elles être représentées dans une hiérarchie de concepts ou dans une hiérarchie lexicale? Et comment articuler ces environnements d'informations? Comment évaluer les (in)dépendances entre ce qui dû aux informations conceptuelles et ce qui est dû au langage lui-même? Il faut donc envisager la mise en place d'un système de représentation sous forme de couches (de feuilleté) en tenant compte que ces couches mettent en jeu des mécanismes ou tâches qui ne sont pas à évaluer sur un même plan; une activation de ces tâches en parallèle ne semble donc pas une solution cohérente. Là encore, ce qui est en question, c'est l'existence d'une organisation (hiérarchique ou autre) de ces tâches. Ce qui est sûr, par contre, **c'est la nécessité de disposer d'éléments de contrôle qui permettent une gestion adéquate de ces processus**. On retrouve là un des grands problèmes de la représentation des connaissances qui est celui de la définition de connaissances de contrôle pour gérer des telles complexités d'interaction :

"Les connaissances de contrôle sont les connaissances sur le système et sur les représentations des connaissances des domaines et portent sur la signification de ces représentations dans le système. Cela provient de l'écart irréductible entre connaissance et représentation. La représentation extraite de la connaissance reçoit un sens informatique dans le contexte du système. Les connaissances de contrôle précisent ce sens. Il y a deux étapes essentielles dans la représentation des connaissances : représenter les connaissances du domaine dans un formalisme de référence indépendamment du système final, représenter les connaissances de contrôle dans un formalisme tiré du système pour concilier représentations du domaine et système final" (Bachimont 1992).

1.1.2. Moulin à N2 : une approche hiérarchique de la construction du sens (Habert & Fabre 1993)

Le travail de (Habert & Fabre 1993) vise à présenter un modèle de la construction du sens pour l'acquisition de dénominations nouvelles: *"Dans l'ensemble des dénominations complexes, nous nous intéressons aux séquences nouvelles (par opposition aux termes attestés) et interprétables, potentiellement construites selon des schémas syntaxiques et sémantiques réguliers"*. Au sein du patron N1 à N2, il s'agit d'examiner les contraintes qui régissent les séquences moulin à N2. Pour cela, leur travail s'appuie sur un outil de représentation hiérarchisée du lexique, via le langage DATR (Gazdar & Evans 1990), et sur des règles à la

[23] Sauf pour "moulin à soleil" (solution 292 du corpus) dont l'interprétation est d'ailleurs difficile.

PATR-II (Shieber 1986) pour montrer la mise en oeuvre de ces contraintes. *"L'examen critique de cette démarche débouche sur la proposition d'utiliser une organisation hiérarchisée des règles de construction du sens comme principe directeur de l'analyse"* (Habert & Fabre 1993).

1.1.2.1. Objectif : un modèle des règles de production de Dénomination Complexe

L'approche suivie s'inscrit dans un cadre de traitement automatique des dénominations complexes non attendues (non présentes dans les bases de connaissances ou les dictionnaires utilisés) pour y acquérir de nouvelles dénominations bien formées. Cette approche repose *"sur le calcul de l'interprétation sémantique de la séquence à partir des sens des composants, et par la mise en évidence de certains schémas de sens productifs"* (Habert & Fabre 1993). Quelles Dénominations Complexes Motivées (c'est-à-dire où le sens global est motivé, dans une proportion plus ou moins grande, par le sens des composants) peuvent être produites? Dans une telle perspective, quelles sont les règles à l'oeuvre?

1.1.2.2. Les règles à l'oeuvre dans le patron `moulin à N2`

Un premier classement

L'examen des expressions `moulin à N2` prend appui sur le travail de (Poncet 1991)[24]. Dans ce dernier, la série des séquences `moulin à N2` entre dans les séquences `N à Ni` désignant des noms d'instruments ou de machines, à l'exception de certains emplois figurés tels que `moulin à paroles` qui sont regroupés dans une classe résiduelle. La répartition des séquences `moulin à N2` est représentée dans la figure qui suit.

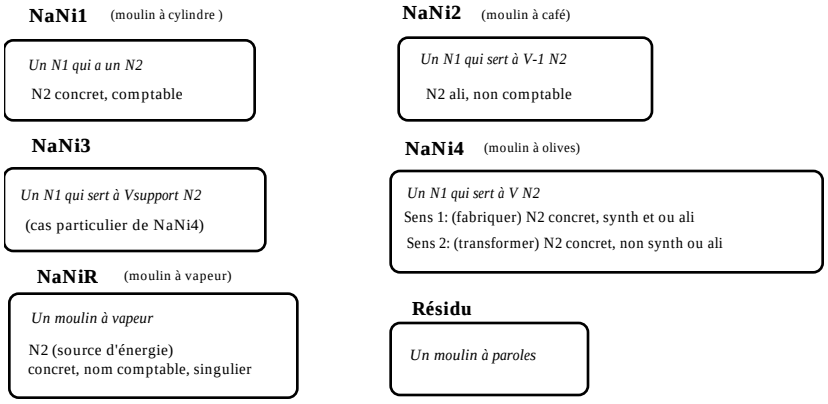


Figure 1.1 : `moulin à N2` : un premier classement.

Le principe de classification sémantique présenté ci dessus utilise des classes sémantiques telles que aliment, noté `ali`, synthétique (pour des référents synthétiques, non naturels), noté `synth`, permettant de résumer un ensemble de traits élémentaires tels que `concret`, `comptable` :

[24] La perspective de (Poncet 1991) est de déterminer les propriétés qui amènent à classer un `Na à Nb` dans la catégorie `Ni`.

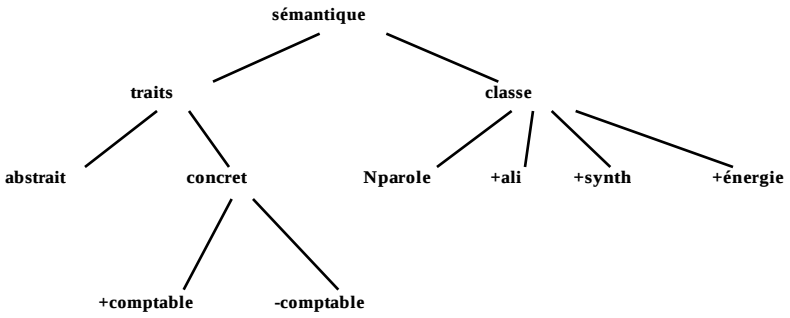


Figure 1.2 : Classification sémantique.

Réexamen de la classe résiduelle pour une analyse unifiante du sémantisme de `moulin`

"Considérer `moulin à paroles` comme figuré, figé, n'est pourtant pas vraiment satisfaisant. Lister de telles séquences et leur associer un sens unique (langue, bavard) ne rend pas compte en effet des possibilités de création sur ce schéma" (Habert & Fabre 1993). On trouve en effet une série de réalisations qui entrent dans la classe résiduelle de la figure précédente : `moulin à sophismes`, `moulin à conversation` (-> annexe corpus moulin). Ces séquences fonctionnent suivant la contrainte suivante : elles attendent un `N2` de type `NParole`, au pluriel ou de sens pluriel, et le `N` résultant est `+humain`. (Habert & Fabre 1993) proposent[25] donc d'associer à `moulin` un prédicat à quatre arguments de la forme suivante[26] :

`PRODUIRE(Entrée,Sortie,Energie,Partie-De)`

Le modèle sémantique proposé permet d'affiner les résultats existants sur la classification des `N1 à N2` (Poncet 1991) en intégrant notamment la classe jusqu'alors résiduelle de `moulin à paroles`, celle-ci s'intégrant au modèle proposé suivant les contraintes sémantiques précisées dans la figure qui suit.

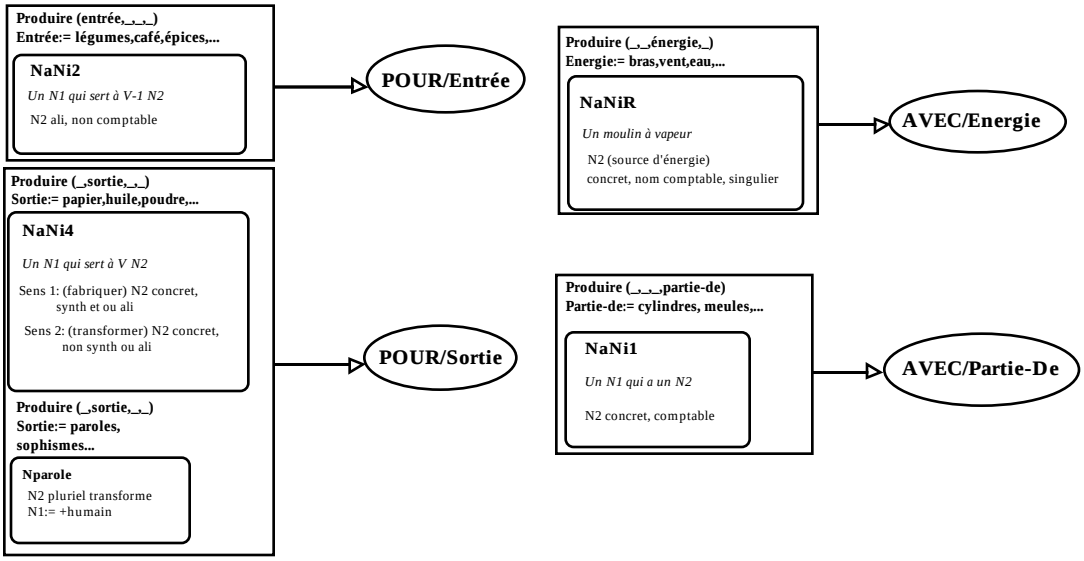


Figure 1.3 : Nouvelle répartition sémantique des `moulin à N2`.

---

25 *"La productivité, restreinte, de ce schéma invite à réexaminer son rapport avec le sémantisme de `moulin`, et à donner de celui-ci une analyse unifiante"* (Habert & Fabre 1993).

26 Pour reprendre ce qui est dit dans la note précédente, l'approche choisie ici vise donc plus à chercher les propriétés qui font de `Nb` un bon candidat pour être apparié à un `Na` dans un `Ni`.

Dans ce modèle, *"les sens dits figurés apparaissent alors comme une spécialisation d'une configuration citée. /./ Un* `moulin à paroles` *produit des paroles, c'est cette inscription dans l'une des configurations possibles pour* `moulin` *qui explique la productivité et la transparence relatives de ce patron. L'utilisation d'un N2 pluriel,* `NParole` *comme deuxième argument, entraîne une transformation régulière de N1 (+*`humain`*). En revanche le trait* `excès` *qui apparaît dans ces dénominations relève probablement d'un autre type de mécanismes : par exemple de maximes à la Grice. Si l'on caractérise une personne par la production de paroles, c'est que celle-ci est excessive. A une métaphorisation qui s'appuie sur les caractéristiques fondamentales du prédicat associé à* `moulin` *s'ajoute un nouveau trait, de manière là encore régulière et reproductible"* (Habert & Fabre 1993). Ce modèle permet aussi de préciser les propositions de (Cadiot 1992). `POUR` se subdivise en `entrée` et `sortie`, `AVEC` en `énergie` et `partie-de`. *"Les relations* `POUR` *correspondent aux arguments essentiels du prédicat, les arguments* `AVEC` *aux arguments secondaires"* (Habert & Fabre 1993).

Le traitement proposé utilise *"des règles de réécriture à base hors contexte où chaque élément de la règle est remplacé par un graphe de contraintes sur l'élément de la phrase"* (Habert & Fabre 1993). Une représentation graphique de cette vision unifiante de la construction du sens est donnée dans la figure qui suit.

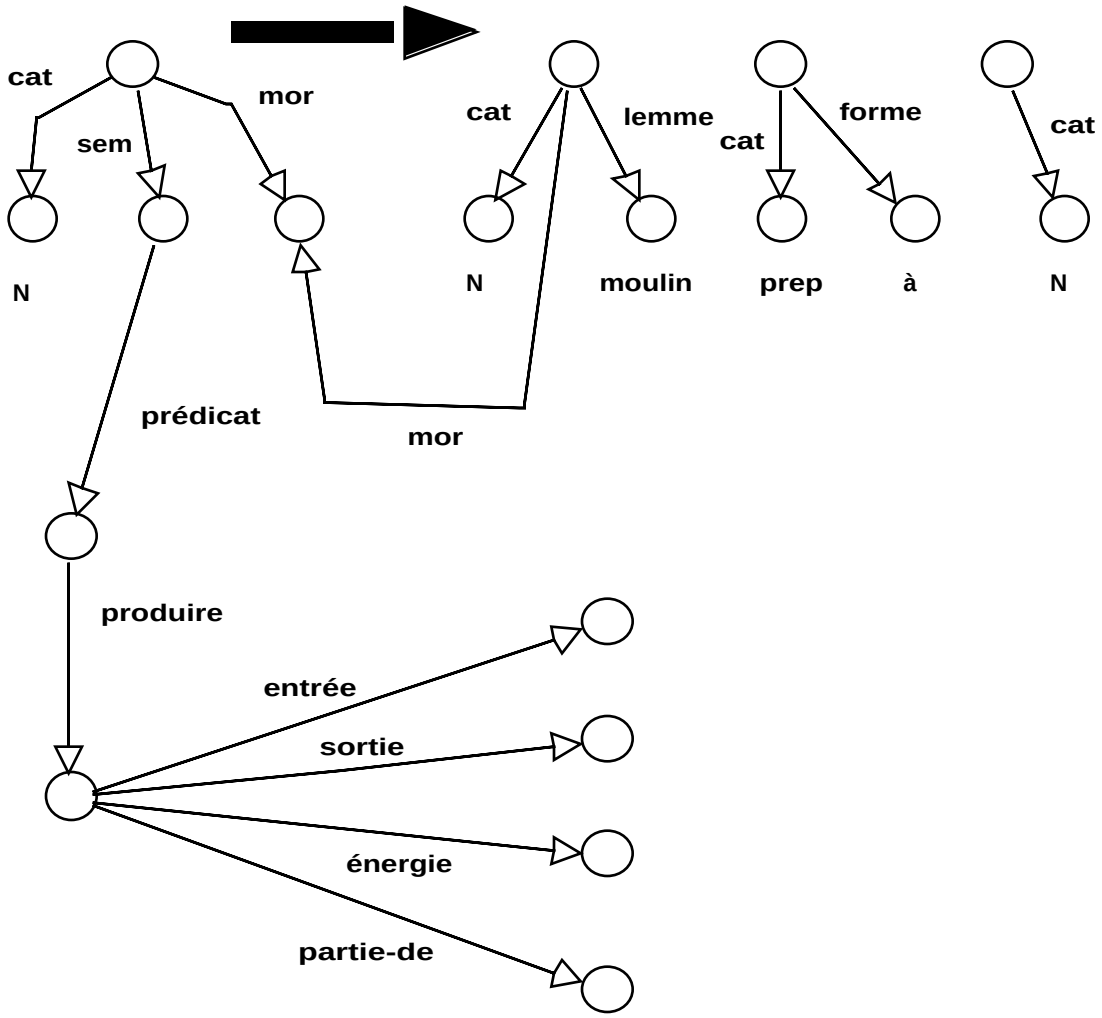


Figure 1.4 : Une vision unifiante de la construction du sens des séquences `moulin à N2`. (1)

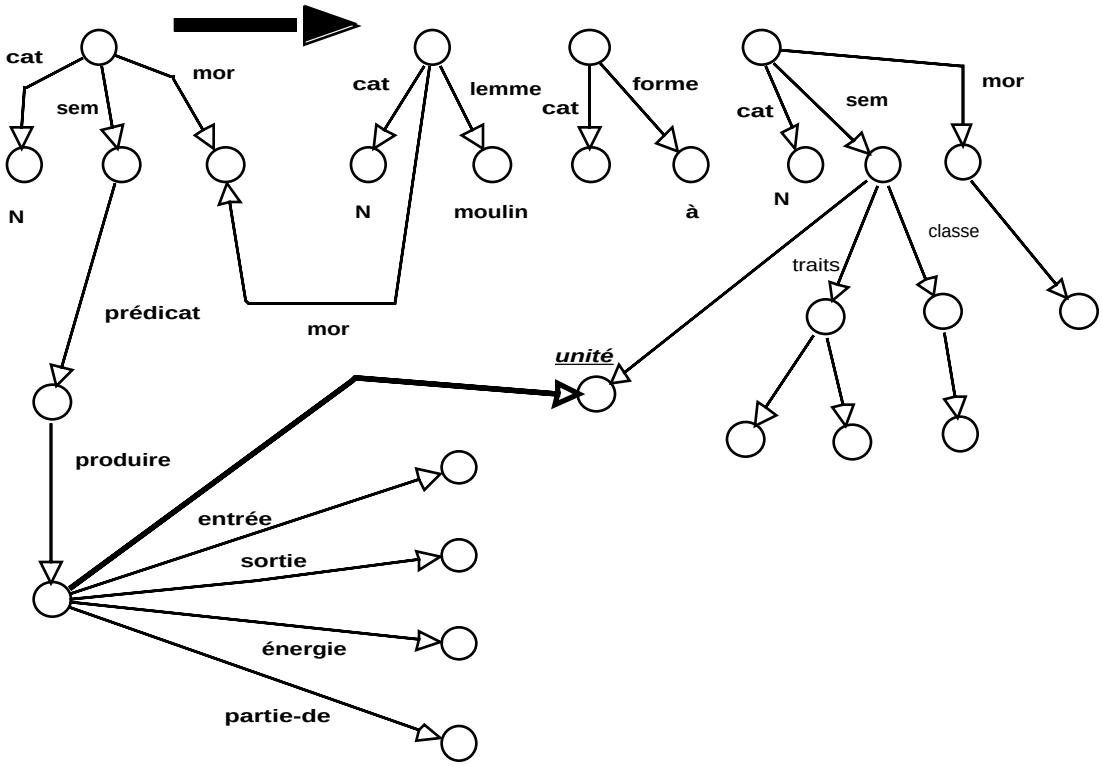


Figure 1.4 : Une vision unifiante de la construction du sens des séquences `moulin à N2`. (2)

Construction du sens des séquences `moulin à N2`

L'approche retenue par (Habert & Fabre 1993) s'inscrit dans un cadre qui privilégie la construction du sens non par juxtaposition et combinaison des sens des entrées lexicales, mais par un affinement d'hypothèses initiales *"quitte à rester dans l'indétermination quand les informations disponibles ne permettent pas d'aller plus avant"* (Habert & Fabre 1993). Le choix d'une telle approche vise à :

- Construire une représentation sémantique très abstraite, très peu déterminée qui est précisée si besoin est;
- Inscrire dans la représentation les scénarios, les processus d'interprétation.

L'analyse à mettre en oeuvre *"est vue comme un affinement progressif d'hypothèses dirigé par la structure hiérarchique des entrées lexicales et par le contexte des têtes lexicales"* (Habert & Fabre 1993). Le point fondamental dans cette approche est de maintenir une continuité entre une sous-détermination initiale et une spécialisation à venir.

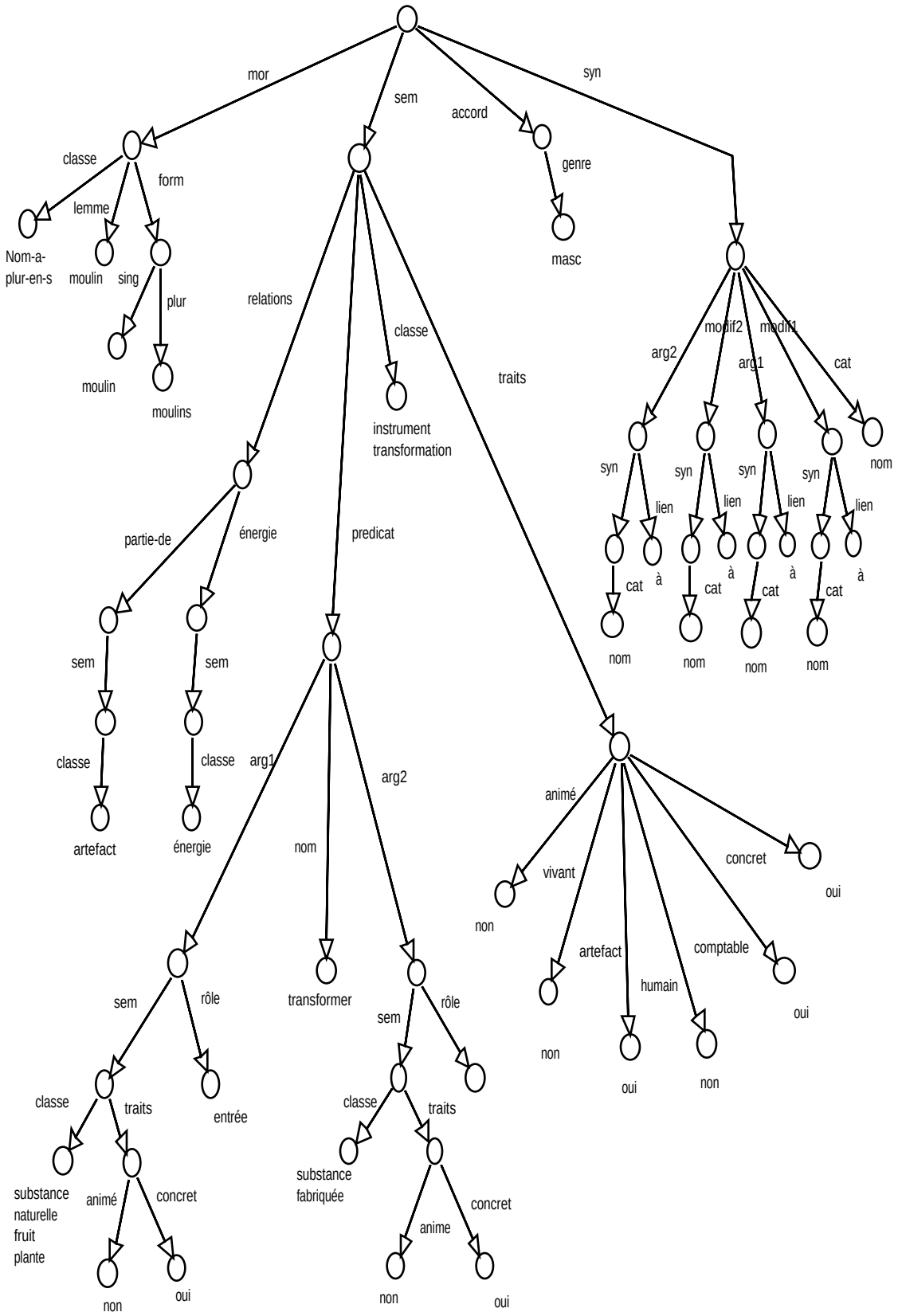


Figure 1.5 : Représentation scénarisée de `moulin`.

Cette approche de la construction du sens basée sur la polysémie et la sous-détermination se conjugue avec une approche de représentations hiérarchisées des connaissances lexicales qui traduit la volonté de limiter les effets d'un traitement homonymique du sens. La représentation lexicale doit prendre en compte la nécessité de disposer de mécanismes exprimant le passage du général au particulier. Il s'agit en particulier de mettre en avant les régularités lexicales qu'il est ensuite possible de préciser pour tel item donné. Pour cela, (Habert & Fabre 1993) utilisent le langage de représentation lexicale DATR (Gazdar & Evans 1990) qui permet un traitement des généralisations et des exceptions basé sur l'héritage multiple et la non-monotonie. C'est aussi une vision où il ne peut y avoir de violation de ce qui est acquis à un niveau supérieur. Il n'est pas sûr que cela ne soit jamais la cas.

Affiner le modèle sémantique proposé

Certains exemplaires de `moulin à N2` ne rentrent pas dans le cadre du prédicat à quatre places retenu : `scie, foulon, ventilateur.` Il s'agit pour ces derniers, non pas de ce que produit le moulin, mais de l'instrument constitué par le moulin : l'énergie du moulin sert à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation : instrument dont l'énergie est apportée par le moulin et non `partie-de`. De plus, l'étude des emplois métaphoriques peut élargir la limitation faite ici au `NParole`. On peut ainsi avoir des séquences `moulin à N2` métaphoriques avec (`N2 pluriel` : `coups, thèmes, images, sophismes` ou avec (`N2 massif singulier : ennui`), dans les deux cas, on a une production massive. La présentation du modèle faite précédemment doit donc prendre en compte ces exemplaires de `moulin à N2` qui apportent des précisions au modèle initial.

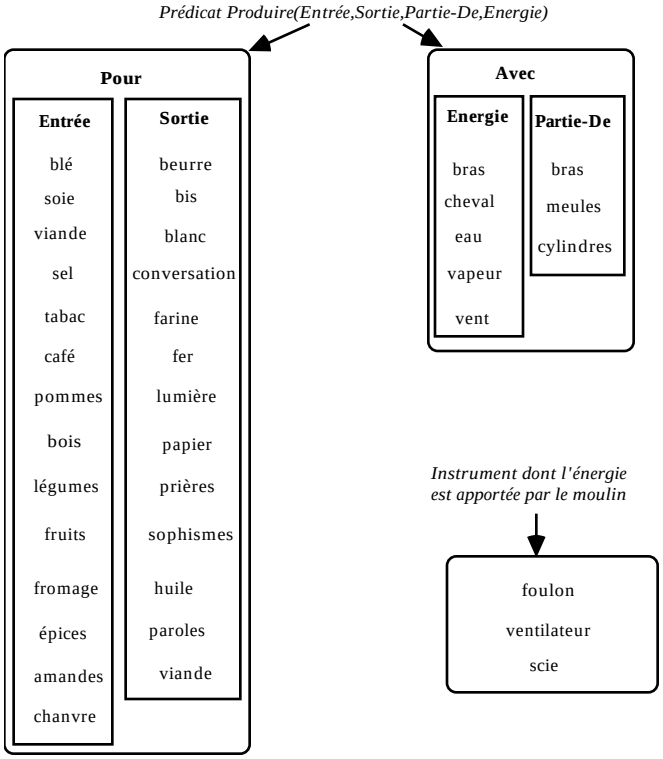


Figure 1.6 : Répartition lexicale dans le modèle sémantique construit.

1.1.3. Changements Réguliers de sens et organisation des connaissances

1.1.3.1. Limites du filtrage des arguments acceptables

Certains exemples mettent en avant les limites d'un simple filtrage des arguments acceptables. Les dérivations de sens invalident ce type de filtrage puisqu'aussi bien l'élément en position d'argument ne

présente pas les traits nécessaires et que ceux-ci doivent lui être ajoutés (moulin devient +humain dans moulin à paroles). Il en va de même pour bras ou cheval dans moulin à bras ou moulin à cheval. Il n'y a en effet aucun intérêt à adapter la description de bras ou de cheval en ajoutant un sens Nénergie. Il s'agit donc de mettre en place des strates qui permettent :

- un filtrage fin en fonction des scénarios associés au nom tête,
- une extension éventuelle en fonction de rôles hérités,
- une inférence plus grande à partir d'informations conceptuelles.

Il est important de souligner malgré tout qu'il semble difficile de prendre en compte des interprétations qui ne seraient pas préalablement inscrites dans les mécanismes prédéfinis.

1.1.3.2. Structurer les connaissances syntaxiques

Les interprétations possibles dans la cadre de la construction du sens des formes moulin à N2 sont toutes placées au même plan, alors qu'un poids différent leur est attaché. On aimerait pouvoir dire qu'une interprétation est plus centrale qu'une autre dans la manière même de représenter les connaissances. Les modèles de représentation pour les composés doivent préserver l'ajustement et l'ambivalence dans la construction du sens de ces composés : il faut permettre la construction du sens, non par juxtaposition et combinaison des sens des termes de la structure complexe en cause, mais par l'affinage progressif des hypothèses initiales (qu'il est préférable de laisser floues) quitte à rester incertain quant à l'issue du calcul en cours pour les enrichir quand de nouvelles informations pertinentes le permettent. Il faudrait ainsi pouvoir disposer pour représenter la construction de sens de dénominations complexes de mécanismes qui permettent :

- d'ordonnancer des règles : une règle doit pouvoir n'être appliquée qu'après l'échec des règles précédentes. cela devant s'accompagner d'une plausibilité de l'interprétation;
- de donner aux règles la possibilité de remodeler tout ou partie des entrées lexicales qu'elles prennent en argument (comme cela se passe pour bras = énergie dans moulin à bras);
- de respecter des mécanismes transversaux (opérations régulières).

En cela, on prend le contre-pied de deux choix habituels dans les formalismes d'unification (-> chapitre 3) : la construction de tous les possibles (qui sont mis sur le même plan) et la non-monotonie. Par ailleurs, c'est une position sinon opposée au lexicalisme, du moins limitant son champ d'action. Cela consiste à dire en effet qu'après chaque passage de règles normales de construction du sens, il y a des règles de récupération qui essaient, par coercion[27] en particulier, de produire tout de même une interprétation. Enfin cela pousse à chercher des modes de structuration du lexique qui fassent la part moins belle à l'homonymie, et qui recherchent un partage réel des informations et non une factorisation statique.

[27] (Pustejowsky 1991).

1.2. Quels savoirs pour les mots ?

Les développements récents des travaux en linguistique manifestent un intérêt grandissant pour la constitution de gros corpus. Il ne s'agit plus d'analyser en profondeur des domaines restreints, mais plutôt d'évaluer quantitativement des performances de systèmes sur des données de plus en plus importantes. Même si les travaux quantitatifs sur des corpus de plus en plus importants permettent d'élargir la couverture des problèmes traités, **les savoirs linguistiques établis sur des corpus ne peuvent être considérés comme figés et non évolutifs**. Le travail d'extraction de savoirs à partir de corpus montre en effet qu'il est difficile de traiter les savoirs linguistiques comme des généralités.

- En dehors de la connaissance précise d'un domaine de spécialité, il est souvent impossible de savoir si une construction est "acceptable" ou non.

A. Daladier (Daladier 1990) indique : *"en biologie, la phrase* `des antigènes ont été injectés dans les anticorps d'un lapin` *est dénuée de valeur informative. Il ne serait pas suffisant de dire que cette phrase représente une information fausse parce qu'il n'y a aucune façon d'obtenir une signification, même fausse, à partir des combinaisons de mots qui la constituent, pour un biologiste. Dans un sous-langage, une phrase est interprétable si elle est acceptable".*

- Les savoirs rencontrés sur un corpus donné peuvent remettre en question des résultats généraux établis par ailleurs.

  En médecine coronarienne, `longue` fonctionne comme un adjectif qui marque le degré quand il est utilisé comme modifieur de `occlusion`. C'est le cas dans la séquence nominale `"une occlusion longue"`. Dans d'autres sous-langages, cet adjectif n'est pas typé pour marquer le degré.

- Les savoirs généraux que l'on peut associer aux mots ne sont pas toujours pertinents :*"La plupart du temps, les généralisations globales ne sont absolument pas fiables, parce qu'il n'existe pas de caractérisation générale d'une langue entière; au contraire, il y a des différences considérables entre registres (ou sous-langage). La description complète d'une langue entraîne par conséquent une analyse composite de traits tels qu'ils fonctionnent dans des registres divers"* (McNaught 1993). Il n'est donc pas satisfaisant de se contenter d'un modèle apriorique pour construire une représentation des comportements des unités lexicales.

- On ne connaît pas toutes les caractéristiques[28] des combinaisons des mots sur un sous-langage donné

  `coronaire` et `coronarien` partagent des contextes[29] :

  `coronaire {artere ~, chirurgie ~, heredite ~, lesion ~, maladie ~, reseau ~, spasme ~}, coronarien {artere ~, chirurgie ~, heredite ~, lesion ~, maladie ~, reseau ~, spasme ~}.`

  Mais les modes de combinaison de ces deux mots sont divergents :

---

28 *"En ce qui concerne la variabilité liée au domaine, J.McNaught (1993) met l'accent sur ce qu'il perçoit comme une des caractéristiques les plus marquées du fonctionnement lexico-syntaxique des sous-langages: des verbes qui sont des homographes de verbes en langue générale ont fréquemment dans un sous-langage donné des structures prédicat-arguments complètements différentes de celles qui leur correspondent en langue générale ou dans d'autres sous-langages. Ceci a trait à la relation extrêmement serrée entre syntaxe, sémantique et domaine conceptuel qui caractérise les sous-langages, et qui a l'attrait de ne permettre en général qu'une seule analyse même dans le cas de séquences longues de syntagmes prépositionnels"* (Péry-Woodley 1995).

29 Les exemples donnés ici sont extraits du corpus MENELAS (Zweigenbaum 1994).

`coronarien` est associé à des adjectifs évaluatifs `{severe, significatif, important}` et `coronaire` ne l'est pas.

- Les comportements lexicaux ne suivent pas des parcours uniformes sur des entités de même catégorie et surtout il est difficile de les prévoir. Les mêmes comportements lexicaux ne se réalisent pas sur tous les mots d'une même catégorie, ils se distribuent sur des sous-familles particulières. De plus ces comportements ne se réalisent pas de manière uniforme sur les différentes familles de mots d'une même catégorie.

  `artère` et `infarctus` entrent dans des relations de localisation qui se réalisent de manières distinctes:

  `artere {coronaire circonflexe diagonal...}, infarctus {anterieur inferieur apical}`

- De plus, *"bon nombre des problèmes de traitement des langues sont en fait limités à la langue utilisée dans un domaine particulier (...). La variété de langue utilisée dans un domaine scientifique ou technologique donné est non seulement beaucoup plus restreinte que la langue dans son ensemble, elle est aussi plus systématique sur le plan de la structure comme sur celui du sens"* (Grishman & Kittredge 1986).

Il y a peu de sens à vouloir faire de l'acquisition sémantique en dehors d'un sous-langage

En suivant ce constat, notre travail s'inscrit dans une approche qui vise à ne pas présumer complètement des choses à représenter. Cette démarche met en place un processus de représentation évolutif : les structures de représentation construites devront pouvoir être affinées dès que de nouveaux savoirs seront mis à jour. On peut en effet considérer que les comportements des mots ne sont pas tous prédéfinis mais que ceux-ci "émergent" dans le contexte dans lequel ces mots "agissent". Il n'est donc pas raisonnable de les considérer comme acquis par définition; il s'agit au contraire de les mettre en lumière ainsi que les corrélations multiples qui existent entre les mots dans un flot continu de discours. Notre démarche consiste en quelque sorte à "faire émerger" les comportements des mots puis à les représenter ou à affiner les représentations existantes.

Pour autant il ne s'agit pas pour nous de nier en bloc toute forme de savoirs stabilisés dans ces processus de représentation automatiques des descriptions linguistiques. Les savoirs généraux en langue ne manquent pas. Les dictionnaires constituent d'ailleurs une trace de la fixation sur le moyen terme de ces savoirs. Il ne s'agit donc pas de tout réinventer. Notre travail vise au contraire à prendre appui sur des connaissances très générales (on utilise au départ un cadre de sous catégorisation approximatif qui sera présenté infra) qui sont affinées voire remodelées et changées au gré des observations rencontrées de la même manière que les savoirs linguistiques évoluent (les dictionnaires sont mis à jour périodiquement...). Il s'agit ainsi pour nous de tendre vers un modèle cohérent qui ajuste les descriptions linguistiques dès qu'une certaine stabilité apparait. De même, dès qu'un certain type de comportement se révèle pertinent, il convient d'ajuster les savoirs définis pour en tenir compte.

1.3. Approche d'une grammaire de sous-langage

1.3.1. Notion de sous-langage

Les caractéristiques des sous-langages[30] (Dachelet 1994) - dimensions réduites, systématicité structurelle et sémantique - ont trait au modèle linguistique dans lequel prend la notion de sous-langage : ce modèle a été développé par Z. Harris à partir des années 60. La thèse de R. Dachelet donne une présentation complète de ce modèle.

1.3.2. L'analyse des sous-langages : la démarche harrissienne

> *"In the absence of an external metalanguage, the entities of each language can be identified only if the sounds, markers, or words of which they are composed do not occur randomly in utterances of the language. That is, the entities can be recognized only if not all combinations occur, or are equally probable. This condition is indeed satisfied by languages. A necessary step, then, toward understanding language structure is to distinguish the combinations of elements that occur in the utterances of a language from those that do not : that is, to characterize their departures from randomness."*
> (Harris 1988)

Harris met en oeuvre une conception "distributionnaliste" de la sémantique : le sens des mots[31] se déduit des constructions[32] dans lesquelles ils figurent : *"In fact we find in language that each word has a particular and roughly stable likelihood of occuring as argument, or operator, with a given other word, though there are many cases of uncertainty, disagreement among speakers, and change through time. These roughly stable likelihoods, and especially the selection frequency, conform to and fix the meanings of words "* (Harris 1988). Harris part de la constatation empirique que les éléments ou les suites d'éléments ne se rencontrent pas arbitrairement les uns par rapport aux autres et les arrangements ou environnements expriment une certaine régularité qui dévoile la structure du texte : *"les parties d'une langue ne se rencontrent pas de façon arbitraire les unes par rapport aux autres; chaque élément se rencontre dans certaines positions par rapport à certains autres éléments"* (Harris 1970). La reconstitution des phrases élémentaires d'un corpus de langue de spécialité débouche sur la mise en évidence de classes d'opérateurs[33] et de classes d'arguments, qui sont supposées fortement liées aux notions et aux relations du domaine considéré. La méthode consiste donc à construire des classes définies distributionnellement. On établit ainsi

---

30 *"Nous utilisons indistinctement sous-languages et sous-langues. Tous deux traduisent le mot anglais 'sublanguage'. C'est un problème conceptuel récurrent que les mots 'langue' et langage' sont tous deux des traductions possibles de l'anglais 'language'"* (Dachelet 1994).

31 *"At what point do words get meanings? One should first note something that may not be immediately obvious, and that is that meanings do not suffice to identify words. They can give a property to words that are already identified, but they don't identify words. Another way of saying this is that, as everybody who has used Roget's Thesaurus knows,* ***there is no usable classification and structure of meaning per se, such that we could assign the words of a given language to an a priori organization of meanings. Meanings over the whole scope of language cannot be arranged independently of the stock of words and their sentencial relations.*** *They can be set up independently only for kinship relations, for numbers, and for some other strictly organized parts of the perceived world"* (Harris 1988).

32 *"The components that go into the making of the structure are the components that go into the making of the information"* (Harris 1988).

33 *"In 'The man asserts that a tree sleeps', the 'asserts' is the operator on the argument pair 'man', 'sleep' (forming 'man asserts sleep'), while 'sleep' is the operator on its argument 'tree'. This is a dependence, , with each word in the sentence requiring particular other words, more exactly any one word of particular classes: 'sleep' requires the classes of 'man, tree'. This dependence is a partial order"* (Harris 1988).

les occurrences et les co-occurrences d'un élément ou la suite d'éléments, puis on établit les équivalences[34]. C'est l'examen des classes d'équivalence dans la relation opérateur-argument[35] qui permet d'obtenir et les classes de mots et leurs relations de co-occurrence. Cet examen se révèle mettre à jour un petit nombre de phrases-type constituées de combinaisons spécifiques de sous-classes de mots.

Propriétés des sous-langages

- En général, le lexique est restreint.

- On y trouve peu de types de phrases différents. Chaque type est caractérisable comme une combinaison particulière de sous-classes de mots.

"Le trait distinctif d'un sous-langage est que pour certains sous-ensembles de phrases du langage, le phénomène de sélection, pour lequel on ne peut formuler de règles pour le langage dans son ensemble, fait partie de la définition de la grammaire. Dans un sous-langage, les classes de mots correspondant aux restrictions de sélection ont des frontières relativement nettes, ce qui reflète la division des objets du monde réel en classes qui sont clairement différenciées dans le domaine" (Sager 1986).

- La contrainte de probabilité y est neutralisée : dans la relation opérateur-argument, il n'existe pas de gradation due aux restrictions de sélection.

- Alors que les sous-langages sont des sous-ensembles de la langue, la grammaire d'un sous-langage n'est pas un sous-ensemble de la grammaire de la langue générale. Les grammaires sont plutôt en intersection.

"The sublanguage has important constraints which are not in the language : the particular word classes, and the particular sentences types made these. And the language has important constraints which are not followed in the sublanguage. Of course, since the sentences of the sublanguage are also sentences of the language, they cannot violate the constraints of the language, but they can avoid the conditions that require those constraints. Such are the likelihood differences among arguments in respect to operators; these likelihoods may be largely or totally disregarded in sublanguages. Such also is the internal structure of phrases, which is irrelevant to their membership in a particular word class of a sublanguage" (Harris 1988).

Obtention de classes d'opérateurs et d'arguments à partir de phrases élémentaires

- Dans la lignée des travaux de Harris, l'équipe de Naomi Sager a développé une méthodologie (Sager & al. 1987) de mise à jour, à partir d'une analyse distributionnelle en phrases élémentaires, des classes de concept et de relations d'un sous-langage lié à un domaine d'activité donné (Habert & Nazarenko 1996).

- (Habert & Nazarenko 1996) présentent une démarche qui reprend partiellement l'approche harrissienne : étude des fonctionnements syntaxiques qui sont propres à un corpus donné, analyse de groupes nominaux, automatisation du traitement.

La démarche harrissienne de mise au point de grammaires de sous-langages est moins connue en France que l'utilisation de sa théorie de la langue générale. Seules les études de textes politiques ont exploré cette voie : l'analyse automatique de discours politique de M. Pêcheux (Pêcheux 1969). M. Pêcheux *"a developpé*

---

34 *"On dira donc que des éléments (parties du texte, morphèmes ou séquences de morphèmes) sont équivalents entre eux s'ils se présentent dans l'environnement d'autres éléments identiques ou équivalents. Tout ensemble d'éléments équivalents entre eux s'appelle une classe d'équivalence"* (Harris 1969).

35 *"In language as a whole almost every word has a unique subset of operators on it (or arguments under it) in respect to which it has 'selectional' (higher than average) frequency. Over its whole range of operators (or arguments) this selectional subset characterizes the central meaning of the word. In respect to each operator many different arguments have different likelihoods"* (Harris 1988).

une entreprise qui, en se fondant sur la scienticificité de la linguistique, visait à élaborer des outils d'analyse capables de révéler les processus idéologiques à l'oeuvre dans le discours /./ Le cadre épistemologique de l'A.A.D se définit comme l'articulation de trois régions de connaissances scientifiques : le matérialisme historique, la linguistique (théorie des mécanismes syntaxiques et des processus d'énonciation), une théorie du discours comme théorie de la détermination historique des processus sémantiques, ces trois régions étant traversées par une référence à une théorie psychanalitique du sujet" (Maingueneau 1991).

1.4. Conclusion

> Puisqu'il est difficile d'appréhender globalement les faits de langue examinons au plus près leurs composants.

La langue présente trop de variations pour pouvoir être appréhendée globalement. Ces variations se retrouvent le plus souvent au niveau des mots où elles sont plus directement accessibles. Dans les systèmes actuels de TALN, le lexique joue un rôle actif à tous les niveaux d'analyse. Notre travail s'inscrit dans une approche lexicaliste et vise à associer aux mots les savoirs minimaux de base qu'il est possible de leur associer et qui permettent des ajustements ou des affinements potentiels à venir. Les savoirs associés aux mots ne sont pas des valeurs définitives ou prédéfinies de manière absolue : nous considérons en effet que ces savoirs ne sont pas prédictibles et qu'ils peuvent évoluer de telle sorte qu'aucune représentation ne soit capable de fixer de prime abord des valeurs définitives pour ces savoirs. De même les savoirs sémantiques que l'on peut associer aux mots constituent des noyaux de base pour la représentation sémantique. Il ne s'agit pas pour autant de valeurs absolues qui exprimeraient un noyau irréductible de signification, aucune prescription ontologique ne guide la définition de ces noyaux de base associés aux mots. Ce noyau de base doit contenir ou indiquer les pistes de sens possibles à partir d'une valeur de base. Notre tâche n'est pas de prescrire le ou les sens mais de permettre de reconstruire ce ou ces sens via le noyau de base i.e. le support pour dériver les différentes pistes de sens possibles.

Chapitre 2

Le Langage : "Si loin, si proche"

Si loin de nous, en ce qu'il est difficile à appréhender.
Si proche, en ce qu'il est une des principales activités humaines.

Ce n'est pas uniquement parce que le langage est difficile à appréhender que nous avons du mal à l'aborder, c'est aussi parce qu'il est proche de nous que nous avons du mal à nous en détacher pour le prendre aux mots ou à la lettre.

2.1. De l'étude de faits linguistiques

2.1.1. Les faits linguistiques sont empiriques

2.1.1.1. Si la linguistique est une science, elle est une science empirique[36]

Le langage possède la propriété remarquable d'être quantifié : il met en jeu divers types d'unités discrètes, les phonèmes qui composent les mots, les mots qui composent les phrases... Il s'agit d'une des rares activités humaines qui puisse être ainsi discrétisée. A partir de cette propriété de quantification du langage, il a paru possible de transformer la linguistique en une véritable science.

2.1.1.2. Le langage est à la fois, support, instrument et fondation de nos activités de représentation

Le principal obstacle à l'étude scientifique du langage réside dans la difficulté d'isoler un domaine objectivable : le langage se confond avec son propre fonctionnement et y cache ses secrets. Le linguiste et son objet sont difficilement séparables, il lui est très difficile de prendre de la distance par rapport à son matériel d'étude. Le langage est *"un système ouvert et au delà du rapport désignatif à des **réalités**, il est l'objet même de ses propres représentations...il est à la fois, support, instrument et fondation de nos activités de représentation et par suite, de nos actions symboliques sur le monde; on ne peut ainsi analyser le linguistique sans prendre en compte cette capacité structurelle de **manipulation des signes**, en conséquence*

36 *"Par empirique, on entend l'ensemble de ce qui est représentable dans l'espace et dans le temps"* (Milner 1989).

d'abstraction générique vers la catégorisation permanente des choses et du rapport à ces choses" (Vignaux 1986).

2.1.1.3. Du travail sur les faits de langue[37]

Etant donné une langue, le problème d'un linguiste consiste d'une part à déterminer ses unités discrètes minimales et d'autre part (et simultanément) à étudier les combinaisons de ces éléments. Les difficultés apparaissent très vite : le linguiste doit procéder à des regroupements et invoquer des argumentations fondées sur la répétition des unités discrètes et leur combinaison à l'échelle de la langue. Le linguiste forme des hypothèses de règles de syntaxe. Il construit alors les assemblages de mots prévus par les règles, il vérifie la validité de ses hypothèses en portant des jugements faisant appel à l'intuition sur l'acceptabilité de ses constructions. Certains faits peuvent ainsi être dégagés au moyen d'intuition d'acceptation et de refus de certaines formes. Ce sont ces jugements d'acceptabilité qui constituent la base expérimentale de la syntaxe et tout le progrès de la syntaxe provient de ce que ces intuitions ont une bonne reproductibilité.

2.1.2. De la finesse des questions linguistiques

Les problèmes traités par la science linguistique touchent en général à des phénomènes d'une extrême finesse[38] : quand on travaille sur la langue, on est contraint d'étudier des microphénomènes. Le travail du linguiste est une recherche d'équilibre entre différentes contraintes; le résultat de ce travail est toujours remis en question par la prise en compte de nouvelles informations. Il est donc condamné à rester provisoire.

2.1.3. De l'interaction de phénomènes divers dans les questions linguistiques

Les questions auxquelles sont confrontés les linguistes mettent en jeu de nombreux micro-phénomènes[39] de natures diverses qui interagissent de manière complexe. Quand on dépasse l'étude des mots isolés (ces derniers n'appartenant pas à l'objet empirique de la linguistique), on est confronté, face à un énoncé linguistique, à des phénomènes lexicaux, syntaxiques, sémantiques... Un énoncé s'inscrit dans une situation sociale et temporelle qui n'est généralement pas sans incidence sur cet énoncé.

2.1.4. Du non-achèvement des résultats dans la science linguistique

---

37 *"La linguistique doit se donner comme des faits primitifs:*

- *le fait de la langue, le factum linguae, le fait que ce que parle un être parlant mérite le nom de langue, à bien distinguer du factum loquendi ou le simple fait qu'il ait des gens qui parlent;*
- *le fait des langues, ou factum linguarum, c'est-à-dire qu'elles soient diverses, tout en contituant une classe homogène; cette multiplicité-homogénéité est censément attestée par le fait de la traduction;*
- *le fait que les langues soient descriptibles en termes de propriétés. Ce fait peut recevoir un nom plus précis: c'est le fait de la grammaire, factum grammaticae. On peut le résumer ainsi: l'activité grammaticale existe dans la plupart des communautés linguistiques. Or cette activité a des caractères propres: elle suppose qu'on puisse attribuer des propriétés à une formulation langagière sans avoir aucun égard ni à celui qui la profère ni à son éventuel destinataire ni aux circontances de la proférat ion. D'où il suit que certaines de ces propriétés seront hors circonstances, c'est-à-dire constantes.*

De là suit la disposition terminologique la plus courante. Tout de même que le terme langage, dans son usage courant, sténographie le factum loquendi, de même, dirait-on, le complexe des trois faits - factum linguae, factum linguarum, factum grammaticae - se trouve sténographié par le terme langue" (Milner 1989).

38 *"Les questions que soulève la science du langage, dans toutes ses versions, sont des questions fines; dès qu'elle dépasse la banalité, une proposition de linguistique concerne peu de données à la fois et elle y fait apparaître généralement ce que l'opinion courante tiendrait pour du détail"* (Milner 1989).

39 *"La donnée minimale de langue, qui est une phrase, est toujours trop complexe par rapport à la proposition minimale de linguistique.(...).Les langues naturelles sont organisées de telle sorte que jamais une propriété ne se trouve à l'état isolé"* (Milner 1989).

Quand on s'attaque à l'étude des activités langagières, le nombre de problèmes à résoudre est considérable et les résultats établis dans la science linguistique ne sont jamais définitifs. L'évolution perpétuelle des langues ne laisse pas intacte les connaissances que l'on peut avoir sur les langues.

On ne peut jamais en savoir assez sur le système osseux et sur le système nerveux de la langue

Vouloir construire des systèmes de représentation de la langue, c'est toujours (re)construire ces systèmes car travailler sur un fait linguistique, c'est toujours (re)travailler sur ce fait de langue.

2.1.5. La science linguistique résiste au formalisme

2.1.5.1. De la formalisation[40] du langage naturel

Un langage formel définit un ordre syntaxique en faisant opérer un ensemble de règles sur une suite de symboles. Le langage formel ne contient pas par définition les références qu'il est possible d'attacher à chacun des symboles qu'il manipule. Il faut pour cela opérer une mise en rapport entre le monde représenté et le système formel construit. Inversement, un code symbolique définit un rapport direct entre les symboles et les objets représentés, mais ne définit pas de lien syntaxique entre les symboles utilisés. Si l'on a pu penser qu'une formalisation des activités linguistiques passait par la construction d'un langage formel symbolisé, où le référentiel est soumis au syntaxique, il est apparu rapidement que la composante pragmatique du langage devait être intégrée à cet ensemble. En fait les langues mettent en jeu des ordres différents (Rastier 1994) :

- L'ordre syntagmatique qui traite de la linéarisation du langage dans l'espace spacio-temporel.
- L'ordre paradigmatique qui permet de donner une valeur à une unité relativement à d'autres qui sont commutables avec elle.
- L'ordre herméneutique qui traite des conditions de production et d'interprétation des textes.
- L'ordre référentiel qui traite des rapports entre les signes, les concepts et les choses, et des rapports entre les phrases, les propositions et les "états des choses".

2.1.5.2. De l'excès dans la langue à la formalisation

Si on considère la langue comme un système et si on veut le modéliser, il faut d'une part essayer de déterminer les connaissances à l'oeuvre dans les activités langagières et d'autre part il faut étudier la mise en oeuvre et la structuration de ces connaissances. Malheureusement, le problème de définir les modes de représentation et les interactions entre ces modèles de représentation n'est pas trivial. Celui de déterminer

---

40 *"Un moment décisif de la science du langage devra consister en ceci qu'on raisonne sur des entités purement littérales, en ne tenant compte que des lois qui régissent les combinaisons de ces entités. A cet égard ce qui se passe du côté des formalismes logico-mathématiques importe au premier chef; ils constituent un réservoir naturel de contraintes aveugles . Or dans ce domaine, l'invention existe et un nouveau type de contrainte (ou une propriété encore mal dégagée d'un formalisme déjà connu) peut toujours se révéler. Comme on l'a souligné, c'est une grande question de savoir si les formalismes logico-mathématiques forment en dernière instance la seule et unique base de littéralisations. Le structuralisme ne le croyait pas. Il n'est pas sûr qu'il ait eu raison. Pour autant, la science du langage ne cesse d'être empirique; il faut donc que le maniement aveugle produise, en fin de parcours, des propositions empiriques réfutables. En retour , choisir tel type de contrainte plutôt que tel autre dépend de la fécondité empirique qu'on peut lui reconnaître. Le croisement entre la réfutabilité (fondée par un différentiel de langue) et la littéralisation (qu'elle soit explicitement empruntée aux formalismes mathématiques ou qu'elle ne le soit pas), voilà donc ce qui doit caractériser la science du langage dans son ensemble"* (Milner 1989).

les différentes types de connaissances nécessaires pour un traitement automatique du langage naturel (TALN) ne l'est pas non plus. Les connaissances nécessaires pour comprendre ou engendrer des textes sont considérables, surtout s'il s'agit de vouloir atteindre un niveau de résultat équivalent à celui produit par un être humain. On touche là un problème crucial pour la construction de modèles de représentation du langage naturel. Comment formaliser une telle complexité d'interactions de processus qui sont mis à l'oeuvre dans la production langagière? Faut-il construire des systèmes qui dépassent nos limitations dans notre intelligence de production et de compréhension du langage naturel ou faut-il suivre le comportement humain dans les choix fondamentaux à effectuer pour modéliser les activités langagières?

2.1.5.3. Le TALN ou le dur désir de formaliser les faits de langue

Les formalismes dont on doit pouvoir disposer pour le traitement du langage naturel (et notamment pour mettre en place des dispositifs de représentation de ces modèles) doivent prendre en compte les complexités inhérentes aux activités langagières. Celles-ci mettent en jeu dans un énoncé des processus complexes interagissants : la construction syntaxique d'un énoncé s'inscrit dans l'espace et le temps et cette structuration progressive et matérielle accumule tout au long de son énonciation des indices sémantiques qui se répartissent de manière non linéaire sur les éléments de l'énoncé. Ces derniers se répondent tout au long de la production de l'énoncé et grâce aux interactions de ces traces sémantiques disposées par les éléments constitutifs de la chaîne produite dans l'espace et le temps de l'énonciation, il est généralement possible de (re)construire un des sens associé à l'énoncé perçu. Certes, modéliser de tels mécanismes n'est pas simple. D'une part, les éléments en jeu dans ce travail interprétatif sont difficiles à appréhender, car il n'est pas simple de pouvoir isoler ces éléments. D'autre part, dans le cadre d'un dispositif informatique de représentation matérielle de ces modèles, les outils manquent encore pour représenter ces mêmes phénomènes : on pense en particulier aux mécanismes réflexifs, perpétuellement à l'oeuvre dans les activités langagières, mais dont l'utilisation qui en est faite dans les langages de programmation de haut niveau n'a pas encore atteint des résultats satisfaisants. Les activités langagières mettent en jeu des mécanismes de rétro-action, d'ajustement, de masquage...; on comprend qu'il soit problématique de représenter de tels processus : ils sont inhérents à notre activité de parole, nous les manipulons en permanence, mais on a beaucoup de mal à les isoler et donc à les représenter.

2.1.5.4. Formalisation et réalité linguistique

Ainsi, même si les langages de programmation, utilisés dans le cadre des dispositifs informatiques, mettent en place des mécanismes puissants (c'est le cas des processus réflexifs pour certains d'entre eux), il n'est pas sûr que ceux-ci soient adéquats pour représenter ces mêmes processus réflexifs à l'oeuvre dans le langage[41] : leur existence dans les langages informatiques n'est pas une modélisation définitive de ces phénomènes, nous ne sommes pas encore en mesure de la réaliser. Il faut donc garder en mémoire que *"toute conception rationaliste voire formaliste du linguistique se trouve en péril non seulement de **figer** la grammaire quand on sait que celle-ci n'est que le produit d'une longue activité phylogénétique mais encore d'ignorer que dès le syntaxique, se trouvent intriquées aux **formes-signes**, les traces des manipulations sémantiques c'est à dire encore pragmatiques de ces formes"* (Vignaux 1986).

2.1.6. La science linguistique et le dispositif

"La science du langage n'a apparemment que deux possibilités : ou bien demeurer vague et renoncer à attribuer à son objet des propriétés spécifiques, ou bien rechercher la précision empirique et dépasser les données observables en proposant un dispositif" (Milner 1989).

---

41 *"Il convient toujours de poser la question lorsqu'on rencontre une théorie linguistique formalisée: les procédures formelles sont-elles dues à une exigence de rigueur, en sorte que l'on aura choisi telle procédure plutôt qu'une autre parce qu'elle est plus conforme à l'idéal métathéorique que l'on s'est proposé? Ou bien les procédures formelles constituent-elles des hypothèses sur les propriétés du langage?"* (Milner 1989)

Un dispositif[42] propose une représentation matérielle des phénomènes traités. Dans notre cadre de travail, les dispositifs, dont il est question, sont constitués par des outils informatiques développés dans le cadre de la programmation à Objets; il s'agira d'établir qu'on a bien une adéquation forte entre les opérations de connaissance que ces outils permettent et la langue. De plus, *"la relation d'une proposition de linguistique à une procédure technique ne saurait se faire directement et simplement."* (Milner 1989). L'activité langagière manipule des processus de contrôle, de rétro-action, de masquage,..., sur les formations qu'elle produit. Les dispositifs (et notamment les dispositifs informatiques que nous manipulons) doivent impérativement prendre en compte ces phénomènes à travers les systèmes de représentation qu'ils développent. C'est en ce sens qu'il convient de les interroger pour que les modèles qu'ils représentent puissent être considérés comme des représentations empiriquement correctes. Ces phénomènes sont au coeur de toutes les problématisations sur les systèmes de représentation des connaissances. En effet, les questions qui sont en jeu avec le traitement automatique du langage naturel dépassent le cadre de l'étude du langage naturel. Il ne faut pas oublier d'autre part que *"le recours à une technique ou à une science comme dispositif est éminemment contingent et déterminé par des valeurs de convenance : on choisit la référence qui, à un moment donné, fournit le dispositif le plus clair et le plus distinct pour la représentation, celui qui comporte le moins d'incontrôlable. Autrement dit ce recours est aussi précaire : une science peut être amenée à changer de dispositif à tout moment, en fonction de son propre progrès"* (Milner 1989).

2.1.7. Des industries de la langue ?

"On peut même noter, entre linguistes et techniciens de la langue, une indifférence et une ignorance réciproques plus accentuée qu'il n'est de mise dans d'autres domaines" (Milner 1989).

L'étude théorique des automates et des langages formels a pendant un temps affecté simultanément les recherches informatiques et linguistiques. Le travail fait par Chomsky sur les langages formels atteste de la relation privilégiée entre l'informatique et la linguistique. On a d'ailleurs très tôt (trop tôt) pensé que l'informatique pourrait constituer une technique pour la linguistique[43]. Encore aujourd'hui il est irréaliste de penser que l'informatique puisse servir de technique (au sens milnérien défini supra) à la linguistique, même si d'aucuns souhaiteraient voir se développer rapidement un marché des industries[44] de la langue et pouvoir ainsi disposer d'outils informatiques adaptés à tous les besoins possibles et imaginables pour résoudre toutes sortes de problèmes linguistiques. En effet, l'informatique est une discipline neuve en perpétuel développement. Or une industrie suppose préalablement une science achevée (provisoirement). Il

---

42 *"Dans le dispositif, il s'agit de proposer une représentation - et en fait une représentation matérielle - de ce qui a été découvert; si l'on adopte un dispositif informatique pour représenter le langage, on ne vise pas seulement des raisonnements, on ne se borne pas à soutenir que la méthode linguistique est analogue à la méthode de l'informatique, mais l'on affirme que, dans le cadre du dispositif et pour toutes les opérations de connaissance qu'il autorise, le langage est effectivement tel qu'on le représente en termes informatiques"* (Milner 1989).

43 Le projet de la traduction automatique a vu le jour en 1960. L'idée de base consistait à effectuer une analyse syntaxique des phrases et à s'appuyer sur la structure générale obtenue pour traduire des textes (en l'occurrence des symboles) particuliers. Les algorithmes d'analyse étaient fondés sur les combinaisons deux à deux des mots et/ou des groupes de mots. Le nombre des combinaisons à vérifier dans un texte pouvant être élevé d'où le recours à la machine, de nombreuses difficultés sont très vite apparues et la traduction automatique a fait une faillite retentissante en 1970. Il ne suffisait pas comme cela était pensé à l'époque de penser que l'information fournie par les grammaires et les dictionnaires du commerce suffiraient à construire de façon non ambiguë les interprétations des phrases.

44 *"Quand on parle d'application industrielle, on songe trop aisément à l'industrie lourde. Or les questions que soulèvent la science du langage, dans toutes ses versions, sont des questions fines; dès qu'elle dépasse la banalité, une proposition de linguistique concerne peu de données à la fois et elle y fait apparaître généralement ce que l'opinion courante tiendrait pour des détails. En bref, il paraît peu vraisemblable qu'à partir de la science du langage, on puisse développer l'équivalent d'une sidérurgie ou d'une aéronautique"* (Milner 1989).

ne saurait en être ainsi pour les langues naturelles : les descriptions sont toujours partielles, partiales et destinées à le rester : *"En tant que la science les saisit, les langues et le langage ne sont pas des matières réalisées; ce sont plutôt les lois qui régissent ces "matières"* (Milner 1989). Le TALN doit placer au centre de ses préoccupations la nécessité d'outils, de dispositifs permettant de mettre au point les connaissances à utiliser, de tenir compte de leur caractère avant tout provisoire, daté et évolutif.

Et, même si le TALN a pendant un temps bénéficié des résultats acquis par le développement de l'informatique, il reste à savoir si l'absence de solution de la continuité entre langages formels, langage naturel et langages de programmation n'est pas un postulat précipité, les points de contacts et de divergences n'ayant pas été explorés. De plus les formalisations disponibles ne permettent pas encore d'exprimer correctement les relations complexes existant, entre autres, dans la représentation des connaissances lexicales et syntaxiques d'une langue. En outre, elles ne disposent pas encore d'outils adéquats pour manipuler les représentations initiales en spécifiant des questionnements précis ou pour examiner les résultats produits grâce à des mécanismes associés à ces formalismes. Il s'agit encore pour le TALN d'utiliser l'informatique comme d'un outil de vérification (d'utilisation) des résultats connus d'une théorie linguistique.

2.1.8. La langue comme modèle de système de représentation des connaissances

"Les langues, utilisées constamment par les êtres humains et continuellement re-modelées par eux en fonction de leurs besoins, constituent des systèmes exemplaires de gestion des connaissances" (Pierre-Yves Raccah in (Dubois 1986)).

Le langage usuel est un moyen de représenter les choses, doté d'une capacité de description et d'explication considérable. L'être humain a créé des langues en fonction de ses besoins de communication et de ses possibilités de traitement de l'information. Nous sommes capables de parler et de comprendre un interlocuteur en temps réel bien que le temps de traitement de nos neurones se mesure en millisecondes. De plus l'être humain parlant structure ses connaissances sur la langue qu'il manipule de telle sorte qu'elles soient accessibles au moment où celles-ci s'avèrent nécessaires. Nous avons accès immédiatement à ce qui nous intéresse. De plus, quand nous rencontrons une ambiguïté, nous choisissons en général la solution la moins coûteuse qui consiste à ne pas retenir plusieurs interprétations et donc à ne pas maintenir plusieurs interprétations et ainsi à ne pas retarder la construction d'une structure (et cela à cause en partie de nos limitations de mémorisation). On garde malgré tout la possibilité de revenir en arrière quand les choix initiaux conduisent à une impasse en effectuant le retour le moins coûteux qui conduit à une solution encore une fois la moins coûteuse. Puisque le langage est adapté à nos besoins et à nos possibilités (ce qui prend en compte les défauts et les qualités de compréhension et de production) on peut peut-être souhaiter que les dispositifs mis en place pour le traitement automatique soient à l'image de ce que nous faisons, c'est-à-dire qu'ils tiennent compte du fait que dans notre activité langagière nous commettons des erreurs (que nous savons gérer dans le cadre de cette activité), ou encore que nous ne comprenons pas toujours tout ce que nous entendons, en précisant que toutes les opérations que nous effectuons dans notre langagière sont faites en temps réel! L'examen de diverses langues semble indiquer que certains principes combinatoires qui régissent l'organisation des unités discrètes sont généraux. D'où un second niveau niveau de problème : comment caractériser le potentiel linguistique d'un être humain? Quels sont les mécanismes qui permettent à un enfant d'apprendre une langue quelconque? Si nous savions quelles sont les méta-connaissances innées permettent à l'enfant d'apprendre à parler et à comprendre, il serait envisageable de disposer de systèmes capables d'agir de manière identique, on en saurait beaucoup plus sur la langue.

2.2. De la construction du sens

> Le destinataire d'un énoncé est un interprète. Au destinataire attentif, nul doute que le sens apparaît, même si les sons sont parfois troublés ou confus. (Une allusion à l'azur, M.Butel, l'Azur n°14)

Le sens : un papillotement de présences et d'absences.

"Ces mots poreux, chargés des discours qu'ils ont incorporés et dont ils restituent au coeur du discours en train de se faire, la charge nourricière et dépossédante, ces mots gigognes, qui se scindent, se déboîtent en d'autres mots, mots kaléidoscopiques dans lesquels le sens, multiplié à leurs facettes imprévisibles, s'échappe, en même temps, et peut, dans le vertige, se perdre, ces mots qui manquent, manquant à dire, manquant pour dire- défectueux ou absents - cela même qu'ils permettent de nommer, ces mots qui séparent, ceux-là même entre qui ils établissent le lien d'une communication, c'est dans le réel des non-coïncidences foncières, irréductibles, permanentes, dont ils affectent le dire, que se produit le sens. Ainsi, est-ce fondamentalement que les mots que l'on dit ***ne vont pas de soi****, mais si l'on veut ...* ***de l'Autre*** *: de l'Autre ouvrant le discours sur son extériorité inter discursive interne, la nomination sur sa perte relativement à la chose, la chaîne sur l'excès de sa* ***signifiance****, la communication sur sa béance intersubjective, et au total l'énonciation, sur la non-coïncidence à lui-même du sujet, divisé, de cette énonciation. Cet espace de non-coïncidences où se fait le sens, nourri de ces hétérogénéités qui le distinguent de la fixité une du signal, est aussi, indissociablement celui dans lequel il pourrait se défaire, si ne le protégeait, s'opposant à sa dispersion, une force de lien, de cohésion, de UN qui fait tenir une parole, qui fait que* ***tenir*** *une parole c'est, entre autres, faire* ***tenir ensemble*** *ce qui ne fait sens que de n'être pas un"* (Authiez-Revuz 1992).

De même que la production langagière est une construction permanente pour généralement dire quelque chose, la compréhension d'un énoncé est parallèlement une construction pour comprendre ce qui est dit. Et ce qui est dit est souvent, volontairement ou non, porteur d'éléments implicites, ambigus, ambivalents,..., auxquels le destinataire du message n'a pas accès ou qu'il peut construire (directement ou par inférence).

L'interprétation oblige souvent à résoudre dans le dire ce que le trajet du dire a produit comme masquages, brouillages, télescopages, etc., dans le vouloir dire.

Malgré tout, il existe un contraste entre l'omniprésence de la polysémie des diverses marques de surfaces dans les activités langagières et le petit nombre d'énoncés réellement ambigus. Quoi qu'il arrive en général dans une activité langagière, nous parvenons à récupérer les manques apparents, à rétablir les liens entre ce qui est perçu et une représentation de ce qui a voulu être dit, à identifier les flous du dire... De plus, *"si les êtres humains transmettent et stockent la plupart de leurs connaissances sous forme de textes en langue naturelle, c'est peut-être parce que l'ambiguïté est une qualité nécessaire à la représentation des significations, et non pas une tare léguée par l'histoire des langues, dont il convient de se débarrasser"* (Kayser 1989). Là encore, le langage manifestant une grande adaptation à nos besoins et à nos possibilités, il est souhaitable que les modèles de représentation du langage utilisés tiennent compte du fait que dans notre activité langagière nous commettons des erreurs (que nous savons gérer dans le cadre de cette activité), ou encore que nous ne comprenons pas toujours tout ce que nous entendons et que malgré tout nous réussissons généralement à transmettre / reconstruire du sens : **La représentation sémantique doit préserver l'ambiguïté et l'ambivalence inhérente au langage naturel.**

2.2.1. Représentation(s) du sens

"Trois problématiques de la signification, centrée sur le signe, dominent l'histoire des idées linguistiques occidentales :

*a) La problématique de la **référence**, de tradition aristotélicienne, définit la signification comme une représentation mentale, précisément un concept. Elle est reprise diversement aujourd'hui par la sémantique vériconditionnelle et la sémantique cognitive.*

*b) La problématique de l'**inférence**, d'origine rhétorique et de tradition augustinienne, définit la signification comme une action intentionnelle de l'esprit, mettant en relation deux signes ou deux objets. Elle est développée aujourd'hui par la pragmatique.*

*c) La problématique de la **différence**, d'origine sophistique, développée par les synonymistes des Lumières, puis par la sémantique dite structurale, définit la signification comme le résultat d'une catégorisation.*

*Enfin la problématique du **sens** prend pour objet le **texte**, plutôt que le signe, et définit le sens comme interprétation, passive ou active.*

/./ Dès que la problématique de la différence tient compte du contexte, elle dépasse le problème de la signification et s'ouvre à la question du sens...La prééminence du paradigme du sens se marque dans le fait que les sèmes inhérents ne sont actualisés qu'en fonction de licences ou prescriptions contextuelles, ce qui place en somme la signification sous le contrôle du sens" (Rastier in (Rastier & al. 1994)).

L'unification des paradigmatiques passe donc par une unification du *"paradigme lexical de la différence et le paradigme du sens, le sémiotique et le textuel, pour rendre compte de l'inférence et de la référence"* (Rastier in (Rastier & al. 1994))

Pour représenter la complexité de phénomènes à l'oeuvre dans la construction du sens d'un énoncé il ne suffit pas d'empiler des couches.

Les langues naturelles constituent un moyen irremplaçable pour communiquer et pour représenter des connaissances et toute activité langagière opère une recherche de marques de repérages que le langage construit afin que la compréhension se fasse. Ce constat ne nous est malheureusement pas suffisant pour déterminer les processus qui permettent d'identifier les connaissances présentes dans un fait de langue. La nature même de ces connaissances (et leurs rapports) restent mal connues.

Aucune énumération, aucune décomposition des unités qui composent un énoncé ne peut donner une somme de sens correspondante.

L'étude d'un énoncé consiste en un examen des articulations entre les différents niveaux de représentations des connaissances que le langage met en place : niveaux morphologiques, lexicaux, syntaxiques, sémantiques, pragmatiques. Si les linguistes ont pu établir de nombreuses théories sur chacun de ces champs particuliers de l'activité langagière, une théorie globale est loin d'être envisageable, d'autant plus que certains de ces champs d'étude sont encore loin d'être couverts. Il va ainsi pour les questions sémantiques.

En effet, l'analyse du langage pose (au moins) deux difficultés importantes quand il s'agit d'étudier les questions sémantiques. La première difficulté résulte de l'ambiguïté inhérente à toute activité langagière. Si nous avons du mal à représenter / formaliser les processus en jeu dans un énoncé, il ne faut pas oublier que l'élaboration de cet énoncé porte elle aussi souvent les traces de notre difficulté à coder sous une forme linguistique logique, rigoureuse et unique le sens d'un énoncé. C'est grâce à cette potentialité du langage de laisser les énoncés ouverts à différentes interprétations que le langage permet une grande richesse d'expressivité; et c'est d'ailleurs, en partie, en raison de cette potentialité de produire des énoncés porteurs de sens flous qu'il est difficile de formaliser les activités de langue. La seconde difficulté résulte de l'immense quantité de connaissances nécessaires pour traiter des questions touchant au langage naturel. On l'a déjà dit précédemment, cette interaction complexe de phénomènes divers pose des problèmes non triviaux. D'autant plus quand on aborde les phénomènes sémantiques qui mettent en place des mécanismes d'analyse à profondeur variable. Une des grandes difficultés de la représentation des phénomènes linguistiques "tourne autour" des questions du sens : la formalisation et donc le codage des processus sémantiques est un tâche d'autant plus ardue que ces processus manipulent des indices d'une portée générale touchant aux activités fondamentales de l'intelligence humaine. On perçoit d'ailleurs très bien qu'une

grande partie de la qualité du TALN est conditionnée par un résultat "global[45]" de cette formalisation. Cela est d'autant plus important que les questions sémantiques ont longtemps été négligées dans les travaux de réflexion sur le langage.

Une phrase signifie toujours plus. Même un mot, pris dans la texture de ses innombrables connotations est généralement irréductible.

2.2.2. De la construction du sens comme un travail

2.2.2.1. L'interprétation : une mise en action

L'interprétation consiste à proposer une cohérence sémantique du dit qui n'est pas directement le reflet d'un sens visé par le vouloir dire.

La piste sonore fournit des pistes de sens. Il s'agit ensuite de choisir parmi les chemins de sens possibles, ceux qui permettent d'accéder à un espace de sens cohérent. La construction du sens n'est pas assimilable à un processus linéaire de lecture d'étiquettes sémantiques. Il ne suffit pas en général de considérer un énoncé comme une entité dont le sens se construirait par une lecture séquentielle des indices sémantiques de ses composants. Les énoncés de langue produits sont porteurs de non-dits, de flou, de polysémie. *"Les expressions de langue ne sont plus formes désincarnées dotées d'un sens littéral, et ensuite enfilées bout à bout pour fabriquer un discours"*, elles représentent *"des indices relatifs à la construction du discours, généralement vu comme une mise en place de domaines liés entre eux, et structurés de façon interne par des modèles cognitifs"* (Fauconnier 1992). Le sens se nourrit de l'espace-entre, entre les fixités des sons produits, entre les définitions, entre les bruits; le sens se nourrit de chuchotements, mais aussi de gonflements. Le sens se négocie. Une conversation tient souvent lieu de négociations sur le sens des mots utilisés.

L'interprétation est une mise en action des matériaux mis en place par le dire.

L'analyse du sens d'un énoncé n'est donc généralement pas déterminée uniquement par la présence et la nature des formes linguistiques. Le sens n'est pas une donnée explicite des énoncés produits : le sens est une construction, le sens est un processus, un travail interprétatif. La production langagière met en place tout un réseau *ouvert* de correspondances entre les indices sémantiques présents dans un énoncé et le calcul interprétatif permet généralement de construire un (des) sens attaché(s) à l'énoncé produit. Chaque texte donne lieu à la construction d'un ensemble de mondes. L'interprétation consiste généralement à décrire les inférences qui permettent de passer de la linéarité du texte à une compréhension plus précise : le découpage du texte, l'inventaire et l'articulation des relations entre mondes jouent un rôle essentiel pour exprimer un des sens possibles du texte visé.

L'interprétation, c'est aussi s'investir dans ce processus de mise en action.

Le raisonnement au cours d'une interprétation est généralement non-monotone. On ne raisonne pas sur la seule base des informations explicitement accessibles. On est souvent amené lors d'un processus d'interprétation à faire des hypothèses, des choix... Un énoncé peut en effet porter et produire, volontairement ou involontairement, plusieurs interprétations, et c'est le résultat de cette confrontation qui fait le sens : l'énoncé "Le coup d'état de grâce" (Habert & Fiala 1989) émis au moment de la réélection de François Mitterrand est volontairement porteur de plusieurs sens, et c'est la confrontation de ces sens visés qui produit l'interprétation. De même, dire *"nous tripions"* (Jean Tardieu, entretien avec Pierre Dumayet) transforme la conversation en *"une action qui nous prend aux tripes"*, ce qu'elle est souvent d'ailleurs.

Les souvenirs, la mémoire peuvent aussi se faire reconnaissance.

---

[45] Ce terme doit être compris en tenant compte du non-achèvement des résultats linguistiques et sur leurs remises en cause continues.

Les unités linguistiques peuvent être les dépositaires d'informations qui dépassent le cadre de leur réalisation. Un mot peut être le "déclencheur" de mécanismes de rappel de situations spatio-temporels antérieures (enregistrées sous forme d'images mentales ou autres). Comment évaluer la portée de tels mécanismes dans les processus d'interprétation?

2.2.2.2. La construction du sens : un télescopage de savoirs

La construction du sens nécessite l'intervention de compétences de natures diverses qui fonctionnent en interaction. Le sens global d'un énoncé peut être assimilé à une espèce de feuilleté, un ensemble de couches articulées les unes aux autres, et dont l'organisation est à déterminer et à construire pour déterminer le sens[46] de cet énoncé. La question de savoir s'il est possible d'organiser (de manière hiérarchique ou autre) ces informations sémantiques se pose d'ailleurs de manière fondamentale dans la représentation des savoirs linguistiques. Une réelle interprétation de la polysémie, du flou, du vague, etc., nécessite la prise en compte de phénomènes qui dépassent parfois le cadre de l'énoncé produit. Aux connaissances statiques dont l'énoncé est porteur, il faut adjoindre des connaissances dynamiques qui éliminent ou atténuent les indéterminations, les imprécisions que l'énoncé révèle. Dans cet exemple, cité par (Kleiber 1994) :

"Je suis passé de la salle de bain dans la cuisine. Le plafond était très haut."
(plafond[cuisine])

"Je suis passé de la salle de bain dans la cuisine. Le plafond était trop haut."
(plafond[salle de bain])

Si l'interprétation contextuelle des deux phrases "Le plafond était très haut", bien que précédées par la même phrase, conduit à un résultat différent, c'est parce que c'est la phrase qui construit le contexte pertinent. Le contexte se trouve construit pendant l'interprétation de la phrase elle-même. De même, si le sens d'un mot hors contexte nous échappe, il peut être possible de retrouver une des significations de celui-ci dans un contexte donné. Ainsi si le verbe "stipendier" n'a pas de signification précise pour bon nombre de locuteurs du français, il ne fait aucun doute que cette signification sera éclairée par la mise en contexte suivante :

"Les jockeys du Prix de l'Arc de Triomphe ont été stipendiés pour fausser la course, ils font l'objet de poursuites judiciaires."

Le contexte est infini : aucune saturation du contexte n'est possible, on peut toujours en dire plus.

La construction du sens d'un énoncé peut s'accompagner d'une instruction à mettre en place un contexte approprié pour pallier les problèmes posés par l'incomplétude de certaines expressions (Kleiber 1994). Ainsi si l'on considère l'énoncé (J. Moeschler 1989 repris dans (Kleiber 1994)) :

"Il fait beau, mais le pape est célibataire"

si l'on nous interroge sur la "validité linguistique" de celui-ci, et si notre intuition nous conduit à le rejeter, c'est peut-être dû au fait qu'il ne nous a pas été possible, *"en temps limité et dans les conditions particulières et artificielles de la demande, de construire une situation appropriée"* (Fauconnier 1992).

---

[46] Quand on utilise l'expression "le sens d'un énoncé", il faut entendre "un des sens possibles que l'on peut attacher à cet énoncé", en particulier un de ceux que le destinataire peut construire quand il interprète cet énoncé.

2.3. Modèles pour la construction du sens

2.3.1. Connexionnisme et Composés

Le traitement des composés peut être considéré comme un domaine d'études intéressant pour l'utilisation de réseaux de neurones appliqué au langage naturel. Si le connexionnisme traite du rapport forme/sens quand il travaille sur les données de langue, les composés constituent un terrain particulièrement propice pour rendre compte des phénomènes complexes qui interagissent dans ce rapport. De plus, l'utilisation de tels réseaux qui mettent en place des mécanismes interprétatifs qui adaptent les paramètres de résolution en fonction des données traitées, peut se révéler pertinente pour traiter certaines configurations nominales dont la construction peut s'accompagner de mécanismes mettant en jeu un affinage progressif de la représentation.

2.3.1.1. Apprentissage et réseaux connexionnistes

Les réseaux de neurones sont des modèles inspirés du fonctionnement des neurones biologiques. Au cours des années 40, des chercheurs ont commencé à travailler sur la modélisation du système nerveux. Les neurones étaient simulés par automates élémentaires reliés entre eux par des liaisons ou connexions (les synapses) qui permettaient de faire passer un influx plus ou moins intense. Chaque neurone réémet des influx en fonction de la force des potentiels reçus. L'information dans un réseau de neurones se situe à deux niveaux :

- Dynamiquement, les influx circulant d'unité en unité représentent la réaction du réseau à des stimuli reçus;

- Statiquement, la force des connexions et la sensibilité des neurones caractérisent la connaissance emmagasinée dans le réseau, sa mémoire à long terme, ils modulent les réactions du réseau.

Mimant les phénomènes de plasticité synaptique observées par les neurobiologistes, les liaisons entre automates sont capables de se modifier dynamiquement, simulant ainsi des phénomènes d'apprentissage. L'apprentissage (avec ou sans modèle correct) consiste à modifier les paramètres du réseau afin d'induire un comportement ultérieur de celui-ci qui soit mieux adapté aux données reçues. L'idée de reproduire des architectures neuronales par des réseaux de neurones artificiels a été essentiellement motivée par des problèmes de reconnaissance de formes ou de classification.

2.3.1.2. Réseau de neurones et TALN

Dans le cadre du traitement des données symboliques telles que le langage naturel, on distingue deux types de réseaux : les réseaux localistes et les réseaux subsymboliques.

- **Les réseaux localistes :**

Ce sont les héritiers directs des réseaux sémantiques. Dans un tel réseau, chaque unité est associée à un des symboles du domaine (symbole de grammaire, mot du lexique ou trait sémantique). Les connexions y ont également une valeur symbolique telle que la compatibilité sémantique ou syntaxique entre des mots lorsqu'elles sont positives ou leur antagonisme dans le cas contraire. Seule l'activité propagée entre les unités est indifférenciée.

- **Les réseaux subsymboliques :**

Aucune valeur interprétative n'est assignée a priori aux composants du réseaux ou aux activités qui s'y propagent. Les stimuli appliqués sur le réseau sont encodés de façon opaque (codage aléatoire) ou parcellaire (utilisation de micro-traits). S'il émerge une organisation interne qui n'a pas été explicitement

introduite initialement, elle est révélée par une analyse des modifications après apprentissage. L'utilisation d'un réseau se passe en deux temps :

Phase d'acquisition : une procédure d'apprentissage et un corpus d'entraînement servent à l'ajustement automatique des paramètres. Dans les réseaux localistes il est possible d'attribuer à la main des valeurs aux paramètres du réseau.

Phase d'exploitation du réseau : on observe les capacités de classification et de prédictivité sur des données nouvelles inconnues.

Les réseaux sont utilisés, par exemple, pour apparier sens et forme et rendre compte de la polysémie et de compositionalités complexes (Fuchs &Victorri 1990).

2.3.1.3. Réseau de neurones et "noms composés"

(Habert & Jacquemin 1993) présentent un panorama des traitements automatiques mis en place pour l'analyse des noms composés. Dans le cadre des méthodes connexionnistes, nous reprenons leur présentation du travail de C. Jacquemin.

- (Jacquemin 1992) met en place un réseau localiste pour la reconnaissance de noms composés subissant des variations tels que `câble à haute tension`. A chaque entrée lexicale est associé un lemme. Le réseau est divisé en sous-réseaux : un sous-réseau a pour tâche de reconnaître un nom composé particulier. *"Par baisse de seuil, le réseau révèle les variations inattendues des composés et un apprentissage supervisé permet d'intégrer cette connaissance dans le réseau par une modification de poids. La reconnaissance ultérieure de la modification ne nécessitera plus alors de baisse de seuil, mais sera acceptée par la nouvelle configuration de poids obtenue par apprentissage"* (Habert & Jacquemin 1993).

- (Jacquemin 1993) met en place un réseau qui s'inspire de la neurophysiologie pour la mise en oeuvre des mécanismes d'apprentissage et de propagation sur deux tâches liées à la composition nominale. *"Le premier ensemble de données introduit dans le réseau est composé d'interprétations sémantiques de composés en `N à N` et des acceptabilités de ce composé pour certaines variations syntaxiques et certains tests interprétatifs. Par exemple `verre à pied` a pour sémantique `AVEC (partie-de)` et accepte des variations telles que la modification adjectivale du deuxième nom dans `verre à pied en métal`. L'observation du réseau met en évidence des connexions renforcées pour certaines associations statistiquement pertinentes"* (Habert & Jacquemin 1993). La deuxième tâche doit rendre compte *"de la compositionnalité dans les composés en `cuisson Prép N` en fonction de la préposition et du nom en jeu /./. Le renforcement de certaines connexions distingue le rôle inférentiel des prépositions incolores (`à,de`) du codage sémantique des prépositions colorées (`sur, nous`) qui imposent l'interprétation finale"* (Habert & Jacquemin 1993).

2.3.2. Représentation hiérarchisées : mots ou choses

2.3.2.1. Hiérarchies

Ontologie vs. Classifications linguistiques

- Les choses telles qu'elles sont. Et les classes d'objets qu'une langue donne à voir.

La construction du sens met en oeuvre une confrontation permanente entre les faits de langue et les existants du monde. Le traitement de la polysémie pose d'ailleurs clairement le problème de *"la conceptualisation des entités de base supposées"* (Habert & Cadiot 1997).

- L'exemple de "livre".

  Un livre peut être considéré comme un Objet-Physique.

  Un livre peut aussi être appréhendé via le contenu qui le compose.

  Il est aussi un élément constitutif du processus de publication ou de commercialisation attaché à la réalisation de ce livre.

Savoir sur le monde et savoir sur la langue

a. Articulation Langagier-Concepts

Définir le sens est une tâche délicate. Si comme Mallarmé, on n'accorde au mot "fleur" que la seule légitimité d'être "l'absente de tous bouquets", on aboutit à une sémantique de l'absence qui interdit toute définition sémantique formelle. Ce qui est en cause fondamentalement dans les problèmes de construction du sens, et plus généralement dans l'étude de modèle de représentation du langage naturel tourne autour des notions de concept, de signe et de référent. Dans le système de la langue, le signe "fleur" est un assemblage arbitraire de deux voyelles et trois consonnes. Il est donc à coup sûr porteur d'une forme. Le passage de ce signe dans le discours va permettre de référer à un objet unique : "cette fleur est bleue". Le sujet de cette phrase désigne un objet du monde identifiable (le référent). Le signe est donc un simple élément capable d'éveiller dans l'esprit une idée et qui n'est pas sa représentation : le signe apparaît comme un déclencheur. Les mots et les phrases ne présentent aucune affinité préétablie avec les objets. *"La problématique de la référence, de tradition aristotélicienne, définit la signification comme une représentation mentale, précisément un concept. Elle est reprise diversement aujourd'hui par la sémantique vériconditionnelle et la sémantique cognitive"* (Rastier & al. 1994). Le concept est le fruit de l'activité de l'intelligence (conception). *"Depuis G.Frege (Frege 1893), un concept peut être vu comme une fonction définie sur un certain domaine et à valeur dans l'ensemble de valeurs de vérité {vrai faux}. Cette fonction a pour but de discriminer les objets auxquels s'appliquent le concept..."* (Haton & al 1991). Selon Lyons, le concept *"exprime toute idée, toute pensée ou toute construction mentale au moyen de laquelle l'esprit appréhende les choses ou parvient à les reconnaître"* (in Haton & al 1991), selon Kayser, un concept est une *"entité symbolique recevant des propriétés inférentielles"* (in Haton & al 1991).

1. Inutilité de définir le sens d'un mot

L'approche de D. Kayser (Kayser 1987,1992) vise à modéliser non pas le sens, mais les processus de compréhension des phrases en leur associant des ensembles de transformations gouvernées par des règles d'inférence plausible. De plus, cette approche vise à redéfinir la notion de concept traditionnellement associée aux mots. "Livre" selon D. Kayser renvoit à une multiplicité de types de référents possibles dont on peut rendre compte en se passant de la notion de sens.

"Livre" peut renvoyer à :

- un élément indéterminé d'une classe d'équivalence

  (1) Ce livre se trouve dans toutes les bonnes librairies

- une entité à partir de laquelle les éléments de cette classe ont été créés

  (2) Ce livre a été tiré à 150000 exemplaires

- un objet manuscrit

  (3) Jean est parti à la campagne pour écrire un livre

- aux idées contenues dans ce livre

(4) Ce livre a fortement influencé les révolutionnaires de 1789

- à la commercialisation de ce livre

(5) Ce livre a été un fiasco pour son éditeur

- à la lecture de ce livre

(6) Ce livre est ennuyeux

La variation de prédicat entraîne également un référent différent et donc un sens différent pour "livre" dans (3), (4), (5).

D. Kayser émet l'hypothèse du mot générateur d'entités dans un réseau sémantique :

"Lorsque l'interprétation d'un mot est souhaitée, on fait fonctionner le générateur au premier niveau. On obtient ainsi un ou plusieurs noeuds (au groupe nominal "un livre" correspond un noeud situé dans la branche hiérarchique dominée par le noeud "Objet-Physique"). Si ce nouveau noeud remplit les conditions souhaitées, le processus peut s'arrêter là. Sinon on demande au générateur de fabriquer un ou plusieurs autres noeuds ("représentant indéterminé d'une classe", actions acceptant "livre" comme objet : la lecture, l'écriture, la publication...). On réitère ce processus jusqu'à trouver un (ou plusieurs) noeuds satisfaisants, ou jusqu'à avoir épuisé les ressources mises à sa disposition" (Kayser 1987).

2. Utilité de définir le sens d'un mot

Pour Kleiber et Riegel (Kleiber & al. 1989), il convient de maintenir pour "livre" l'hypothèse d'un sens unique, capable d'expliquer à la fois :

- la variation référentielle des SN comportant "livre" dans les énoncés (1) et (7);

- la possibilité d'avoir avec un référent stable des effets interprétatifs différents dus à ce qu'ils appellent le principe de métonymie intégrée (certaines caractéristiques de certaines parties peuvent caractériser le tout).

1. Pour Kleiber et Riegel, il n'est guère besoin de procéder à un éclatement sémantique de "livre" pour rendre compte du changement de référent des SN des énoncés suivants :

(7) Ce livre est sale est déchiré

Ce livre se trouve dans toutes les bonnes librairies

Il s'agit de mécanismes référentiels généraux liés à la distinction type/token. La structure conceptuelle du mot "livre" (i.e. son sens) révèle l'existence d'entités individuelles regroupant elles-mêmes des occurrences.

2. On peut résoudre les interprétations (3), (4) et (5) en optant pour un référent qui reste le même et en expliquant les phénomènes d'interprétation globale au moyen du principe de la métonymie intégrée.

b. Représentation des concepts

"Un des grands problèmes relevés dans la modélisation des connaissances en Langage Naturel est celui du principe d'une relation "simple" entre le mot (réalisation lexicale) et le concept (conception cognitive)" (Kayser 1991).

Traditionnellement, on considère le concept comme un atome de signification que l'on traduit en logique par des prédicats ou par des constantes. On peut aussi le représenter sous forme de molécule qui se décompose en primitives atomiques. Dans les deux cas, un concept se traduit par un objet unique. Dans l'approche développée par Kayser, un concept `C` est représenté par une famille ouverte d'objets : l'ensemble des `x` pour lesquels `C` est un NOM-VALIDE dans un certain contexte `Contexte`.

```
Représentation de C : = { x tels que NOM-VALIDE(C, x, Contexte)}
```

D. Kayser (Kayser 1988a) décrit *"un certain nombre de caractéristiques que possèdent les concepts utilisés dans la vie courante"* et montre que *"ces caractéristiques ne sont pas des traits superficiels, liés au langage au niveau lexical mais des traits profonds, liés à la nature même de nos connaissances"*.

"Il existe un contraste frappant entre le nombre de situations que nous pouvons discerner et le nombre d'éléments de description que notre appareil cognitif nous permet de manipuler. En effet, la finesse de nos perceptions nous apporte une quantité d'informations sans commune mesure avec nos facultés de manipulation et de mémorisation; faute d'être capables de gérer des assemblages de concepts distincts pour représenter des situations distinctes, nous réutilisons les mêmes concepts, mais nous savons que ces concepts n'ont plus alors de définition précise, et des mécanismes d'analyse à profondeur variable nous permettent à chaque occasion de reconstituer l'interprétation la plus adéquate du concept considéré" (Kayser 1992).

c. Le Concept : un point d'entrée pour des inférences

Kayser avance que *"les suggestions habituellement proposées (logiques multi-valuées et (ou) non monotones) pour tenir compte de la "typicalité" ne cernent pas parfaitement le phénomène. L'idée force consiste à séparer la notion de concept de celle d'élément de base (prédicat, noeud) et de représenter un concept par une famille ouverte d'entités du système. Chaque entité dénote une interprétation possible du concept, et ces interprétations sont ordonnées par profondeur croissante"* (Kayser 1988a). Les concepts sont donc des points d'entrée qui permettent de déclencher des inférences, de plus ils dominent des graphes ouverts et évolutifs car *"il importe de pouvoir modifier le grain de représentation au cours d'un raisonnement"* (Kayser 1992).

Ainsi à propos du concept de `"lion"` : on peut dire que *"ce concept ne se traduit pas par une seule entité dans le système de représentation, mais par une famille ouverte, et partiellement ordonnée, d'interprétations de ce concept."*

Kayser pose les définitions suivantes :

notion : ce que l'on veut représenter,
entité : élément du système de représentation,
concept : cas où l'on postule une correspondance biunivoque entre les deux.

"Au lieu de représenter une notion par une seule entité du système de représentation, nous décidons de lui faire correspondre une famille ouverte d'entités. On accède à cette famille par un point d'entrée (un mot du langage indexe un point d'entrée, sauf dans les cas d'homonymies "accidentelles"); on ne substitue à ce point d'entrée d'autres éléments de la famille que si les inférences en cours l'exigent" (Kayser 1992).

"Cette approche permet :

- *de réduire la distance entre mot et concept : `C` est le nom d'un concept; c'est aussi un mot de la langue : un mot ne correspond à une multiplicité de concepts qu'en cas d'homonymie accidentelle;*
- *de rendre compte de certains mécanismes de coréférence : le principe circonscriptif tend à maximiser la coréférence, et la logique des défauts permet de formaliser le choix de l'antécédent contextuellement le plus plausible;*

- *mais surtout de traiter certaines régularités des phénomènes d'extension de sens et de glissement de sens : une même expression sert à désigner différentes notions qui entretiennent entre elles des rapports constants; /./ De tels ajouts à l'ensemble des significations possibles d'un mot sont représentables par une règle générale d'extension de sens (règle de type (RD) pour prendre en compte les exceptions locales); de plus, un tel ensemble se construit dynamiquement, au fur et à mesure que la profondeur d'analyse augmente"* (Kayser 1988b).

On s'écarte donc de la notion traditionnelle du sens[47] dans la mesure où il n'existe pas d'espace de sens et qu'il est donc impossible d'exhiber un objet (formule, graphe, point, ...) en affirmant : voici le sens de cette expression (Kayser 1992).

d. `Moulin` et la profondeur variable

L'approche suivie par D. Kayser vise à établir un équilibre entre des savoirs établis (non modifiables par inférence) et le raisonnement (les processus qui affinent par inférence les savoirs établis). La profondeur variable suppose en effet une hiérarchie de concepts aux noeuds de laquelle sont attachées des connaissances, des rôles qui permettent des inférences. Cela structure le savoir sur le monde effectivement mis en oeuvre. Cette hiérarchie est le support de l'inférence. Le savoir sur le monde est structuré et cette organisation guide le travail inférentiel.

Ainsi dans le cadre des séquences `moulin à N2`, on peut mettre en oeuvre la profondeur variable défendue par D. Kayser : on construit au départ une représentation sémantique initiale peu déterminée (uniquement le prédicat pour `moulin`) qui peut ensuite être affinée en tenant compte des éléments d'informations complémentaires permettant d'étendre la représentation initiale. On retrouve là la notion du sens comme processus : le sens n'est pas une donnée, il est reconstruit par affinements successifs. Ce type d'approche permet tous les remodelages possibles. **Il reste à pouvoir contrôler ces ajustements ou ces remodelages** (comment éviter de surgénérer ?) et donc à fournir des processus de contrôle sur les entités visées si l'on veut évaluer les modifications réalisées. Cette approche met aussi clairement en avant la difficulté qu'il y a à articuler les différents niveaux de savoirs manipulés. La mise en place d'articulations dynamiques entre les savoirs décrits n'est pas un problème qui se laisse résoudre facilement surtout si les structures de représentation utilisées ne permettent pas de tenir compte de tels ajustements toujours possibles.

Une approche conceptuelle à la Kayser met aussi en lumière les difficultés de penser et de représenter une articulation cohérente des connaissances conceptuelles et des connaissances lexicales habituellement manipulées dans les modèles de représentation du langage naturel : de bons outils de ce type ne sont pas encore disponibles en raison justement de cette difficulté à mettre en place une telle articulation. D'autant plus qu'il ne s'agit pas de privilégier une pratique par rapport une autre : à l'inverse d'une approche lexicaliste, l'approche conceptuelle a ainsi du mal à rendre compte de certains phénomènes comme les régularités sémantiques ou morphologiques. Cette approche met aussi en lumière les limites des formalismes d'unification : comment, dans un cadre symbolique de représentation, peut-on rendre compte des probabilités différentes des interprétations ?

2.3.2.2. Accès à des scénarios (Pustejovsky & Boguraev 1991)

Deux approches pour traiter la polysémie

a. Approche lexicographique traditionnelle

---

[47] *"Ce que nous avons dit des concepts et des mots correspond à l'idée qu'ils n'ont pas de signification propre, et que cette signification peut se modifier au cours des processus inférentiels. On arrive ainsi à des notions d'ambiguïté conceptuelle, voire de pensée ambiguë. De telles notions sont incompatibles avec des modèles vériconditionnels du sens. Elles pourraient donc contribuer à orienter la sémantique formelle dans des directions fort différentes de celles où elle s'est dirigée jusqu'ici"* (Kayser 1992).

Cette approche consiste à créer autant de d'entrées qu'un mot a de types sémantiques. On dira qu'il y a deux entrées "livre", une pour l'objet physique et l'autre pour son contenu.
Cette approche pose les problèmes suivants :

- Elle postule qu'il est possible de lister tous les sens et ne rend pas compte de la possibilité de créer des sens en contexte.
- Elle génère aussi de l'ambiguïté lexicale et pose donc le problème de la désambiguïsation.
- Elle n'explique pas non plus le lien qui existe entre les différents sens (le lien entre l'objet physique et son contenu pour "livre").
- Elle traite enfin toutes les ambiguïtés de la même manière.

b. Approche semi-polymorphique du lexique génératif de Pustejovsky

Le travail de Pustejovsky vise à une prise en compte de l'évolution permanente du lexique et à intégrer ces phénomènes dans le travail de représentation de celui-ci. Cette évolution permanente est le résultat de la création perpétuelle de mots nouveaux mais aussi le résultat d'une mise en contexte des éléments lexicaux capable de produire des transferts, des élargissements, des glissements de sens. Une représentation statique du lexique n'est évidemment pas apte à rendre compte des mécanismes de sélection du sens d'un mot dans la mesure où elle ne met pas en place les liens qui permettraient une discrimination entre les différents sens possibles. De plus les facteurs contextuels ne sont pas toujours suffisants pour préciser le sens d'un mot.

Repérer les régularités lexicales

Il ne s'agit pas de créer autant de représentations du sens d'un mot qu'il y a de réalisations syntaxiques de celui-ci produisant une interprétation différente mais au contraire de capter les généralisations pertinentes entre les différents sens d'un mot (Pustejovsky 1991) tout en définissant la notion d'inférence lexicale. Le but de cette approche est de ne pas avoir un type sémantique par mot, mais de donner aux mots une riche représentation typée qui peut être considérée comme une réserve de types, en indiquant les différentes extensions sémantiques possibles d'un mot. Chaque mot est donc représenté par plusieurs niveaux de représentation qui sont connectés entre eux.

- La structure argumentale spécifie le nombre et le type des arguments d'un item.
- La structure événementielle permet de définir le type et la structure événementielle.
- La structure des qualia indique les caractéristiques sémantiques essentielles de l'item lexical, c'est-à-dire les propriétés et les événements qui lui sont liés, et ce dans quatre rôles :
    - Un **rôle constitutif** qui indique la relation entre un objet et ses constituants ou parties : matériau, poids, parties.
    - Un **rôle formel** qui distingue l'objet à l'intérieur d'un domaine plus large : orientation, grandeur, forme, dimension, couleur, position.
    - Un **rôle télique** qui exprime la fonction et la finalité de ce que décrit l'entrée lexicale visée : but, fonctions constitutives.
    - Un **rôle agentif** qui spécifie ce qui se rapporte à l'objet en général, en particulier à son origine : créateur, artefact, espèce naturelle.

Une entrée lexicale est donc associée à un prédicat au sein de la structure que constitue le qualia. La structure de qualia est décomposée sous forme de champs descriptifs qui permettent la composition sémantique.

```
livre
CONSTITUTIF = pages
TELIQUE = lire
AGENTIF = écrire
FORMEL = objet physique
```

Les différentes facettes de signification d'une entrée lexicale sont donc réunies dans une structure qualia et les relations d'héritage permettent d'infléchir l'interprétation des entrées lexicales. Cette hiérarchie doit traduire les liens transversaux qui existent entre les entrées lexicales et les liens orthogonaux que l'on peut définir entre la représentation lexicale et une représentation des concepts organisée hiérarchiquement. La signification d'une entrée lexicale est saisie au travers d'une série d'oppositions plutôt qu'en décomposant cette entrée en une suite de primitives sémantiques.

Mais cette structure traduit aussi un certain point de vue sur le mot "livre" et sur son rapport au monde. On peut modifier les valeurs associées aux rôles associés à la structure qualia si on adopte un autre point de vue : on peut en effet considérer que le livre est constitué de parties ou de chapitres différents (CONSTITUTIF = chapitres), qu'il est porteur d'informations (FORMEL = information), qu'il est conçu pour être acheté (TELIQUE = acheter) et enfin qu'il est produit par un imprimeur (AGENTIF = imprimer). La question de savoir quelle attitude adoptée pour appréhender ce mot reste donc ouverte et la structure qualia ne propose en fait que la mise en place dans la définition du mot que d'une description particulière.

L'opération de coercion

La coercion est une opération sémantique qui convertit l'argument d'une fonction en un type attendu par celle-ci :

"if α is of type <b, a>, and β is of type b, then $\alpha(\beta)$ is of type a" (Pustejovsky 1991)

Si on suppose que chaque expression a à sa disposition un ensemble d'opérateurs de conversion, Σ_α, qui peuvent agir sur elle (en changeant son type et sa signification), on peut s'y référer comme à des alias de cette expression. De plus, il est possible de faire appel directement à l'un de ces alias dans les règles définies pour la description des items lexicaux.

"if α is of type <b, a>, and β is of type c, then :
(i) if type c = b, then $\alpha(\beta)$ is of type a;
(ii) if there is $\sigma \in \Sigma_\beta$, such that $\alpha(\beta)$ results in an expression of type b, then $\alpha(\sigma((\beta))$ is of type a;
(iii) otherwise a type error is produced" (Pustejovsky 1991)

Cette opération permet la prise en compte des phénomènes de coercion de sens sur l'énoncé :

"Cette 2CV conduit dangereusement"

Comme il n'est pas souhaitable de classer "2CV" dans les Nhumains, cette opération permet justement, dans ce contexte particulier et temporairement, de modifier le type initialement associé à "2CV". Cette opération permet aussi de prendre en compte l'interprétation des configurations nominales du type moulin à paroles qu'il est possible de considérer comme une coercion de type.

L'exemple de "fast"

Pustejovsky (Pustejovsky & al. 1993) illustre son travail de représentation de la polysémie lexicale sur l'entrée lexicale "fast" à laquelle on peut attribuer au moins les différents sens suivants :

- fast1 : se déplacer rapidement

"The island authorities sent out a fast little government boat, the Culpeper, to welcome us"

- fast2 : réaliser une action rapidement;

"a fast typist"

- fast3 : disposer d'un court laps de temps pour réaliser quelque chose;

"a fast book"

- fast4 : prendre rapidement des initiatives.

"You may decide that a man will be abble to make the fast, difficult decisions"

a. Analyse de "fast" comme s'appliquant au rôle télique du nom qu'il modifie

Comme on l'a vu précédemment, il ne s'agit pas de proposer une représentation qui énumère les différents sens recensés. Pustejovsky propose au contraire de construire une représentation lexico-sémantique qui indiquera que l'adjectif agit sur le prédicat de type "event" (en modifiant le type d'activité) contenue dans le rôle télique du nom qui lui est associé : "fast" peut différencier le prédicat attaché au nom tête qu'il modifie.

car (*x*
Const : { body, engine,...}
Form : physobj(*x*)
Telic : drive(T,y,*x*)
Agentive : artifact(*x*)

Considérons "fast" comme partie intégrante du prédicat associé au rôle télique du nom qu'il modifie. Ici "car" dans "a fast car". Dans cette expression, c'est l'activité de conduire qui est modifiée par l'adjectif. Les mêmes remarques s'appliquent à (nom tête:"typist", rôle télique : type(T,y,*x*)) dans "a fast typist" ou à (nom tête:"reader", rôle télique : read(T,y,*x*)) dans "a fast reader".

La structure qualia et l'opération de coercion définies par Pustejovsky donnent la possibilité de générer différentes interprétations d'une entrée lexicale sans avoir à les lister toutes de manière rigide.

Les opérations génératives qui opèrent sur cette représentation sont :

- la possibilité de générer, à partir de la structure des qualia, les différents types sémantiques du mot;
- la coercion de types qui permet à un prédicat de changer le type de son argument vers celui qui est requis par le prédicat;
- le liage sélectif qui permet à un prédicat de ne modifier qu'une partie de la structure des qualia;
- la cocomposition, qui permet à un argument de changer la sémantique du prédicat en composition.

Représentation de la polysémie nominale

Cette approche distingue deux types de nom : les noms non-polysémiques qui ont des types sémantiques simples et les noms polysémiques qui ont des types sémantiques complexes. Un nom a un type simple s'il réfère à un seul type. Dans ce cas, il a un seul argument, qui correspond directement au type du rôle formel. Un nom a un type complexe quand il réfère à un type pointé[48] (t1.t2). Ce type correspond au produit cartésien des types définis dans la structure argumentale, c'est-à-dire t1 et t2. Si le nom a deux arguments : le rôle formel définit la relation entre les deux arguments; le nom dénote trois types différents t2.t3, t2, t3. L'ensemble des types dénotés par un item constitue son "paradigme conceptuel lexical". `"Livre"` par exemple a un type complexe : il a deux arguments dont le rôle formel indique la relation : y contient x. Sa fonction est d'être lu et il est créé par l'acte d'écriture. Le nom `"livre"` peut donc projeter séparément les trois types de son "paradigme conceptuel lexical" : le type pointé, `informationnel` et `phys-objet`. Les types événements seront aussi accessibles par coercion ou liage sélectif.

Evaluation paresseuse par nécessité (Copestake & Briscoe 1994)

A la manière de Pustejovsky, (Briscoe & Copestake 1993) étudient les mécanismes d'inférence sur le lexique qui permettent d'articuler analyse syntaxique et interprétation en proposant une décomposition lexicale du même type. Là encore leur approche met en avant la nécessité de définir une hiérarchie des concepts en parallèle à la hiérarchie lexicale. Leur approche utilise un langage de représentation lexicale qui manipule des attributs typés pour définir les entrées lexicales : les types permettent une structuration des entrées lexicales qui sont associées à des structures d'attributs typés. Là encore, on est confronté à une difficulté liée à la mise en place de hiérarchies rigides qu'il est souvent difficile de faire évoluer.

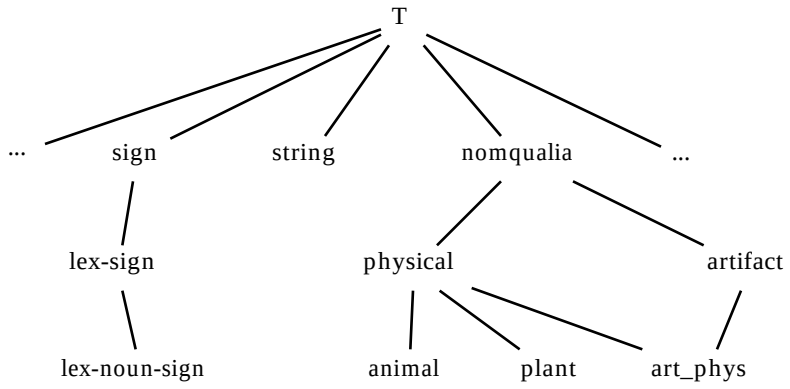


Figure 2.1 : Fragment de la hiérarchie de types (Briscoe & Copestake 1993)

Une description lexicale est par exemple la structure suivante :

```
book 1
< > = lex-noun-sign
< QUALIA > = art_phys
< QUALIA TELIC PRED > = read
< QUALIA FORM > = indiv
```

Cette utilisation de la structure de `qualia` (qui utilise une représentation hiérarchisée de concepts) permet de décrire des entrées lexicales par héritage de structure `qualia`. C'est le cas pour un des sens associé à l'entrée `"novel"` :

```
novel 1
```

[48] L'introduction du type pointé permet ainsi de lever l'objection soulignée supra sur le choix du rôle formel associé à `livre`.

```
< QUALIA > < < book_1 < QUALIA > .
```

qui spécifie que cette entrée hérite d'une structure `qualia` (via le symbole <) construite pour `"book"`. Il est aussi possible d'affiner une relation d'héritage en rendant modifiable une partie des attributs pour rendre possible la redéfinition de certaines entrées lexicales dans la structure hiérarchique. Certaines valeurs d'attributs peuvent ainsi être marquées comme modifiables (via un `"/"` à la droite de la valeur), ou non-modifiables (via un `"/"` à la gauche de la valeur). Pour inscrire `"dictionary"` dans la structure hiérarchique initiée la représentation de `"book"`, il est nécessaire de modifier le rôle télique de `"book"` et le rendre modifiable dans la mesure où si `"dictionary"` peut hériter de la structure `qualia` définie par `"book"`, il est nécessaire d'ajuster le rôle télique en `"refer-to"`.

```
book 1
< > = lex-noun-sign
< QUALIA > = art_phys
< QUALIA TELIC PRED > = /read
< QUALIA FORM > = indiv

dictionary 1
< QUALIA > < < book_1 < QUALIA > .
< QUALIA TELIC PRED > = /refer_to
```

a. Traitements de la polysémie

- Polysémie de construction (Constructional Polysemy)

"There are many cases of apparent polysemy which we would argue are better treated as 'constructional' polysemy, in that the lexical item is assigned one (often more abstract) sense and processes of syntagmatic combination or 'co-composition' are utilised to specialise this sense appropriately. We treat this as a process of sense modulation, represented by specialisation in the Lexical Knowledge Base, in contrast to the process of sense extension /./, which we represent using lexical rules" (Briscoe & Copestake 1993).

Cette notion correspond à la nécessité de choisir l'aspect de sens approprié (dans le possible défini) à une situation donnée plutôt qu'un changement de sens (exemple de `"fast"` in (Briscoe & Copestake 1993) p20).

- Extension de sens (Sense extension)

"By contrast with constructional polysemy, we argue that there are systematic polysemies which are best represented as lexical rules, which we refer to as sense extensions; that is predictable creation of different but related senses" (Briscoe & Copestake 1993).

Au lieu de lister toutes les extensions de sens dans le lexique, ce qui impliquerait d'ailleurs une remise à jour permanente du lexique, il s'agit de représenter via des règles lexicales les mécanismes qui réalisent ces extensions de sens. C'est le cas des emplois métaphoriques des noms d'animaux appliqués pour décrire des êtres humains.

b. Les règles lexicales

"Lexical rules are used to create derived sense. /./ We use the lexical rules to dynamically generate alternatives during constraint resolution of nodes with lexical types" (Briscoe & Copestake 1993).

Les règles lexicales peuvent être vues comme des processus de conversion. Dans l'emploi métaphorique des noms d'animaux pour qualifier les êtres humains, il suffit d'ajuster la représentation sémantique initiale, en ajoutant un type `animé` et deux sous-types `humain` et `animal`, puis construire un mécanisme de conversion qui réalise l'interprétation de ces emplois métaphoriques.

Telle entrée lexicale : tels scénarios?

a. Tous les noms sont-ils descriptibles par une structure prédicative ? Tous les noms ont-ils des rôles téliques ?

"Pustejovsky ne cherche pas fonder linguistiquement les distinctions qu'il établit (sauf pour les types P, S et T). Celles-ci restent avant tout cognitives même si les sous-champs discriminés diffèrent dans l'utilisation linguistique qui en sera faite. Il est même plausible de soutenir que les deux ordres de discriminations (cognitif et linguistique) doivent être maintenus et ne sont pas forcément isomorphes. Le point important, sur lequel l'accord paraît établi, est qu'il convient de circonscrire l'accès à la Représentation Sémantique lexicale" (Fradin 1993). Les champs définis dans la structure `qualia` peuvent-ils s'appliquer à toutes les entrées lexicales? Fradin (Fradin 1993) pense que non, notamment pour ce qui concerne la distinction rôle formel et rôle constitutif. De plus, *"ces distinctions ont-elles une pertinence linguistique. Rien n'est moins sûr, car la frontière de l'inaliénabilité, par exemple, passe à l'intérieur de chacune"* (Fradin 1993). *"Conceptuellement, ce qui ressortit au rôle télique est plus clair. D'un point de vue formel, il devrait regrouper des formules avec plusieurs arguments. D'autres cependant resteraient sous le rôle agentif, notamment toutes celles exprimant l'origine de l'objet référé. tel que nous le comprenons, le rôle agentif a trait à la catégorisation de l'objet (ou de l'évenance) et à sa place dans un chaînage de causalité"* (Fradin 1993). Fradin propose d'ailleurs de réorganiser la structure `qualia` en la réduisant sous la forme suivante :

- Un rôle aspectal qui *"concerne les propriétés aspectales et constitutives : rapports tout/partie, figure/site et les propriétés servant à identifier l'objet en question"* (Fradin 1993).
- Un rôle télique qui indique les propriétés téliques au sens large : *"la fonctionnalité de l'objet aussi bien que son origine"* (Fradin 1993).
- Un rôle qui traduit le mode d'existence de l'objet, *"essentiellement sa catégorisation par rapport aux enchaînements de causalité"* (Fradin 1993).

b. Aspect fini de la représentation : on fixe le nombre de scénarios

Ce type de représentation reste en effet en deçà des possibles de réalisation d'une entité donnée. Puisque les faits de langue sont soumis à des évolutions permanentes, leur représentation se trouve en permanence remise en question. Il faut pouvoir les ajuster en tenant compte des incidences potentielles que les conditions de leur réalisation révèleront et il est difficile de prévoir tous les scénarios possibles à attacher aux mots.

2.3.2.3. Les classes de mots

G. Gross (Gross 1994) travaille dans le cadre théorique d'un lexique composé de phrases et non de mots isolés et propose un modèle pour le traitement informatisé du lexique à des fins notamment de traduction automatique. Il construit pour cela des classes de noms clairement spécifiées. Les classes de noms sont définies non pas "dans l'absolu", mais toujours en fonction d'un opérateur approprié (plus précisément d'une construction syntaxique). Les classes d'objets représentent en fait le domaine d'arguments de cet opérateur. Les classes d'objets sont donc définies syntaxiquement et elles sont destinées à classer les noms et à décrire le domaine d'arguments des prédicats verbaux. Cette approche vise à établir une véritable approche sémantique via une méthode de classification par opérateurs syntaxiques. D. Le Pesant (Le Pesant 1994) en donne une illustration sur les compléments du verbe `"lire"`. Cette étude porte sur 1500 noms environ, classés en fonction de leurs opérateurs appropriés. Ce classement formel constitue aussi un classement sémantique qui distingue les "textes" des "supports de texte" et établit des sous- classes sémantiques homogènes.

N0+h lire que P
N0+h lire du N1 nph

(1) **N0+h lire N1(-/+)h**

N0+h : groupe nominal sujet humain
que P : proposition complétive
N1 : groupe nominal complément d'objet
nph : nom propre humain
(-/+)h : humain ou non humain

"Je suis en train de lire que la critique ne considère pas Pynchon comme un romancier américain"
"Je suis en train de lire du Pynchon"
"Je suis en train de lire un romancier américain"

(2) **N0+h lire N1-h**

"Je suis en train de lire un poème"

N0+h V N1-h (dans + sur) N2 inc

(3) V : déchiffrer, décrypter, lire, relire
Restriction : (dans + sur) N2 est une détermination non de N0 mais de N1
inc : inanimé concret

"Je lis un poème sur un radiateur"

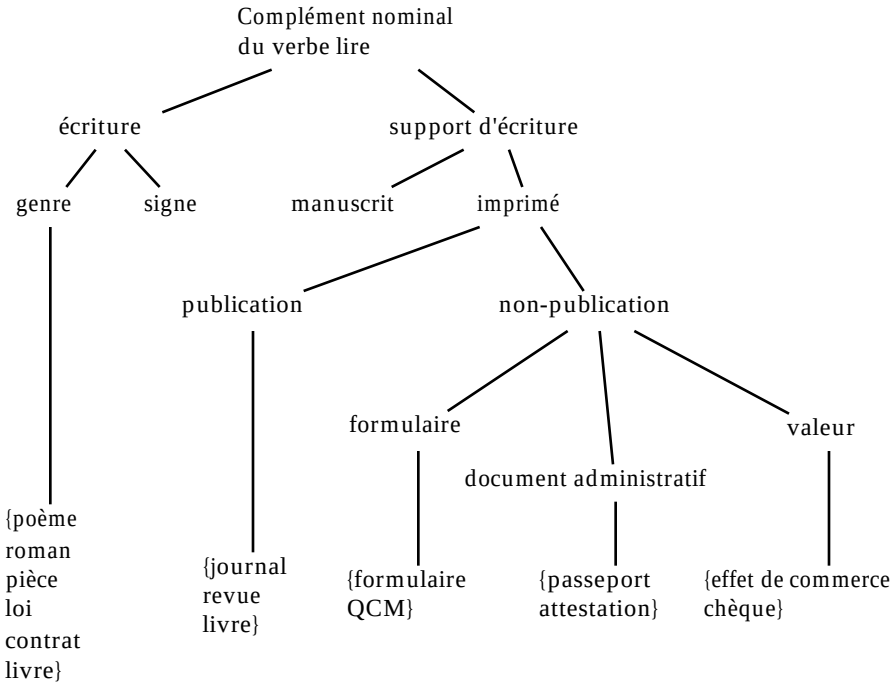


Figure 2.2 : Classement des compléments nominaux du verbe lire.

Ce type d'approche pose des problèmes cruciaux pour la représentation des connaissances des faits de langue. On décrit un certain état de la langue, à un moment donné. Et le classement produit vise à stabiliser une somme de savoirs en construisant une hiérarchie qui subdivisionne à l'infini les savoirs classés pour prendre en compte leur extrême finesse. Ce classement doit aussi traduire le fait qu'un mot peut appartenir à plusieurs classes :

livre : `<support de l'écriture> <texte>`

Le point important à souligner est qu'on aboutit à une représentation hiérarchisée qui tend à "clore" ou à fermer le domaine représenté (aspect fini de la représentation) : on fixe le nombre de classes, et il est difficile de revenir sur cet acquis. Ce type de représentation reste donc en deçà des possibles de réalisation d'une entité donnée. Puisque les faits de langue sont soumis à des évolutions permanentes, leur représentation se trouve en permanence remise en question. Il faut pouvoir les ajuster en tenant compte des incidences potentielles que les conditions de leur réalisation révèleront. On peut penser à une mise à jour de la classification par spécialisation. Mais il est souvent insuffisant de créer des sous-types pour traduire les extensions de sens potentielles sur les mots. A contrario, créer des sur-types peut conduire à des conflits d'héritage. Mais surtout, il reste délicat voire impossible de changer dynamiquement un graphe d'héritage construit. Surtout quand on sait que généralement, les représentations des savoirs linguistiques peuvent construire des hiérarchies de savoirs parallèles qui articulent les différents types de savoirs à l'oeuvre dans le langage naturel.

De plus un examen de réalisations textuelles révèle le plus souvent que les comportements des mots sont soumis à des variations quand on passe d'un corpus à un autre.

"Two findings repeatedly come out of corpus-based studies : first, individual linguistic features are distributed differently across registers, and second, the same (or simular) linguistic features can have different functions in different register" (Biber 1993).

En effet, le travail sur les sous-langages révèle que les classements généraux que l'on peut établir sur la langue ne sont pas toujours pertinents. L'utilisation des mots dans des domaines de spécialités particuliers peut induire des comportements linguistiques nouveaux qui mettent justement en lumière un nouveau type de rapport avec l'objet visé dans ce domaine particulier (Cadiot & Nemo 1997a).

"Corpus-based research shows that our intuitions[49] *about lexicals patterns for grammatical structures, for many words there is no general pattern of use that holds across the whole language; rather, different word senses and collocational patterns are strongly preferred in different registers"* (Biber 1993).

Si notre travail s'éloigne sur ce point de l'approche suivie par Gross, il en retient la reprise de l'approche harissienne (Daladier 1990, Dachelet 1994, Sager & al. 1987, Habert & al. 1996) : en fait la détermination de classes d'opérateurs et d'opérandes par le fonctionnement linguistique. Il ne s'agit donc pas d'une sémantique ontologique i.e. ayant des existants hors de toute réalisation linguistique.

2.3.2.4. Types et distributions (Godard & Jayez 1995)

(Godard & Jayez 1995) propose le multitypage des entrées lexicales (livre : `objet matériel & objet informationnel`) pour prendre en compte les différentes réalisations sémantiques d'une entrée lexicale. Ce typage (structure de type simple ou complexe) est réalisé dans une approche linguistique avec le dessein d'obtenir des classes en nombre limité (à assez gros grain).

---

49 *"The collocational patterns (confirmed by concordance listings) identify two surprising fact about (these) words : 1) 'certain' is commonly used to mark uncertainty rather than certainty; and, 2) certainty is rarely expressed in social science at all. Thus, the most common collocations for 'certain' in social science reflect a kind of vagueness, marking a referent as possibly being known (by someone) but not specified in the text (e.g., a certain kind of..., in certain cases..., there certain indications that...). These collocations are relatively rare in fiction. In contrast, those collocational patterns for 'certain' that directly state that someone or something is certain - he/she/they/you/it + BE + certain and i/we + BE + certain - are extremely rare in social science but relatively common in fiction"* (Biber 1993).

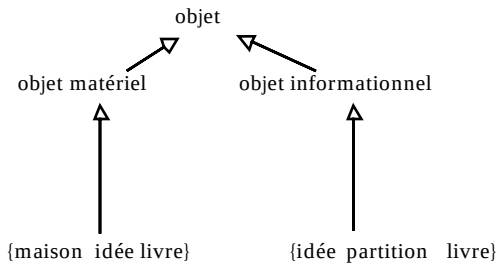


Figure 2.3 : Multitypage des entrées lexicales.

Là encore, on retrouvre le problème d'une approche de représentation qui tend à figer les savoirs et qui limite leurs évolutions potentielles. Si le type d'un mot préexiste aux occurrences de ce mot dans des textes, on limite l'interprétation de ces occurrences au types prédéfinis.

2.3.3. Polysémie et changements de sens

2.3.3.1. Changements réguliers de sens (CRS) et hiérarchies

Multitypage et choix en contexte : (Godard & Jayez 1995)

Le multitypage permet un regard en avant sur des possibles que les énoncés en situation doivent ajuster comme dans l'exemple suivant dans lequel le mot `livre` est associé à deux types différents :

Paul écrit un livre qui pèse déjà un kilo et qui fera du bruit.

livre : `objet matériel & objet informationnel`

On est donc confronté ici à une difficulté liée à la mise en place de hiérarchies rigides qu'il est souvent difficile de faire évoluer dans la mesure où le multitypage s'accommode d'un graphe des types figés.

Les règles lexicales (Briscoe & Corperstake 1993)

Les règles lexicales peuvent être vues comme des processus de conversion : elles sont utilisées pour créer les sens dérivés. Elles permettent ainsi de générer automatiquement des alternatives possibles de sens lorsque la résolution de contraintes l'exige. L'utilisation de règles lexicales repose donc de manière différente le problème de la fermeture de la représentation du domaine étudié. Si une approche hiérarchique tend à "clore" la représentation du domaine et à limiter les évolutions potentielles des savoirs ainsi décrits, les règles lexicales permettent d'agir sur ce type de représentation statique en modifiant temporairement les valeurs attachées à tel noeud de la hiérarchie (quand cela est nécessaire). Ce type de clôture reste malgré tout plus proche d'un artifice calculatoire. Si la volonté initiale de ne pas lister tous les sens d'un mot se traduit par la mise en place d'une liste de règles lexicales (que l'on peut étendre à l'infini), le modèle de représentation ne traduit pas adéquatement ce qui est réellement en jeu.

Coercion de type (Pustejovsky 1991)

Dans l'énoncé qui suit, la coercion permet au prédicat de changer le type de son argument vers celui qui est attendu par le prédicat:

`"Il a commencé le livre"`

commencer attend un événement, le prédicat ajuste donc le type de livre en fonction des rôles associés à l'argument.

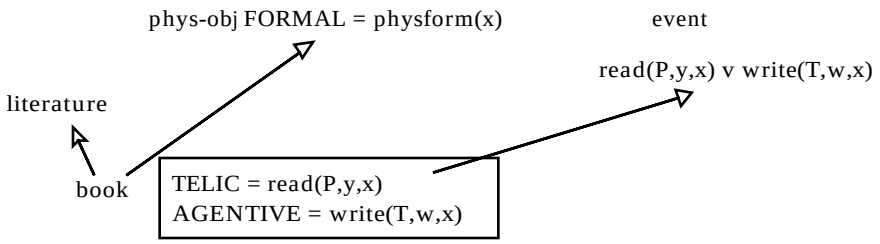


Figure 2.4 : Coercion de type sur livre.

On peut aussi penser à la métonymie intégrée de Kleiber. Et là encore, on est confronté à une difficulté liée à la mise en place de hiérarchies rigides qui s'accommodent d'un graphe des types figé.

2.3.3.2. Contraintes

Continuité du passage d'un type à un autre

a. Un noyau de sens pour conduire l'interprétation

1. Du discret au continu

La problématique développée par C. Fuchs met en avant la difficulté à définir les modes d'application des mécanismes qui réalisent la construction du sens d'un énoncé. Comment attribuer des valeurs aux unités linguistiques à représenter, et quelles valeurs retenir? Si l'on fige la représentation sémantique d'une unité de langue, celle-ci peut être modifiée intégralement dans une interprétation donnée. Sachant que l'interprétation peut conduire à une utilisation de valeurs sémantiques éloignées des valeurs habituellement attachées aux entités manipulées. Il convient donc d'exprimer la notion de variation graduelle, c'est-à-dire *"se représenter la signification en termes non plus en termes de valeurs discrètes et disjointes mais en termes d'un continuum de valeurs"* (Fuchs & Le Goffic 1985). La formalisation mise en place pour représenter ce continuum de valeurs se situe dans le cadre d'une modélisation utilisant les mathématiques du continu.

1.1. Modèle mathématique de la polysémie

A toute expression polysémique sont associés deux espaces :

- **un espace des indices co-textuels** : il *"permet de représenter les données susceptibles par leur présence dans l'énoncé d'influer sur l'interprétation de l'expression étudiée"* (Fuchs & Victorri 1990),

- **un espace sémantique** : dans celui-ci, *"les diverses significations de l'expression seront représentées comme des régions plus ou moins vastes"* (Fuchs & Victorri 1990).

De plus, *"le processus d'interprétation est modélisé par une fonction qui associe à chaque point de l'espace des indices une dynamique sur l'espace sémantique"* (Fuchs & Victorri 1990) : à chaque énoncé correspond un relief sur l'espace sémantique et *"les points de basse altitude sur ce relief correspond(e)nt aux significations de l'expression dans l'énoncé considéré"* (Fuchs & Victorri 1990). Dans ce modèle, on associe à chaque énoncé une dynamique sur l'espace sémantique, et la signification de l'expression dans un contexte ne se réduit pas à un ou plusieurs points, mais est matérialisée par des régions (les bassins attracteurs de la dynamique). *"L'espace sémantique associé à une expression polysémique est un espace géométrique continu, structuré de telle manière que des significations proches soient représentées par des régions voisines"* (Fuchs & Victorri 1990).

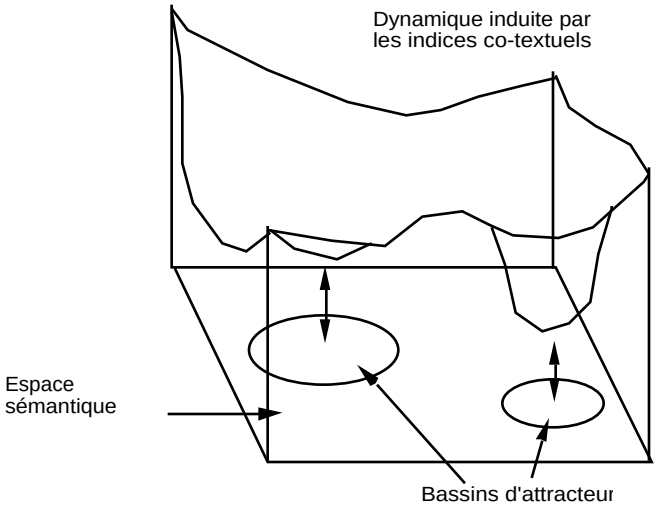


Figure 2.5 : Dynamique associée à un énoncé.

Une ambiguïté est alors représentée par l'existence de deux points d'équilibre stable dont les bassins d'attraction ont des formes traduisant le type d'ambiguïté (alternative, ambivalence, indétermination, etc.).

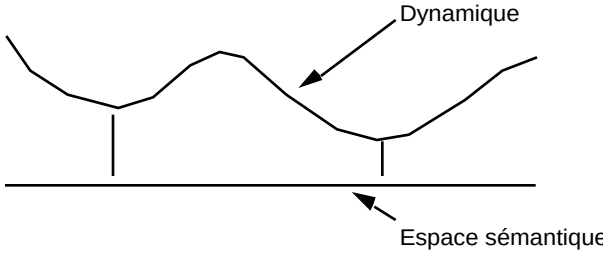


Figure 2.6 : Interprétation avec ambiguïté-alternative.

Un bassin unique correspond à un interprétation univoque :

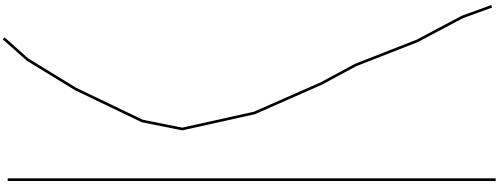


Figure 2.7 : Interprétation univoque.

Un bassin très large et peu profond traduit une signification peu déterminée :

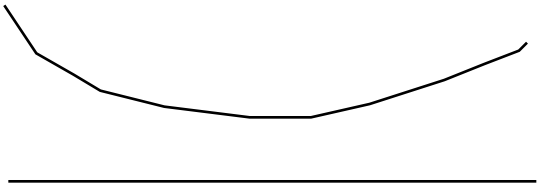


Figure 2.8 : Interprétation avec indétermination.

Ce modèle, implémenté à l'aide de réseaux connexionnistes qui fonctionnent par apprentissage, permet le traitement du sémantisme de certaines unités grammaticales. Le point fondamental dans cette étude est la construction de l'espace associé à l'expression étudiée : *"loin de ne constituer qu'un codage arbitraire et sans conséquence, cette construction implique de définir une topologie sur ces espaces qui va conditionner les capacités de discrimination et de généralisation du système"* (Fuchs & Victorri 1990).

1.2. Hypothèse méthodologique pour le classement des significations

Pour une expression polysémique, *"on postule l'existence de significations discriminales, et à propos desquelles il est possible d'émettre des jugements de proximité"* (Fuchs & Victorri 1990). Aussi, il s'agit d'établir *"des jugements qualitatifs relatifs assez spontanément mis en oeuvre dans la pratique langagière quand on cherche par exemple à expliciter le sens précis d'une expression de la langue dans un contexte quelque peu inhabituel"* (Fuchs & Victorri 1990).

1.3. Accès aux valeurs sémantiques

Pour cela, on utilise des jugements de recoupement du type :

"La signification de l'expression dans l'énoncé E_2 recoupe la signification que l'on observe pour cette expression dans l'énoncé E_1" (Fuchs & Victorri 1990),

et la variante plus forte :

"La signification de l'expression dans l'énoncé E_2 inclut la signification que l'on observe pour cette expression dans l'énoncé E_1" (Fuchs & Victorri 1990).

Ce type de jugement implique *"que toute signification possède une certaine extensionnalité dans l'espace sémantique : les significations les plus indéterminées seront celles de plus grande extension, et inversement les plus précises correspondront à de très petites régions de l'espace sémantique"* (Fuchs & Victorri 1990). La notion de valeur peut être définie comme étant *"une signification d'extension minimale (c'est-à-dire telle que toute signification qui la recoupe la contienne entièrement)"* (Fuchs & Victorri 1990). A la différence des approches componentielles, ce modèle pose *"l'existence de ce niveau de valeurs, qui ne sont pas des éléments de sens (sèmes, sémèmes, traits, etc.), mais des sens purs, d'extension minimale ou plutôt d'extension nulle, obtenus en réduisant progressivement l'extension des sens attestés jusqu'à zéro par passage à la limite"* (Fuchs & Victorri 1990).

2. Le noyau de sens

Le noyau de sens *"ne serait pas à proprement parler une valeur, mais une description minimale valable"* (Fuchs & Victorri 1990) pour l'entrée lexicale considérée, *"à partir de laquelle on pourrait définir des dimensions canoniques qui caractérisent chacune des valeurs"* . Le noyau de sens doit permettre *"de choisir un bon repère de l'espace sémantique associée à l'expression"* (Fuchs & Victorri 1990). Il est "*une abstraction sous-déterminée, que l'on peut se représenter comme un opérateur abstrait : ce qui est défini en effet c'est l'opération, unique, qui doit s'effectuer chaque fois que l'on se donne des instanciations concrètes"* (Fuchs & Victorri 1990) des paramètres décrivant ce noyau de sens. A chaque expression polysémique, par exemple, est associé un noyau de sens sous-déterminé et des directions dans lesquelles peuvent se déployer des significations plus déterminées en contexte. Si l'analyse n'exige pas de précisions, on en reste au noyau de sens. Dans le cas contraire, *"le calcul de significations plus déterminées fait appel à des règles qui selon la complexité de l'expression, sont soit des règles symboliques, soit des règles implicites apprises par un réseau connexionniste à partir d'exemples"* (Fuchs & Victorri 1990).

3. Définir un noyau de sens comme un opérateur abstrait auquel on peut faire correspondre une opération cognitive

Le noyau de sens ouvre la voie à une représentation cognitive : il peut être vu comme *"un schéma conceptuel dont les ingrédients (un domaine, le parcours d'une trajectoire dans ce domaine, une proposition à valeurs de vérités définie sur ce domaine) peuvent se retrouver dans de nombreuses opérations de pensée"* (Fuchs & Victorri 1990).

b. Interpréter, c'est opérer une confrontation frontale de tous les indices sémantiques présents dans un énoncé

En plus de la construction d'un espace sémantique (décrit par le noyau de sens), il faut définir l'espace des indices co-textuels qui influent sur l'interprétation. Puis évaluer les relations entre les présences/absences de ces indices sur le cas de figure interprétatif visé.

- De la tension interaction vs calcul ordonné (hiérarchisé)

"Ce qui est en question, c'est la possibilité d'un calcul hiérarchiquement organisé, où la valeur sémantique de chaque composant ne dépend que de la valeur de ceux qui, du fait de leur position dans l'arbre, ont été calculés avant lui. L'ordre de prise en compte des constituants dépend des formes présentes dans l'énoncé et doit donc être calculé d'abord lui-même avant d'être utilisé. Cela enlève déjà en grande partie l'intérêt de mener un calcul hiérarchique. Et ce n'est pas tout. Il nous faut prendre en compte un autre phénomène qui rend encore plus problématique l'application d'un tel calcul : les indices présents dans l'énoncé interviennent avec une certaine "force" en faveur de telle ou telle valeur et ces forces peuvent se combiner de différentes manières. En particulier, plusieurs indices faibles peuvent concourir à imposer une valeur, malgré la présence d'un indice "fort" en faveur de la valeur opposée" (Fuchs & al. 1991).

C'est dans son processus de création, inscrit dans l'espace et le temps de son énonciation, que le sens d'un énoncé se dévoile : il y a obligation de réaliser un calcul dynamique du sens en contexte.

"On est alors conduit à une vision plus globale du problème : tous les indices contribuent à construire la signification de l'énoncé pris comme un tout, et c'est en fonction de ce tout ce que l'on peut assigner à chaque constituant une valeur sémantique. Le calcul des valeurs des différentes caractéristiques sémantiques doit donc se mener "de front", à partir de toutes les formes présentes dans l'énoncé. /.../. La représentation du calcul sémantique doit prendre en compte :

- *les relations exprimant l'influence des formes présentes dans l'énoncé sur la valeur des différentes caractéristiques sémantiques;*

- *les contraintes (plus ou moins fortes) entre les caractéristiques sémantiques"* (Fuchs & al 1991).

Modèle discret vs. continu : notre problème est ailleurs

La notion de continuité pour décrire les processus sémantiques n'est pas une nécessité absolue : (Kayser 1994) montre qu'il est possible avec des modèles discrets de représenter une continuité conceptuelle. Mais l'utilisation de la continuité pour la modélisation des processus sémantiques (Fuchs & Victorri 1994) met surtout l'accent sur les difficultés rencontrées pour mettre en oeuvre (dans un dispositif) les phénomènes linguistiques ainsi décrits. En effet le point important soulevé par ce type d'approche est clairement énoncé dans (Victorri 1994) : *"/./ representing continuity on a machine is all but a simple problem"*, et cette question ne se laisse pas résoudre facilement. La mise en oeuvre d'un dispositif informatique pour représenter ce type de phénomène rejoint d'ailleurs un point important de notre travail : la notion de contrôle et la méta-représentation (-> partie 3 chapitre 9). Si le connexionnisme par exemple est capable de rendre compte des phénomènes de la continuité, *"it has a drawback that prevents it from playing in continuous modeling the same role as classical IA tools play in discrete modeling. This flaw is related to an important notion developped in IA : the notion of control. A connectionist network remains 'a black box' which does not allow much reasoning about its functioning"* (Victorri 1994). Or, notre travail vise aussi à

mettre en avant la nécessité de disposer de processus de contrôle pour le TALN. Si les descriptions / processus linguistiques à représenter (de manière discrète ou continue) sont potentiellement évolutifs, il est impératif de pouvoir évaluer les évolutions réalisées et donc de disposer d'outils de contrôle.

2.3.3.3. Statut des CRS

Les extensions de sens, dans quelle mesure relèvent-elles :

- d'une hiérarchie de concepts (Kayser 1992), (Pustejovsky 1991),
- de mécanismes pragmatiques très généraux (Kleiber & al. 1991).

Les entrées lexicales inversement se réduisent-elles à l'addition d'informations héritées d'une hiérarchie conceptuelle ?

Mécanismes pragmatiques très généraux (Kleiber & al. 1991)

- Le principe de métonymie intégrée

Certaines caractéristiques de certaines parties peuvent caractériser le tout.

"Paul a le dos bronzé"

caractérise le tout (Paul) au moyen d'une propriété assignée à l'une de ses parties.

Ce principe permet ainsi d'assimiler la commercialisation à une "partie du livre" dans la mesure où on peut considérer d'une part que la commercialisation fait partie du concept "livre" : un livre est un objet destiné par définition à la commercialisation. D'autre part, on peut dire qu'un livre est destiné à être lu par le plus grand nombre possible de lecteurs : là aussi, la commercialisation et la publication constituent à cet effet des facteurs importants. Le principe de métonymie exige en fait plus. Il ne suffit pas que la partie soit jugée suffisamment importante pour le tout, il faut encore que la caractéristique attribuée à cette partie soit caractéristique du tout. S'il n'en va pas ainsi, l'attribution au référent entier risque d'être bancale. Ainsi le prédicat "être un fiasco pour son éditeur" doit-il être pertinent non seulement pour la commercialisation, mais aussi pour le livre : il faut qu'il dise quelque chose en plus sur le livre en question. En disant qu'un livre a connu un fiasco éditorial, on n'indique pas seulement que la commercialisation ou publication du livre a échoué, mais on révèle en même temps indirectement quelque chose sur le livre lui-même : l'échec commercial est en même temps l'échec du livre en tant que livre.

Liés à une hiérarchie de concepts (Kayser 1992), (Pustejovsky 1991)

La profondeur variable de Kayser, comme le lexique génératif de Pustejovsky, suppose une hiérarchie de concepts aux noeuds de laquelle sont attachées des connaissances, des rôles qui permettent des inférences. L'interprétation de moulin à paroles et des expressions similaires peut relever des mécanismes d'extension de sens à la Kayser, ou encore des coercions de type à la Pustejovsky. On a même intérêt à l'expliquer ainsi, dans la mesure où il faut rendre compte du sentiment qu'il s'agit d'un sens "figuré". Par exemple pour moulin, on peut associer le rôle partie-de à objet-concret, énergie à instrument et entrée/sortie à instrument de transformation. Cela structure ainsi le savoir sur le monde effectivement mis en oeuvre. Cette hiérarchie est le support de l'inférence. Le savoir sur le monde est structuré, et cette organisation guide le travail inférentiel (d'où l'appellation paradoxale de Pustejovsky "héritage projectif", où la création dynamique s'appuie sur une structure statique). Pour suivre Kayser, on aboutit à des représentations sémantiques très abstraites, très peu déterminées (le prédicat pour moulin), qui sont précisées si besoin est. Et qui sont très éloignées des référents de même nom. Les idées de coercion, ou de dérivation de sens, invalident pour partie le filtrage "statique", aussi fin soit-il, puisqu'aussi bien

l'élément en position d'argument précisément ne présente pas les traits nécessaires et que ceux-ci doivent lui être rajoutés, au besoin en supprimant des traits présents et contradictoires (`moulin` devient `+humain` dans `moulin à sottises`).

2.3.4. Evolution des classifications

2.3.4.1. Hiérarchies rigides

S'il apparaît très clairement qu'un choix de représentation qui manipule la notion d'objet dans une taxonomie de classes est adéquate pour fixer un domaine de savoir établi dans une structure hiérarchisée[50], il s'avère parfois délicat de construire puis de prendre en compte les évolutions de telles représentations hiérarchiques quand celles-ci décrivent des domaines de savoir non stabilisés. Dans une telle approche, la mise en contexte des instances de classe peut révéler la nécessité d'une mise à jour de la structure initialement adoptée pour la définition des classes. Dans le cas de la construction d'une hiérarchie lexicale dans laquelle on souhaite inscrire dans la définition des entrées lexicales une articulation forme-sens, une telle représentation doit affronter le problème complexe de la confrontation d'un savoir établi et d'un savoir à venir (non pris en compte par le précédent). La prise en compte de nouvelles informations, de résultats issus de l'analyse, de configurations énonciatives particulières..., peut conduire à modifier, à affiner la représentation initiale. Dans le cadre de notre travail sur la modélisation des configurations nominales du type `moulin à N2`, une telle évolution sur le savoir initial est à l'oeuvre sur, par exemple, des formes du type `moulin à paroles` pour laquelle `moulin` devient `+humain`. Et c'est justement la travail de mise à jour d'une hiérarchie qui pose problème. Plus on structure un domaine de savoir, plus les liens de dépendance entre les noeuds du graphe d'héritage se complexifient, et il devient ainsi délicat de modifier cette structure hiérarchique.

2.3.4.2. Changer la hiérarchie

Vouloir classer les choses de manière figée pose donc des problèmes diffilement compatibles avec une nécessaire évolution dans les représentations construites des savoirs linguistiques à un moment donné. On sait bien que les classifications évoluent : la connaissance du monde bouge, le classement des mots change, un même domaine est aussi sujet à des classifications distinctes. Aussi, on pourrait vouloir changer les hiérarchies construites en spécialisant (créer des sous-types) ou en généralisant (créer des sur-types : le livre électronique comme objet matériel informatique (avec fichier, programme...), mais quel sur-type doit on considérer pour `"un livre est un livre"` ?). Il reste délicat voire impossible de changer dynamiquement un graphe d'héritage construit (comment remplacer (dynamiquement) un graphe de types par un autre?). Surtout quand on sait que généralement, les représentations des savoirs linguistiques peuvent construire des hiérarchies de savoirs parallèles qui articulent les différents types de savoirs à l'oeuvre dans le langage naturel.

---

[50] Le multitypage ou la coercion de type ou les règles lexicales s'accommodent d'un graphe des types figé.

2.4. Conclusion

Notre travail s'inscrit fondamentalement dans la lignée des approches présentées supra, même si nous nous sommes montré critique sur certains aspects de ces modèles. Ces approches visent toutes à concevoir la construction du sens comme la mise en action de mécanismes. Ces approches s'écartent d'un appariement de termes et de sens. Elles ne consistent pas non plus à recourir uniquement à de l'induction ou à du raisonnement. Il s'agit plutôt d'établir un compromis entre des informations figées et des heuristiques qui peuvent enrichir ces informations. Ce qui est en jeu, c'est la confrontation entre la représentation de savoirs minimaux et les mécanismes qui permettent d'étendre ces savoirs en fonction des réalités contextuelles. Il s'agit pour certains d'organiser les savoirs de manière hiérarchique de telle sorte que des processus attachés à ce type de représentation permettent de reconstruire le sens; pour les autres, il s'agit au contraire de décrire plus précisément les mécanismes qui réalisent la construction du sens. Le problème fondamental pointé ici tourne donc autour de la difficulté à représenter les savoirs à l'oeuvre dans les descriptions linguistiques dans la mesure où ces derniers ne sont pas donnés une fois pour toutes.

2.4.1. Représenter, c'est figer un peu.

- Comment représenter la base de savoirs initiaux pour tenir compte du fait que l'on a affaire à des informations évolutives et non figées ? Comment enrichir ces savoirs établis quand de nouveaux types de descriptions linguistiques révèlent une certaine stabilité ?

Si les descriptions linguistiques se bornent à refléter un savoir partagé sur le monde, en particulier si le lexique est organisé (de manière hiérarchique ou non) sur la base de connaissances (le monde objectif), on perd une grande partie des mécanismes à l'oeuvre dans le langage naturel. L'interprétation permet en effet la mise en oeuvre de processus sémantiques qui créent de nouvelles acceptations et de nouveaux emplois non dérivables de la connaissance encyclopédique. Dans ce cas les parcours (arborescents ou non) n'apprennent rien que l'on ne sache déjà (Rastier 1995). Figer les savoirs linguistiques dans des taxonomies figées et non évolutives ne convient guère à la structure sémantique des langues et assez mal à celle des discours de spécialité (Biber 1993) : il n'est pas certain que les différents niveaux de l'arbre appartiennent à une même pratique (Rastier 1995). Un travail de représentation des faits de langue doit donc s'attacher à rendre les savoirs de base évolutifs en sachant qu'un ajustement est toujours potentiellement à venir sur ces savoirs.

2.4.2. Ajuster les processus d'ajustements : approches de la réflexivité

L'utilisation d'heuristiques pour étendre les savoirs initiaux représentés dans des configurations particulières pose quant à elle au moins deux types de problèmes.

- Comment représenter le fonctionnement des heuristiques pour rendre compte à la fois de leurs évolutions potentielles et de leurs applications sélectives (éviter de surgénérer) ?

Un dispositif pour le TALN est contraint d'enrichir ou affiner les processus inductifs mis en place (il en va de même pour les savoirs représentés). Dans la mesure où il est impossible de lister et de prévoir tous les scénarios possibles à attacher aux mots, il est nécessaire de pouvoir ajuster les processus définis au cas où telle heuristique doit justement être remodelée. On est donc confronté au problème fondamental qui consiste à travailler sur un nombre important de connaissances interconnectées, toujours en mouvement, remodelées par des processus évolutifs et capables de s'enrichir.

Chapitre 3

Modèle de classification pour le Langage Naturel

3.1. Classification

"La connaissance - qu'elle s'exerce sur les données immédiates de la perception ou sur le donné élaboré par la science, qu'elle s'applique à réaliser industriellement ses schémas ou à construire les êtres mathématiques - classe nécessairement ses différents actes, mieux, la classification est sa vie même."
G. Bachelard, Essai sur la connaissance approchée.

Le terme de classification utilisée en IA doit être pris au sens de classement : étant donné un objet, à quelle classe d'objets appartient-il ? Dans les domaines de savoir où il existe une taxonomie établie, le raisonnement par classification est adéquat pour organiser, rechercher ou maintenir les connaissances ainsi structurées. Dans les domaines de savoir où il n'existe pas encore de structuration établie des données, la catégorisation est l'opération qui consiste à construire une hiérarchie dans le domaine en question en classifiant les données une à une.

3.1.1. Les systèmes à héritage

1. Système à héritage et représentation des connaissances :

Un système à héritage permet de représenter des connaissances sous la forme d'une hiérarchie d'objets.

2. Représentation et objet :

Un objet est défini par une collection de propriétés structurelles et procédurales qui décrivent son état et les opérations qu'il est capable d'exécuter.

3. Objets et héritage :

Les objets sont organisés par niveau d'abstraction et ils partagent des propriétés par l'intermédiaire du mécanisme d'héritage : un objet hérite des propriétés des objets qui sont plus généraux que lui tandis que ses propriétés sont héritées par les objets qui sont plus spécifiques que lui (Haton & al 1991).

3.1.1.1. Les langages de classes

Les langages de classes permettent de modéliser un domaine à l'aide d'une hiérarchie de classes (-> présentation de la PàO). Une classe est une structure décrivant un ensemble de propriétés que l'on associe à des instances de cette classe. Une instance de classe est une entité élémentaire représentant un élément potentiel de sa classe d'appartenance. L'utilisation de classes implique la mise en place d'un mécanisme descriptif centralisateur : une instance se réfère à sa classe pour déterminer son comportement. La classe est en effet un moule à partir duquel sont créées les instances. Un graphe d'héritage se construit par spécialisation des classes existantes. Une classe dérivant d'une classe hérite des variables et des méthodes définies pour sa superclasse. De plus, les propriétés définies sur une classe s'ajoutent à celles définies pour ses superclasses. Les langages de classes se déterminent donc par deux types de relation :

- La relation d'instanciation qui indique l'appartenance d'un objet à une classe donnée.
- La relation de spécialisation qui permet de redéfinir ou d'ajouter certaines propriétés au sein d'une classe dérivée d'une classe donnée.

3.1.1.2. Les langages à prototypes

A la différence des langages de classes, les langages à prototypes utilisent une autre approche pour modéliser un domaine, ou un concept donné. Ainsi, un concept donné, décrivant une famille d'objets, peut être représenté par un exemplaire particulier de cette famille qui est porteur des propriétés les plus fréquemment associées aux membres de la famille d'objets considérés : cet objet particulier est un prototype. Le prototype est une manière de représenter les connaissances par défaut. Tout objet est une spécialisation d'un prototype donné et l'on créé les nouveaux objets par copie différentielle d'un prototype.

3.1.1.3. Les langages hybrides

La représentation de concepts avec des classes est appropriée lorsqu'il existe une abstraction qui permet de décrire un concept de façon complète et définitive. Elle se révèle moins pratique pour prendre en compte des connaissances typiques ou incomplètes. Certains langages objets dédiés à la représentation des connaissances ont évolué vers une famille de langages hybrides qui intègrent les caractéristiques des langages de classes et des langages à prototypes....

3.1.2. Le Filtrage

On parle de filtre pour désigner les opérations d'appariement, de reconnaissance de modèles, de correspondance de mots ou de coïncidence de formes... Le filtrage est l'opération élémentaire pour le raisonnement par classification. Il permet d'identifier ou de différencier des objets en mettant en correspondance un certain nombre de constantes et de variables avec des données. Cette opération permet donc de retrouver toutes les instances qui correspondent au modèle que constitue le filtre. Un langage de filtrage peut comprendre un certain nombre d'opérateurs pour définir des patrons, les combiner et éventuellement leur associer des actions. Un système de filtrage associe ce langage, et une procédure de filtrage qui aura pour rôle de mettre en relation le patron et la donnée. Le filtrage a été utilisé dès les premiers programmes cherchant à simuler une interaction homme-machine en langue naturelle. Il permet en effet d'extraire les éléments jugés informatifs dans les énoncés de l'utilisateur (Eliza (Weizenbaum 1966)). OLMES0 (Habert 1991) présente un système de filtrage adapté à la reconnaissance et à la détection des expressions figées.

3.1.3. La classification dans les systèmes à héritage

3.1.3.1. Les langages de classes

Classer une instance, i.e trouver sa classe d'appartenance, n'a pas de sens dans les langages à classes, puisqu'en effet la relation d'instanciation n'existe qu'après la définition de la classe. Le mécanisme de classification de classes étudie la possibilité de repositionner une classe donnée dans un graphe d'héritage, i.e il s'agit de déterminer les classes qui peuvent devenir des superclasses de cette classe et celles qui peuvent devenir des sous-classes. Cette opération repose sur les liens de sous typage qui peuvent exister entre différents types d'un même attribut.

3.1.3.2. Les langages à prototypes

Dans un tel cadre, un mécanisme de classification ne peut définir l'appartenance sur la base de propriétés nécessaires et suffisantes partagées par l'ensemble des membres de la famille d'objets issue d'un prototype donné. Il s'agit plutôt d'envisager une relation de similitude qui peut exister entre les objets.

3.1.3.3. Savoir harmoniser certaines complémentarités des formalismes sous-jacents aux modèles de représentation utilisés

On pourrait envisager des systèmes travaillant sur les représentations des connaissances utilisant d'une part une approche fondée sur les taxonomies de classes dans les domaines de savoir représenté où il existe un figement de ce savoir et d'autre part utilisant une approche prototypique de la représentation des connaissances dans les domaines de savoir où la classification est floue ou mouvante. Une approche qui privilégie la prise en compte des irrégularités, des rapports contextuels de courte portée est capable de rendre compte puis de fournir des clés pour y voir plus clair. Il faut lui adjoindre un module qui "mette de l'ordre" là où il est possible de le faire. Celui-ci doit modéliser les régularités (et elles ne manquent pas dans la langue). On dépasse ainsi les représentations individuelles pour construire des représentations plus stabilisées. De même que pour classer une quantité d'éléments disparates, on peut procèder par un examen de tous les éléments, puis par une répartition dans différents sous-ensembles suivant les caractéristiques définitoires de chacun de ces éléments que l'on arrive à identifier, tout en mémorisant ces mécanismes qui nous font placer tel élément dans tel sous-ensemble. Pour y voir plus clair, il faut réorganiser le "chaos" initial et construire des ensembles structurés stables. Dans le cadre de l'analyse automatique, on pourrait ainsi disposer d'un module d'analyse qui travaille sur des domaines de représentation des connaissances pour l'analyse basée sur une hiérarchie de classes des éléments de l'analyse. Et d'un module d'analyse privilégiant une représentation prototypique, qui si les mécanismes de classification mis en place dans celui-ci permettent une stabilisation ou un figement des savoirs manipulés permettrait une adaptation des connaissances manipulées dans le module précédent.

3.2. Le TALN et la classification

Les représentations hiérarchisées des connaissances se sont imposées pour le TALN pour les raisons suivantes :

- 1. La place croissante donnée au lexique dans les formalismes d'unification, et la nécessité de disposer d'outils pour gérer les redondances.

- 2. La convergence des représentations du type de celles utilisées par les réseaux sémantiques et de la place croissante des langages à objets.

- 3. L'"écho" relativement naturel entre les types de hiérarchies manipulées en linguistique et celles de la PàO.

L'émergence d'approches de la catégorisation en termes de prototypes n'a pas débouché sur des projets descriptifs d'ampleur et la prédominance des approches fondées sur les taxonomies de classes reste de mise. Et cela peut s'expliquer de la manière suivante : les classements utilisés en TALN reposent pour la plupart sur des décennies voire des siècles de tradition. Les catégories et leurs subdivisions, si elles ne font pas vraiment toujours un accord unanime, convergent globalement. Le TALN s'est dans l'immédiat consacré aux segments de savoir sur lesquels il y a le plus de recouvrement : morphologie et syntaxe. Pour la sémantique, la version qui en est retenue est minimaliste. Certains secteurs d'analyse toutefois, où la catégorisation à opérer est mouvante, posent la question du rapport entre deux modes d'organisation des connaissances, un mode apriorique et un mode inductif.

3.2.1. De la connaissance et du TLN.

Le problème du TALN s'inscrit dans un domaine où il n'existe évidemment pas de modèles mathématiques aptes à résoudre les problèmes de génération ou compréhension du langage naturel. Ce sont les connaissances dont on dispose sur le langage naturel qui sont modélisées pour résoudre les problèmes qui se posent pour le TALN. Il s'agit de modéliser les connaissances que l'on a sur les langues pour résoudre ou aider à résoudre des problèmes de traitement automatique du LN. Dans le TALN, on utilise un nombre important de connaissances hétérogènes que l'on peut provisoirement dissocier ainsi (Pitrat 1991) :

- Les connaissances morphologiques qui expriment la formation des mots, leur décomposition en morphèmes. Ces connaissances qui regroupent les formes canoniques des mots, les racines et leurs terminaisons possibles ainsi que les règles qui régissent les associations, peuvent être stockées dans des bases de connaissances telles que les dictionnaires (les entrées d'un dictionnaire correspondent en gros à ce type de connaissance).
- Les connaissances syntaxiques expriment la façon dont les formes canoniques qui constituent une phrase s'organisent dans la phrase. Ces connaissances permettent donc de valider les phrases légales, c'est-à-dire soit de construire des phrases correctes si l'on travaille dans un cadre de génération automatique, soit de reconnaître une structure grammaticale valide si l'on travaille dans un cadre de compréhension du langage naturel.
- Les connaissances sémantiques construisent des représentation du sens à partir des significations que véhiculent les mots.
- Les connaissances pragmatiques expriment des notions de sens commun sur les contextes d'utilisation des mots. *"On peut définir les connaissances pragmatiques comme étant les connaissances auxquelles il est nécessaire de faire appel lorsque les connaissances syntaxiques et les connaissances sémantiques ne sont pas suffisantes pour produire une interprétation satisfaisante"* (Prince 1993).

Il faut ajouter à cela que les langues peuvent exprimer aussi :
- Des connaissances incomplètes qui, par exemple, permettent de supprimer l'acteur au passif, ou de supprimer les marqueurs temporels, locatifs...

- Les connaissances approximatives.
- Les connaissances incertaines.
- Les connaissances contradictoires.
- La nécessité d'exécuter une action.
- Les appréciations sur un texte étroitement mêlées au texte lui-même.

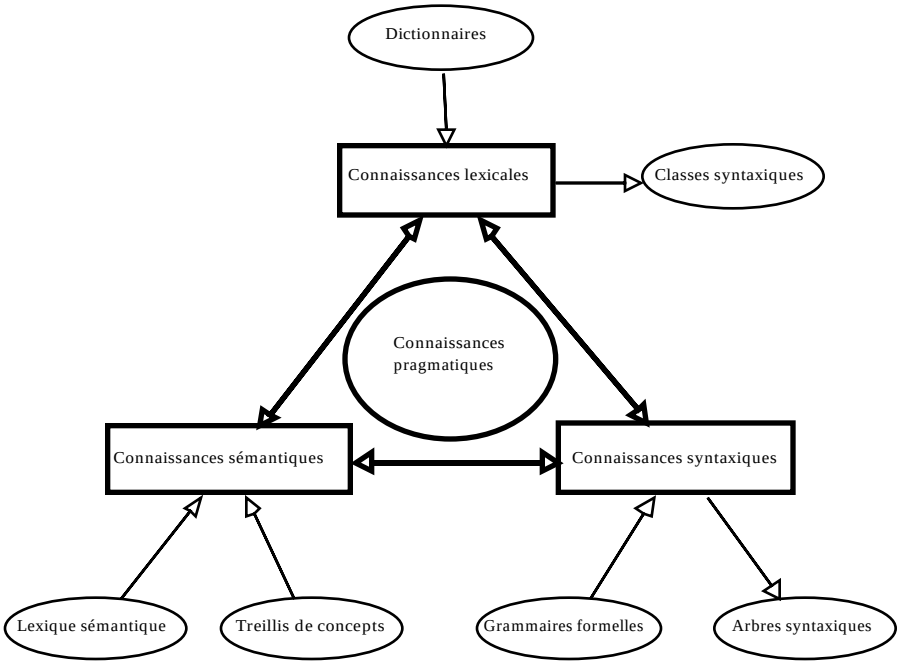


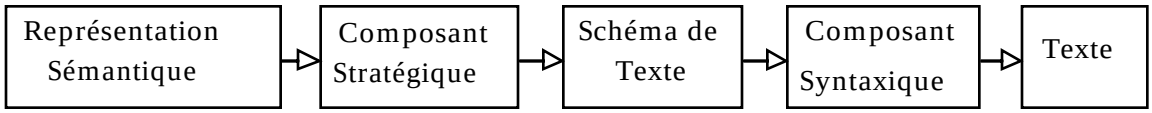
Figure 3.1 : Connaissances et TALN.

Les linguistes ont depuis longtemps construit des grammaires de la langue pour déterminer si un texte est correct ou non. Ces grammaires doivent théoriquement avoir l'avantage d'être indépendantes d'un domaine particulier, la même grammaire est valable quelque soit le domaine dont on parle. Or pour les êtres humains, il vaut souvent mieux connaître le domaine du texte que bien connaître la langue : les connaissances pragmatiques semblent être déterminantes pour comprendre un texte, plus importantes que les connaissances syntaxiques. Certains programmes dotés de connaissances assez complètes sur ce qui peut raisonnablement se produire dans un domaine restreint arrivent à des performances satisfaisantes bien qu'ils aient peu de connaissances syntaxiques (Frum & Dejong 1979).

3.2.1.1. Modularité ou intégration.

Un des problèmes du TALN est de savoir comment travailler avec ces types de connaissances. Faut-il utiliser ces connaissances indépendamment les unes des autres ou faut-ils tenter de les intégrer en un modèle de représentation plus autonome ? On rencontre là un problème crucial qui oppose en fonction des arguments linguistiques théoriques choisis une approche visant à une intégration des connaissances à une approche visant au contraire à faire coopérer des connaissances modulaires. Chacun des types de connaissances pré-cités est souvent représenté par un objet exprimé dans un formalisme particulier. L'approche qui vise à intégrer les connaissances cherche quant à elle à fondre des connaissances hétérogènes dans des modules de connaissances de différents niveaux. Il importe donc ici de savoir comment interagissent les connaissances ainsi regroupées. Le problème majeur de cette approche est qu'elle s'adapte mal aux domaines de savoirs sur lesquels on travaille à une mise au point des connaissances, ou dans des domaines dans lesquels il n'existe pas encore de modèles théoriques complets qui permettent d'évaluer les rapports hiérarchiques, ou autres, entre les différents modules de connaissances qui composent le domaine en question. En effet, dans un tel cadre, une modification est coûteuse puisqu'elle nécessite souvent une refonte totale de l'organisation interne du module de représentation choisi.

Le travail présenté par L. Danlos (Danlos 1985) en génération de textes en langage naturel précise qu'un algorithme cohérent de génération ne devrait pas être modularisé, pour permettre une interaction entre les décisions prises pour la génération. Ces décisions sont effet dépendantes les unes des autres : décisions sur le choix des phrases simples, sur l'ordre des informations, sur la segmentation des phrases, sur les contraintes syntaxiques... La difficulté de concevoir cette non-modularisation a conduit malgré tout à retenir un système de génération tel qu'il est présenté ci-dessous :



Le composant stratégique produit le schéma de texte en traitant les aspects de la sélection des structures de discours, des choix lexicaux pour les prédicats, du nombre de phrases, de la hiérarchie des constituants syntaxiques. Le schéma de texte peut être considéré comme la structure profonde du texte à produire. Le composant syntaxique qui produit le texte final travaille sur les aspects de la synthèse des étiquettes (prise en compte de la pronominalisation), puis opération de placement (hormis pour les particules pré-verbales dont l'ordre d'apparition conditionne leur synthèse), de la conjugaison des verbes, de l'application des règles morpho-syntaxiques. D'autres travaux en génération ne présentent pas ce choix d'une non-modularisation des éléments constitutifs du générateur. Dans le même sens, les travaux menés par (Sabah & Rady 1983) ont montré les limites d'une approche non modulaire dans le cadre d'un travail de compréhension (limité à l'analyse de phrases). Si ce programme construit des représentations qui intègrent les aspects morphologiques, syntaxiques et sémantiques, les auteurs ont constaté qu'il était difficile de constituer les connaissances nécessaires et de pouvoir ensuite les gérer efficacement afin d'étendre cette technique à tous les processus du TALN. En effet, il s'avère délicat d'ajouter de nouvelles connaissances pour traiter de nouveaux problèmes si on ne veut pas créer d'interférence avec le pré-existant, sachant que les connaissances ne sont plus indépendantes. La nécessité de modulariser les connaissances devient indispensable. Les connaissances de même nature sont regroupées modulairement et coopèrent en s'échangeant des informations. Un obstacle important existe dans ce type d'approche pour le TALN. Il réside dans le fait que les différents modules de connaissances liés à la langue sont interdépendants. Chaque module a besoin des autres modules pour contraindre son fonctionnement. Il est nécessaire d'apporter un élément supplémentaire. L'intégration des connaissances revient à coder les connaissances plutôt qu'à les représenter. L'intégration se situe davantage dans une approche informatique. A l'inverse, la coopération se situe dans une problématique typique de celle de l'intelligence artificielle. On peut d'ailleurs résumer ces deux approches ainsi (Bachimont 1992) :

Coopération = Représentation + Interprétation
Intégration = Représentation + Utilisation

Une des hypothèses fondamentales de l'IA est que les mécanismes intelligents utilisés pour la résolution d'un problème sont des mécanismes de transformation de l'information. Les processus de compréhension automatique consistent à effectuer des changements de représentation d'un langage formel dans un autre.

3.2.2. Manières de concevoir les connaissances.

*"Ce que nous appelons **connaissance** peut être ainsi définie comme prenant la forme de structures stabilisées dans la mémoire à long terme. Ce sont en vérité tous nos savoirs servant à la reconnaissance, à la compréhension des situations et à l'action. Ces savoirs vont se manifester comme **représentations**, c'est-à-dire des états provisoires de connaissance, résultant de nos activtés de construction de sens et d'interprétation des situations ou des évènements. Certaines de ces représentations vont être stockées en mémoire puisque pertinentes ou correspondant à des savoirs réutilisables; elles prendront de ce fait le statut de **connaissances**, et on parlera à ce propos d'acquisition ou de construction des connaissances."* (Vignaux 1991)

Nos connaissances sont structurées de telle sorte que nous avons accès directement à ce qui nous intéresse pour produire ou comprendre du langage naturel. Les systèmes manipulant le langage naurel doivent eux-aussi avoir les connaissances structurées de manière à effectuer une utilisation efficace de celles-ci.

Connaissance et Utilisation des connaissances

- Connaissances statiques ou connaissances évolutives (mises à jour à la main, en donnant des outils de mise à jour, s'automodifiant...).

- Connaissances stratifiées (hiérarchisées en pipe-line / interagissant) ou connaissances modulaires.

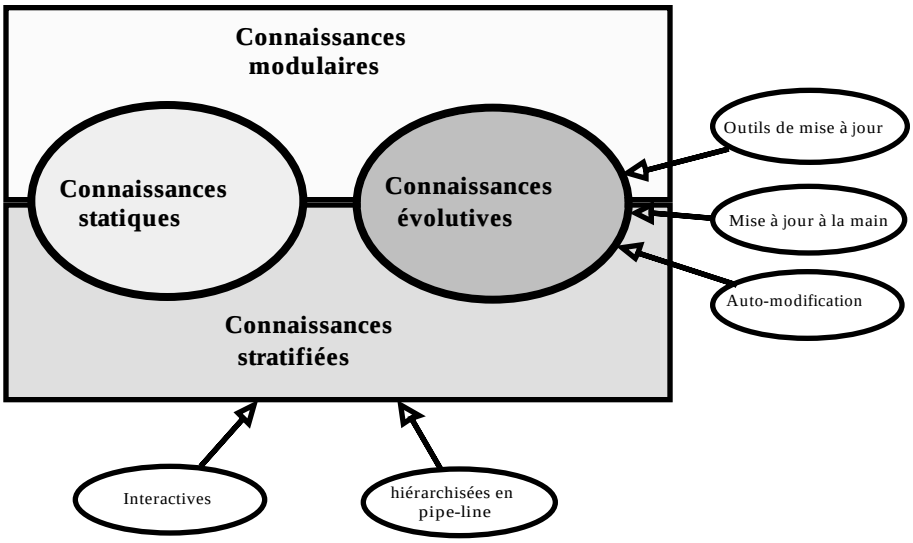


Figure 3.2 : Connaissance et utilisation des connaissances.

Les sources des connaissances dans un dispositif de TALN sont d'ordre très divers. Que les représentations utilisées soient procédurales ou déclaratives, les architectures des systèmes associés reflètent ces distinctions. De plus l'interaction entre les processus est un problème essentiel lors de la réalisation d'un dispositif complet de compréhension du langage. Une architecture simple est nécessaire pour permettre une collaboration efficace de ces processus. Le véritable problème est de construire des dispositifs qui permettent d'obtenir "rapidement" (en temps informatique) les informations aussi bien pour le déroulement des programmes mis en place pour le dispositif traitant de la solution du problème que pour les outils à mettre en place pour spécifier des questionnements (dépendants de l'utilisateur), c'est-à-dire pour réaliser une interface agréable pour l'utilisateur en fonction de ses besoins spécifiques. La difficulté majeure pour les systèmes traitant du langage naturel est de résoudre les problèmes liés à la manipulation d'une grande quantité de connaissances, évolutives en quantité et inter-agissantes de manière complexe. Il semble pour cela nécessaire de disposer de connaissances définies de manière déclarative c'est-à-dire en les séparant de leur mode d'emploi. De plus, si on met en place un système de représentation des connaissances sous forme déclarative, il est nécessaire de disposer parallèlement de connaissances capables de traiter les connaissances initiales (les modes d'emploi). On touche là au problème considérable que constitue l'utilisation de méta-connaissance, ce problème dépasse d'ailleurs largement le traitement du langage naturel.

3.2.3. Paradigmes récents en TLN : vers plus de déclarativité.

Les années 70 ont été dominées, pour l'analyse syntaxique automatique, par les ATN (Augmented Transition Networks) de Woods, et, pour la linguistique, par les grammaires transformationnelles. Les deux cadres de travail ont progressivement été remis en cause. L'un et l'autre prêtaient le flanc à une critique commune : ces formalismes sont trop puissants (ils ont la puissance des machines de Turing), ce qui rend problématiques

tant leur utilisation informatique que leur apport dans les structures produites des énoncés. En outre, l'écriture de grammaires un peu complexes avec des ATN produit des réseaux dont la compréhension et l'évolution deviennent difficiles. Le début des années 80 a vu la montée des grammaires d'unification, qui ont donné une place centrale au lexique, et par conséquent à des structures catégorielles complexes, et à l'unification sur ces structures. L'essor de ces formalismes est lié au moins à deux influences. En premier lieu, celle de la programmation déclarative, issue en particulier de Prolog : on entend désormais, chaque fois que c'est possible, définir un programme non par la suite des actions élémentaires qui le constituent, mais par les faits et relations qu'il doit utiliser pour "trouver" les solutions recherchées. Les formalismes grammaticaux ont alors tenté, avec des succès divers, de respecter cet idéal de déclarativité. En second lieu, un mouvement de retour vers les grammaires indépendantes du contexte en traitement du langage naturel. (Chomsky 1957) avait, semble-t-il, tranché le débat sur le type de grammaire nécessaire à la description du langage naturel : *"A context-free phrase structure grammar is not sufficient to describe or analyze the whole range of syntactic constructions"*. Depuis cette assertion, il était généralement tenu pour acquis que les grammaires hors contexte étaient inadéquates pour décrire des langues naturelles, à la fois en termes de capacité générative faible (en ce qui concerne le filtrage des phrases acceptables) qu'en termes de capacité générative forte (pour ce qui regarde les structures attribuées aux phrases). Les arguments mis en avant allaient de phénomènes d'accord (qui sont pourtant traitables par des grammaires hors contexte) - c'est le genre d'arguments que (Gazdar & Pullum 1987) appellent le "folklore" - à des dépendances non bornées, en passant par des constructions manifestant des enchevêtrements comme celles avec l'adverbe "respectivement" (Desclés 1973), (Gazdar & Pullum 1987). (Gazdar & Pullum 1987) ont tenté de réfuter ces arguments, et de montrer qu'à tout le moins, l'inadéquation des grammaires hors contexte pour les langues naturelles n'est pas vraiment démontrée. (Shieber 1987), pour sa part, examine une construction enchevêtrée en suisse allemand, dont la description requiert une grammaire légèrement sensible au contexte. Le renouveau dans les dix dernières années d'un débat que beaucoup croyaient tranché montre que la question reste largement à explorer. En outre, les phénomènes qui sembleraient demander des grammaires plus puissantes que les grammaires hors contexte semblent occuper une place somme toute marginale dans la langue, voire constituer des difficultés pour les locuteurs. Certains linguistes-informaticiens ((Gazdar 1985) par exemple) soutiennent alors qu'après tout, la majeure partie des constructions des langues naturelles peut être traitée dans ce cadre. Ils entendent limiter au strict nécessaire la puissance des formalismes utilisés. C'est en ce sens que les grammaires indexées, légèrement sensibles au contexte, ont suscité un certain intérêt.

Les formalismes informatiques proposés en analyse automatique ont progressé vers plus de déclarativité. Ils ne permettent pas pour autant à des linguistes de spécifier aisément et immédiatement leurs questionnements et leurs résultats. De plus les formalismes induisent des biais dans leur manière de représenter les connaissances lexicales et syntaxiques d'une langue. Les règles de réécriture même enrichies occupent souvent une place centrale. L'articulation du lexique et de la syntaxe est d'autant plus délicat. Et cela s'avère pas toujours adéquat pour traiter des langues non positionnelles. On retrouve ici l'idée de se servir de l'informatique pour vérifier ou utiliser une théorie linguistique. On examinera donc les formalismes, présentés infra (-> présentation de la PàO puis de la PàP) en examinant des deux point particuliers que sont leur pouvoir expressif et leur déclarativité. Le premier point n'ayant d'intérêt qu'au regard du suivant. La déclarativité d'un formalisme linguistique étant gênée par la déclarativité des informations à utiliser.

3.2.3.1. Les ATN

Les ATN (pour Augmented Transition Networks) sont des réseaux de transitions récursifs enrichis par la manipulation de registres et qui sont formellement équivalents à des machines de Turing. Les ATN permettent de reconnaître tout langage constitué d'un ensemble récursivement énumérables de phrases. Les règles des grammaires dans le cadre de tel langage sont de la forme :

a -> b , a et b désignant des catégories terminales ou non-terminales.

Un ATN contient un nombre illimité de registres afin de conserver toute l'information en mémoire et de permettre d'accèder à cette information dans tous les registres disponibles. En effet, puisque l'agencement des constituants présents dans la structure profonde ne correspond pas forcément à celui de la structure de

surface, il est souvent indispensable de disposer des informations disponibles tout au long de la construction d'une analyse syntaxique afin de modifier l'arbre de surface. Les ATN permettent donc grâce à cette mémoire illimitée que constituent les registres, de se référer à tout moment aux divers éléments construits pendant l'analyse. De plus, les arcs d'un ATN peuvent être étiquetés par des conditions et des actions. Une condition fait référence au contenu des différents registres et doit être satisfaite pour que l'arc qui la spécifie puisse être emprunté. Cette condition peut conduire à l'exécution d'une action modifiant le contenu d'un registre. L'illustration qui suit illustre l'utilisation d'un ATN pour la construction d'un groupe nominal :

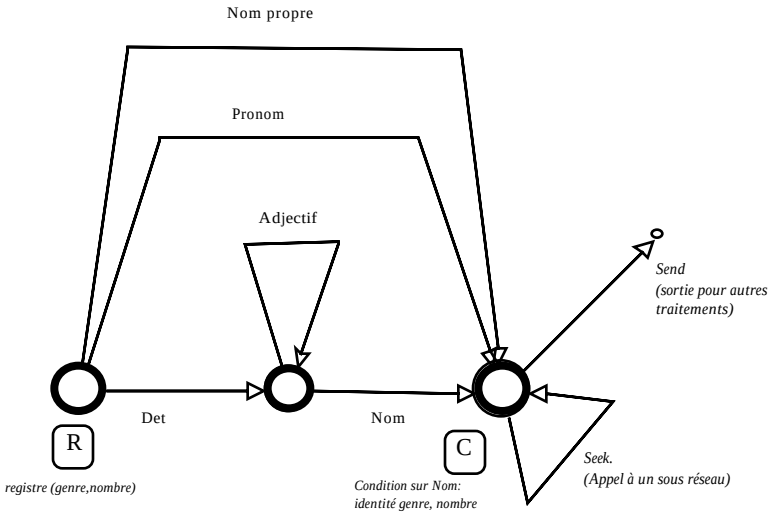


Figure 3.3 : ATN et contruction d'un groupe nominal.

Un ATN est un langage de programmation dont la fonction première est l'écriture d'analyseurs pour langue naturelle et est essentiellement un système procédural dans lequel grammaire et analyse sont inséparables. Il est important de noter ici que l'on assiste ici à un mélange savoir/contrôle. De plus, si la grammaire contient des règles récursives à gauche, l'analyse ne se termine pas : les ATN ont en effet une stratégie de contrôle descendante.

3.2.3.2. Les Grammaires d'unification

Limites des grammaires hors contexte

SN -> DET N

Cette règle accepte: *un règle

Figure 3.4 : Grammaire Hors Contexte : Une Règle trop générale.

SN -> SNsing l SNplur

SNsing -> DETsing Nsing

SNplur -> DETplur Nplur

Ces règles permettent: *un règle

Figure 3.5 : Grammaire Hors Contexte : rajout de traits de `nombre`.

SN	**->**	**SNsing l SNplur**
SNsing	**->**	**SNmascsing | SN fémsing**
SNmascsing	**->**	**DETmascsing Nmascsing**
SNfémsing	**->**	**DETfémsing Nfémsing**
SNplur	**->**	**SNmascplur | SN fémplur**
SNmascplur	**->**	**DETmascplur Nmascplur**
SNfémplur	**->**	**DETfémplur Nfémplur**

Ces règles permettent: ? ma circonstance

Figure 3.6 : Grammaire Hors Contexte : rajout de traits de `genre`.

Le nombre de traits reste fini, et il faut réécrire toute la grammaire quand on en rajoute.

Les grammaires indexées

Les grammaires indexées, décrits par (Aho 1968), constituent une sur-classe des langages hors-contexte et une sous-classe des langages sensibles au contexte. Les grammaires indexées se distinguent des grammaires hors-contexte par la possibilité d'attacher aux non-terminaux de la grammaire des chaînes d'indices et de manipuler des piles d'indices dans les règles, en ajoutant des indices ou en en supprimant, ou en les copiant. On définit une grammaire indexée par la donnée d'un 5-uple `(N,T,I,P,S)` où `N` est l'ensemble des non-terminaux, `T` celui des terminaux, `I` celui des indices, `P` celui des productions et `S` l'axiome de la grammaire. On dispose de plus de trois types de production :

1. La chaîne d'indices associée à la partie gauche de la règle est recopiée auprès de chaque non-terminal de la partie droite de la règle :

$$A[...] \text{ ----> } X_1[...]\ X_n[...]$$

2. Un nouvel indice est ajouté à la chaîne d'indices de l'élément droite de la règle qui remplacera l'élément gauche de celle-ci :

$$A[...] \text{ ----> } B[i...]$$

3. Le premier indice de la chaîne d'indices de l'élément gauche de la règle est retiré de la chaîne d'indices associée à chacun des non-terminaux de la partie droite :

$$A[i...] \text{ ----> } X_1[...]\ X_n[...]$$

(Hopcroft & Ullman 1979) présentent la grammaire indexée suivante :

`G` telle que `N=(S,T,A,B,C)`, `T=(a,b,c)`, `I=(f,g)`, `S`. Et les règles suivantes :

```
S --> Tg
T --> Tf
```

```
T --> ABC
Af --> aA
Bf --> bB
Cf --> cC
Ag --> a
Bg --> b
Cg --> c
```

On peut construire la dérivation suivante :

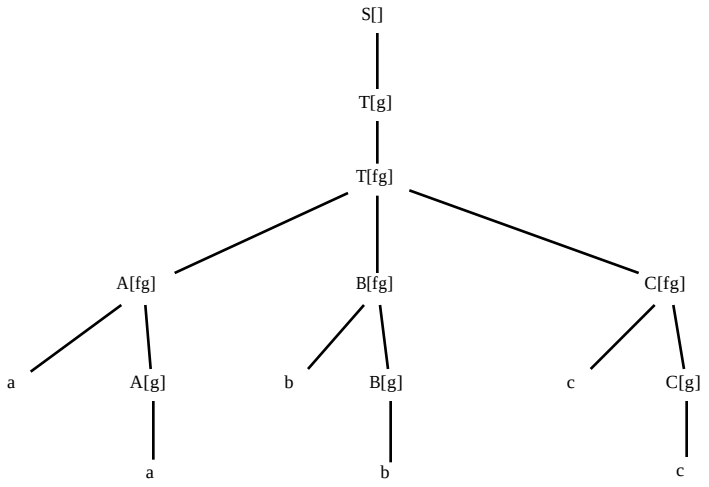


Figure 3.7 : Grammaire indexée : dérivation pour $a^n b^n c^n$.

On génère $a^n b^n c^n$ qui, on le sait, n'est pas un langage hors-contexte. Les langages indexés manipulent (via la chaîne d'indices attachée à un non-terminal), un nombre indéfini de catégories syntaxiques ce qui les fait sortir du cadre des grammaires hors-contexte. La circulation des indices permet de contraindre les dérivations ou de transmettre des informations d'une partie d'un arbre syntaxique à une autre. On peut ainsi rendre compte des dépendances à distance ou d'enchevêtrements.

Grammaires basées sur l'unification (-> incise 1)

Une théorie linguistique vise à créer un modèle reflétant les propriétés de la langue naturelle. Pour ce faire, les linguistes ont développé des langages à l'aide desquels ils tentent de construire cette théorie. Un méta-langage de ce type permet d'une part de fournir un instrument de description de la langue naturelle et d'autre part de délimiter la classe des langues naturelles possibles. En suivant Schieber (Schieber 1987), on évalue un méta-langage selon trois critères fondamentaux que sont l'adéquation linguistique, le pouvoir expressif et l'adéquation informatique. Le formalisme PATR (Schieber 1986) est un formalisme linguistique basé sur l'unification favorisant la définition de grammaires à traits et leur utilisation pour l'analyse. Dans cet ensemble de formalismes on distingue deux types de formalisme (Miller & Torris 1990) : ceux qui ont été conçus comme des outils pour linguistes et ceux qui se veulent des théories linguistiques. Les formalismes de type "outil" (PATR en fait partie) se distinguent par des mécanismes généraux destinés à accroitre leur pouvoir expressif tandis que les formalismes de type "théorie" (LFG en fait partie) ont tendance à incorporer des instruments définis pour des besoins spécifiques liés aux hypothèses auxquelles ils adhèrent. La compréhension des choix conceptuels faits dans les formalismes du premier type est relativement indépendante des analyses linguistiques qu'ils servent à exprimer. Au contraire, ces deux aspects sont liés dans les formalismes du deuxième type où les choix formels sont censés exprimer des principes universels.

3.3. Classifier en terrain connu : taxonomies bien identifiées et langages à taxonomies de classes

Et les objets, quelle doit être l'attitude vis-a-vis des objets ? Tout d'abord en faut-il ? Quelle question. Mais je ne cache pas qu'ils sont à prévoir. Le mieux est de ne rien arrêter à ce sujet, à l'avance. Si un objet se présente, pour une raison ou pour une autre, en tenir compte.
"L'innommable", Samuel Beckett, Ed. de Minuit.

3.3.1. La PàO

Mise en oeuvre immédiate d'une taxonomie référentielle et illusion de transparence.

La programmation à Objets a été conçue à partir de représentations des connaissances pour résoudre certains problèmes pratiques de programmation. Dans un système de représentation Objet, les objets abstraits et individuels sont structurés des plus abstraits vers les plus concrets au moyen d'une hiérarchie basée sur une relation de généralisation / spécialisation. Cette relation est définie en termes de "Sorte-de". Chaque objet est défini par un ensemble d'attributs valués, les attributs non spécifiques à l'objet sont accessibles par un mécanisme d'héritage. L'ensemble des attributs d'un objet peut se lire en parcourant le graphe d'héritage à travers ses ancêtres. Une approche Objet de la représentation des connaissances peut être synthétisée ainsi :

- **1. Un Objet est complètement déterminé par ses attributs et par les opérations qui le manipulent.**
- **2. Tout peut être représenté sous la forme d'un objet.**
- **3. Les objets sont dans un rapport d'adéquation avec le monde. Ils ne peuvent être affirmés, postulés ou réfutés.**

La notion de réification est essentielle dans une approche objet. Cette approche définit une opération qui permet de définir informatiquement une chose quelconque sous forme d'un objet : passage d'une chose vers une entité qui représente cette chose ou passage d'un événement vers une entité qui le représente etc.

3.3.1.1 Présentation de la programmation Objet avec CLOS

CLOS (Common Lisp Object System) ((Keene 1989), (Steele 1990), (Habert 1996)) est un langage orienté objet de haut niveau développé à partir de la proposition de normalisation de LISP qui a débouché sur le standard de fait que constitue Common Lisp (Steele 1990).

Objets et classe

Un objet est l'association d'un état, modifiable, et d'opérations permettant d'accéder à cet état et de le modifier. Un objet peut mémoriser les traitements qui lui sont appliqués. Les informations qu'il stocke ne sont pas accessibles directement, mais seulement par l'intermédiaire des messages qui sont reçus. L'objet constitue ainsi une "boîte noire". Les informations sont stockées dans des "champs" et les "méthodes" représentent la manière dont les autres objets vont pouvoir questionner un objet ou lui demander de se modifier. Une classe regroupe les objets qui comportent les mêmes attributs et les mêmes comportements. C'est à la fois un moule : elle permet de créer des objets qui ont tous la même forme (interne), et la mise en facteur des méthodes des objets en cause. Il y a partage des savoir-faire. La factorisation des connaissances

resterait limitée si les classes étaient simplement juxtaposées. Ce n'est pas le cas. Une classe peut hériter d'une autre, c'est-à-dire qu'elle ajoute aux champs que sa super-classe comprend les siens propres, et qu'elle additionne aussi ses comportements à ceux dont elle hérite. Elle peut aussi redéfinir à son niveau certains champs ou certains comportements, masquant par là même les caractéristiques sur ce point de la super-classe. Une classe donnée peut même hériter de plusieurs classes. La programmation à objets a donc comme objectif explicite un meilleur lien entre les données utilisées par le système et les entités du domaine représenté. Un objet n'étant utilisable qu'au travers du protocole constitué par les comportements attachés à sa classe, ceux-ci constituent de la sorte un (méta) savoir sur les informations que cet objet stocke. Le graphe d'héritage des classes, par la taxonomie qu'il figure (Wegner 1987), représente également un (méta-) savoir sur les relations propres au domaine modélisé.

Définir une classe

Définir une classe équivaut d'abord à créer un moule, celui sur lequel seront fabriquées toutes les instances de la classe. En suivant notre domaine de représentation, la classe des mots peut se construire de la manière suivante :

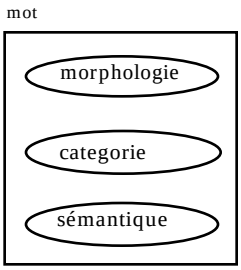


Figure 3.18 : Classe `mot`.

Voici une première définition de la classe correspondante :

```
(defclass mot()(morphologie categorie semantique))
```

Utilisons toutefois immédiatement toutes les possibilités qui nous sont données par CLOS pour donner une valeur par défaut aux champs, préciser leur type, leur associer des fonctions de lecture ou d'écriture, etc. On obtient alors la définition suivante :

```
(defclass mot
  ()
  ((morphologie
      :initarg :name
      :initform ""
      :type string
      :reader get-morphologie
      :writer set-morphologie
      :documentation "Morphology of a word")
   (categorie
      :initarg :name
      :initform ""
      :type string
      :reader get-categorie
      :writer set-categorie
      :documentation "Category of a word")
   (semantique
      :initarg :name
      :initform ""
      :type string
      :reader get-semantique
      :writer set-semantique
      :documentation "Semantism of a word"))
```

```
  (:documentation
"Concrete class.
A <word> knows its morphology, category and semantism"))
```

Pour éviter les efforts de mémorisation inutiles, on utilise le préfixe `get-` pour les lecteurs de champ et le préfixe `set-` pour les écriveurs. La figure 3.19 détaille les différentes parties de définition d'une classe. Ce qui est en gras doit figurer tel quel dans la définition. Il en va ainsi de `defclass`, comme des parenthèses qui entourent la forme de définition, ainsi que ses sous-parties : liste des super-classes éventuelles, liste éventuellement vide de descriptions de champs, liste des options éventuelles de la classe. Les deux premières listes, même vides, figurent obligatoirement dans la définition, la troisième peut être omise.

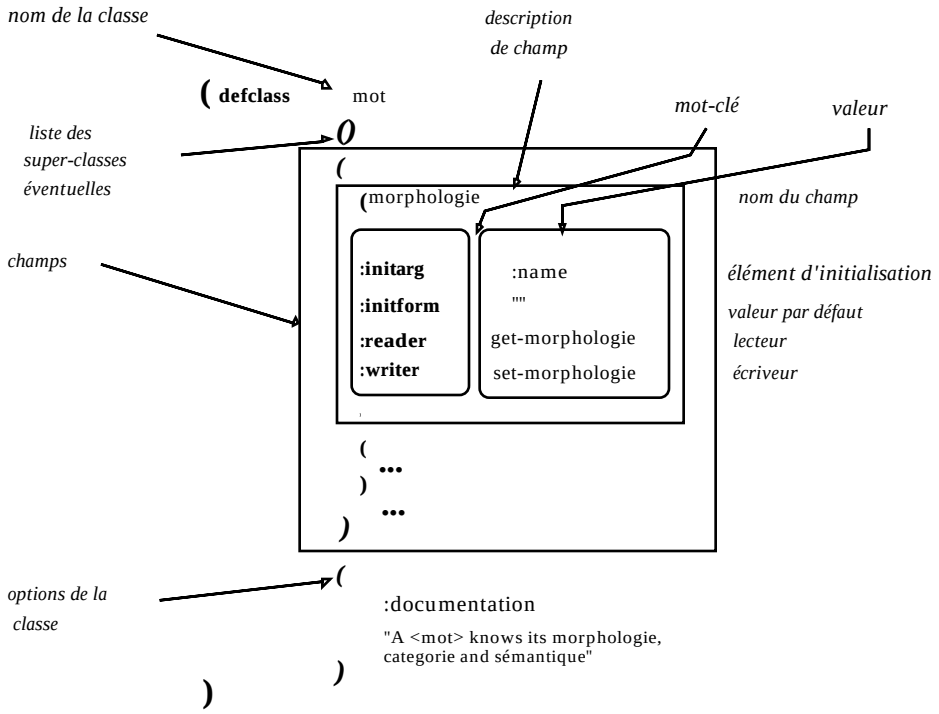


Figure 3.19 : Définition d'une classe.

La description d'un champ peut se réduire au nom seul. Mais elle s'accompagne en général d'informations permettant de donner une valeur par défaut à ce champ, de fournir des fonctions de lecture ou d'écriture etc. Ces informations sont constituées par une suite d'associations entre un mot-clé et une valeur. Ainsi la classe *mot* possède-t-elle un champ `categorie`, qui aura pour valeur par défaut la chaîne vide. `Get-morphologie` permettra de lire le contenu de ce champ, `set-morphologie` d'y écrire une valeur, `:name` servira à donner une valeur autre que la valeur par défaut au moment de la création d'une instance.

Créer des instances

C'est ce que permet l'appel à la fonction `make-instance`. Cette fonction évalue son argument (comme l'indique la quote devant `mot`), ce qui permet de calculer de quelle classe on veut créer une instance.

```
( make-instance 'mot )
```

Figure 3.20 : Créer des instances.

Comme on le voit, un tel appel "nu" aboutit à donner aux champs de la classe leur valeur par défaut.

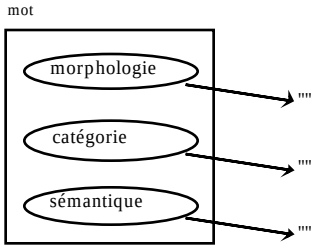


Figure 3.21 : Appel nu à `make instance`.

Il est également possible d'utiliser tout ou partie des éléments d'initialisation, suivis à chaque fois de la valeur à utiliser, et ceci, dans n'importe quel ordre.

```
(setf foo1 (make-instance ‘mot
              :categorie    “Nom”
                     :morphologie  “moulin”
                     :semantique   “Instrument”))
```

La classe sert à fabriquer les instances. Elle permet d'en constituer la structure, et elle leur associe les comportements nécessaires. Pour autant, elle ne connait pas ses instances. A l'inverse, il est possible, à partir d'un objet, de savoir de quelle classe il est l'instance.

Graphe d’héritage et masquage

Une classe peut hériter d'une autre classe. Elle ajoute ainsi aux champs hérités de sa super-classe ses propres champs. Elle peut ainsi redéfinir à son niveau certains comportements masquant ainsi les champs ou les comportements de sa super-classe. L'héritage peut étre simple (une classe possède une super-classe directe et une seule) ou multiple (une classe peut hériter de plusieurs super-classes). La structure obtenue par la relation d'héritage peut être soit un arbre de profondeur quelconque soit un graphe dont la constitution doit obéir à des régles précises sous peine de contradictions dans la définition des caractéristiques des classes ce réseau.

Fonction générique et les méthodes associées

Un langage objet est l'association d'une structure de données et d'un mode d'emploi intrinséquement lié à cette structure de données. L'un des traits originaux de CLOS par rapport aux autres langages orientés objet est la généricité et la combinaison des méthodes. Les méthodes constituent les protocoles d'accès aux objets. Quand un objet est défini, c'est à dire quand une structure de données est définie, ce n'est plus cette structure de données que l'on connait mais ce sont les protocoles d'accès à cette structure. La nature exacte des opérations éxecutées par une méthode dépend donc de la classe à laquelle appartient l'objet destinataire du message. Si cette méthode est définie pour plusieurs sur-classes d'une classe donnée, c'est la méthode la plus spécifique (i.e la classe la plus proche) qui sera appliquée. Les fonctions génériques jouent un rôle fondamental en CLOS. Les méthodes n'ont d'ailleurs pas d'existence en dehors d'elles : une méthode n'est accessible que via une fonction générique de même nom. La fonction générique fournit une interface : elle indique le patron syntaxique commun aux diverses méthodes associées. L'appel d'une fonction générique conduit à un filtrage et à un choix de la méthode correcte : la fonction générique opére un aiguillage[51] et un tri en fonction des arguments qui lui sont passés. De plus, on peut associer à une méthode donnée (dite primaire) une méthode `before` et une méthode `after`. Lors de l'appel de la méthode primaire, la

[51] L'appel d'une fonction générique conduit aux étapes suivantes : (1) Vérification syntaxique des arguments. La fonction générique définit une lambda-liste commune à toutes les méthodes de même nom. La forme de l'appel doit correspondre au patron que représente cette lambda-liste. (2) La fonction générique sélectionne les méthodes qui lui sont applicables. Une méthode est applicable quand les arguments fournis à la fonction générique sont des instances des classes (ou des sous-classes) indiquées pour chaque paramètre dans la définition de la méthode. (3) S'il n'y a pas de méthode applicable, une erreur est déclenchée. (4) La méthode effective est appliquée : c'est le résultat de l'application de la combinaison de méthodes spécifiées pour la fonction générique à la liste de méthodes applicables. Le résultat obtenu est celui de la fonction générique.

méthode `before` est exécutée d'abord, puis la méthode primaire, puis la méthode `after`. Le résultat est celui de la méthode primaire. Les méthodes `before` et `after` servent aux effets de bord.

Documentation

Une autre caractéristique de CLOS est de fournir la possibilité d'intégrer de la documentation au sein du code. En effet des chaînes de documentation peuvent être attachées aux classes, aux champs d'une classe, aux fonctions génériques et aux méthodes. On voit très vite l'intérét de l'utilisation de cette fonctionalité pour renseigner l'utilisateur sur les objets manipulés par une application implémentée avec un tel langage. Il est aisé de rendre cette documentation accessible à l'utilisateur en lui fournissant les méthodes idoines.

3.3.2. Représentation de lexique et Objets

3.3.2.1. DATR : un outil de représentation hiérarchisée

DATR (Gazdar & Evans 1990) est un langage formel basé sur l'héritage qui permet d'exprimer simplement le mécanisme d'exception / généralisation que l'on rencontre dans le cadre de la représention des connaissances lexicales. Ce langage permet la représentation des informations lexicales en formulant les connaissances de manière déclaratives (exceptions et généralisations). Il se situe dans le cadre général de la représentation par graphe d'héritage. Et donc, au lieu de définir un élément lexical comme irrégulier s'il ne satisfait pas les propriétés typiques d'un élément voisin, il s'agit dans un tel cadre de considérer que cet élément est régulier sauf pour telle ou telle propriété : il hérite donc de toutes les propriétés d'une classe d'éléments, sauf en ce qui concerne telle ou telle particularité. DATR a été développé pour fournir un outil de modélisation du lexique pour les formalismes du TALN et notamment les grammaires basées sur l'unification. D'ailleurs, DATR utilise des notations proches de celles utilisées par les grammaires d'unification et en particulier PATR-II. Les entrées lexicales sont représentées sous forme de graphe. Plus on descend dans un graphe de représentation lexicale, plus la spécification est grande et plus les propriétés héritées augmentent. DATR intègre l'héritage (simple ou multiple) grâce au principe de combinaison d'héritage ainsi que la non-monotonie par le biais du masquage (d'information). Une propriété héritée peut être redéfinie sur n'importe quel noeud du graphe.

3.3.3. Représentation de règles et Objets

Les analyses des expressions figées faites au LADL ont montré que la plupart des expressions figées françaises obéissent aux règles générales de la syntaxe en y rajoutant des contraintes arbitraires. Intégrer la connaissance des expressions figées dans l'analyse automatique peut pousser à créer autant de règles et à perdre des liens entre des règles similaires. Le fait de travailler à la réalisation d'analyseur automatique dans le cadre de la programmation à Objet doit mener plutôt à une spécialisation des règles syntaxiques générales jusqu'à l'obtention de la finesse de contrainte souhaitée, donc à l'organisation des règles nécessaires par des relations d'héritage (Habert 1991c). Le modèle qu'est la PàO joue dans ce cas un rôle d'heuristique et pèse dans la représentation choisie des rapports entre idiosyncraties et règles générales pour les expressions figées. Dans une analyse automatique, il est souhaitable de pouvoir ajuster à une règle syntaxique des contraintes particulières en la spécifiant. L'approche Objet semble ici toute indiquée puisqu'elle factorise au maximum les comportements et les structures. Pour ce faire la solution retenue par OLMES (-> partie 3 chapitre 8) de représenter la grammaire comme un réseau de classe permet de définir des méthodes spécialisées sur les différents éléments de ce graphe d'héritage. Chaque classe définie à la compilation correspond à une occurrence d'un constituant de règle. Il est possible d'ajouter sur chacune de ces classes ainsi définies des méthodes / contraintes particulières qui spécifient des traitements particuliers. On peut de plus grâce à CLOS, spécifier qu'il faut "ajouter" les différentes contraintes que l'on met en place si des problèmes de masquage bloquaient le calcul de toutes les contraintes possibles.

3.3.3.1. `Moulin à N2` et DATR

(Habert & Fabre 1993) présente *"une approche hiérarchique de la construction du sens de dénominations complexes de la forme N1 à N2"* qui utilise les mécanismes mis en place par DATR. La représentation morpho-sémantique des termes en jeu dans les séquences étudiées a l'allure suivante :

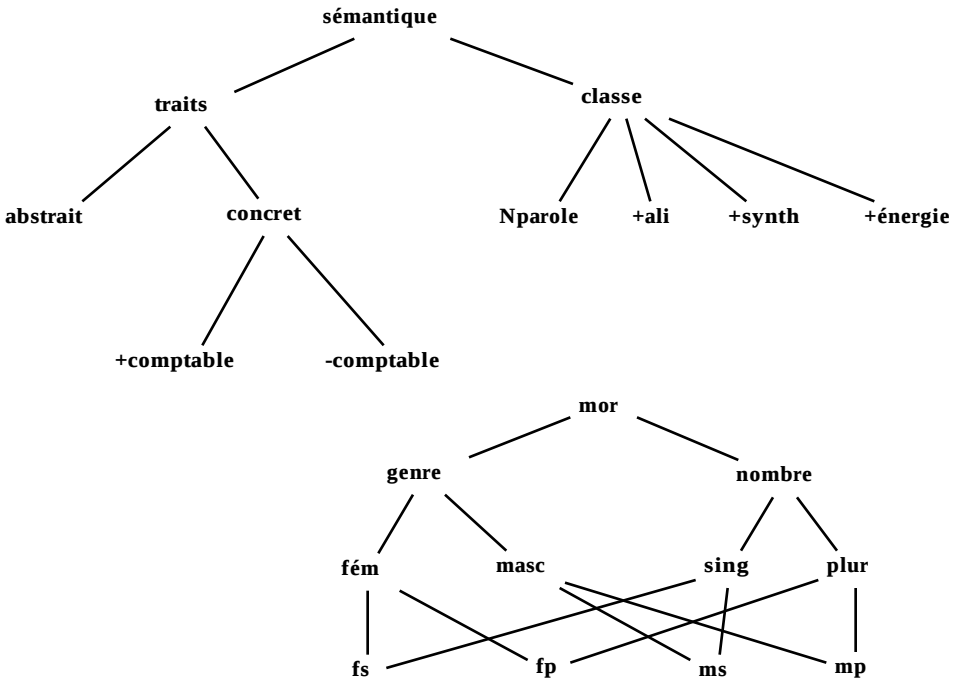


Figure 3.24 : Représentation morpho-sémantique de N1 à N2.

Le recours au langage hiérarchisé DATR permet dans ce cadre de limiter l'expression des redondances et de rendre compte de la hiérarchisation des traits. Les règles utilisées pour la construction du sens des expressions étudiées sont des règles de réécriture à base hors contexte où chaque élément de règle est remplacé par un graphe de contraintes sur l'élément correspondant de la phrase. On a ainsi une représentation graphique équivalente de la représentation à la PATR-II :

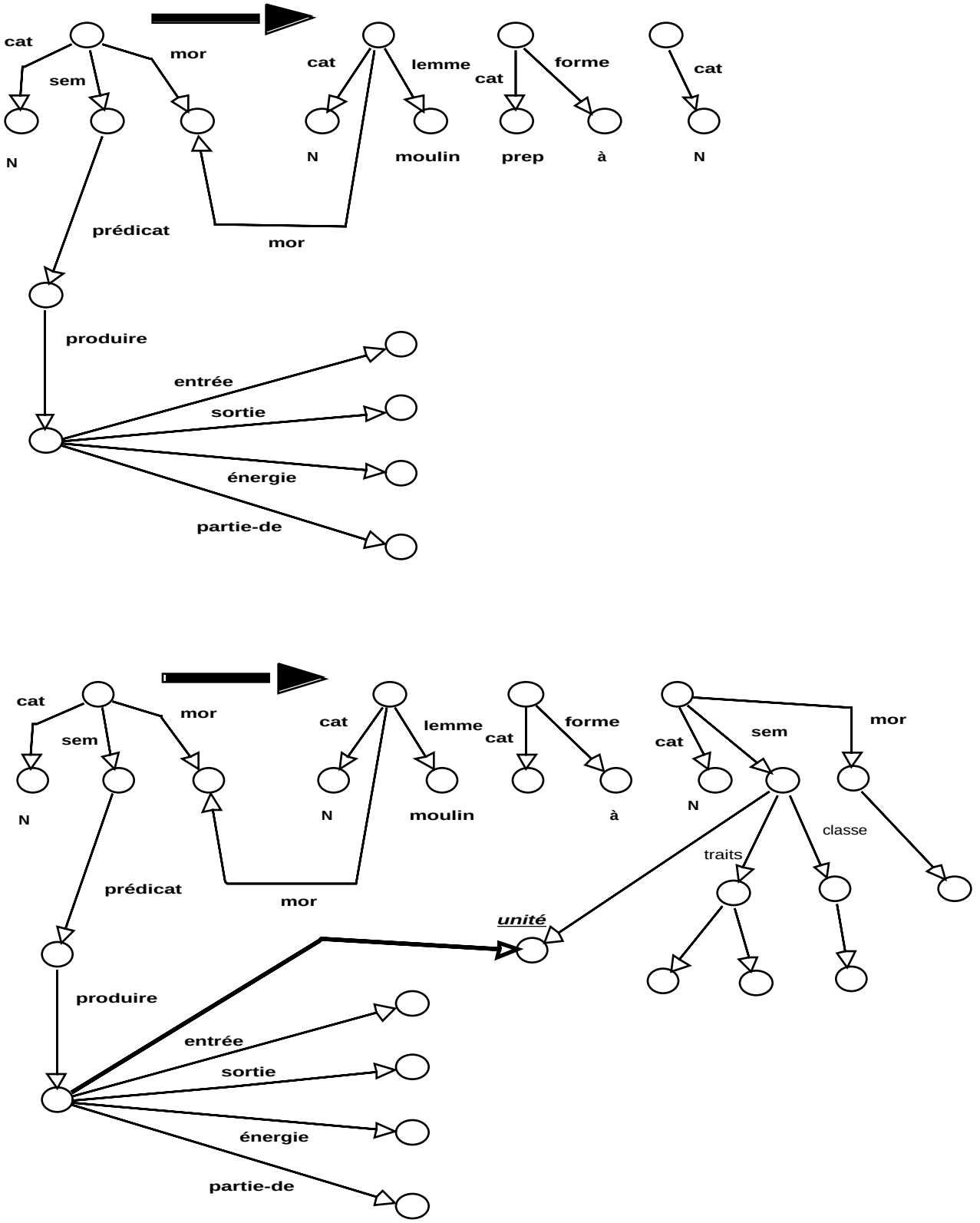


Figure 3.25 : `Moulin à N2`, représentation hiérarchisée des connaissances et construction du sens.

Le modèle proposé permet de formuler des hypothèses sur l'occupabilité des places argumentales dans le moule `moulin à N2`. Une partition des possibilités peut se matérialiser par les ensembles suivants :

moulin POUR/sortie moulin POUR/entrée moulins AVEC/énergie moulin AVEC/partie-de

L'ordre d'écriture des ensembles ainsi défini de réalisations des structures possibles sur le moule `moulin à N2` correspond à la réalisation quantitative de telles structures : les occurrences attestées et regroupées dans les ensembles ci-dessous diminuent quand on lit les ensembles dans l'ordre donné. Cette analyse offre une vision unifiante de la construction du sens des structures étudiées. Elle permet d'une part d'éliminer des séquences données si celles-ci ne rentrent pas dans le cadre ainsi défini, d'autre part de classer les séquences `moulin à N2` en fonction des compatibilités observées. De plus, on peut ainsi définir des bases pour des heuristiques évaluant la classification des lexèmes à entrées multiples dans le cadre défini par la classification supra.

3.3.3.2. La hiérarchie au service de la sous-détermination

Le travail présenté par (Habert & Fabre 1993) s'inscrit dans un cadre d'analyse tel que la construction du sens s'établit non pas par la juxtaposition et la combinaison des sens des entrées lexicales mais par la construction progressive en précisant pas à pas les hypothèses initiales, quitte à rester dans l'indétermination quand les informations disponibles ne permettent d'aller plus loin. Il s'agit de mettre en place un travail d'affinage des constructions développées par l'analyse qui progresse du plus général vers le particulier (-> glossaire : raisonnement non-monotone).

3.3.4. Limites : comment prendre en compte l'évolution des connaissances

3.3.4.1. Les savoirs lexicaux résistent aux classements hiérarchisés

Si on choisit de représenter les savoirs linguistiques dans une taxonomie de classes, on se heurte de front au modèle de représentation sous forme d'objet dans lequel "un objet est (principalement) ce qu'il est". Les outils pour la représentation dans une telle approche ont tendance à figer les choses. Quand il s'agit de représenter une matière organique, les problèmes surgissent rapidement. Si on accepte de penser en termes d'objet, c'est-à-dire d'admettre que tout est réductible en un ensemble de propriétés définitoires stables, une telle approche se justifie. Malheureusement, on n'en saura jamais assez sur les faits de langue. Le langage crée en permanence. Un fait de langue peut prendre le contre-pied de résultats établis, bousculer les règles définitoires. On est donc contraint à des esquisses de représentation.

- La PàO permet d'articuler les représentations sur des domaines de savoirs stabilisés.
- Or les faits de langue sont difficilement réductibles en termes d'objets figés : Il faut pouvoir "donner du mouvement au jeu" de la représentation.
- Une approche de représentation pour la TALN doit permettre un affinage dans les structures de représentation construites.

Le figement dans la représentation ne permet pas de rendre compte des articulations dynamiques forme-sens (interactions et concurrences). Même si un choix de représentation qui manipule la notion d'objet dans une taxonomie de classes est adéquat pour fixer un domaine de savoir établi dans une structure hiérarchisée, il s'avère parfois délicat de construire puis de prendre en compte les évolutions de telles représentations hiérarchiques quand celles-ci décrivent des domaines de savoir non stabilisés. Dans une telle approche, la mise en contexte des instances de classe peut révéler la nécessité d'une mise à jour de la structure initialement adoptée pour la définition des classes. Dans le cas de la construction d'une hiérarchie lexicale pour laquelle on souhaite inscrire dans la définition des entrées lexicales une articulation forme-sens, une telle représentation doit affronter le problème complexe de la confrontation d'un savoir établi et d'un savoir à venir. La prise en compte de nouvelles informations, de résultats issus de l'analyse, de configurations énonciatives particulières... peut conduire à modifier, à affiner la représentation initiale. Plus on structure un domaine de savoir, plus les liens de dépendance entre les noeuds du graphe d'héritage se complexifient, et il devient ainsi délicat de modifier cette structure hiérarchique.

3.3.4.2. Représentation de moulin et construction du sens des séquences moulin à N2

L'interprétation des configurations nominales du type moulin à paroles peut relever des mécanismes d'extension de sens à la Briscoe, des coercions de type à la Pustejovski , ou encore de la profondeur variable à la Kayser .

- On peut inscrire dans la représentation les scénarios, les processus d'interprétation.

Il reste cependant difficile de prévoir tous les savoirs que l'on peut associer à la configuration nominale traitée : si un certain nombre de "scénarios" sont associés à un mot donné (permettant par exemple des extensions de sens potentielles), cette somme de savoirs n'est pas nécessairement définitive. Le langage naturel et sa représentation pour un traitement automatique s'accomode mal d'une prévision qui clot les possibles.

- On peut construire une représentation sémantique très abstraite, très peu déterminée qui est précisée si besoin est.

Ainsi dans le cadre des séquences moulin à N2, on peut mettre en oeuvre l'approche de la profondeur variable défendue par Daniel Kayser : on construit au départ une représentation sémantique initiale peu déterminée (uniquement le prédicat pour moulin) qui peut ensuite être affinée en tenant compte des éléments d'informations complémentaires permettant d'étendre la représentation initiale. Par contre ce modèle se heurte au problème des remodelages potentiels des savoirs représentés : il peut être difficile voire impossible de faire évoluer dynamiquement une représentation hiérarchisée de ces savoirs.

De plus dans la représentation présentée supra, les interprétations possibles dans la cadre de la construction du sens des formes moulin à N2 sont toutes placées au même plan, alors qu'un poids différent leur est attaché. On aimerait pouvoir dire qu'une interprétation est plus centrale qu'une autre dans la manière même de représenter les connaissances. Et donc ordonner ces différences de statuts sémantiques dans la représentation construite.

3.3.4.3. Représenter en hiérarchisant pas à pas les interprétations

Les processus de représentation doivent rendre possible une phase de représentation transitoire où la hiérarchie est à construire et donc où la structuration des savoirs n'est pas définitive. Il faut utiliser des outils qui permettent une mise à jour d'un savoir stabilisé sous forme hiérarchique sans prédéterminer ce classement mais en le reconstruisant et en laissant ouvert le classement produit. Pour le langage naturel, les résultats acquis restent transitoires. Leurs représentations doivent pouvoir évoluer. Vouloir classer les choses de manière figée pose donc des problèmes difficilement compatibles avec une nécessaire évolution dans les représentations construites des savoirs linguistiques à un moment donné. Il ne s'agit pas ici de dire que la représentation sous forme hiérarchique est inconcevable mais d'adapter les structures hiérarchiques aux processus qui sont à l'oeuvre dans la construction du sens des faits linguistiques. Ce travail vise aussi à établir une comparaison critique du pouvoir expressif de deux cadres de représentation : la Programmation à Objet dans une taxonomie de classes et la Programmation à Prototypes. Dans un cas, il est plus facile de représenter des savoirs établis donc de fixer ces savoirs dans des hiérarchies. Dans l'autre cas, le processus de représentation est contraint par des savoirs non connus à l'avance : il est donc plus difficile de présumer de ces savoirs et de leur mode de structuration et d'organisation. Il s'agit donc aussi d'évaluer les apports pertinents de chacun de ces cadres pour la représentation du langage naturel, ce dernier articulant très bien ces deux aspects de la manipulation des connaissances.

3.4. Classifier en terrain mouvant : La PàP

3.4.1. Prototypie et représentation des connaissances

Le problème de la prototypie dans le domaine de la programmation à Objets s'inscrit dans un cadre général de réflexion sur la représentation du savoir adapté à la résolution de problèmes informatiques. Le modèle associé à la PàO donne la primauté à la représentation des concepts comme des classes (ensembles), la seconde présentée ici propose la représentation des concepts comme des prototypes. Ces deux systèmes de représentation font appel à deux mécanismes distincts pour formuler le partage des connaissance : l'héritage pour le premier et la délégation pour le second. La Programmation à Prototypes s'est développée à la fin des années 80, à la suite notamment des travaux de Liebermann (Lieberman 1986) ou des informaticiens de Stanford (Ungar & al. 1987, 1988) qui ont par la suite mis au point le langage à prototypes Self dont nous reparlerons plus loin.

3.4.1.1. Prototypie et représentation du savoir : une présentation

Une des problématiques majeures en matière de représentation des connaissances en IA tourne autour de l'opposition Classe / Catégorie et leurs processus afférants (Classification / Catégorisation). Les classes sont définies de manière abstraite, désambiguïsée suivant des modèles de nécessité et de suffisance : elles modélisent des concepts. Cependant, dans l'exercice de nos activités intellectuelles nous n'utilisons pas directement la notion de classe. On se base très souvent sur des entités individualisées dans les conditions empiriques déterminées par le contexte. C'est la contextualisation de la connaissance qui détermine les conditions de leur activation et les limites de leur validité.

3.4.1.2. La PàP : les langages à prototypes, un outil qui permet l'hétéro-modification

PàO vs. PàP

Pour introduire la présentation de la PàP, il convient de souligner la différence d'approche qui existe entre l'approche la programmation dans les langages de classe et celle de la PàP. Ces deux modèles de représentation sont implémentés en utilisant deux mécanismes distincts : l'héritage pour les langages à classes et la délégation pour les langages à prototypes.

Dans les langages de classe, on définit au préalable une classe abstraite dans laquelle sont définies des propriétés, et à partir de cette abstraction (le moule) on peut créer des objets : les instances de cette classe. On dit qu'un objet hérite des propriétés définies dans la classe dont il est le représentant. C'est la relation "Une-sorte de" qui définit l'appartenance d'un élément donné à la classe ainsi définie. Les divers éléments qui répondent aux propriétés essentielles attachées à une classe donnée partagent donc une certaine somme d'information. Les propriétés spécifiques des éléments de la classe donnée sont définies de manière individuelle de telle sorte que cette spécification ne soit pas partagée. En termes de représentation informatique, on définit donc la classe comme une entité abstraite dans laquelle sont définies des propriétés. On utilise ensuite ce moule pour créer des objets.

Les langages à prototypes utilisent une démarche inverse. Les langages à prototypes sont caractérisés par l'absence de structuration abstraite, du type de celle qui est proposée par le modèle à classes. Dans un système à prototypes pur, les objets de base sont les seules entités du système. Dans cette approche, on utilise le savoir que l'on a sur un élément particulier pour créer une représentation prototypique de la famille de l'élément retenu. A partir de cet objet prototypique, on construit un nouvel objet qui possède les mêmes propriétés que le prototype initial mais qui pourra aussi posséder des propriétés individuelles

spécifiques. Il n'existe donc pas dans la PàP de distinction du type classe/instance, les objets y sont complètement autonomes et contiennent toute l'information nécessaire pour les représenter. Le premier avantage des langages à prototypes est lié à la possibilité de ne pas avoir à pré-déterminer les propriétés pertinentes dans la création des objets.

La PàP réalise aussi du partage local d'informations entre les objets créés. La délégation est le mécanisme qui réalise ce partage des connaissances entre les objets :

"To create an object that shares knowledge with a prototype, you construct an extension object, which has a list containing its prototypes, which may shared by other objects, and personnal behavior idiosyncratic to the object itself. When an extension object receives a message, it first attemps to respond to the message using the behavior stored in its personal part. If the object's personal characteristics are not relevant for answering the message, the object forwards the message on to the prototypes to see if one can respond to the message" (Lieberman 1986).

La délégation repose sur une factorisation des comportements communs à un ensemble de prototypes.

Illustration de la représentation prototypique avec la catégorie grammaticale des noms

L'exemple qui suit est une illustration d'une première approche que l'on peut avoir de la notion de prototype appliqué à certaines entités de la langue.

Un nom = un prototype

Si l'on considère le mot `moulin` comme un représentant typique des mots généralement regroupé dans la catégorie grammaticale des noms, on peut considérer qu'il peut servir de modèle pour la création d'un prototype de nom. A partir du savoir de sens commun que l'on a sur ce nom, on créé un objet avec des attributs qui porte ces valeurs.

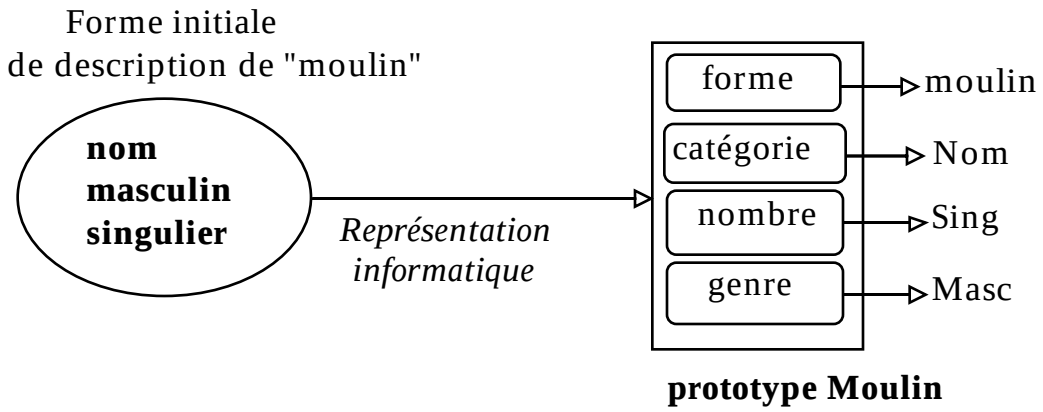


Figure 3.27 : De la description individuelle au prototype Nom.

Représentation d'un autre nom par clonage et différenciation

Si on considère maintenant le nom `machine`, on a affaire à un autre représentant de la catégorie grammaticale des noms. On peut donc créer un prototype associé au mot `machine` à partir du prototype `initial` associé à `moulin` par clonage. On ajuste ensuite les attributs du prototype résultant en leur donnant les valeurs adéquates.

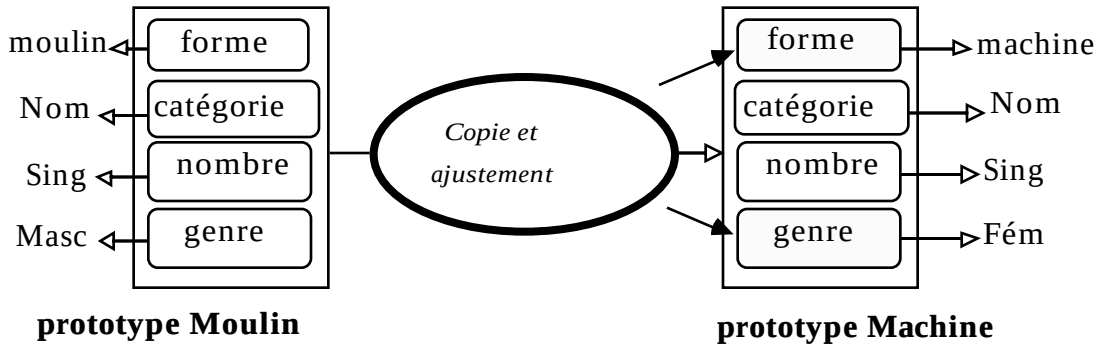


Figure 3.28 : Clonage des prototypes Noms.

Partage des attributs communs par délégation

On peut considérer que l'attribut `catégorie` est partagé par l'ensemble des prototypes représentant des noms. On peut donc décider de créer un objet qui portera cet attribut, et tous les prototypes représentant des noms délègueront à cet objet leur nom de catégorie.

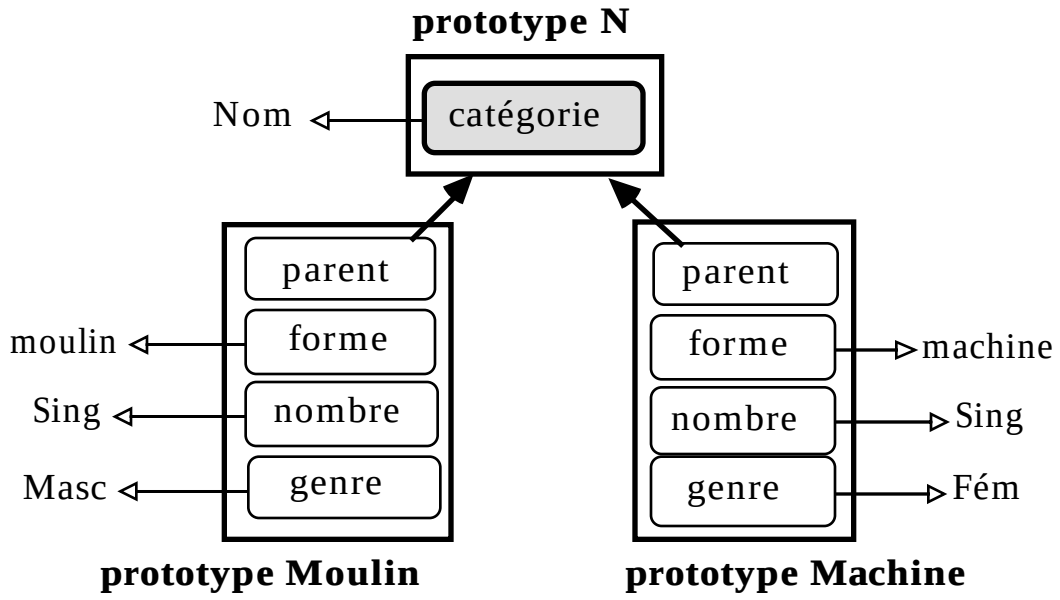


Figure 3.29 : Prototype et délégation chez les Noms.

La représentation prototypique se caractérise donc comme un mécanisme de reprise (via l'opération de clonage) et de différenciation. La délégation permet de définir un objet avec une somme d'informations partagées par un ensemble d'objets et avec une somme d'informations locales / spécifiques / privées attachées à cet objet particulier. Le point fondamental dans ce type de représentation est que le partage d'information est réalisé au niveau des éléments concrets et non au niveau d'une abstraction comme dans le cadre de la représentation en classe.

3.4.1.3. Le langage Self (Ungar & al 1987)

"Self(Ungar & Smith 1987) was designed to aid in exploratory programming by optimizing expressiveness and malleability. It is essentially a template-based language; however, Self templates are as much a matter of convention as of language design. New Objects are created by cloning an existing one..." (Ungar & al. 1988).

*"Self is more than a language. It is the combination of a language, a set of reusable objects, a programming environment, a user interface, and a virtual machine. /./ Self is based on prototypes and has no classes or meta-classes. Simplicity is ensured through uniformity : all data are objects, all expressions are message sends, and computation is performed solely by sending messages" (*Agesen 1996).

Self est le langage qui est utilisé ici pour représenter les mots et leurs comportements. Self est un langage qui permet l'héritage multiple et l'héritage dynamique. Il a été conçu en 1986 par David Ungar & Randall Smith ("Self : the power of Simplicity" OOPSLA'87). La première implémentation a été réalisée à Stanford en 1987, la dernière version (Self-4.0) est disponible depuis juillet 1995. Ce langage est désormais développé par Sun Microsystems avec beaucoup de moyens, ce qui semble indiquer l'importance de ce type de représentation dans le développement de la Programmation à Objets. Un prototype ou un objet en Self est une entité composée d'attributs. Ces attributs peuvent porter des données, des méthodes ou pointer sur d'autres objets. Les objets définis dialoguent entre eux via un mécanisme d'envoi de messages.

Présentation

Le modèle de calcul de Self est basé sur la notion de prototypes. Les langages à prototypes privilégient la flexibilité plutôt que les organisations rigides. Dans un langage à classes, les abstractions (les classes) doivent être créées avant les instances individuelles. Les langages à prototypes ne construisent pas de modèles abstraits aprioriques permettant la création d'instances individualisées. Ce sont les propriétés intrinsèques aux objets représentés qui définissent la structure des prototypes à créer. Cette absence de contraintes dans l'organisation des langages à prototypes peut mener à l'écriture de programmes difficiles à lire. Pour réduire ce manque d'organisation, Self propose une méthodologie de programmation basée sur des objets appelés `traits`. Les objets `traits` agissent comme des bibliothèques partagées contenant des comportements communs à tous les objets qui délèguent ces types de comportement aux objets `traits`. La méthodologie de programmation utilisant les `traits` consiste à définir de nouveaux types d'objets en créant d'abord un objet `traits` définissant le comportement commun au nouveau type d'objet puis en créant une instance prototypique du type d'objet qui servira par la suite au clonage pour la création de nouvelles instances.

La définition d'une classe dans un langage de type PàO peut être assimilée à la construction d'un moule dont les éléments définitoires seront repris et instanciés par chacun des objets qui seront attachés à cette classe : les objets seront complètement déterminés par ce filtre définitoire. La notion de `traits` n'est pas assimilable à cela, en ce sens que même si elle peut prédire ce que sera une partie de la définition des prototypes à venir elle n'est pas suffisante pour décrire ce que peut être le prototype à construire : cette structure ne porte en effet que le savoir partagé par les prototypes qui inscriront un lien d'héritage avec ces comportements. Les deux notions ont en commun le fait d'exercer un regard en avant sur les objets qu'il sera possible de manipuler, mais pour une classe il est complet (les objets seront ce que la classe permet), alors que pour les `traits`, il n'est que partiel (les objets seront ce qu'ils font d'une part et ce qu'ils sont d'autre part).

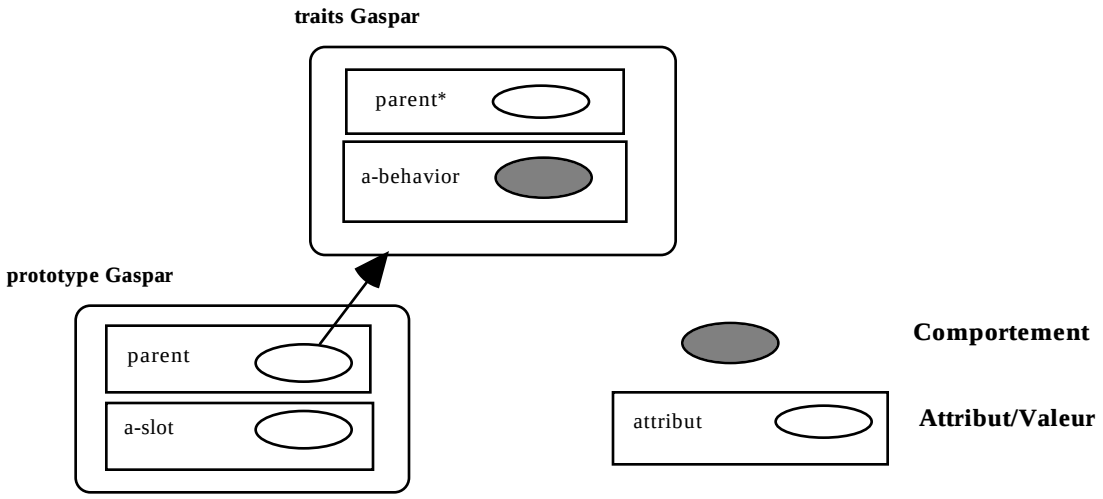


Figure 3.30 : `Traits` et prototypes.

<u>Concept de Bases (Cointe & al 1991)</u>

- Objets autonomes :

  Les objets en Self sont complètement autonomes, et contiennent toute l'information nécessaire pour répondre aux envois de messages. Un objet est une collection d'attributs.

- Unification des concepts d'"instance" et de "méthode" :

  L'état interne d'un objet Self est représenté par un ensemble de "méthodes" retournant une valeur constante et qui sont accessibles par envoi de messages.

- Partage des connaissances :

  Chaque objet est doté d'un attribut `parent` qui lui permet de désigner les objets dont il partage les connaissances. La spécification du partage se fait donc objet à objet.

- Activation par envoi de messages :

  Le mécanisme de calcul de base est l'envoi de messages. A la réception d'un message, un attribut correspondant au sélecteur est recherché dans le receveur puis exécuté s'il est trouvé.

- Mise en oeuvre du partage, la délégation :

  Lorsqu'un objet ne possède pas d'attribut correspondant au sélecteur de message, celui-ci est délégué au parent du receveur. Dans le cas où il existe plusieurs parents, ceux-ci sont ordonnés suivant un ordre indiquant la priorité de délégation. Si un attribut est trouvé dans un des parents du receveur, le message sera exécuté dans le contexte des attributs du receveur initial.

- Clonage :

  L'opération de base pour la création de nouveaux objets est le clonage. Tout objet peut servir de modèle pour cette opération. Le clonage se fait par l'envoi du message `clone` à un objet, message dont l'exécution retourne une copie superficielle du receveur.

Les éléments constitutifs pour une modélisation d'un domaine avec Self peut être définie ainsi :

- 1. On définit un réservoir de comportements (les `traits`).
- 2. On définit un prototype distingué en effectuant une délégation vers l'objet contenant les comportements partagés par ce prototype.
- 3. On créé un nouvel élément par clonage du prototype défini précédement.

Objets et attributs Self

Un prototype ou un objet en Self est une entité composée d'attributs. Ces attributs peuvent porter des données, des méthodes ou pointer sur d'autres objets. La définition structurelle d'un objet peut être modifiée à tout moment. **On dispose avec Self de primitives qui permettent d'ajouter ou de supprimer dynamiquement des attributs aux objets**. Ce mécanisme est fondamental car il permet de modifier voir de remodeler complètement un prototype de manière dynamique. **Ce type de mécanisme n'est pas disponible avec la PàO**.

Les mécanismes d'ajout(s) ou retrait(s) dynamique(s) d'attributs permettent de construire des représentations par affinements successifs : on peut à tout moment ajouter à un objet de nouveaux comportements ou modifier les comportements de cet objet. Les objets définis dialoguent entre eux via un mécanisme d'envoi de messages.

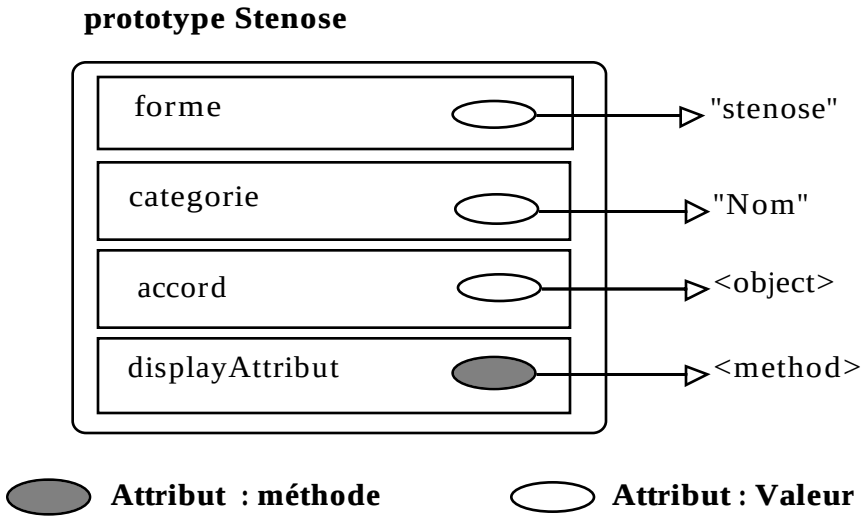


Figure 3.31 : Objet Self.

Héritage et Self : la délégation, une vision dynamique de l'héritage

L'héritage dans la PàP se réalise au travers de la notion de délégation. La délégation est une opération qui permet de factoriser localement des comportements[52] partagés par un ensemble quelconque de prototypes. Pour mettre en place l'heritage avec Self :

- On commence par créer des objets qui vont porter les comportements partagés.
- On établit ensuite un lien entre ces objets et les prototypes qui délèguent les comportements factorisés en inscrivant dans ces prototypes le chemin de délégation défini. Pour inscrire un chemin de délégation

[52] Utiliser un langage de programmation pour modéliser des objets du monde réel conduit à faire référence à deux types de comportement. D'une part les comportements de l'objet réel décrit, dans notre cadre de travail, cela concerne par exemple les comportements syntaxiques ou sémantiques des mots. D'autre part les comportements de l'objet informatique qui sert de support à l'objet décrit, par exemple des opérations qui permettent d'afficher ou d'imprimer l'objet informatique visé (Lemaire & Rechenmann 1995). La factorisation des comportements partagés possible avec la PàP peut concerner ces deux niveaux de description.

dans un objet, on ajoute un attribut `parent` (nom de l'attribut suivi d'une étoile) qui pointe sur l'objet qui porte les comportements délégués.

- Dans le langage Self, un parent commun à plusieurs prototypes est appelé un objet `traits`.

Dans l'exemple présenté ci-dessous le prototype `stenose` délègue deux comportements à deux prototypes distincts via les deux attributs `parent1` et `parent2` :

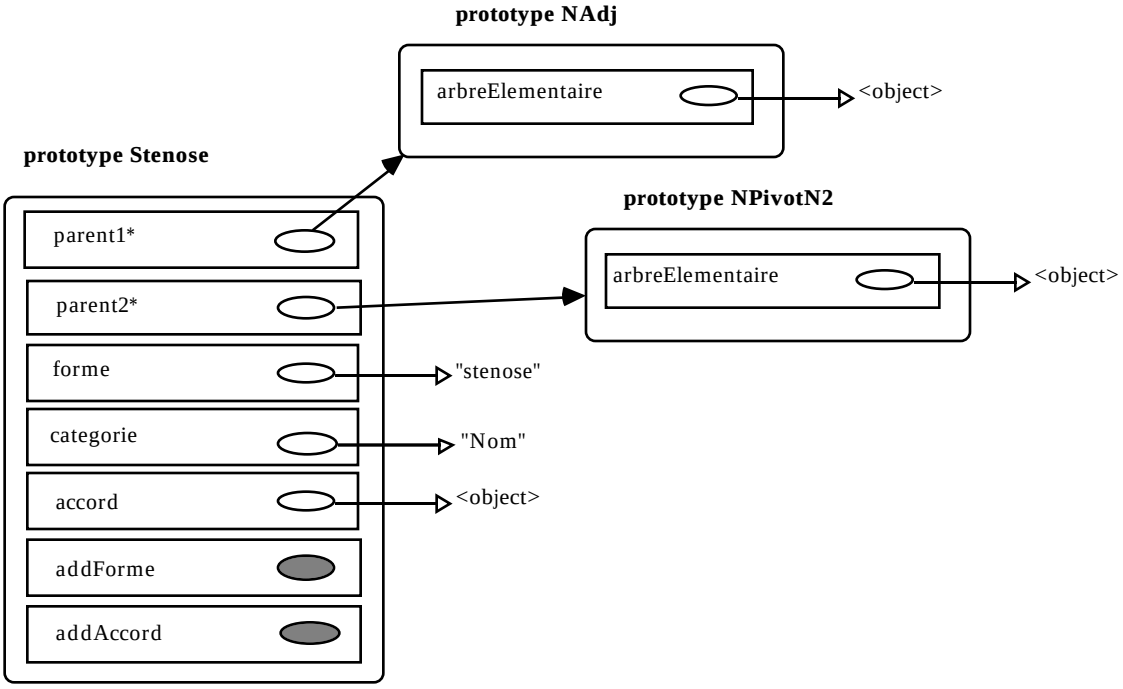


Figure 3.32 : Prototype et délégation en Self.

Permettre l'hétéromodification

Pour illustrer l'hétéromodification applicable avec les langages à prototypes, nous prendrons à l'image de (Ungar & al 1986), l'exemple d'une application sur les polygones.

Les langages à taxonomie de classes

Si l'on considère la classe générale des polygones, on peut définir ceux-ci par les arcs qui construisent ces polygones. Si ensuite on considère que les rectangles constituent un sous-ensemble des polygones, le choix initial de caractérisation des polygones par leurs arcs constitutifs peut être remis en cause : il est plus commode de considérer un rectangle comme la donnée de quatre points au lieu de quatre arcs. Ainsi, la définition d'une sous-classe des rectangles à la classe des polygones initiale nécessite la mise en place d'un biais illustré par la figure suivante :

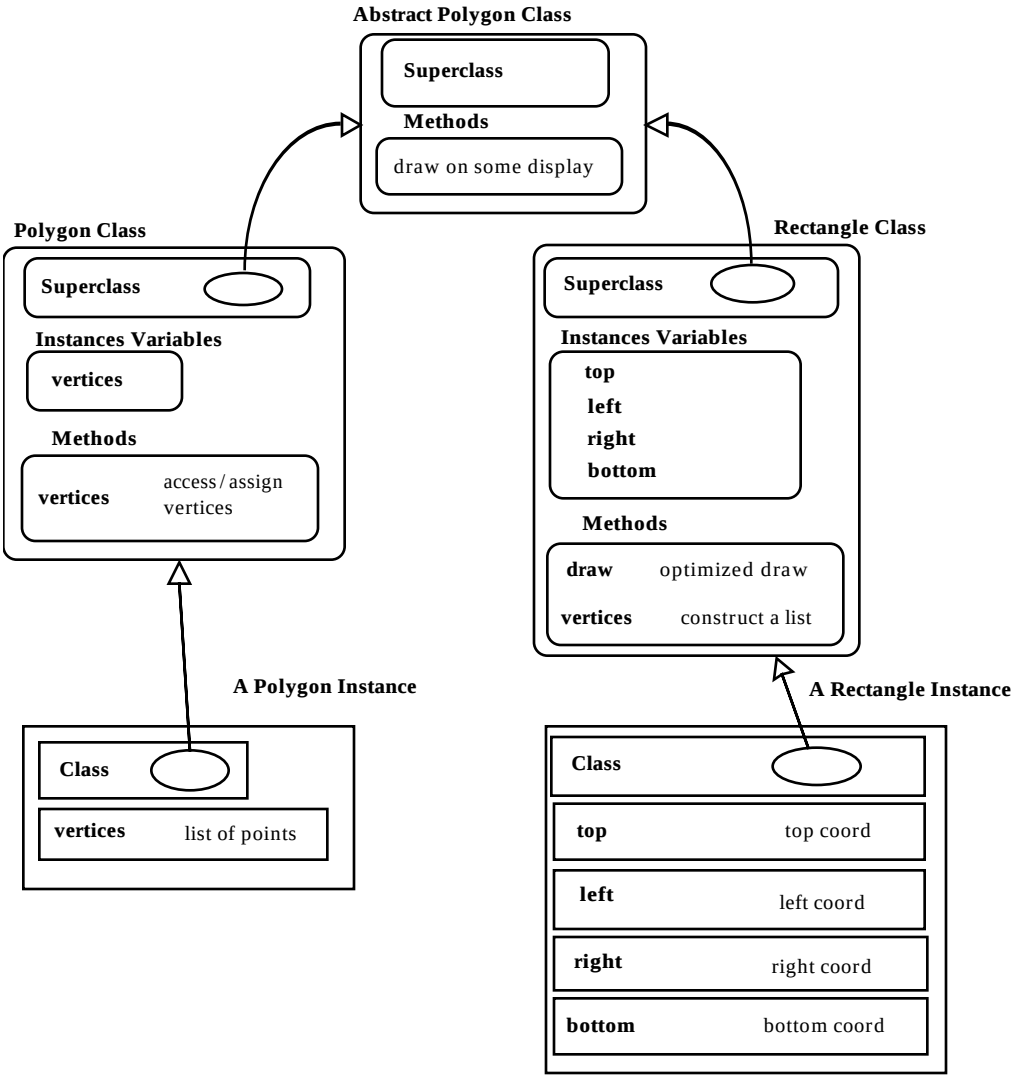


Figure 3.33 : Utilisation de classes abstraites pour prendre en compte des modifications.

Les langages à prototypes

Le prototype rectangle qu'il est possible de définir dans un langage à prototypes prend en compte la donnée des quatre points définissant le rectangle et d'un comportement associé aux prototypes rectangle, il ne comprend pas de définition d'un héritage "parental". Ce choix permet de décider explicitement que la représentation des rectangles diffère de celles des polygones, ce sont les comportements de représentation qui définissent les deux types de constructions des objets.

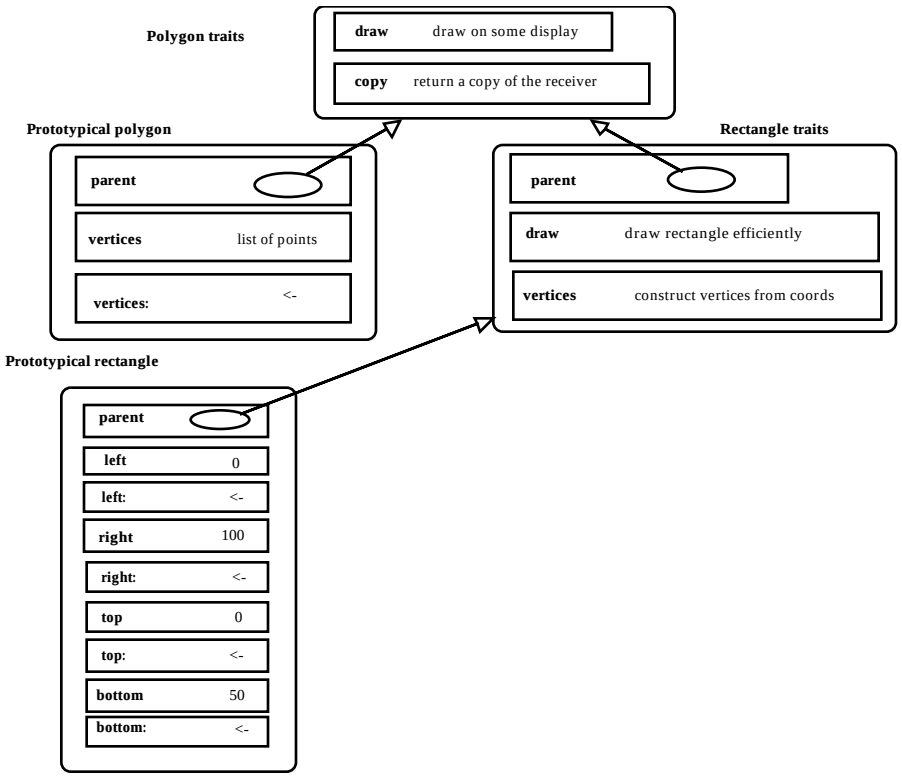


Figure 3.34 : Prototype polygone et modification de comportement.

Le comportement associé au rectangle modifie le comportement initial attaché aux polygones, en construisant une méthode de construction d'arcs par la donnée de quatre points.Si l'on adresse un message `vertices`, si le receveur est un polygone on accèdera aux données définissant les polygones, si le receveur est un rectangle on accèdera aux données définissant un rectangle.

Changer dynamiquement les comportements

Si on souhaite associer des comportements particuliers à certains objets, plusieurs choix sont possibles. Soit on met en place des étiquettes qui détermineront le mode de comportement attaché à cette étiquette. Cette étiquette est testée pour éventuellement déclencher un certain type de comportement. Dans ce modèle de présentation des modes de comportement, il peut s'avérer délicat d'ajouter de nouveaux comportements à moins de modifier une grande quantité de structures déjà construites. Il est aussi possible de considérer directement les comportements comme des alternatives possibles au cadre le plus général défini en utilisant par la suite l'héritage pour associer tel comportement à telle entité.

Dans le cadre des langages de classe, il est coûteux (voir impossible) de permettre à un objet de changer de classe de référence, on ajoute donc des comportements supplémentaires spécialisant les modes de comportements à un niveau très précis de spécification. A l'inverse dans les langages à prototypes, il est possible de modifier dynamiquement une délégation de comportements en utilisant les assignations parentales adéquates. Dans l'exemple des polygones, si on peut définir deux méthodes pour dessiner les polygones et si on choisit de rendre l' attribut `parent` assignable, il est possible de modifier le comportement de représentation graphique des polygones en fonction des résultats souhaités.

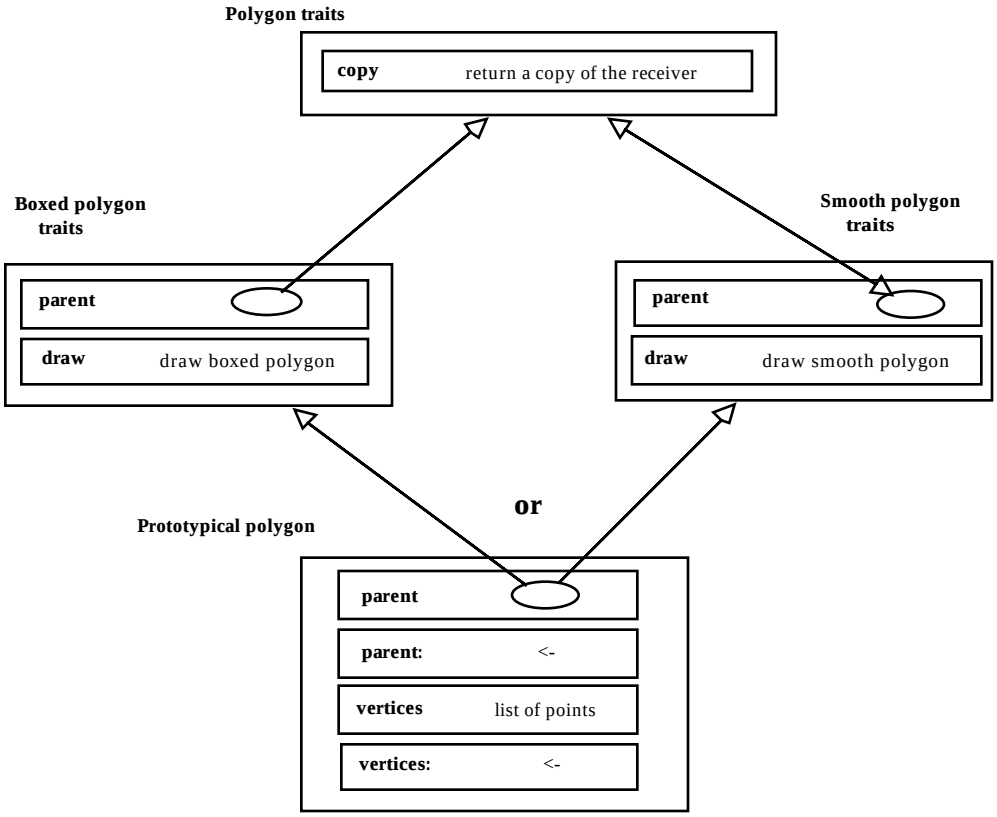


Figure 3.35 : Modification dynamique de comportement.

On perçoit déjà que ce type de mécanisme peut se révéler adéquat pour des processus de représentation des faits de langue : ces derniers manifestent très souvent des différences de comportement dans des contextes particuliers. On peut aussi penser à des mécanismes de classification capables de restructurer les savoirs représentés et de spécialiser les comportements de ces savoirs dans des situations contextuelles données.

3.4.2. Prototype et psychologie cognitive

Les travaux en psychologie cognitive initiés par Rosch visent à mettre à jour les universaux qui régissent l'organisation cognitive. En considérant parallèlement que c'est l'organisation cognitive qui structure le monde qui nous entoure. Ce qui conduit à un ajustement des structures lexicales sur les structures cognitives.

3.4.2.1. Prototype : une version standard

Le prototype comme meilleur exemplaire

La théorie du prototype met en avant la mise au premier plan de propriétés ou attributs prototypiques, saillants, caractéristiques de la catégorie, qui se séparent des conditions nécessaires et suffisantes (CNS) en ce qu'elles ne sont pas nécessaires.

"Le concept de typicalité traduit donc le fait que, en mémoire humaine, les représentations des différents exemplaires d'une catégorie ne sont pas identiques, mais se distribuent selon leur degré de représentativité par rapport à un exemplaire particulièrement typique ou représentatif, le prototype. La typicalité n'est pas adéquatement décrite par des conditions nécessaires et suffisantes d'appartenance mais en termes de distance au prototype, encore désignée par les notions de ressemblance, de similitude ou d'air de famille" (Dubois 86).

Dans une tradition classique de la catégorisation, les exemplaires d'une catégorie sont liés par une relation d'équivalence : l'appartenance à une catégorie est fondée sur le partage de propriétés communes et ces propriétés sont nécessaires et suffisantes pour valider l'appartenance à la catégorie. Les travaux de Rosch (Rosh 1975) ont montré que les catégories i.e "l*es représentations sémantiques d'objets du monde réel*" (Rosch 1976) sont organisées dans des structures hiérarchiques de type taxonomique et les catégories se déterminent par des relations de discrimination inter-catégorielle et non pas de manières indépendantes les unes par rapport aux autres selon des critères de conditions nécessaires et suffisantes d'appartenance.

Prototype et catégorisation

La version standard prend appui sur les thèses suivantes (Kleiber 1991) :

- La catégorie a une structure interne prototypique.

Les travaux et expériences de Rosch introduisent donc la notion de prototype comme étant le meilleur exemplaire ou encore la meilleure instance, le meilleur représentant ou l'instance centrale d'une catégorie. *"L'idée fondamentale est que les catégories ne sont pas constituées de membres "équidistants" par rapport à la catégorie qui les subsume, mais qu'elles comportent des membres qui sont de meilleurs exemplaires que d'autres"* (Kleiber 1991).

- Le degré de représentativité d'un exemplaire corespond à son degré d'appartenance à la catégorie.

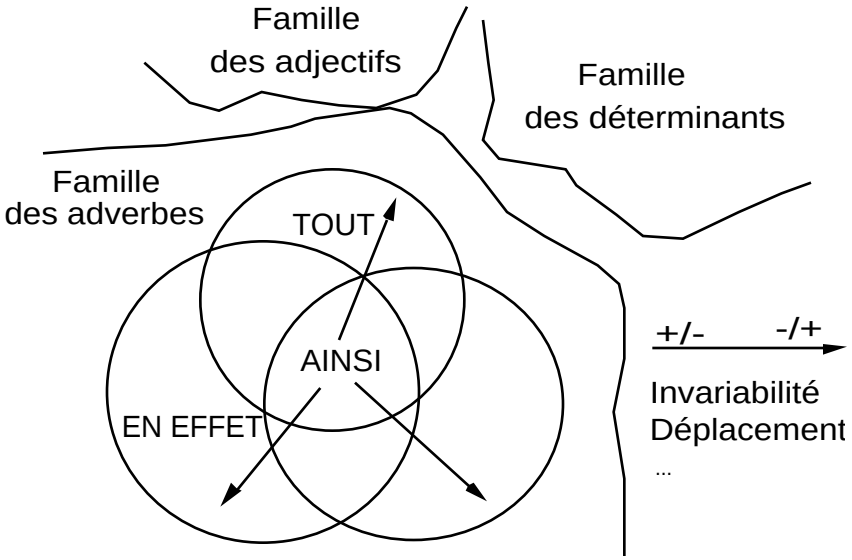
"Le prototype fournit en même temps le principe de catégorisation : les entités sont rangées dans une catégorie selon leur degré de ressemblance avec le prototype. Le degré de représentativité devient ainsi équivalent de degré d'appartenance dans des catégories conçues comme ayant des frontières floues. La notion de prototype est elle-même sujette à des glissements définitoires qui la font passer de "meilleur exemplaire" à "combinaisons de propriétés typiques" (Kleiber 1991).

- Les frontières des catégories ou des concepts sont floues.

- Les membres d'une catégorie ne présentent pas des propriétés communes à tous les membres; c'est une ressemblance de famille qui les regroupe ensemble.

- L'appartenance à une catégorie s'affectue sur la base du degré de similarité avec le prototype. Elle ne s'opère pas de façon analytique, mais de façon globale.

L'opération de catégorisation est une opération d'appariement et un non une opération de vérification de CNS. Le prototype sert de point de référence cognitif pour les catégories et systèmes de classification. Dans la langue, les comportements adverbiaux respectifs de "ainsi" et de "tout" ne sont pas équivalents.



La catégorie est organisée autour d'un meilleur exemplaire, le prototype.
"ainsi" est plus prototypique que "tout" dans la famille des adverbes

Le niveau de base

Les objets des catégories naturelles seraient donc plus adéquatement décrits par des corrélats d'attributs et des relations de ressemblance perceptive ou fonctionnelle que par des listes de traits indépendants les uns des autres. Et à un même niveau d'organisation, il est possible de définir les exemplaires les plus représentatifs -les prototypes-, qui possèdent une place centrale dans l'organisation des catégories : c'est autour des prototypes que les catégories s'organisent. On définit un gradient de typicalité[53] pour évaluer la représentativité d'un exemplaire par rapport au prototype qui définit la classe.

Le niveau de base[54] est le "*niveau d'abstraction par lequel les catégories possèdent le plus d'attributs en commun qui impliquent des mouvements communs, qui présentent une similitude de formes objectives et qui sont identifiables à travers le dessin des formes moyennes*" (Dubois 1986).

3.4.2.2. Prototype : une version étendue

Les travaux initiés par Rosch ont été révisés par Rosh et plus tard par Lakoff (1986 et 1987) et ont remis en cause une part importante des hypothèses faites sur l'organisation catégorielle. Dans cette version étendue, Rosch n'assigne plus au prototype le rôle de repère central pour la catégorie : "*Les prototypes ne constituent pas un modèle de calcul particulier pour les catégories*" (Rosch 1978). La seconde version du prototype "*relègue le prototype au rang d'effet. Il n'est plus le principe structurant la catégorie et ne donne plus non plus la réponse à la question de la catégorisation. Corollairement, les thèses du vague des catégories et de l'équivalence entre degré de représentativité et degré d'appartenance se trouvent abandonnées. La notion de ressemblance de famille, déjà présente dans la version standard, est à l'origine de la constitution de la version étendue. Elle a toutefois pour conséquences cruciales, d'une part, un détournement de la notion de prototype qui, même comme phénomène superficiel, arrive à perdre son trait définitoire initial de* ***meilleur représentant pour les sujets*** *au profit de simple trait* ***central*** *ou* ***basique****, et, d'autre part, un élargissement*

---

53 *"Le degré de typicalité peut s'approcher au travers d'expérimentation simple. Demander à des sujets de citer un type de pomme illustrant la catégorie des pommes peut donner des résultats convergeant pour désigner "la golden" comme élément "plus typique" de l'ensemble des pommes... Le niveau d'abstraction renvoie au fait que lorsqu'on demande à des sujets de nommer hors contexte une entité, ils choisiront volontiers des termes situés à un certain niveau d'abstraction - "c'est une vache", "c'est un poisson" -, et qui n'est pas forcément le niveau scientifique classificatoire communément enseigné : vache = animal = ruminant = mammifère. C'est ce niveau d'abstraction intermédiaire qu'on nomme communément le "niveau de base", et qui exprimerait l'existence de sous-catégories opératoirement plus représentatives de la catégorie ou que la catégorie elle-même, impliquant donc qu'il suffit, pour évoquer ou résumer une catégorie, de mentionner ses sous-catégories les plus typiques"* (Vignaux 1991).

54 *"Le codage des catégories au niveau de base au moyen des prototypes semblerait être la forme de représentation cognitive qui peut fournir la plus grande information sur la catégorie que l'on peut retenir en une seule image concrète"* (Rosh & Lloyd 1978).

aux items polysémiques qui entraîne une assimilation indue entre catégorie conceptuelle (ou référentielle) et catégorie (linguistique) de sens. La version étendue constitue par là même une rupture par rapport à la version standard" (Kleiber 1991). En refusant au prototype d'être l'entité organisatrice d'une catégorie, on lui enlève aussi tout pouvoir pour expliquer l'appartenance d'une entité à une catégorie. Le fait de renoncer à considérer le prototype comme étant directement la représentation des concepts a pour conséquence immédiate de lever l'exigence d'une représentation prototypique du sens d'un terme. Du coup, les représentations ou définitions les plus diverses retrouvent droit de cité. La version étendue s'écarte donc radicalement de la version standard sur les points suivants :

- La notion de prototype comme représentant des concepts des catégories et comme structuration de la catégorie disparaît : la notion de prototype ne répond plus à la définition initiale du prototype comme meilleur exemplaire.

- La relation qui unit les exemplaires d'une catégorie est une relation de ressemblance de famille : elle aboutit à une conception catégorielle différente où ce n'est plus le concept ou son correspondant sur le plan sémantique, l'acceptation ou le sens, qui constituent l'indicateur d'une catégorie, mais l'unité lexicale. Il en résulte une version polysémique ou multi-catégorielle qui, plutôt que d'expliquer pourquoi telle ou telle entité particulière appartient à telle ou telle catégorie, rend compte de ce qu'un même mot peut regrouper plusieurs sens différents, c'est-à-dire peut renvoyer à plusieurs types de référents ou de catégories.

La notion de ressemblance de famille laisse donc de côté la notion première de représentation prototypique. En fixant le prototype comme le pivot d'appariement référentiel, la version standard en reste à une conception de la catégorie comme regroupant un type de référents : le fait que tous les exemplaires de la catégorie doivent partager au moins un trait du prototype empêche l'éclatement de la catégorie en des sous-catégorie pouvant ne plus rien avoir de commun entre-elles. A l'inverse, la version étendue ouvre la possibilité d'une conception de la catégorie comme une entité multi-catégorielle : "*La polysémie apparait comme étant un cas spécial de la catégorisation à base prototypique, où les sens d'un mot sont les exemplaires d'une catégorie. L'application de la théorie du prototype à l'étude du sens des mots met de l'ordre là où il n'y avait auparavant que chaos*" (Lakoff 1987). La notion de ressemblance de famille permet à la théorie du prototype de s'appliquer à tous les phénomènes de catégorisation polysémiques. Le mot "veau" regroupera dans une même catégorie les sous-catégories référentielles "animal", "viande de cet animal"... L'appellation générique s'explique par des principes généraux référentiels d'enchaînement et d'association qui produisent les similarités nécessaires à l'intégration dans la même famille catégorielle. La situation du prototype est modifiée avec cette définition catégorielle polysémique : le prototype est réduit à un phénomène de surface, il peut prendre diverses formes et il n'est plus l'exemplaire qui est nécessaire comme étant le meilleur exemplaire de référence pour la catégorie.

"Toutes ces approches en termes de représentations privilégiées n'ont d'autre pertinence qu'en regard de classements hiérarchisés généraux de ce qui serait des types, ordonnant à leur tour des représentations locales. Cette hiérarchisation est supposée rendre compte d'une stabilité de l'organisation psychologique des informations sémantiques. La séduction de l'idée provient de ce qu'après tout elle ne fait que retrouver des conceptions anciennes et même aristotéliciennes des fonctionnement catégoriels de notre esprit, en supposant, pour ce faire des entités stables de référence sans trop se préoccupper, en dépit des nombreux biais cognitifs observés, des procédures mêmes qui quotidiennement vont travailler ces entités dans l'échange psychologique entre humains" (Vignaux 1991).

3.4.2.3. PàP vs. Prototypie cognitiviste

A ce niveau de la présentation de la PàP, il convient de préciser que l'approche de la PàP n'a rien à voir avec celle adoptée par les cognitivistes tels que Rosch dans leur présentation de la prototypie.

Dans la version standard des prototypes développée par Rosch, les catégories sont essentiellement hétérogènes. Elle sont organisées autour d'un meilleur exemplaire : le prototype. Et les éléments d'un catégorie sont rangés en fonction de leur degré de ressemblance avec le prototype.

Par rapport à cette version, la PàP se distinguent sur les points suivants :

- Dans la PàP, on n'a pas de meilleur représentant de la catégorie représentée, la catégorie ne s'articule donc pas autour d'un meilleur représentant.
- L'opération de clonage produit une copie conforme de l'objet cloné : la notion de degré de ressemblance par rapport à un élément privilégié ne se pose donc pas.
- Dans la PàP, l'opération du clonage invalide la notion de flou des catégories mis en avant dans la prototypie cognitiviste.
- La PàP met en place une représentation à plat, sans hiérarchie ou niveau privilégié.

Dans la version étendue des prototypes développée par Rosch, le lien qui existe entre les éléments d'une catégorie est une relation de ressemblance de famille, la notion de prototype devient résiduelle ou superficielle. Là encore, la PàP se distingue de cette version parce qu'elle met en avant, via l'opération de clonage, la ressemblance tout court entre l'objet et son clone.

Incise Chapitre 3

Incise 1. Grammaire basée sur l'unification

Structure d'information

Les grammaires d'unification utilisent toutes des ensembles structurés de traits ou structures catégorielles, auxquels sont associées des valeurs, qui peuvent constituer elles-mêmes des traits.

```
structure catégorielle:= { <association trait-valeur> +}
 <association trait-valeur>::= {<nom de trait><valeur>}
 <nom de trait>:= <atome>
 <valeur>:= <atome> | <structure catégorielle>
```

Il n'est pas possible pour un même trait de recevoir deux valeurs différentes dans la même structure catégorielle :

```
{{catégorie nom}{genre masculin}{genre féminin}}
```

est ainsi une structure catégorielle mal formée.

On pourra ainsi noter l'association au trait `nombre` de la valeur `singulier` et de la valeur `féminin` au trait `genre` :

```
{{nombre singulier}{genre féminin}}
```

ou encore :

```
nombre : singulier
genre : féminin
```

Une autre notation possible, sous la forme de graphe, est :

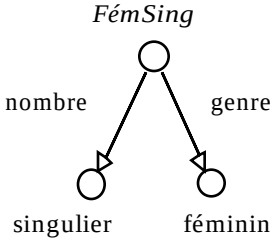


Figure 3.8 : `FémSing`

Les valeurs d'un trait donné peuvent constituer elles-mêmes un ensemble structuré (qu'on appellera par exemple NomFémSing) :

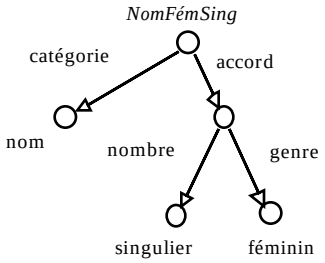


Figure 3.9 : NomFémSing

```
catégorie : nom
accord :
     nombre : singulier
     genre : féminin
```

Le (sous-)trait accord a pour valeur l'ensemble structuré vu précédemment (FémSing). On parlera de structure simple pour l'association catégorie : nom et de structure de traits complexe pour FémSing ou pour NomFémSing.

Dernière caractéristique : dans un ensemble de traits, deux (ou n) traits peuvent partager une valeur donnée (ce qui est distinct de l'égalité de valeur pour les deux traits : dès qu'un des traits prend une valeur donnée, l'autre la prend également).
Dans

```
Catégorie : GN
accord : 1
Tête :
Catégorie : N
accord : 1
Modifieur :
Catégorie : Adj
accord : 1
```

L'indexation par 1 indique que le Groupe Nominal constitué d'un Det, d'un N et d'un Adj se caractérise par le partage de la valeur du trait accord entre l'Adj, le N, le Det et le GN. On exprime ainsi de manière très simple la cohésion en genre et nombre entre les composants d'un Groupe Nominal, et entre ceux-ci et le Groupe constitué.

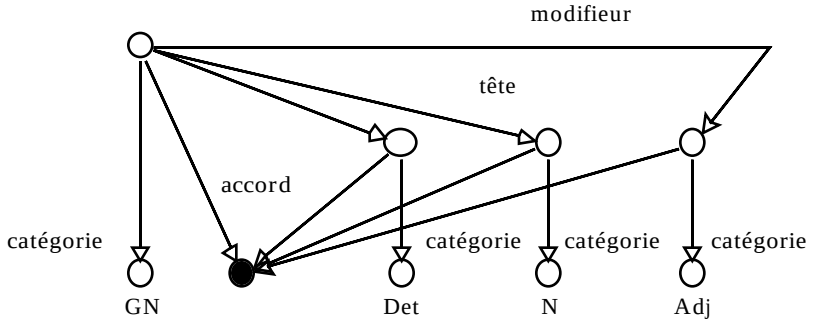


Figure 3.10 : Partage de valeurs entre DAGs

Conventions : D(f) notera la valeur associée au trait f dans la structure D : par exemple, `FémSing(genre) = féminin`. Un chemin dans une structure est une suite de traits permettant d'accéder à une sous-structure : `NomFémSing(<accord nombre>) = singulier`.

Ce type de notation de structures informationnelles complexes présente plusieurs avantages :

- En premier lieu, le linguiste qui associe de telles structures à des entrées du lexique dispose pour ce faire de traits d'une arbitraire complexité, qui peuvent a priori satisfaire des besoins très divers (les informations sémantiques s'y intègrent par exemple harmonieusement).

- Deuxièmement, les structures obtenues à un moment donné ne sont pas figées pour autant. Leur caractéristique principale est au contraire l'ouverture. Il est toujours possible, à quelque niveau que ce soit, de rajouter des informations qui viennent compléter celles qui existent sans que le "format" des entrées du lexique soit à reconsidérer en même temps. Or l'expérience montre que les traits attribués à des entrées lexicales sont rarement satisfaisants du premier coup : il est donc particulièrement important de pouvoir procéder par retouches et ajouts, voire par changement radical de cap.

- Dans l'écriture de règles de grammaire et dans la manipulation de traits qui en résultent, on peut ne s'intéresser qu'au sous-ensemble de traits qui sont pertinents à ce moment-là, sans avoir à penser à la totalité des structures catégorielles qui sont en jeu.

- Enfin, les traits manipulés sont explicites. Ce ne serait pas le cas si les structures révêtaient la forme suivante : `<valeur trait1><valeur trait2> .. <valeur trait n>`. Par exemple, avec les 5 traits suivants : `catégorie, sous-catégorie, genre, nombre, lemme,` une catégorie possible pour `monstres` est : `nom commun masculin pluriel MONSTRE`. Cette formulation oblige à retenir une disposition particulière des valeurs permettant de savoir que la première représente la catégorie syntaxique, la dernière le lemme etc. Dans la formulation explicite, il suffit de suivre le chemin qui y mène pour comprendre à quel trait renvoie telle valeur.

```
catégorie :
     principale : nom
     sous-catégorie : commun
accord :
     genre : masculin
     nombre : singulier
lemme : MONSTRE
```

Unification

L'unification consiste, de manière intuitive, à combiner deux (n) structures compatibles pour obtenir une nouvelle structure éventuellement plus spécifiée, porteuse alors de davantage d'informations. Soit les entrées de lexique `le` et `ferme`, et les structures de traits associées :

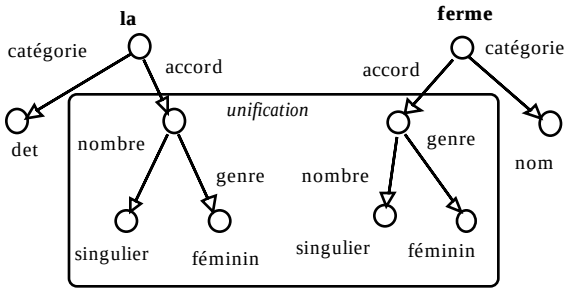


Figure 3.11 : Unification possible sur le trait `accord`

L'unification entre leurs deux sous-structures `accord` est possible. C'est :

```
accord :
 nombre : singulier
 genre : féminin
```

Par contre, l'unification du trait `accord` de `le` est impossible avec le trait accord de `ferme` :

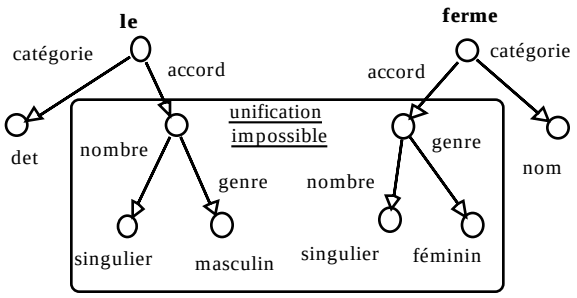


Figure 3.12 : Unification impossible sur le trait `accord`

L'unification produit une structure plus "informative" quand les deux structures mises en relation portent des informations qui se complètent sans se contredire. Ainsi l'unification du trait accord de `le` avec celui de `voile` (le genre n'est pas spécifié, puisqu'il peut s'agir d'un ou d'une voile) :

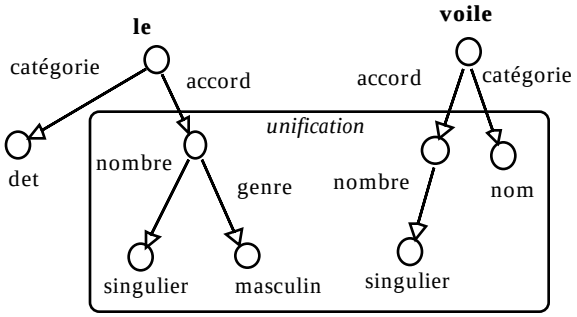


Figure 3.13 : Unification ajoutant de l'information

produit la structure suivante :

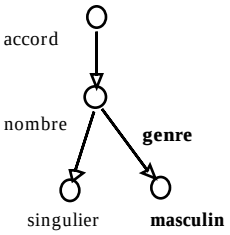


Figure 3.14 : Précision du trait `accord` de `voile` grâce à l'unification

Il faut ajouter au DAG associé à `voile`, pour le chemin `<accord genre>` la valeur `masculin`, pour que les deux sous-DAGs accord de `voile` et de `le` soient unifiables, i.e. compatibles. Quand elles ne se complètent ni se contredisent, l'unification réussit, mais sans enrichir les structures de départ (c'est le cas des traits `accord` de `la` et `ferme` supra). Elle fonctionne différemment selon qu'elle s'applique à des valeurs similaires ou à des valeurs partagées. Le résultat est le même quel que soit l'ordre dans lequel les unifications sont opéreés entre n structures. Cette indépendance du résultat par rapport à l'ordre choisi rend l'unification particulièrement apte à servir dans des types d'analyseurs extrêmement variés : un formalisme basé sur l'unification ne présuppose rien sur la stratégie d'invocation des règles de réécriture (

approche montante / descendante) ni sur l'exploration des alternatives (en largeur / en profondeur d'abord), ni enfin sur la direction d'exploration de la phrase (gauche-droite, droite-gauche, bi-directionnelle).

L'unification produit une structure plus "informative" quand les deux structures mises en relation portent des informations qui se complètent sans se contredire. L'unification est une opération de combinaisons qui repose sur la notion de subsomption et peut être comparée à l'opération de l'union dans la théorie des ensembles. On dit qu'une structure de traits T1 subsume une structure de traits T2 si T2 est au moins aussi informative que T1. On définit l'unification de deux structures catégorielles de traits T1 et T2 comme la structure de traits T la plus générale telle que T contienne T1 et T2.

Règles de réécriture et contraintes additionnelles : DAGs

Des règles vont permettre de construire des constituants plus importantes à partir de constituants de base, à condition en outre que puisse se réaliser l'unification de traits spécifiés liés aux éléments des règles. La construction de constituants plus amples se fera par le recours à des règles du type de celles de grammaires indépendantes du contexte. La combinaison des traits se fera par l'indication d'unifications à réaliser entre des sous-structures des structures associées aux éléments de la règle. On suit la notation de PATR-II, le formalisme de Shieber, qui se veut selon les mots mêmes de Shieber *"a powerful, simple, least common denominator of the various unification-based formalisms"*.
Soit la règle :

```
GN -> DET NOM
<GN accord> = <NOM accord>
<DET accord> = <NOM accord>
```

La première ligne de la règle indique les constituants à concaténer dans cet ordre pour produire un GN. La partie inférieure de la règle[55] rajoute un certain nombre de contraintes sur tout ou partie des structures catégorielles associées à ces constituants. Cette règle appliquée à les voile (c'est-à-dire plus précisément aux structures de traits associées à ces entrées du lexique).

```
les
      catégorie det
      accord :
      nombre : pluriel
voile
      catégorie nom
      accord :
      nombre : singulier
```

ne peut qu'échouer, tandis qu'avec la voile, elle réussit, et le trait accord du GN a désormais pour valeur :

```
accord :
      nombre : singulier
      genre : féminin
```

tout comme le trait accord du N (l'unification a enrichi l'information associée à voile dans le texte, et concrètement l'a désambiguïsé pour ce qui regarde le genre). On voit comment de tels formalismes rendent aisé le traitement des phénomènes d'accord : La règle ci-dessus est une abréviation commode (correspondant aux habitudes des lecteurs de grammaires hors contexte) pour :

```
X0 -> X1 X2
<X0 catégorie> = GN
<X1 catégorie> = DET
```

[55] Suivant en cela (Shieber 1986), le signe = renvoie à un essai d'unifier les structures qui l'encadrent.

```
<X2 catégorie> = NOM
<X0 accord> = <X2 accord>
<X1 accord> = <X2 accord>
```

C'est-à-dire que les symboles utilisés dans la partie hors contexte de la règle sont en fait la valeur du trait `catégorie` des DAGs mis en relation par la règle.

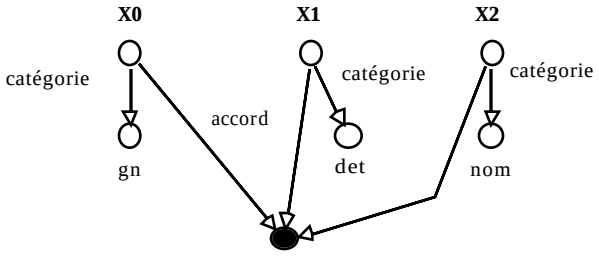


Figure 3.15 : Règle PATR-II "expansée"

On préférera généralement la première forme, plus claire. Dans certains cas, toutefois, l'utilisation de variables (notées X indicé : X_i) dans la règle permet d'écrire une règle générale de manière concise et élégante. C'est le cas par exemple pour la coordination, qui pourrait se traiter ainsi (en adaptant la présentation de (Gazdar & Mellish 1989, p 117) :

```
X0 -> X1 Coord X2
<X0 catégorie> = <X1 catégorie>
<X0 catégorie> = <X2 catégorie>

avec

   et
        <cat> = Coord
   ou
        <cat> = Coord
```

Cette règle exprime en bref le fait qu'une catégorie donnée peut être constituée de deux instances de cette même catégorie reliées par un mot de `<catégorie> = Coord`, c'est-à-dire la coordination pour toutes les parties du discours et pour toutes les catégories syntagmatiques. On imagine toutes les règles que celle-ci subsume :

```
P -> P Coord P
GN -> GN Coord GN
GV -> GV Coord GV
Adj -> Adj Coord Adj
```

etc.

En dernière instance, c'est dans le lexique utilisé que se trouve l'association initiale entre des chaînes et des structures de traits et de valeurs.

Unification et analyse automatique

Les formalismes d'unification (PATR notamment) permettent donc de spécifier des grammaires dont les catégories sont des structures de traits. Et ce type de formalisme sert directement pour une utilisation informatique. Cette utilisation informatique se décompose en général en deux étapes, une première étape de compilation puis une étape d'analyse. La compilation sert à optimiser l'efficacité de l'analyseur en

transformant une entrée lexicale en un objet représentant la structure de traits implémentant cette entrée et une règle en une séquence d'objets représentant la séquence de structures de traits implémentant cette règle. On donne en annexe une illustration d'analyse automatique avec OLMES (-> annexe chapitre 3).

Unification et représentation des connaissances

Les grammaires d'unification tendent à concentrer l'information dans le lexique. Il convient donc de définir très rationnellement le cadre de représentation des connaissances dans ce type de formalisme. Dans la manière de concevoir les connaissances, les grammaires d'unification se caractérisent donc comme évolutives avec mise à jour à la main et non modulaires. Les règles de réécriture même enrichies occupent une place centrale dans ce type de formalisme, ce qui rend délicat l'articulation du lexique et de la syntaxe. Le lexique a longtemps constitué un receptacle des irrégularités de la langue, i.e de toutes les données qui résistent au codage sous forme de règles générales. Les formalismes du TALN ont eu tendance dans leur développement récent à accroître la lexicalisation de ces formalismes (Karttunen 1989) en associant des règles de combinaison simples à des structures lexicales complexes (Pollard & Sag 1987). Le mode de combinaison des entrées lexicales qui peuvent jouer le rôle de têtes est alors exprimé au niveau du lexique. Chaque emploi d'une entrée tête précise ses propres contraintes de combinaison. On élimine ainsi le recours à des règles générales inadéquates car conditionnées par des phénomènes lexicaux. Cette tendance amène une multiplication d'emplois distincts c'est à dire un traitement de l'homonymie. De plus, cette lexicalisation implique d'une part la mise en place de mécanismes d'expression lexicale aptes à limiter les redondances (Kameyama 1988) (Daelemans & al. 1992) et d'autre part à réduire la dissémination du sens en rendant compte prioritairement des régularités lexicales pour limiter les effets pervers d'un traitement homonymique du sens. Ces préoccupations sont traités grâce à des méthodes de représentation des généralisations et de leur contre-coups (exceptions et blocages) basées sur le recours à l'héritage et à la non-monotonie. On présentera (-> partie 3 chapitre 9) une illustration de ces techniques de représentation.

Héritage simple

Une forme-type sert à décrire un ensemble de propriétés communes à plusieurs entrées lexicales : on peut ainsi "capturer" des généralisations s'appliquant à une classe d'éléments.

Héritage par défaut

Il est parfois souhaitable qu'un noeud dans le réseau défini n'hérite que d'une partie des informations contenues dans les noeuds qui le dominent.

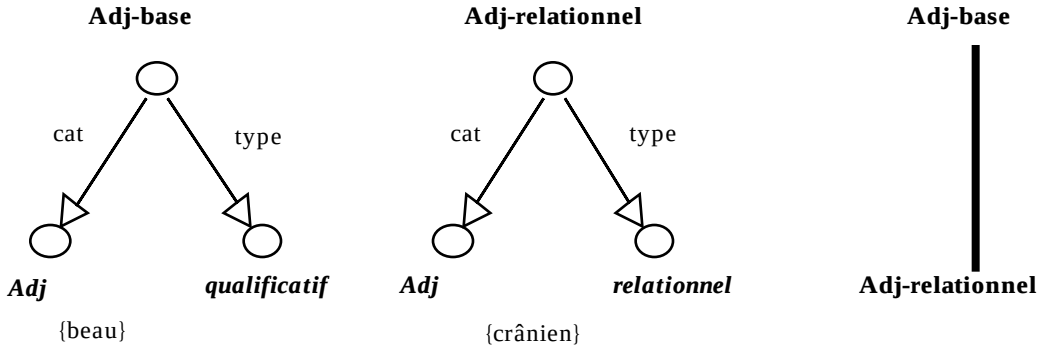


Figure 3.16 : Héritage et redéfinition.

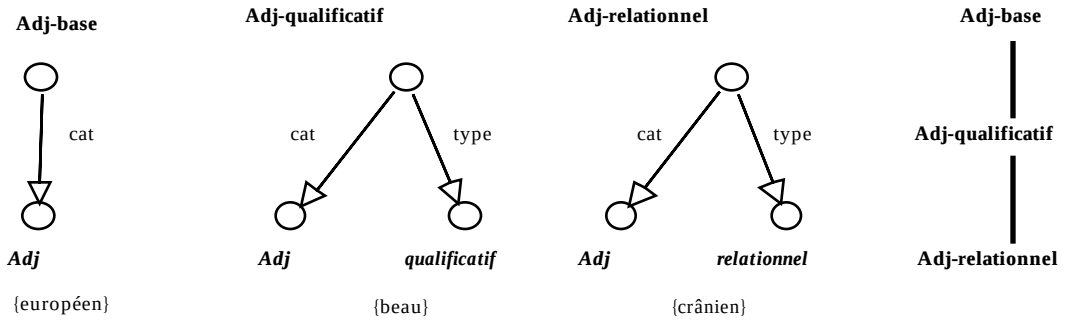
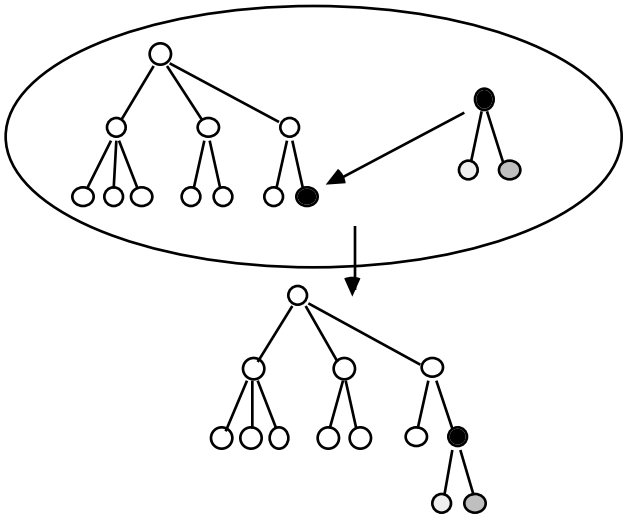


Figure 3.17 : Héritage et ambiguïté.

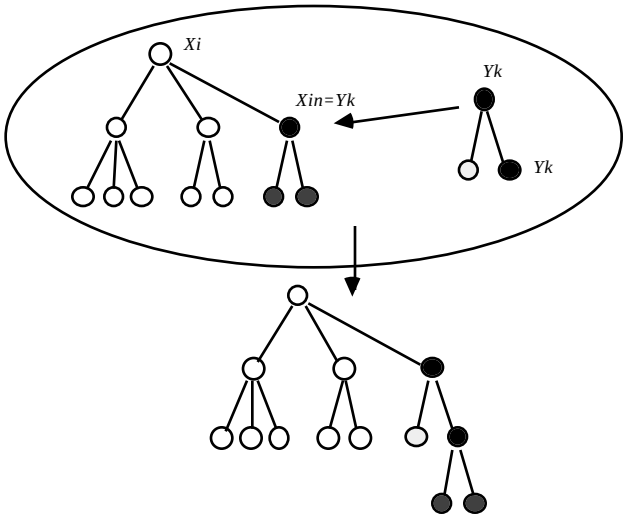
Incise 2. Les grammaires d'arbres adjoints (TAG)

TAG est un formalisme arborescent (Joshi 1987) qui manipule des structures élémentaires plus étendues que les grammaires basées sur des règles de réécriture hors contexte. Une grammaire d'arbres adjoints est un ensemble d'arbres élémentaires. Ces arbres peuvent être de deux types : les arbres initiaux et les arbres auxiliaires. Ces arbres sont de profondeur quelconque et ils ont à leurs feuilles des terminaux ou des noeuds à substituer. Les arbres auxiliaires possèdent une feuille (appelée `pied`) portant un noeud non terminal et de même catégorie que leur racine. La langage TAG est produit par la combinaison de ces arbres au moyen de deux opérations: la substitution et l'adjonction. Les arbres obtenus sont appelés les arbres dérivés.

- L'opération de substitution insère un arbre initial (ou dérivé d'un arbre initial) à la frontière d'un arbre élémentaire ou dérivé.



L'opération d'adjonction insère un arbre auxiliaire (ou dérivé d'un arbre auxiliaire) à un noeud quelconque de même catégorie dans un arbre élémentaire ou dérivé.



Les grammaires d'arbres adjoints lexicalisées ont ajouté une contrainte supplémentaire au formalisme initial : la lexicalisation. Dans cette grammmaire, tout arbre élémentaire doit avoir à sa frontière au moins un terminal servant de tête lexicale et la catégorie de cet item lexical est cet arbre élémentaire. Le

formalisme des TAGs lexicalisées (et avec unification (Vijay Shanker 1987, Abeillé 1989)) est utilisé par un analyseur Earley (Earley 1986). Il est incorporé à un environnement graphique interactif qui permet l'affichage des arbres manipulés et la mise à jour des grammaires et du lexique. Ce dispositif fonctionne sur Sun (Paroubek & al.1992).

Chapitre 4

Un dispositif ou le dur désir de représenter

C'est à l'étude des problèmes de représentation et construction du sens des configurations du type `moulin à N` dans une analyse automatique qu'est consacré la première partie de notre travail. Celui-ci vise donc à construire un dispositif pour représenter matériellement les phénomènes complexes en jeu pour une modélisation des connaissances de telles expressions et en particulier les phénomènes décrivant la construction du sens de ces expressions. Notre approche se situe dans le cadre d'une représentation des connaissances et d'une construction du sens des fragments de langue étudiés par un affinage progressif de ces mécanismes de représentation et de construction.

Dans les approches retenues pour la construction d'un dispositif pour l'étude des séquences `moulin à N2`, notre propos sera le suivant :

> Un dispositif pour une analyse automatique des formes `moulin à N2` doit identifier, classer et décrire ces formes pour construire une taxonomie ouverte et évolutive de cette classe particulière de composés.

On doit donc étudier les procédures de traitements des rapports forme-sens qui sont à l'oeuvre dans les dispositifs informatiques construits : comment à partir de représentations statiques du lexique, mettre en place des opérations dynamiques pour l'interprétation?

4.1. Préambule : Le dispositif est au bord d'un gouffre

Du désir de représenter l'excès dans la langue au possible de représentation dans un dispositif

Sommes-nous à même aujourd'hui de représenter le langage naturel? La réponse est clairement non. Des représentations locales, partielles sont possibles, elles ne sont pas à même de prendre la mesure de l'étendue des possibles dans les énoncés. Si les faits de langue maximalisent les informations, la finesse de ces informations, leurs connexions, ..., leurs représentations tendent malgré tout à les réduire.

La langue use ses représentations

La langue évolue : construire des représentations matérielles du langage naturel, c'est donc être contraint à réviser en permanence les représentations. De plus, à un moment donné on n'en sait jamais assez sur un fait de langue; notre savoir évolutif sur la langue peut donc conduire à modifier les résultats initiaux. De même,

le développement de techniques nouvelles peut aussi amener à interroger les outils utilisés pour la représentation.

Le complexe de la langue n'est pas convertible dans "une monnaie" de représentation à "taux fixe"

Il faut donc s'attacher à des esquisses de représentation. Qu'il s'agit ensuite de faire évoluer : "dégrossir les esquisses". Ce qui est représenté aujourd'hui, peut être à re-représenter demain. De même pour la représentation de la construction du sens : ce qui sensé aujourd'hui peut devenir non-sens demain.

Le fossé entre le complexe de la langue et les outils pour sa représentation est considérable

La principale difficulté à laquelle est confronté le travail de représentation des faits de langue tient dans le rapport inégal qui existe entre d'une part, la complexité des savoirs qui agissent dans ces faits de langue et d'autre part, les outils disponibles pour la mise en place d'une telle représentation. Les énoncés linguistiques réalisent une imbrication des différents types de savoirs qu'ils véhiculent. Leur articulation est partie intégrante de la structure de ces énoncés. La richesse, la finesse des savoirs qui participent dans un fait de langue ne se laissent pas appréhender facilement. On est donc loin d'être en mesure de représenter ce qu'un être humain est capable de faire dans son activité de production d'énoncés. Vouloir représenter une telle complexité de phénomènes met en lumière la nécessité de disposer d'outils adéquats pour saisir les interactions de savoirs qui se mettent en place dans les énoncés visés. Les outils informatiques existants ne permettent pas encore de prendre la mesure de la réalité complexe d'un fait de langue. De plus, comme ces outils pour la représentation matérielle du langage naturel ne nous sont pas fournis de "toutes pièces", il reste à les construire, tout au moins à essayer de le faire. Et c'est dans la réalisation de cette construction d'outils pour la représentation que l'on prend la mesure du gouffre qui existe entre l'objet à représenter et les moyens à mettre en oeuvre pour cette représentation.

Le dispositif ou l'impatience de représentation

Un dispositif pour le TALN ne peut donc pas encore réaliser la représentation de l'excès de la langue, il en reste au possible du représentable.

4.2. Un dispositif qui construit, utilise et modifie des hiérarchies

Un dispositif qui privilégie un cadre de représentation des connaissances sous forme d'objets dans une taxonomie de classes a du mal à prendre en compte le fait que dans un domaine particulier où les connaissances ne sont pas encore clairement établis, il faut faire évoluer les représentations initiales pour construire pas à pas un modèle possible. Si les éléments constitutifs d'un modèle de représentation sont organisés de manière hiérarchique, il est souvent difficile de modifier cette hiérarchie quand les observations révélées par l'expérience révèlent une telle évolution[56]. La modélisation d'un domaine de connaissance sous forme d'une hiérarchie de classe se révèle délicate pour représenter un domaine où les connaissances sont peu (ou ne sont pas) organisées de manière définitive[57].

4.2.1. De la complexité de l'objet à représenter

S'il s'agit de représenter les éléments constitutifs d'une analyse automatique pour la construction du sens, on est confronté à une masse de phénomènes. Les interférences lexicales, syntaxiques, sémantiques, contextuelles sont nombreuses et complexes entre les différents mécanismes qui réalisent la construction du sens d'un énoncé.

4.2.2. Organiser le savoir lexical

Le début des années 80 a vu la montée de formalismes qui ont donné une place centrale au lexique, et par conséquent à des structures de traits complexes, et à l'unification sur ces structures. L'utilisation de tels formalismes pose clairement le problème de représenter les articulations forme-sens qui se jouent au niveau des éléments lexicaux : penser l'organisation lexicale. Une organisation hiérarchique avec héritage de propriétés pose de nombreux problèmes en raison justement de la complexité des phénomènes à représenter. Dans notre domaine particulier des séquences `moulin à N2`, la construction d'une telle hiérarchie n'est pas triviale et pousse à réfléchir sur l'organisation des connaissances dans un tel cadre. Là encore il s'agit de bien comprendre que ce n'est pas tant de construire une hiérarchie qui importe le plus (ce n'est d'ailleurs pas le plus difficile, avec la programmation objet tout est objet, i.e. on peut ainsi tout représenter dans une taxonomie de classes hiérarchisées), mais bien de donner les moyens de faire évoluer cette hiérarchie. Plus précisément, dans notre domaine restreint, l'interprétation du patron `moulin à N2` met en lumière les difficultés à organiser le savoir sur des formes qui autorisent des extensions de sens : faut-il pour cela considérer que ces extensions de sens doivent être prises en compte dans une hiérarchie de concepts, ou bien dans une hiérarchie lexicale? De plus quel est le lien de dépendance entre ces types de hiérarchies? Les entrées lexicales se réduisent-elles à l'addition d'informations héritées d'une hiérarchie conceptuelle?

4.2.3. Du possible dans la représentation

Les problèmes en jeu dans une telle approche mettent donc en avant les difficultés d'organiser le savoir sous forme hiérarchique : si l'on construit une hiérarchie de savoir statique, on a peu de chances de rendre compte des phénomènes en jeu quand les connaissances de ce domaine de savoir évoluent; si l'on veut construire une hiérarchie qui structure le savoir de manière à préserver l'indétermination ou l'affinage

---

[56] Dans ce cadre, on ne dispose en effet que d'un seul mode d'évolution: la spécialisation (c'est-à-dire la création de sous-classe).

[57] Même si les résultats de la science linguistique sont à jamais provisoires, on peut considérer qu'à un moment donné, certains phénomènes de la langue se prêtent à un classement hiérarchique.

potentiel de certaines représentations, on se heurte à des problèmes complexes. C'est pourtant dans cette deuxième approche de travail qu'il faut se situer si l'on veut espérer modéliser les domaines de savoir évolutif du langage puis construire des dispositifs cohérents capables de gérer ces évolutions. Vouloir représenter, c'est donc réfléchir sur la mise en oeuvre d'outils capables de réaliser les représentations adéquates du domaine visé. Si ces outils tendent à organiser hiérarchiquement les savoirs à représenter, il s'agit tout d'abord de représenter en articulant au mieux les savoirs représentés, puis d'évaluer ce qui dans ce type de représentation, permet de rendre compte des phénomènes visés et enfin de prendre la mesure de ce qu'il reste à faire pour s'en approcher au plus près.

4.2.4. Attacher des scénarios aux entrées lexicales : prédire les possibles d'interprétation

Le travail de Pustejovsky est une illustration de cette volonté de formaliser les éléments lexicaux en mettant en oeuvre des structures qui permettent une compositionalité sémantique. Il s'agit via la notion de *qualia* (inspirée d'Aristote) de regrouper dans cette structure les attributs essentiels d'une entrée lexicale. Même si une entrée lexicale peut être associée à plusieurs interprétations potentielles dans la mesure, *"il est arbitraire de créer des mots de sens différents pour une même entrée lexicale juste parce que ces mots entrent dans des réalisations syntaxiques différentes"* (Pustejovsky 1991). Ce qui est jeu, c'est la possibilité de regrouper au sein d'une structure globale les informations qui permettent de spécifier les différents niveaux de représentation capables de générer et composer le sens. Ce choix de représentation met en place une articulation hiérarchique qui assigne une structure de qualia à chaque entrée lexicale, et qui spécifie les liens entre ces entrées lexicales en pouvant restreindre ces liens sur certains aspects de leur structure lexicale respective (Pustejovsky 1991). Dans le domaine des composés, l'utilisation de la prédicativité nominale peut permettre de restreindre les interprétations possibles des constituants du composé (Habert & Fabre 1993, Fabre 1996). On associe pour cela aux noms à structure argumentale, les mots qui apparaissent dans des scénarios typiques. Il faut malgré tout rester vigilant sur l'identification de tels noms prédicatifs au risque de faire la part trop belle au référentiel et donc d'oublier le contextuel.

4.2.5. Difficulté de l'affinement d'une hiérarchie lexicale en fonction des observations faites lors d'une analyse

4.2.5.1. Les faits de langue sont difficilement réductibles en termes d'objets figés

Représenter dans une taxonomie de classes, c'est se heurter de front au modèle de représentation sous forme d'objet dans lequel "un objet est (principalement) ce qu'il est". Les outils pour la représentation dans une telle approche ont tendance à figer les choses représentées. Et donc quand il s'agit de représenter une matière vivante, les problèmes surgissent rapidement. Si on accepte de penser en termes d'objet, c'est-à-dire admettre que tout est réductible en un ensemble de propriétés définitoires stables, une telle approche se justifie. Malheureusement, on n'en saura jamais assez sur les faits de langue. Le langage crée en permanence. Un fait de langue peut prendre le contre-pied de résultats établis, bousculer les règles définitoires. On est donc contraint à des esquisses de représentation. Reste à savoir si les différentes esquisses en disent toujours plus (et mieux) sur ce qui est représenté.

4.2.5.2. "Donner du mouvement" au jeu de la représentation

S'il apparaît très clairement qu'un choix de représentation qui manipule la notion d'objet dans une taxonomie de classes est adéquate pour fixer un domaine de savoir établi dans une structure hiérarchisée, il s'avère parfois délicat de construire puis de prendre en compte les évolutions de telles représentations hiérarchiques quand celles-ci décrivent des domaines de savoir non stabilisés. Dans une telle approche, la mise en contexte des instances de classe peut révéler la nécessité d'une mise à jour de la structure

initialement adoptée pour la définition des classes : celles-ci constituent les moules définitoires dont les caractéristiques seront reprises par les objets dans la relation "une-sorte-de". Dans le cas de la construction d'une hiérarchie lexicale dans laquelle on souhaite inscrire dans la définition des entrées lexicales une articulation forme-sens, une telle représentation doit affronter le problème complexe de la confrontation d'un savoir établi et d'un savoir à venir (non pris en compte par le précédent). La prise en compte de nouvelles informations, de résultats issus de l'analyse, de configurations énonciatives particulières, etc..., peut conduire à modifier, à affiner la représentation initiale. Et c'est justement le travail de mise à jour d'une hiérarchie qui pose problème. Plus on structure un domaine de savoir, plus les liens de dépendance entre les noeuds du graphe d'héritage se complexifient, et il devient ainsi délicat de modifier cette structure hiérarchique. Dans le cadre de notre travail sur la modélisation des configurations nominales du type `moulin à N2`, une telle évolution sur le savoir initial est à l'oeuvre sur, par exemple, des formes du type `moulin à paroles` pour laquelle `moulin` devient `+humain`. Il ne s'agit pas ici prendre le parti de dire que la représentation sous forme hiérarchique est inconcevable; au contraire il s'agit d'adapter les structures hiérarchiques aux processus qui sont à l'oeuvre dans la construction du sens des faits linguistiques.

4.2.5.3. Articuler les représentations

La construction d'une taxonomie ouverte et économique (Milner 1989) doit donc prendre en compte la nécessité de mettre en place des représentations des savoirs qui d'une part articulent ces représentations et d'autre part ne forcent pas les représentations en fonction de rencontres de surface : même si l'on peut rencontrer `"cette 2CV conduit dangereusement"` (G.Gross), il n'est pas souhaitable de classer `"2CV"` dans la classe des `Nhumains`. A l'inverse, c'est justement une représentation qui manipule des mécanismes inférentiels capables de construire, dans ce contexte particulier et temporairement, cette association de `"2CV"` à la classe des `Nhumain` qu'il faut privilégier.

4.2.5.4. Permettre un affinement dans les représentations

Les phénomènes qui participent à la construction du sens d'un énoncé restent encore mal connus. Leur formalisation et leur représentation doivent préserver la non-finitude dans ce qui est formalisé, représenté. Il faut laisser ouvert le champ de travail sur cette modélisation des connaissances qui interagissent dans la construction du sens. En particulier, puisqu'un tel dispositif doit traiter du rapport forme-sens qui est à l'oeuvre dans la construction du sens, il est nécessaire de permettre des articulations entre les savoirs représentés. Comme il s'agit encore de représenter en proposant des esquisses de modélisations pour ces interactions, il est nécessaire que les moyens mis en oeuvre permettent de construire une représentation matérielle de ces articulations de savoirs en ne figeant ni les processus d'interprétation ni les savoirs représentés.

4.2.6. Inscrire des mécanismes inférentiels au sein des représentations

Pour D. Kayser, qui parle de profondeur variable pour la représentation des connaissances, ou pour Pustejovsky qui construit un lexique génératif, il s'agit, dans les deux cas, de construire une hiérarchie de concepts, puis d'attacher aux noeuds de celle-ci des rôles qui guident les inférences éventuelles. On peut d'ailleurs considérer que la construction du sens des expressions `moulin à N2` relèvent de mécanismes d'extension de sens (à la Kayser) ou encore des coercions de type (à la Pustejovsky). Et ces deux cadres révèlent l'inadéquation d'un filtrage statique pour l'étude des séquences `moulin à N`, en effet l'élément en position d'argument ne présente pas les traits nécessaires, ceux-ci doivent lui être ajoutés et au besoin en supprimant des traits initialement présents mais contradictoires (`moulin` devient `+humain` dans `"moulin à paroles"`). Ces deux approches restent cependant éloignées dans les hypothèses initiales suivies. La sémantique lexicale suivie par D. Kayser est d'abord inférentielle. Elle permet la mise de oeuvre de processus interprétatifs capables d'ajuster ou de remodeler les unités manipulées. A l'inverse, l'approche suivie par Pustejovsky impose un point de vue initial particulier sur les unités lexicales représentées sous la forme de structure qualia. Même si certains mécanismes sont définis pour étendre les représentations construites, cette approche s'accommode d'un graphe de types figés qu'il est difficile de faire évoluer.

4.3. Un dispositif qui manipule des prototypes

Notre tâche vise à construire un dispositif dans un cadre de représentation des connaissances qui manipulent des prototypes. La construction de ce dispositif prend appui sur les objectifs suivants :

Ne pas figer les représentations

Dans une approche `Objet(Instance de Classe)`(<- présentation de la PàO), on est contraint de prédéterminer les propriétés pertinentes pour la création des objets : si l'on en sait suffisamment sur un domaine particulier de savoir, une telle approche se révèle généralement adéquate pour représenter ce domaine. Par contre si l'on ne sait pas vraiment quelles sont les propriétés pertinentes des objets que l'on va devoir représenter, une telle approche devient problématique.

S'appuyer sur un savoir minimum établi et affiner la représentation

L'approche `Objet(Prototype)` (<- présentation de la PàP) privilégie une représentation des objets sans référence à un moule initialement prédéterminé. Les objets sont construits "de toutes pièces", en fonction des besoins révélés par le contexte dans lequel cet objet est créé. Un autre `Objet(Prototype)` peut être construit à partir de l'objet initial, il possèdera tout d'abord les mêmes propriétés que celui-ci et pourra ensuite se différencier par ses propres propriétés spécifiques. La notion de prototypes privilégie donc une définition contextuelle des éléments représentés, tout en permettant une évolution de cette représentation suivant les évènements qui obligent à la préciser.

Construire le sens en contexte

Les phénomènes à l'oeuvre dans la construction du sens d'un énoncé révèlent généralement l'importance fondamentale des indices contextuels dans la résolution de l'interprétation de cet énoncé.

- La construction du sens confronte les indices sémantiques dont l'énoncé est porteur : l'interprétation est en effet le résultat de la mise en concurrence des divers indices que le dire parsème dans son trajet de production. Si l'on considère la configuration nominale `moulin à bras` dans un énoncé dont le thème général est centré sur la notion d'énergie, son interprétation sera facilitée par une association de `bras` à `source d'énergie`.

Ces constats conduisent donc à envisager la représentation matérielle de la construction du sens des configurations nominales visées dans un environnement qui, a priori, se prête à une définition contextuelle et évolutive des éléments manipulés. Il s'agit donc dans un premier temps d'identifier le savoir initial minimum qu'il est possible d'attacher aux éléments définis, puis de gérer les évolutions dans la représentation de ce savoir en fonction des résultats que la mise en contexte des éléments définis laisse apparaître. A l'inverse d'une prédiction sur les scénarios possibles que l'on peut attacher aux entrées lexicales et qui limite les interprétations sur le monde fermé du possible, nous souhaitons associer aux entrées lexicales des comportements qui se précisent au regard des évènements contextuels rencontrés.

4.3.1. Construire une analyse automatique guidée par des prototypes

Dans un premier temps, notre tâche est de construire un analyseur syntaxique guidée par une représentation prototypique des différents éléments manipulés (-> partie 2 chapitre 5). Il s'agit ensuite de réaliser un couplage entre analyse syntaxique et analyse sémantique dans un cadre précis de représentation de la construction du sens (celui des configurations nominales de la forme `moulin à X`) (-> partie 2 chapitres 5, 6).

4.3.2. Représentation des connaissances et Objet

4.3.2.1. Des prototypes pour coder la construction du sens de certains composés nominaux

Ce dispositif à approche prototypique illustre les problèmes de représentations des connaissances sous forme prototypique (les notions de catégorie et de prototype seront analysées), puis propose des approches de solution pour la construction du sens des formes `moulin à N2` dans une approche prototypique. Puisque les activités langagières font la part belle à la production d'erreurs (contrôlées), de flou, d'ambiguïté..., et que ces phénomènes sont généralement inscrits dans un contexte particulier, il apparaît cohérent de vouloir s'intéresser à un cadre de représentation des connaissances qui, a priori, offre une grande adaptabilité pour prendre en compte de tels phénomènes. Il s'agit de vérifier si les résultats produits (in)valident cette intuition initiale. Pour illustrer les approches radicalement différentes des dispositifs étudiés, on peut tout de suite noter le grand écart existant entre une approche hiérarchisée et une approche prototypique :

- **Une approche descendante.** La première approche privilégie une représentation sémantique très abstraite pour `moulin`, qu'il faudrait préciser en fonction des besoins rencontrés par le biais de scénarios, de processus inscrits dans la hiérarchie définie.

- **Une approche montante.** Au contraire, la seconde approche met en avant l'utilisation d'"individus" concrets. Il s'agit d'évaluer comment construire un modèle évolutif des connaissances à partir d'un ensemble d'individus concrètement définis et à priori non inscrits dans une hiérarchie prédéterminée.

4.3.2.2. Quels objets linguistiques devons-nous admettre dans notre propre manière de rendre compte des faits ?

> Puisqu'il est difficile d'appréhender globalement les faits de langue examinons au plus près leurs composants.

- Tel mot, tels comportements?

On se trouve donc confronté au problème de définir les individus à représenter. Comment réaliser cette définition prototypique des éléments qui participent à la réalisation des faits de langue. En particulier, quel type de savoir doit-on attacher aux entrées lexicales manipulées : tel mot, tel savoir?

- Au commencement était le mot...

Puisque le sens et la signification sont traditionnellement définis au niveau du mot, nous avons choisi de privilégier une approche de représentation qui associe à chaque entrée lexicale un prototype. D'autant plus qu'il ne s'agit pas ici de dépasser, dans un premier temps, le cadre de la composition nominale, nous en restons en effet à une étude des rapports forme-sens sur "une distance généralement courte". Il convient, malgré tout, de ne pas oublier que certaines caractéristiques textuelles peuvent influencer le sens particulier des mots (nous y reviendrons infra).

- Percevoir les régularités sur les mots

Notre travail vise donc à associer aux prototypes lexicaux une structure définitoire initiale (qui s'appuiera sur la tradition grammaticale établie).

- Si nous pouvons nous "jouer des mots", que leur représentation essaie d'en tenir compte

Il convient ensuite d'affiner les représentations en fonction du but poursuivi (la reconnaissance et la construction du sens de formes nominales) et des résultats que la mise en contexte de ces prototypes lexicaux révèlent.

4.3.3. Etablir une évaluation critique des pouvoirs expressifs des modèles de représentation

Ce travail vise aussi à établir une comparaison critique du pouvoir expressif de deux cadres de représentation des connaissances : le cadre `Objet(Instance de Classe)` et le cadre `Objet(Prototype)`. Si dans un cas il est a priori plus facile de représenter les savoirs établis, et si dans l'autre une représentation contrainte par les informations contextuelles est a priori plus adéquate, il s'agira d'évaluer les apports pertinents que permet chacun de ces modèles pour la représentation du langage naturel, quand on sait que ce dernier articule très bien ces deux aspects de la manipulation des connaissances.

4.3.4. Représenter les mécanismes de construction du sens

Le dispositif prototypique doit permettre la mise en place d'outils pour la classification des formes `moulin à N2` : dans ce domaine de savoir où il n'existe pas encore de structuration établie, il s'agit de catégoriser, c'est à dire de construire une hiérarchie en classifiant les données une à une.

4.3.4.1. Représenter en tenant compte des absences du représentable

En particulier, notre dispositif doit permettre de corriger les représentations initiales arbitrairement choisies pour construire des résultats cohérents avec les savoirs déjà validés par notre modèle de construction du sens des formes `moulin à N2`. Ainsi, le dispositif propose dans un premier temps de classer les formes `moulin à N2` de telle sorte que si le résultat produit ne correspond pas à une des facettes sémantiques du modèle initial choisi pour le classement des `moulin à N2`, on puisse intervenir directement sur les représentations initiales des éléments lexicaux pour éventuellement les "faire entrer" dans la taxonomie en cours de construction.

4.3.4.2. Repérer les mécanismes d'extension de sens et (re)construire ou affiner les représentations

Il s'agit dans un deuxième temps de construire des règles pour prendre en compte ces "erreurs" de représentation et pour les ajuster de telle sorte qu'il soit possible de les identifier. Le dispositif doit pouvoir ajuster en supprimant/ajustant des informations si besoin est : ce sera le cas pour `moulin à sottises`, structure dans laquelle `moulin` devient `+humain`. Nous verrons infra (-> partie 2 chapitre 6) qu'il met en place un filtrage évolutif basé sur la structure d'évènements associés aux noms têtes. Des outils capables de gérer les extensions éventuelles en fonction des rôles hérités sont à définir.

4.3.5. Privilégier une représentation qui permette la spécialisation

4.3.5.1. Construction des composés : une hiérarchie implicite

Puisque une approche prototypique privilégie l'utilisation d'éléments définis contextuellement et modifiables suivant les informations nouvelles rencontrées, nous essaierons de mettre en place un système de traitement qui permette la spécialisation des éléments représentés et construits pour conduire l'analyse. Ainsi, s'il s'agit de construire une configuration nominale du type `N1 à N2` où `N1 = moulin`, l'analyse se déroule en essayant de construire un objet représentant le composé qui ira du plus général (un objet `N1 à N2` sans sémantique définie) au plus particulier (un objet `moulin à N2` avec calcul sémantique intégré). Cette construction par spécialisation est mise en place dans et par les éléments représentés. `Moulin` permet de construire un composé du type `N1 à N2`, et le composé `N1 à N2` se comporte de manière particulière quand

le N1 vaut moulin. Dans ce cas on essaie de construire une interprétation de ce composé. Si ce calcul interprétatif réussit, c'est l'objet le plus spécifique qui est considéré comme construit, c'est-à-dire l'objet qui possède une plus grande cohérence de représentation par rapport à ce qu'il représente. Dans le cas contraire, on ne bloque pas la reconnaissance, quitte à produire un résultat non défini ou incomplet (sémantiquement dans notre cas).

4.3.5.2. Des représentations au "jeu" sur les représentations

Il s'agit donc de préserver, au niveau de la représentation puis au niveau de l'implémentation, la possibilité de prendre en compte une évolution à venir dans la mise en place du dispositif. Même si notre travail se limite dans un premier temps à la représentation des problèmes d'interprétation des configurations du type moulin à N2, il doit pouvoir, d'une part, ne pas bloquer la reconnaissance de configuration nominale proche de celle-ci, et d'autre part, permettre une prise en compte, à moindre coût, d'éventuelles évolutions de ce modèle initial pour y intégrer la prise en compte de nouvelles représentations supplémentaires (la construction du sens de configurations nominales proches).

4.3.6. Développer des outils de contrôle pour l'analyse

Puisque notre approche se situe dans un cadre de modélisation des connaissances par la construction de représentations qu'il s'agit de faire évoluer, d'affiner, on conçoit aisément qu'il soit nécessaire de disposer d'éléments "dégagés" de ces représentations et qui permettent d'évaluer ce qui est en jeu pour ensuite permettre une prise en compte statique / dynamique des éléments pertinents susceptibles de faire évoluer les représentations initiales. Notre difficulté à isoler les interactions complexes des savoirs à l'oeuvre dans la production d'énoncé accentue la représentation et la construction de tels mécanismes. Il semble malgré tout possible d'y voir plus clair sur une utilisation de processus de contrôle à un méta-niveau si l'on restreint leur étude à des niveaux de complexité moindre. Notre réflexion porte donc aussi sur cet aspect fondamental dans la représentation des connaissances pour un dispositif informatique. D'autant plus que les langages de haut niveau que nous utilisons mettent en place des mécanismes qui favorisent les approches réflexives. Celles-ci peuvent éclairer une étude sur la construction de mécanismes de contrôle pour l'analyse et sur la définition même de ces mécanismes : peut-on construire de tels mécanismes capables de restructurer les représentations initiales, de modifier les opérations pré-définies... C'est en tout cas ce que nous essaierons de présenter dans notre cadre linguistique choisi.

4.4. Prototypes, représentation des connaissances et modèles du sens

4.4.1. Noyau de sens et prototypes

La notion de prototype semble adéquate pour représenter un noyau de sens tel que celui que C. Fuchs décrit. Il reste à penser les procédures de contrôle pour déterminer et définir les manières dont les chaînes de sens peuvent s'étendre et les directions dans lesquelles elles opèrent. Il convient pour cela de définir des outils qui permettent à partir de représentations de base volontairement sous-déterminée de préciser les informations ainsi représentées. On peut ainsi envisager de systématiser les phénomènes de reprise sur indétermination qui peuvent se produire dans la construction du sens mais que l'on parvient toujours à atténuer ou à éliminer en faisant appel à des connaissances supplémentaires.

En termes de représentation prototypique à la Self (Self Group 1987-1995) par exemple, la définition d'un noyau de sens peut se matérialiser par l'association d'une entrée lexicale et d'une entité prototypique portant ce noyau de sens. Cette dernière structure peut être considérée comme un comportement commun à un ensemble d'unités lexicales qu'il reste à préciser dans certaines configurations interprétatives. Il s'agit donc d'ordonner les comportements sémantiques que l'on peut attacher à une entrée lexicale, c'est-à-dire déterminer les chemins de sens (en termes de comportements particuliers) et leurs directions. Si la notion de prototypes rend possible dans la représentation l'ordonnancement de comportements, il reste à vérifier si les comportements sémantiques d'une entrée lexicale se prêtent à un tel ordonnancement sémantique.

4.4.2. Adaptation d'une représentation prototypique et d'une approche à la Pustejovsky ?

La notion de prototype reste cohérente avec la volonté de capter les régularités lexicales dans les représentations lexicales associées à des mécanismes d'inférence sur ces représentations.

4.4.2.1. Inscrire des processus d'interprétation dans la représentation prototypique

En tant qu'objet, le prototype peut être défini comme un élément qui structure les différents niveaux de représentation des éléments lexicaux. On peut toujours ajouter de l'information dans une représentation prototypique et à moindre coût. Il est toujours possible de faire évoluer les éléments représentés en les spécialisant ou en modifiant leur lien d'héritage : les prototypes n'héritent pas d'un moule structurel définitoire, mais de comportements partagés par un ensemble de prototypes.

L'évolution dans un système de représentation à base de prototypes n'est pas uniquement descendante :

- On peut ajouter et/ou supprimer de l'information dans les prototypes construits.
- On peut inscrire de la délégation[58] de comportements communs et jouer sur ce lien d'héritage : d'une part il est possible de modifier ce lien d'héritage; d'autre part, on peut affiner les comportements partagés et les spécialiser.

Pour une entrée lexicale donnée, il est possible d'associer à la structure définitoire représentant cet élément lexical une suite de comportements différents qu'il est possible d'ordonner (en tenant compte par exemple de

[58] La notion de délégation sera exposée infra.

leur fréquence de réalisation); il est aussi possible de factoriser ces comportements sur un ensemble d'unités lexicales (peu importe a priori leur "qualité" linguistique, il suffit que l'unité représentée se comporte suivant les processus auxquels elle est attachée).

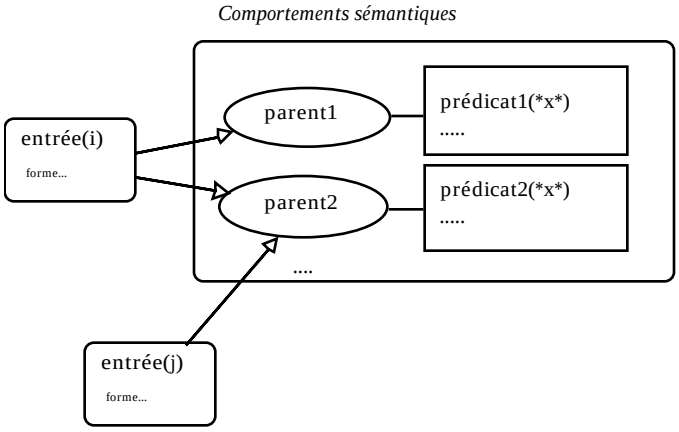


Figure 4.1 : Telle entrée lexicale, tels comportements ?

On peut aussi représenter les unités lexicales en proposant des directions de sens qu'il est possible de suivre en tenant compte des informations rencontrées. S'il y a blocage dans l'application de tel comportement sémantique donné, il faut pouvoir récupérer un chemin de sens adéquat pour la configuration visée. Dans l'exemple de "fast" traité par Pustejovsky, on peut inscrire dans le comportement sémantique de l'adjectif qu'il peut modifier le prédicat attaché au rôle télique du nom auquel il est associé.

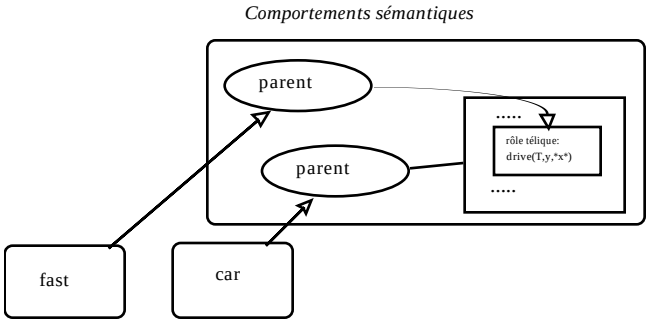


Figure 4.2 : "Fast car"

Une représentation qui structure les informations de manière hiérarchique est toujours difficile à modifier : il est coûteux voire impossible de modifier des liens d'héritage sans remettre en cause l'intégralité du domaine représenté. A l'inverse, une approche prototypique est plus ouverte pour une représentation évolutive des liens entre les éléments représentés.

On peut ainsi envisager différentes solutions pour illustrer une approche à la Pustejovsky à l'aide de prototypes qui associerait aux entrées lexicales des scénarios guidant l'interprétation.

- Solution n°1 : on crée des hiérarchies locales.

On crée une structure définitoire de base qu'il convient d'affiner si nécessaire pour produire de nouvelles instances prototypiques.

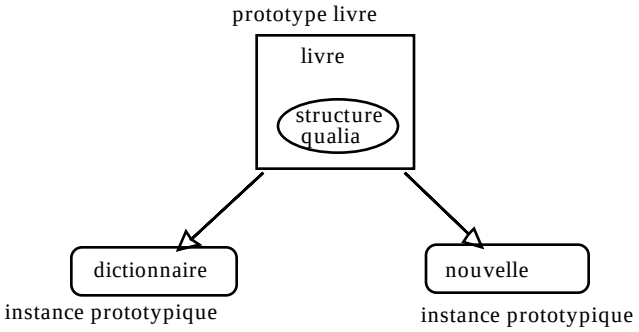


Figure 4.3 : Des prototypes comme moules définitoires pour les entrées lexicales.

Les instances prototypiques de "dictionnaire" et de "nouvelle" utilisent la définition du prototype "livre" comme s'il s'agissait d'un moule définitoire (à la manière d'une approche Objet(Instance de Classe)). Chacune d'elles porte une structure définitoire adaptée à l'entité représentée.

- Solution n°2 : toutes les entités sont au même niveau.

On considère un élément comme le plus typique d'une famille d'éléments à représenter. On peut ainsi amorcer la représentation en utilisant cet élément initial.

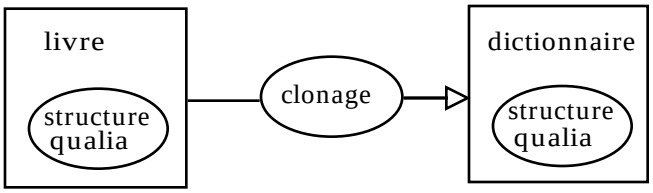


Figure 4.4 : De la multiplication des entités scénarisées.

L'opération de clonage consiste en une copie du prototype initial, on peut ensuite ajuster les éléments définitoires contenus dans la structure prototypique résultante pour l'adapter à la nouvelle entité représentée. Cette solution peut se combiner avec la mise en place de hiérarchie de comportements partagés par les entités représentées. Si le rôle télique est caractéristique des entités représentées, on peut factoriser les autres éléments définitoires.

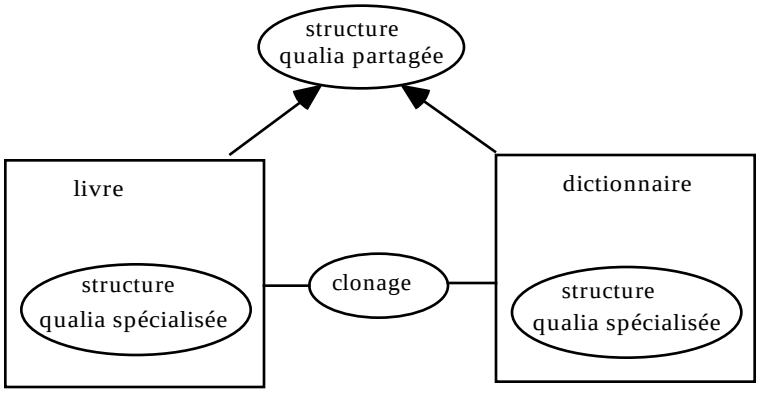


Figure 4.5 : Des scénarios adaptés à la réalité prototypique.

Par défaut, le comportement sémantique initial est composé du comportement sémantique défini par la structure qualia partagée et par la structure qualia spécialisée caractéristique de l'entité représentée. On peut ajuster cette dernière structure (par spécialisation), on peut aussi être amené à modifier la délégation de comportement partagé si nécessaire, ou à la spécialiser elle aussi (en ordonnant la délégation).

- Solution n°3 : tous les éléments sont définis en termes de prototypes.

On ne distingue plus la notion de comportement de la notion de prototype. Les entrées lexicales sont associées à des structures prototypiques qui peuvent être décrites par une association (à définir) avec des prototypes qui précisent leur rôle sémantique. Les entités voisines peuvent se définir par des opérations de copie et d'ajustement des valeurs associées.

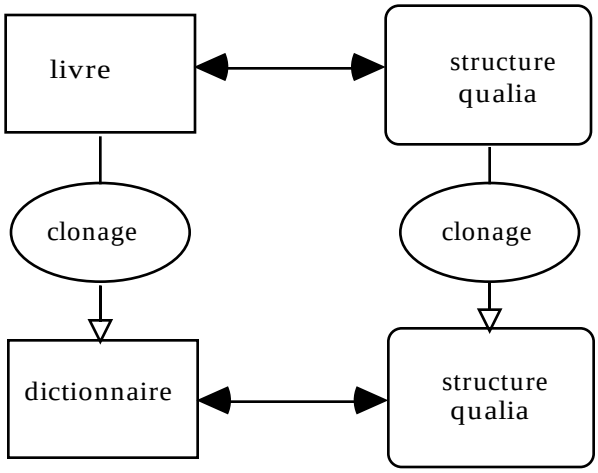


Figure 4.6 : Tout est "prototypisable".

- Solution n°4 : on ordonne la représentation prototypique précédente.

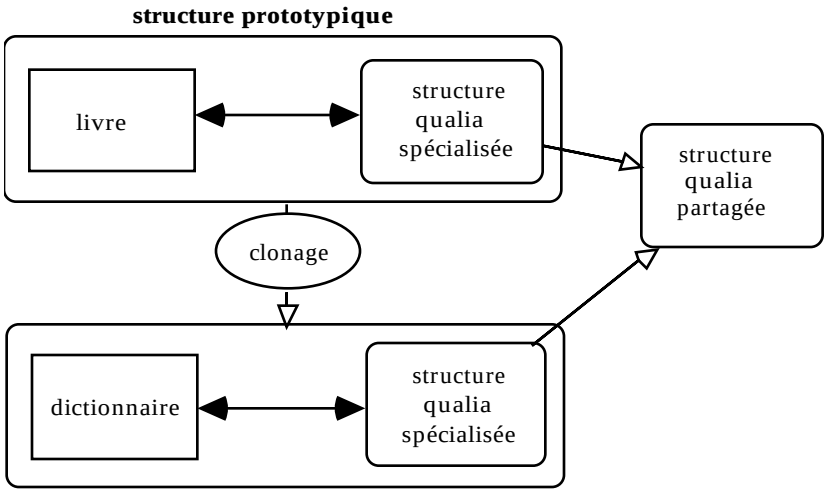


Figure 4.7 : Mise en ordre et prototypie.

Comme précédemment, on hiérarchise le savoir sur le domaine représenté en factorisant les comportements (en particulier les comportements sémantiques).

Il convient une nouvelle fois d'insister sur la nécessité de construire des mécanismes qui proposent des alternatives pour la construction du sens : il s'agit de définir les chemins de sens potentiels pour une entrée lexicale puis d'organiser au mieux les directions de ces parcours sémantiques afin de répondre à des besoins particuliers que les configurations interprétatives rencontrées révèleront. Ces comportements sémantiques doivent traduire les phénomènes de reprise, d'extension, de précision qui opèrent dans le processus d'interprétation. Le choix d'une telle approche vise donc à opérer une réflexion sur l'organisation des savoirs sémantiques que l'on peut attacher à une entrée lexicale (ces savoirs étant assimilés à des comportements). En particulier, il ne faut pas perdre de vue la nécessité de structurer l'ensemble des comportements sémantiques définis. Une prototypisation des savoirs peut conduire rapidement à une dilution incontrôlable et donc peu efficace pour établir à terme la mise en ordre des éléments représentés. Cette mise en ordre doit malgré tout préserver la possibilité d'une mise à jour éventuelle des comportements initialement définis. Si le comportement sémantique attaché à une entrée lexicale peut être affiné, il faut pouvoir spécifier dans la représentation sémantique le détour sémantique rencontré parmi les chemins de sens existants.

Cela revient à traiter parallèlement les points suivants :

- Structurer le savoir sémantique pour chaque entité représentée tout en laissant ouverte la représentation de ce savoir.

- Organiser les savoirs sémantiques sur l'ensemble des entités représentées en définissant les moyens pour ajuster ce classement.

Pour illustrer le premier point, on peut d'ailleurs s'interroger sur les modalités à mettre en oeuvre pour bien cerner les couples discriminatoires qui permettent l'activation de processus sémantiques particuliers. Pustejovsky utilise des oppositions en termes de traits différentiels pour distinguer les comportements sémantiques qu'il est possible d'attacher à une entrée lexicale. Le problème se pose donc de savoir s'il peut y avoir recouvrement entre les couples discriminants et les comportements induits.

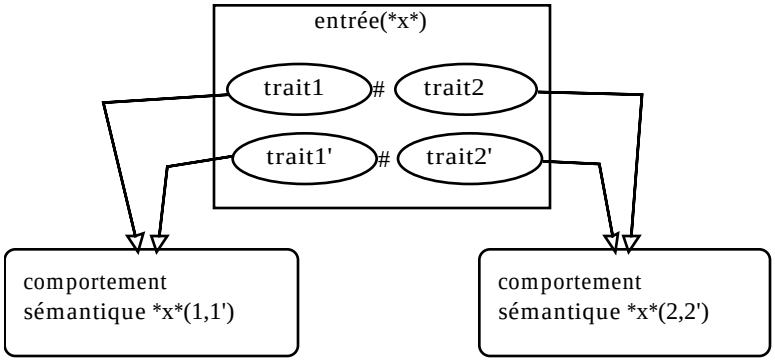


Figure 4.8 : Recouvrement de comportements sémantiques.

Sur quel type de "qualité" sémantique faut-il distinguer les couples discriminatoires activant des processus interprétatifs? Est-il (im)possible, souhaitable d'unifier ces types discriminatoires?

Dans la figure précédente, si on ne maintient pas la double opposition dans la définition des comportements sémantiques induits par ces traits, on risque de ne pouvoir activer le comportement sémantique adéquat. En effet si la structure définitoire ne tient compte que du couple (trait1,trait2), il peut s'avérer que ce couple soit non pertinent dans un contexte donné. Alors que le couple (trait1', trait2') le serait dans ce même contexte.

On rejoint ici encore des problèmes déjà mentionnés. Si ces oppositions constituent des savoirs établis, il faut les organiser pour ne pas laisser de côté des configurations interprétatives. Si ces savoirs sont révélés par l'analyse, il convient de les récupérer et de réorganiser les comportements sémantiques attachés aux objets concernés.

4.4.2.2. De l'unité dans la représentation

Une représentation hiérarchique structure les liens entre les noeuds inscrits dans le graphe d'héritage induit par cette représentation. La définition de mécanismes d'inférences lexicales dans une telle approche peut donc être assimilée à des parcours de ces liens d'héritage afin d'y saisir les informations pertinentes. Seuls les liens pré-définis sont valides; les inférences lexicales sont donc restreintes/contraintes par cette pré-définition de la hiérarchie initiale. Si une structuration du lexique ou des concepts sous forme d'une représentation hiérarchique permet de réaliser des liens d'héritage entre les éléments de la hiérarchie décrite, une telle structuration limite les liens entre les éléments représentés aux chemins qu'il est possible de parcourir dans le graphe d'héritage induit : la spécialisation traduit le passage de propriétés générales vers des propriétés particulières. Qu'en est-il des relations transversales qu'il peut être possible d'envisager entre les éléments d'un graphe d'héritage sur un domaine lexical ou conceptuel?

Le choix d'une représentation prototypique, c'est-à-dire le choix de l'absence, à priori, de structuration globale pour les éléments représentés, se heurte au problème crucial des liens à établir entre ces éléments surtout quand il s'agit de représenter des éléments qui agissent ensemble dans des processus de construction du sens. Il est donc impératif de définir les connexions qui existent entre les éléments représentés de manière prototypique : la prototypie doit s'organiser pour ne pas diluer les savoirs sur des entités qui s'ignorent. S'il s'agit de donner une représentation matérielle de la construction du sens en termes de prototypes, il faut que les éléments prototypiques définis aient les moyens de se (re)connaître mutuellement : enregistrer les liens initiaux entre les entités représentés et aussi pouvoir en construire.

De plus, il est important de bien distinguer ce qui relève de la représentation morphologique ou sémantique de ce qui relève de la représentation des processus qui manipulent ces précédentes formes de savoir. A priori puisque nous travaillerons dans un système qui ne manipule que des prototypes, ces derniers pourront être aussi bien des récepteurs de savoirs (figés ou non) que des passeurs agissant sur les précédents.

Il peut être délicat de restreindre les modalités de représentation de phénomènes "qualitativement" distincts sur un type particulier d'éléments. S'il peut s'avérer adéquat d'utiliser des prototypes pour décrire des problèmes précis, ce mode de représentation peut se trouver en difficulté pour la représentation d'autres domaines de savoir. Il ne s'agit donc pas de dire que le choix des prototypes s'impose pour traiter globalement le problème de la représentation matérielle des faits de langue, mais qu'il peut être adapté pour aborder certains phénomènes à l'oeuvre dans les faits de langue. Là encore il ne s'agit pas de privilégier une pratique par rapport une autre : l'approche prototypique a, semble-t-il, beaucoup de mal à rendre compte de certains phénomènes relevant d'une organisation hiérarchisée et figée du savoir. Il faut entendre la difficulté précédemment énoncée en termes d'inadaptation pour traiter le type de problème visé et non en termes d'impossibilité.

4.5. Approche pour une formalisation prototypique de la construction progressive du sens des formes moulin à N2

4.5.1. Une première approche guidée par les hypothèses sémantiques développées par (Habert & Fabre 1993)

4.5.1.1. Un calcul sémantique à base de traits

Cette approche pour la construction du sens des formes moulin à N2 (qui est développée en détail infra (-> partie 2 chapitres 5, 6)) peut être résumée de la manière suivante :

- **Etape 1** :
  Construction de la représentation du Problème.
  Approche pour une représentation par prototypes des éléments lexicaux.
  Mise en place d'une analyse automatique guidée par des prototypes.

- **Etape 2** :
  Affinement progressif des mécanismes de représentation et d'analyse.
  Mise en place d'un traçage de l'analyse.
  Prise en compte d'un modèle sémantique pour la construction du sens des moulin à N2 : assimilation des résultats produits à des classes connues pour lesquelles on possède des critères d'évaluation et de reconnaissance. Avec notamment un ajustement dynamique de la sémantique attachée à moulin qui devient +humain dans la série des moulins à paroles. On donne infra une présentation des filtres qui permettent une classification provisoire des séquences étudiées.

- **Etape 3** :
  Activation de procédures de contrôle qui permettent de modifier les données initiales si l'Etape 2 ne réussit pas :
  - modification des valeurs sémantiques de manière externe (à "la main" par une intervention de l'utilisateur).
  - ajout de filtres (pour les N2 déverbaux).

- **Etape 4** :
  Mémorisation des "évènements critiques" dans l'Etape 3 qui ont permis d'atteindre à la reconnaissance d'une classe.
  Ces évènements amènent à repenser le modèle sémantique choisi pour prendre en compte les faiblesses révélées par l'analyse.

Le calcul sémantique mis en place dans les Etapes 2 à 4 peut être décrit comme suit :

Soit Traits l'ensemble ordonné des traits sémantiques utilisés par le calcul sémantique :
Traits = {abstrait, concret, nParole, ali, synth, energie, comptable}

Soit Mots l'ensemble des mots x du lexique tels que les formes moulin à x existent.

Soit $Trait_t$:

Mots -> { +,-}

mot -> $Trait_t$(mot), valeur du trait sémantique t pour le mot considéré

Soit Sem :

Mots -> {(x1,...xi,...)/ xi= +,- et i=2....}
mot -> ($Trait_{t1}$(mot),$Trait_{t2}$(mot),....,$Trait_{tn}$(mot))

où n est le nombre de traits sémantiques utilisés par le calcul sémantique des formes moulin à N2 et où les ti appartiennent à Traits.

Soit Classes l'ensemble des classes sémantiques connues :
Classes = { PourSortie1, PourSortie2, PourSortie3, PourSortie4, PourSortie5, PourSortie6, PourEntrée1, PourEntrée2, AvecEnergie, AvecPartieDe}

Chaque élément de Classes correspond à un filtre de traits sémantiques associé à une sémantique possible pour le composé. Ces filtres sont des suites ordonnées de valeurs des traits sémantiques (abstrait, concret, nParole, ali, synth, energie, comptable) définies comme suit :

PourSortie1 = (-,+,+,-,-,-,+)
PourSortie2 = (-,+,+,-,-,-,-)
PourSortie3 = (-,+,-,+,+,-,+)
PourSortie4 = (-,+,-,+,+,-,+)
PourSortie5 = (-,+,-,-,+,-,-)
PourSortie6 = (-,+,-,-,+,-,+)
PourEntrée1 = (-,+,-,+,-,-,-)
PourEntrée2 = (-,+,-,-,-,-,-)
AvecPartieDe = (-,+,-,-,-,-,+)
AvecEnergie = (-,+,-,-,-,+,-)

Soit x ∈ Mots tels que moulin à x existe.

Calcul sémantique de moulin à x

1 Si Sem(x) ∈ Classes, l'analyse réussit.

2 Si Sem(x) ∉ Classes :
Dans ce cas, on est confronté aux alternatives suivantes :

2.1. Ajout de Sem(x) à Classes :
Classes:= Classes U {Sem(x)}. Il convient donc dans ce cas de réajuster le calcul sémantique pour prendre en compte la nouvelle classe reconnue.

2.2. Modification des valeurs sémantiques de x :
Pour t ∈ Traits, $Trait_t$(x):= + ou - . Dans ce cas, il s'agit plutôt de modifier les valeurs sémantiques initiales attachées au N2 pour "faire entrer" le composé produit dans le modèle construit. Après une mise à jour des valeurs sémantiques, on opère une nouvelle analyse (retour supra Calcul sémantique).

2.3. On ne fait rien, mais on garde en mémoire la configuration sémantique qui a échoué. Dans ce cas, il s'agit ni de forcer la représentation (2.2) ni d'intégrer trop tôt une nouvelle classe (2.1). On en reste au modèle initial tout en gardant en mémoire les faiblesses de celui-ci.

2.4. On évalue d'autres traits sémantiques ou morphologiques non encore utilisés par le calcul sémantique initialement défini. On garde aussi en mémoire la configuration sémantique qui a permis d'atteindre ou non à la reconnaissance. Il s'agit en fait d'une extension du modèle sémantique initial qui agit notamment pour le traitement des N2 déverbaux.

4.5.1.2. Limites de ce type de représentation sémantique en termes de traits

Il semble en effet délicat de restreindre la définition des représentations sémantiques en termes de traits binaires. Tout d'abord, on est souvent confronté dans la description des unités linguistiques à des données qui ne se laissent pas appréhender par une simple opposition entre deux possibles. Il s'agit au contraire d'évaluer des propriétés qualitatives dont la représentation ne se prête pas à une telle réduction. Il convient de réfléchir à des descriptions plus raffinées pour appréhender les entités représentées. Ensuite, la prise en compte de nouvelles informations peut conduire à une multiplication incontrôlable du nombre de ces traits binaires. De plus est-il légitime de définir une unité de langue comme une liste d'étiquettes qu'on empile? Qu'est-ce qu'un trait en général? Peut-on prendre n'importe quoi? Que recouvrent réellement ces attributs? (Sur ces problèmes (Rastier & al. 1994)). Dans la mesure où le travail développé ici vise à représenter la construction du sens au travers de processus interprétatifs qui agissent à partir d'une signification minimale que les informations rencontrées sur une configuration donnée permettent de préciser, il ne semble pas adéquat de figer les informations sous une forme de liste d'étiquettes disparates. Il s'agit au contraire de préciser les chemins de sens que l'on peut construire sur un item donné (reconstruire un espace sémantique possible en termes d'opérations sémantiques) et dont les modalités de descriptions finales (sous forme de traits dont la nature sera à préciser, ou sous un autre forme) seront déterminées et ajustées suivant les directions de sens mises en place pour cet item.

4.5.2. Moulin : (Re)construire une Représentation Sémantique Prototypique[59]

Nous considérons tout d'abord la forme initiale de description de moulin adoptée supra :

moulin : nom masculin singulier

Si l'on considère moulin comme un représentant typique des mots généralement regroupés dans la catégorie grammaticale des noms, on peut considérer qu'il peut servir de modèle pour la création d'un prototype nom (noté prototype N). Cette description sommaire que l'on peut faire de ce mot va permettre la création du prototype N qui servira de base pour la création de nouveaux prototypes N.

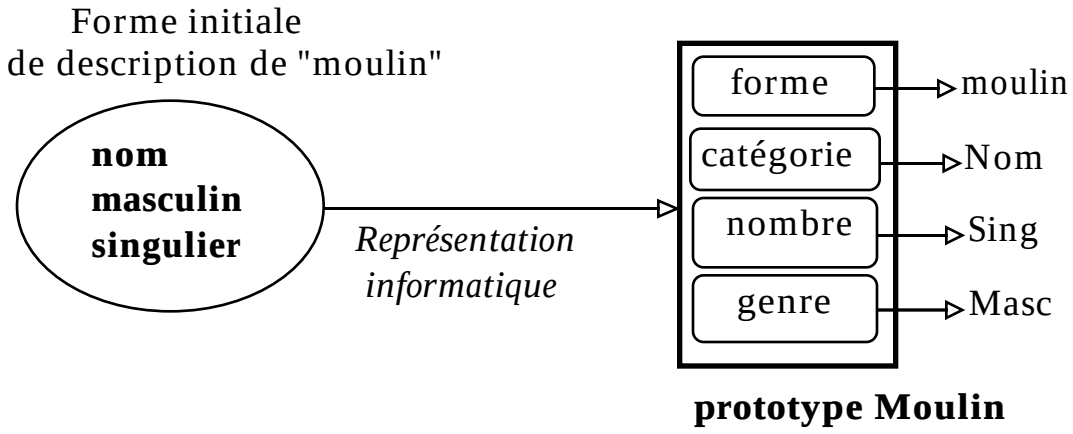


Figure 4.9 (1) : Moulin est à l'origine de la création du monde des noms : première étape, la création.

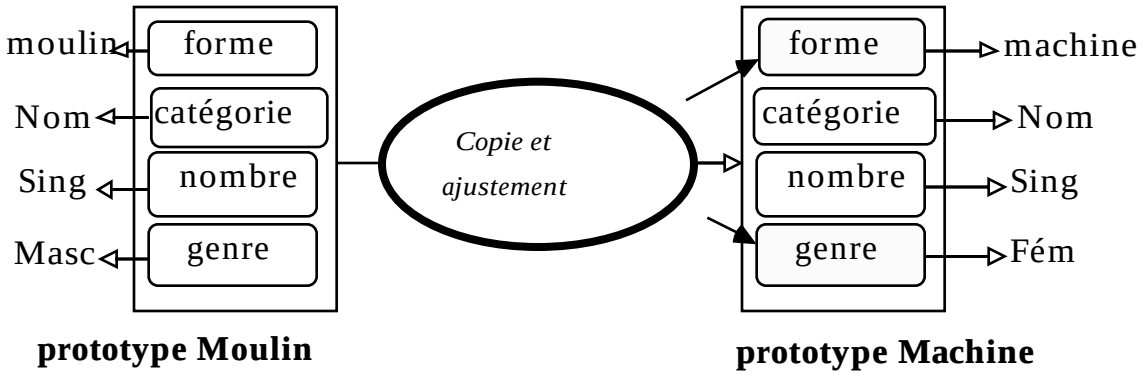


Figure 4.10 (2) : Moulin est à l'origine de la création du monde des noms : deuxième étape, la reproduction.

Sur notre exemple, machine est un autre représentant de la catégorie grammaticale des noms, il possède donc au moins les mêmes attributs définitoires que moulin. On peut donc créer un prototype associé à machine à partir du prototype initial associé à moulin. Les seuls attributs à modifier pour représenter ce nouveau mot sont les attributs décrivant la forme et le genre de celui-ci. Essayons maintenant d'attacher une représentation sémantique à moulin. Il s'agit ici de définir l'information sémantique en termes de comportements particuliers définis par les configurations interprétatives rencontrées. Pour cela, nous souhaitons attacher à moulin une représentation sémantique initialement peu déterminée et ensuite

[59] On donne infra une présentation de l'implémentation proposée pour traiter les problèmes décrits ici (-> partie 2 chapitres 5, 6 -> annexe partie 2 chapitre 5).

affiner cette représentation pour tenir compte des extensions ou des changements de sens qu'il est possible de rencontrer dans certaines réalisations contextuelles.

4.5.2.1. Moulin emballe la machine prototypique

Notre prototype initial attaché à moulin peut donner naissance à deux représentants prototypiques distincts dans leurs comportements sémantiques. Cette première décomposition ne dit encore rien de la description sémantique qui est attachée à chacun des prototypes créés. Elle met malgré tout en lumière les problèmes à résoudre pour construire une représentation des connaissances de manière prototypique.

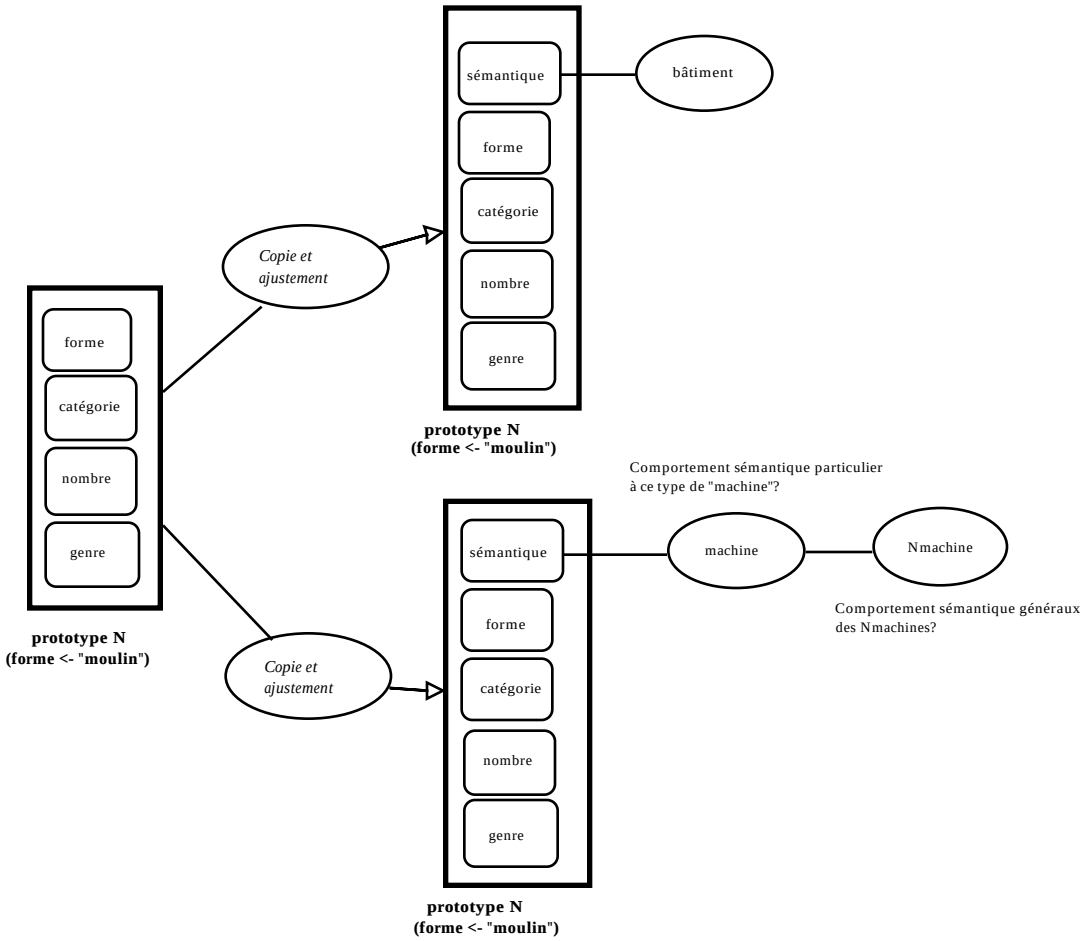


Figure 4.11 : Moulin, quelle représentation sémantique?

Tout d'abord, si on considère que moulin est effectivement un représentant typique des noms de machines, il peut servir de base pour la création d'autres exemplaires de cette classe de noms. Il reste évidemment à vérifier si cette classe de noms organise des comportements sémantiques particuliers, et parmi ces comportements, quels sont ceux qui intéressent moulin ? Ensuite, cette décomposition amène à dresser la liste de tous les possibles de sens de moulin. Pour reprendre Pustejowsky (Pustejowsky 1991), on peut essayer au contraire de capter les régularités lexicales et sémantiques puis spécifier les réalisations dans la représentation.

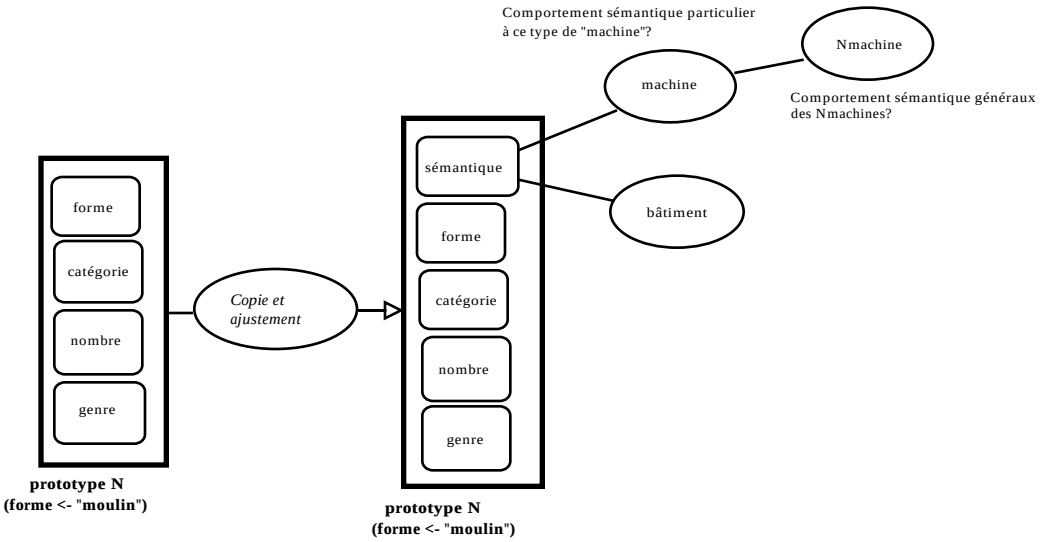


Figure 4.12 : Un prototype `moulin` qui capte les régularités lexicales et sémantiques.

Pour aller plus loin, on peut aussi vouloir définir sur moulin un noyau de sens ("à la Fuchs") qui organise la construction sémantique du prototype en fonction des informations auxquelles celui-ci aura accès dans un contexte donné. Le noyau de sens que nous visons à définir se place dans une perspective d'ébauche de sens. Il s'agit de ne pas émietter dans la représentation des sens possibles, mais au contraire de partir d'une représentation "grossière" puis de la modifier si nécessaire.

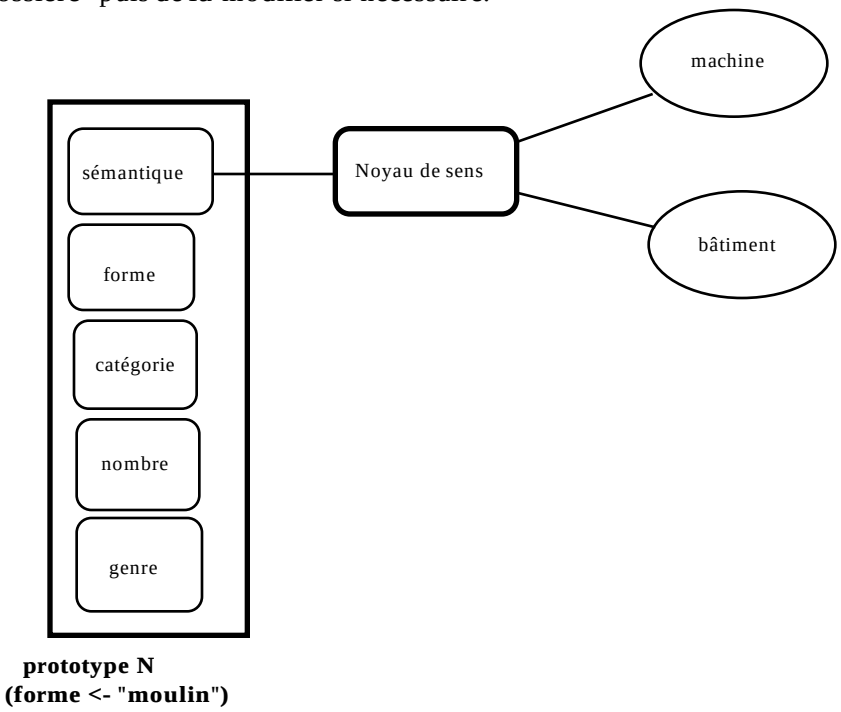


Figure 4.13 : `Moulin`, un noyau de sens ?

Là encore, on est confronté à des choix importants. Si on s'oriente vers une représentation d'un noyau de sens qui va du "général au particulier" sémantique, il convient de ne pas perdre de vue que les descriptions faites sur `moulin` peuvent intéresser d'autres représentants lexicaux "sémantiquement proches". Sans présumer de ce que sera la description sémantique de `moulin`, on peut par exemple vouloir décrire précisément les comportements sémantiques des noms d'instruments et se servir de cette description particulière de `moulin` pour amorcer une description sémantique des noms d'instruments qu'il conviendra de préciser par une étude complémentaire d'autres noms d'instruments particuliers. On peut ainsi considérer que la transformation

machine -> lieu, non articulée dans la représentation précédente, est une opération générale qu'il est possible d'appliquer à bon nombre de noms de machine. Cette transformation dépendant en partie de la taille de la machine visée : "Je suis entré dans le réacteur" vs "Je suis entré dans le concasseur". Nous revenons donc sur la représentation initialement adoptée afin d'articuler cette transformation de sens qui peut s'appliquer sur le noyau de sens associé à moulin.

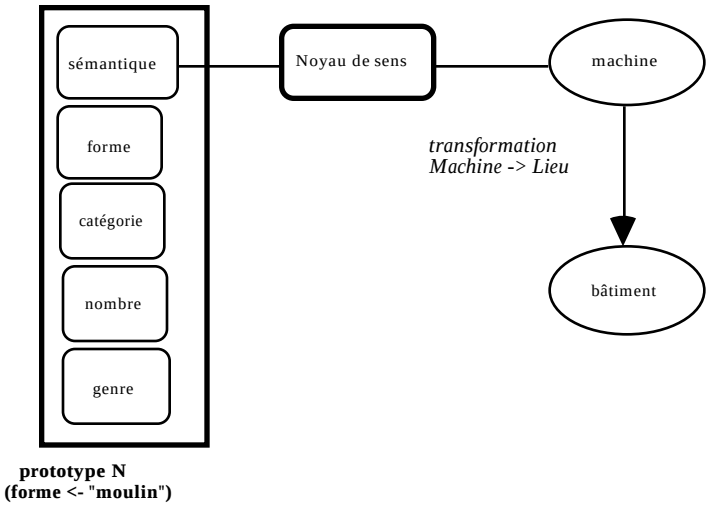


Figure 4.14 : Moulin, un noyau de sens qui articule les transformations de sens observées

Le noyau de sens attaché à moulin doit tendre à tracer une direction "sémantique" adéquate pour la description de moulin comme instrument. Il s'agira ensuite de détacher dans les comportements sémantiques définis ceux qui relèvent plus généralement de la classe des noms d'instruments, tout en permettant de retrouver ces informations particulières dans cette classe de noms et notamment celles qui concernent le comportement sémantique de moulin (mise en oeuvre d'héritage de comportement sémantique). On peut d'ailleurs se demander si ce processus d'abstraction est cohérent avec une approche prototypique.

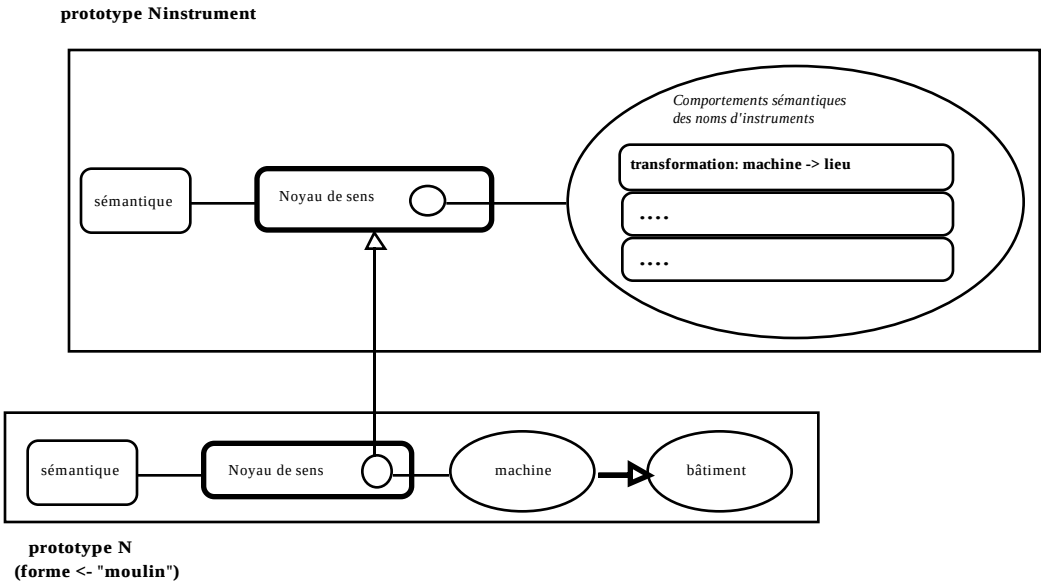


Figure 4.15 : Certaines extensions peuvent s'appliquer aux noms d'instruments.

4.5.2.2. Des moulins aux moulins à N2

Il est possible à ce stade d'étendre la représentation sémantique de `moulin`. Il peut être associé à un prédicat dont la valeur prédicative dépend du `N2`. Si `N2` est un produit naturel, la valeur prédicative traduit une transformation. Si par contre, le `N2` est un artefact, celle-ci traduit plutôt une création. On se place donc ici dans une approche qui élargit la notion de prédicativité nominale à des constituants autres que des déverbaux. Ces nouvelles informations nous permettent de préciser notre représentation prototypique de la manière suivante[60].

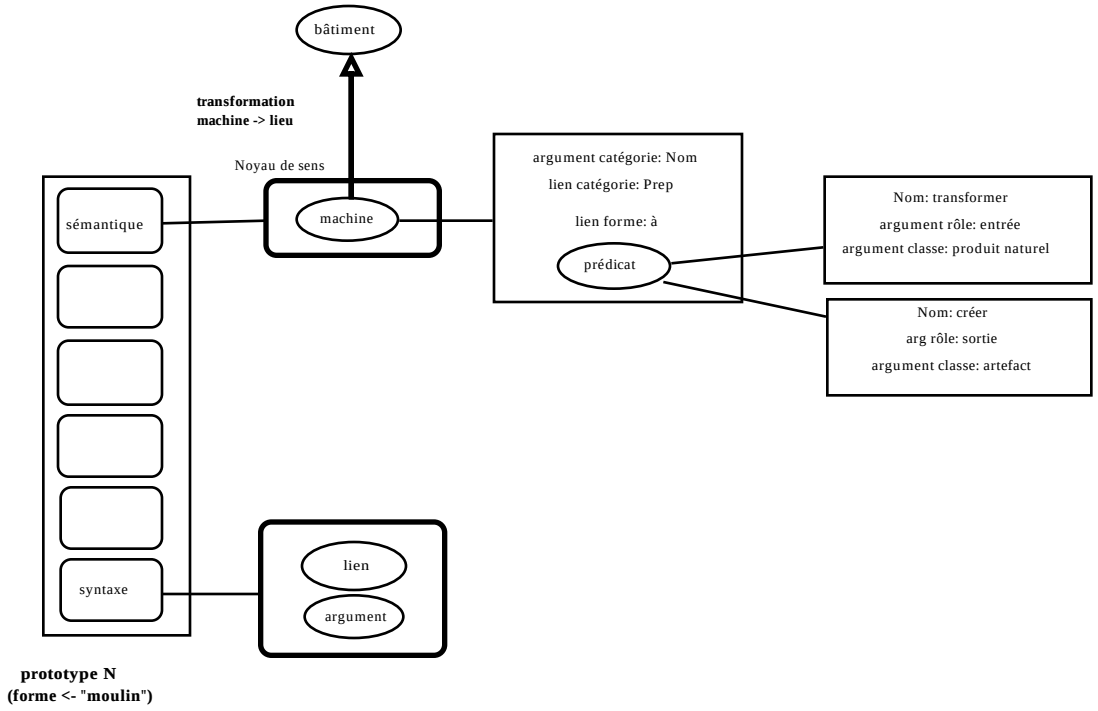


Figure 4.16 : Des `moulins` aux `moulins à N2`.

Les réalisations telles que `moulin à vent`, `moulin à eau`, `moulin à bras` sont partiellement prises en compte dans cette représentation sémantique attachée à `moulin` : la valeur prédicative `transformer` reste valide, par contre la classe de l'argument doit être étendue à des arguments de type `énergie`. De plus la nature de la relation est elle aussi affectée, il s'agit plutôt d'une relation `partie-de`. L'argument est dans une relation de partie constitutive dans le processus de transformation : le moulin transforme une énergie (naturelle en général) en énergie mécanique ou autre, et l'énergie (en entrée) est partie intégrante du processus de transformation réalisé par le moulin. Il semble d'ailleurs que cette dernière remarque nous permette de préciser davantage la représentation prédicative associée à `moulin` . On peut en effet considérer que `moulin` implique un changement d'état qui recouvre plusieurs réalisations lexicales suivant que l'argument soit un objet fabriqué, un produit naturel ou un nom d'énergie. On peut ainsi définir un prédicat générique unique `faire-changer-d'état` dont les manifestations lexicales dépendent de l'argument : cette solution est assez proche de la co-composition de Pustejowsky.

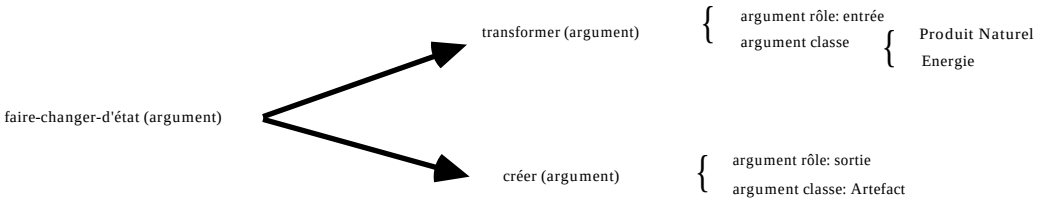


Figure 4.17 : Un prédicat générique pour `moulin`.

---

[60] La représentation proposée ici ne prend pas encore en compte des séquences telles que "un moulin à blé à vent" ou "*un moulin à farine à blé"; ce qui revient à distinguer les status de l'argument et du modifieur, on y reviendra par la suite.

Ce choix permet tout d'abord de maintenir une représentation qui s'accorde avec bon nombre de noms d'instrument; ensuite, la sous-spécification du prédicat en `transformer` ou `créer` peut être vue soit comme un problème de spécialisation lexicale, soit comme un phénomène de co-compositionalité. La figure 4.18 reprend la représentation précédente en ajustant la définition de la nature du prédicat associé à `moulin`.

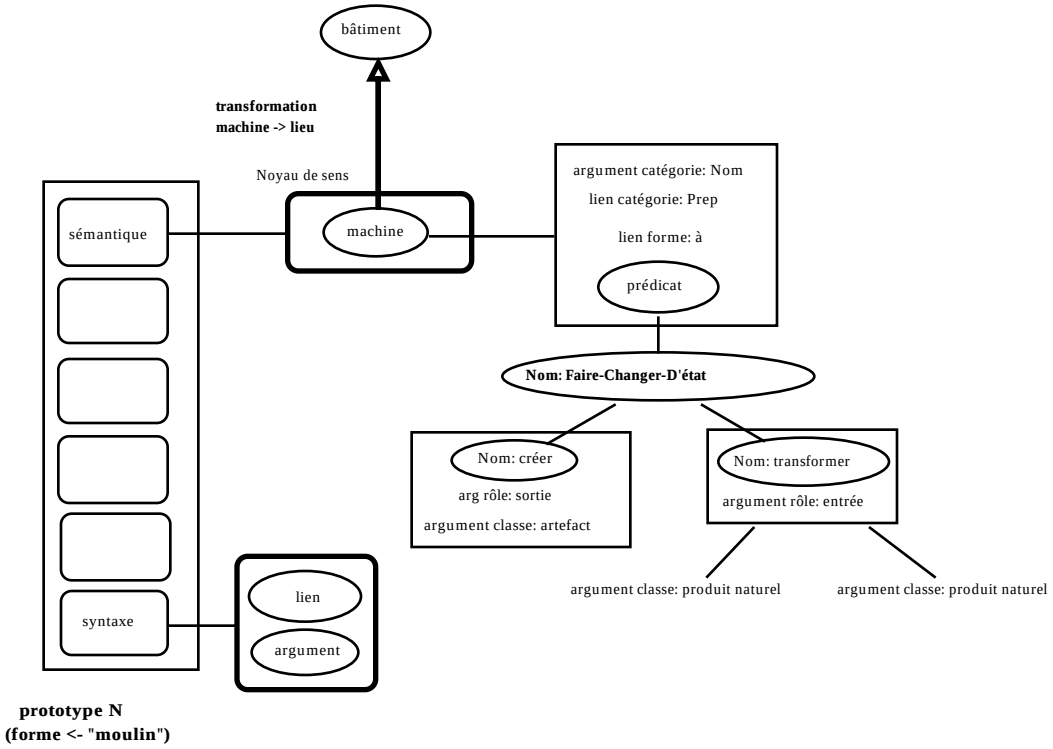


Figure 4.18 : Une représentation prédicative sous-spécifiée pour `moulin`.

En ce qui concerne les emplois comme `moulin à bras`, `moulin à cylindres`, `moulin à meules`, les arguments décrivent des éléments intégrés au moulin (l'argument est dans une relation de partie constitutive de l'instrument décrit) : cette caractérisation par la facette `partie-de` met d'ailleurs en évidence des parties saillantes du moulin décrit. On peut définir un rôle constitutif (à la Pustejovsky) ou un rôle aspectal (à la Fradin) pour prendre en compte ce genre d'extension. La valeur prédicative relativement neutre peut d'ailleurs rester de mise tant que l'on n'en sait pas plus sur le moulin visé : quitte à spécifier et/ou sous-spécifier cette valeur si nécessaire (-> figure infra).

4.5.2.3. Des moulins à prières aux moulins à paroles

`Moulin à prières` est semble-t-il central pour la prise en compte de certains sens figurés du patron `moulin à N2`. A priori deux attitudes auraient été possibles pour inscrire `moulin à prières` dans la représentation sémantique initiée ci-dessus. Si `moulin à prières` est, dans la définition précitée, un instrument[61] constitué d'un "*cylindre renfermant des bandes de papier recouvertes de formules sacrées pour acquérir les mérites attachés à la répétition de cette formule*", on peut considérer que ce type d'instrument transforme des formes/signes graphiques en source sonore : les prières (i.e. il s'agit en quelque sorte d'une opération de décodage). Si l'on considère maintenant la relation qui existe entre `tourne-disque` et `disque`, on peut considérer que cet instrument opère un décodage du sillon inscrit sur le disque en une production de sons. A l'image de cette relation, on pourrait envisager d'associer à `moulin à prières` le

---

61 La solution 264 du le corpus donné en annexe en donne une définition précise: "*/./ un thibétain inscrit sa prière sur une bande et l'enferme dans un cylindre qu'il fait tourner dans un moulin à prières...*"

prédicat transformer avec [argument rôle : sortie], à la différence de la relation de transformation tourne-disque et disque pour laquelle [argument rôle : entrée]. Et c'est justement ce dernier point qui semble le plus inadéquat pour retenir le prédicat choisi. En effet, le processus de transformation privilégie fortement la donnée de l'objet à transformer plutôt que la donnée de l'objet issue de la transformation. Au niveau conceptuel, un tel choix serait donc maladroit pour décrire un instrument de transformation pour lequel l'objet transformé serait masqué. De plus, au niveau de la représentation, ce choix se serait révélé coûteux : un tel choix impose de disposer de filtres supplémentaires pour le prédicat choisi et surtout on verra que la solution suivante se révèle plus productive, notamment pour la prise en compte des sens figurés du patron moulin à N2 tels moulin à paroles ou moulin à conversation. Si l'on choisit en effet une solution qui associe à moulin à prières le prédicat qui traduit la création de l'argument donné, le lien logique entre le processus décrit et l'argument donné syntaxiquement est cohérent : le processus de création privilégie fortement la donnée de l'objet créé. Il faut donc compléter la définition des filtres initialement attachés au prédicat dans notre représentation sémantique précédente pour prendre en compte la classe des NParole pour le prédicat créer ([argument classe : NParole]). Ensuite, il semble possible, à partir de cette interprétation, de retrouver par dérivation les sens figurés que l'on rencontre pour les séquences moulin à N2. Les emplois tels moulin à paroles peuvent eux-aussi être associés au prédicat créer dans la mesure où il s'agit dans ce cas de la création d'un excès de paroles[62]. Moulin désigne donc dans ce cas une entité de type +humain. Nous devons donc ajuster ici d'ajuster la représentation du noyau de sens : moulin "ajuste" certains traits de sa définition sémantique initiale de machine; il ne s'agit plus de décrire initialement une machine dont les caractéristiques sont précisées par la structure argumentale donnée, mais il s'agit de décrire une personne[63] qui telle une machine produit une grosse quantité de paroles (c'est aussi la structure argumentale qui permet cette dérivation de sens). En termes de représentation prototypique, si l'on considère que moulin est typiquement une machine, on doit malgré tout pouvoir retrouver ce détour par rapport à la préférence sémantique attachée à moulin si les informations rencontrées le nécessitent. Cette nouvelle direction de sens pour moulin peut se réaliser sous la forme d'un clonage du prototype initial attaché à moulin (le prédicat est maintenu : créer, argument rôle : NParole, argument : +excès) avec parallèlement une assignation d'un nouveau type de comportement sémantique qui décrit l'attachement de ce représentant de moulin à la classe des Nhumains. On peut d'ailleurs remarquer que l'interprétation initiale de moulin à prières contient déjà le trait excès dans la production décrite par l'argument et que ce trait est maintenu dans le prototype dérivé. De plus, cette solution permet de concilier une autre interprétation pour moulin à prières : il peut s'agir d'une autre dénomination pour grenouille de bénitier (Habert & Fabre 1993). La figure 4.19 tente de rassembler les points de modification annoncés dans la Représentation Sémantique Prototypique attachée à moulin.

---

62 Entendu dans la bouche d'un locuteur canadien de langue française: "Les français sont des verbaux moteurs... de vrais moteurs à parler".

63 La solution 12 du corpus présente également la forme "moulin à papier" auquelle on peut associer "personne qui produit une grande quantité de papier" i.e. un copiste.

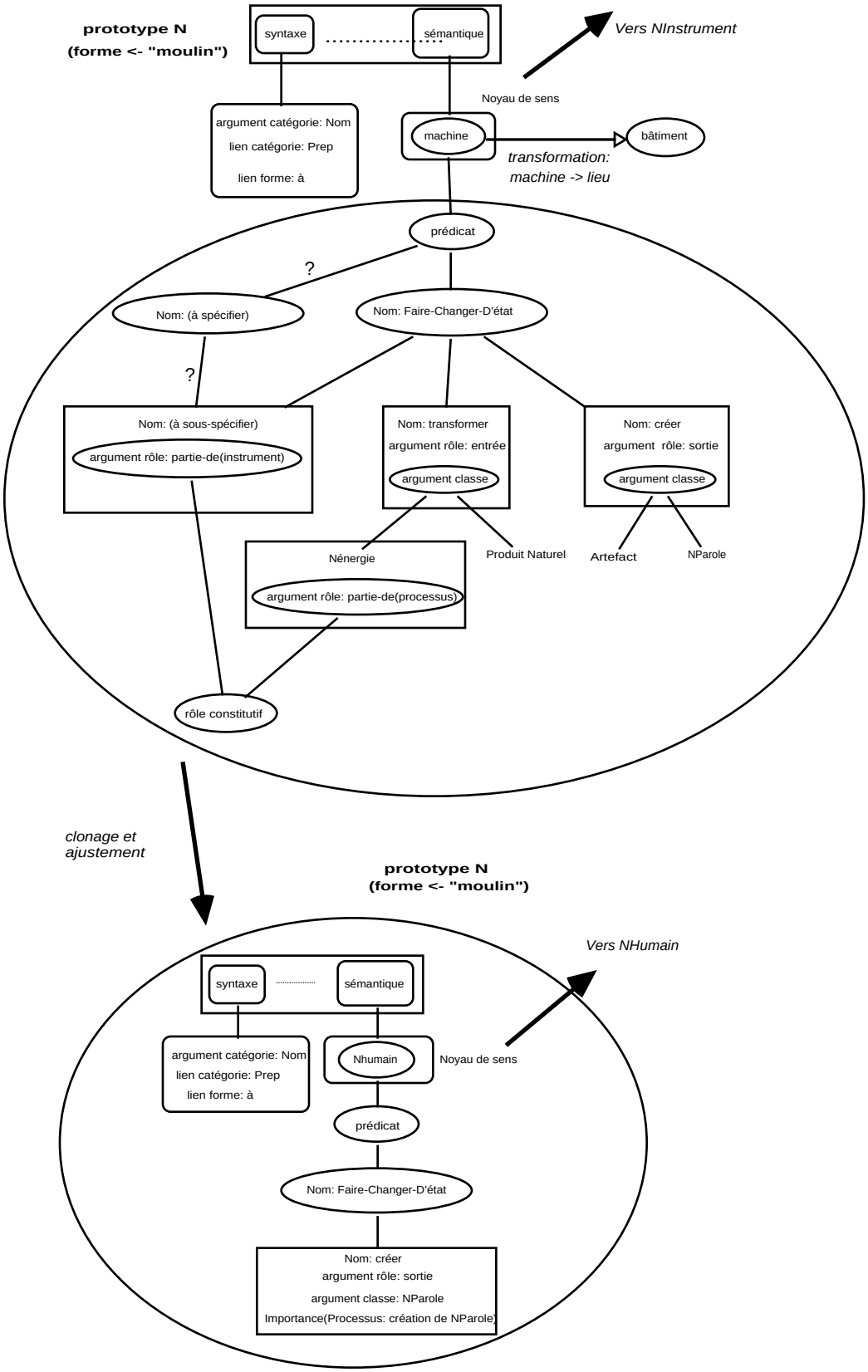


Figure 4.19 : Représentation Sémantique Prototypique.

Les emplois métaphoriques s'accordent semble-t-il avec la valeur prédicative retenue : il y a un changement d'état du processus de création en sortie de l'argument donné, celui-ci devient important et le résultat produit est massif.

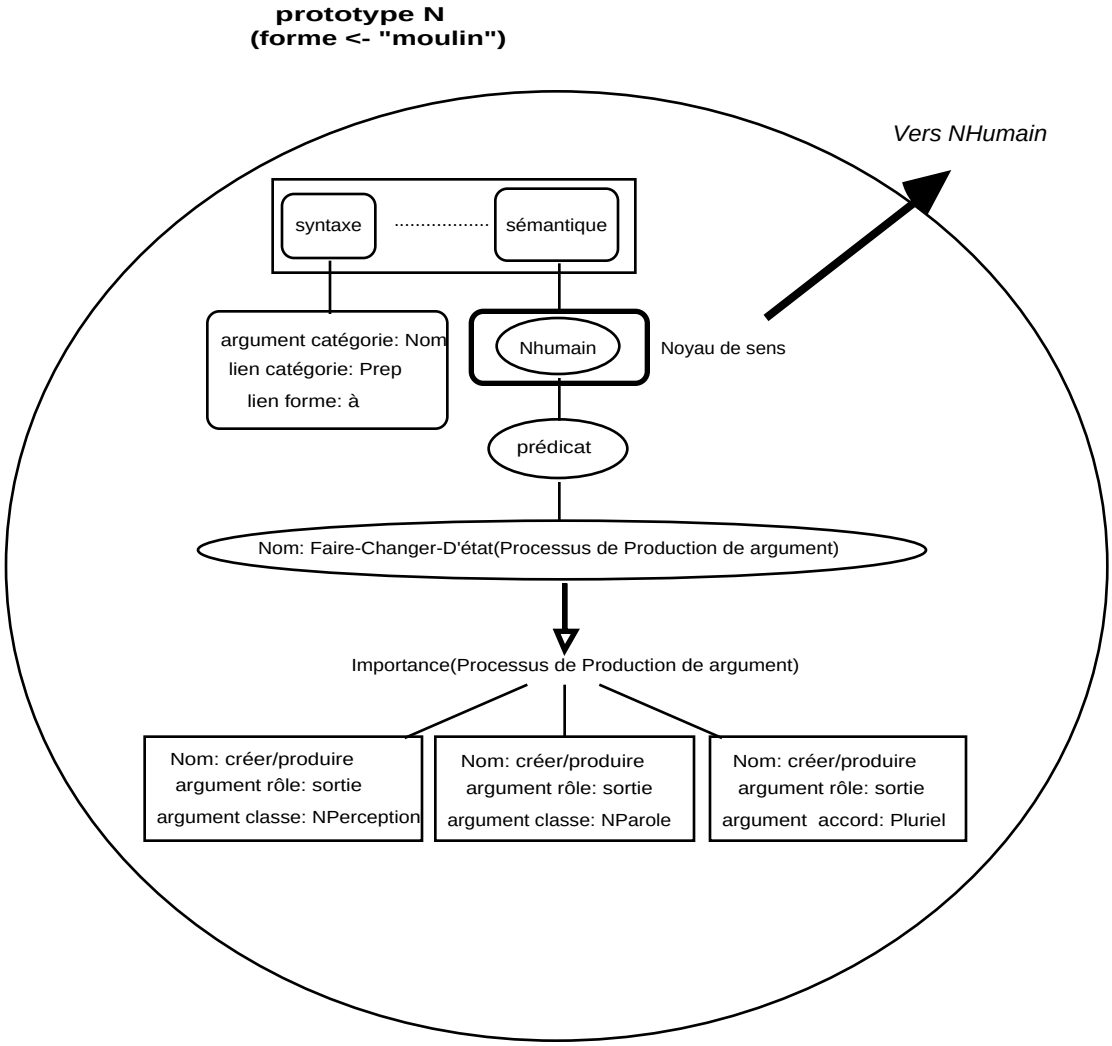


Figure 4.20 : Représentation Sémantique Prototypique pour `moulin Nhumain`.

4.5.2.4. `Moulin` et les déverbaux

Comme on l'a vu précédemment, certains exemplaires de `moulin à N2` ne décrivent pas ce que produit le moulin ni ce qu'il transforme, mais ils représentent un instrument constitué par le moulin. On retrouve là des processus généraux qui sont à l'oeuvre dans de nombreux composés en `N1 à N2` : une combinaison de deux prédicats qui s'articulent suivant la part que prend chacun des constituants du composé. On trouve par exemple dans (1) `étuve à incubation` ou (2) `ventilateur à aspiration` le même genre de relation entre un processus secondaire (le moyen) et un processus primaire (la fin) qui se réalise en sens inverse : (1) l'incubation se fait au moyen de l'étuve, (2) la ventilation se fait au moyen de l'aspiration. L'information sémantique portée par le `N2` peut ainsi prendre le pas sur celle du nom tête : dans (1) le `N2 déverbal` "pèse" sur la détermination du prédicat sous-jacent; alors que dans (2) c'est le "poids" du `N1 déverbal` qui importe. On rejoint donc ici une analyse des composés en `N1 à N2` qui force à réévaluer la place du `N2` dans la combinaison et qui tend à proposer un mode de calcul du sens en fonction de la nature des deux `N`. Le travail initié par (Habert & Fabre 93) s'oriente d'ailleurs depuis vers une analyse des `N à Ni` (noms composés désignant des instruments) en suivant le principe de co-compositionalité. Les différentes analyses

possibles s'ordonnent en effet sur un continuum en fonction de la part de chacun des composés donnés en suivant une échelle de co-compositionalité. On reprend ici le schéma proposé par Cécile Fabre :

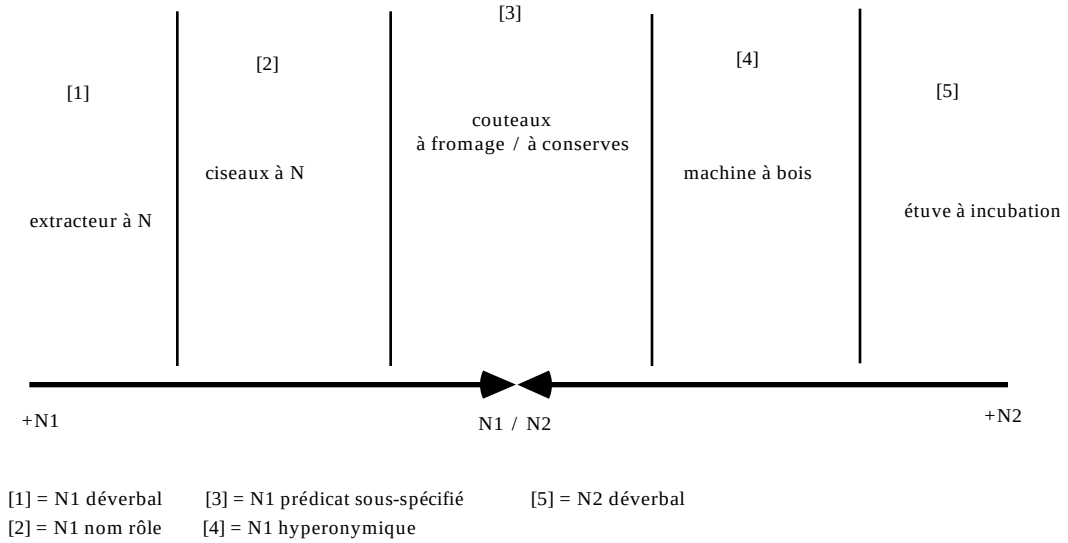


Figure 4.21 : La co-compositionalité pour l'analyse des `N à Ni`.

Dans les réalisations `moulin à (N2 déverbal)` mentionné supra, il en va ainsi pour la détermination du prédicat sous jacent : la ventilation se fait au moyen du moulin. Que reste-t-il malgré tout de la valeur prédicative attachée à moulin? Il reste possible de dire que l'on a un changement d'état de la valeur prédicative initialement attachée à `moulin` i.e. `Faire-Changer-D'état` (le prédicat comme agent de la réflexivité!) : le prédicat s'applique à lui-même si le `N2` est un déverbal et change sa valeur prédicative.

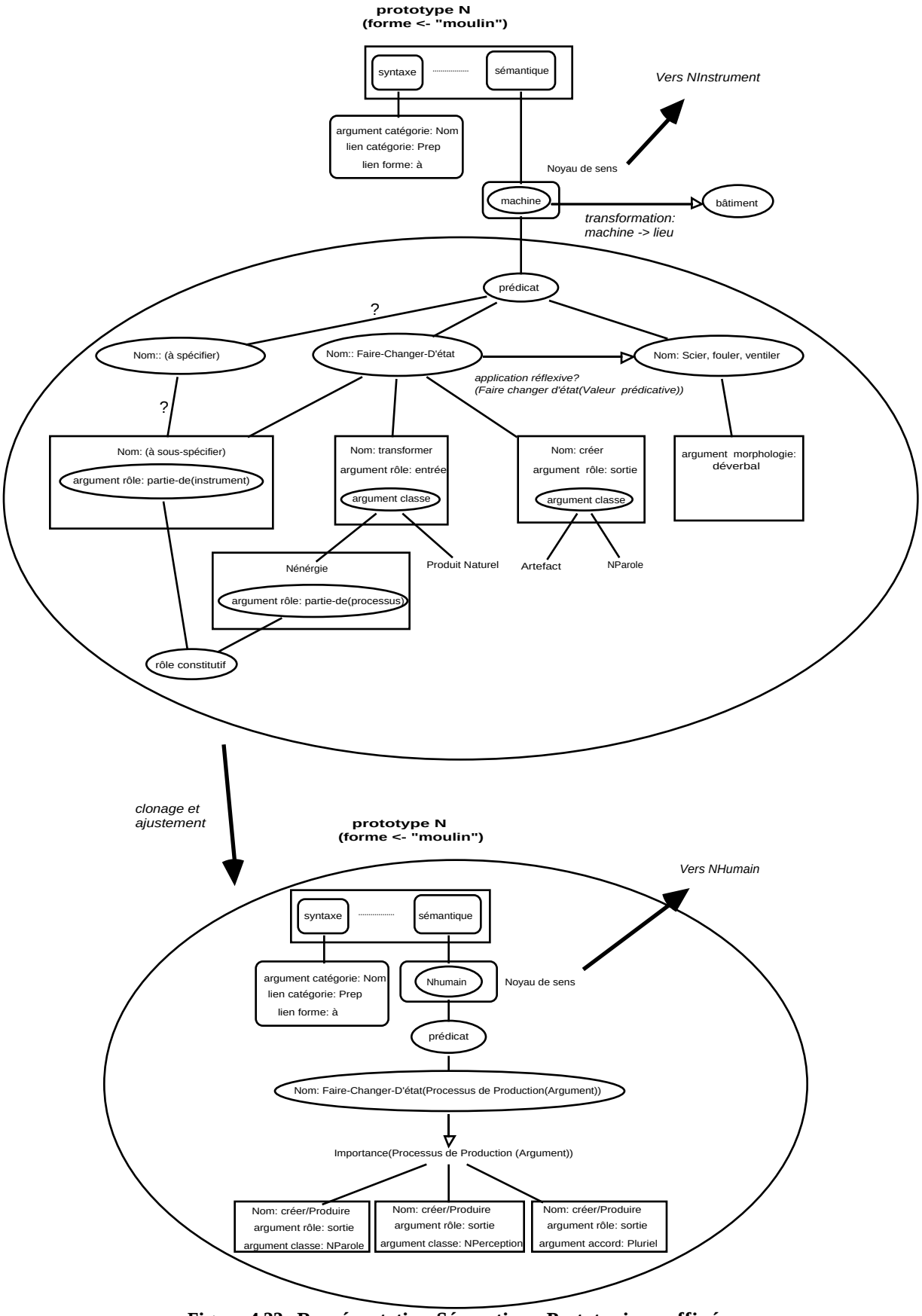


Figure 4.22 : Représentation Sémantique Prototypique affinée.

4.5.2.5. Articuler les niveaux de représentation

Cette représentation sémantique provisoire nous permet de repréciser la définition initiale attachée à `moulin` :

Un moulin[64] est :

- une machine :
  - dont le type est précisé par la structure argumentale qui lui est associée :
    `(tête : moulin , lien[catégorie : Préposition, forme : à], argument[catégorie : Nom]),`
    `(prédicat : Faire-Changer-D'état)`
    - une machine dont l'argument donné est un composant du moulin (`argument rôle : partie-de`) ($moulin_D$); la valeur prédicative donnée peut être à sous-spécifier ou à respécifier si nécessaire i.e. si des informations contextuelles supplémentaires le permettent.
    - une machine qui transforme un produit naturel ($moulin_{A1}$) : la valeur prédicative est sous-spécifiée en `transformer`, `[(argument classe : produit naturel) , (argument rôle : entrée)]`; elle peut être aussi une machine qui transforme une source d'énergie en une autre source d'énergie ($moulin_{A2}$) : la valeur prédicative est sous-spécifiée, de la même manière, en `transformer [(argument classe : Nénergie), (argument rôle : partie-de(processus))]`, ce dernier point, secondaire, indique simplement que la source d'énergie en entrée est partie intégrante du processus de transformation décrit.
    - une machine qui crée un artefact ($moulin_{B1}$) : la valeur prédicative est sous-spécifiée en `créer`, (`argument classe : artefact`); elle est aussi une machine qui produit des prières ($moulin_{B2}$) : la même valeur prédicative `créer` s'applique, (`argument classe : NParole`); ($moulin_{B2>}$) par extension décrit aussi un être humain qui telle une machine crée un excès de paroles, de conversation, (dans ce cas (`argument classe : NParole`) entraîne (`tête classe : Nhumain`)); la valeur prédicative sous-spécifiée traduit de plus le changement d'état du processus de production de l'argument donné : production/création importante ou massive.
    - une machine qui scie, ventile, foule et dont le moulin est un élément intégré à cette machine (`argument morphologie : déverbal`) ($moulin_C$); ici, c'est la valeur prédicative initiale qui a changé d'état.
  - du type moteur d'avion ou de voiture ($moulin_F$); on peut d'ailleurs remarquer que cette interprétation rejoint fortement ($moulin_C$).

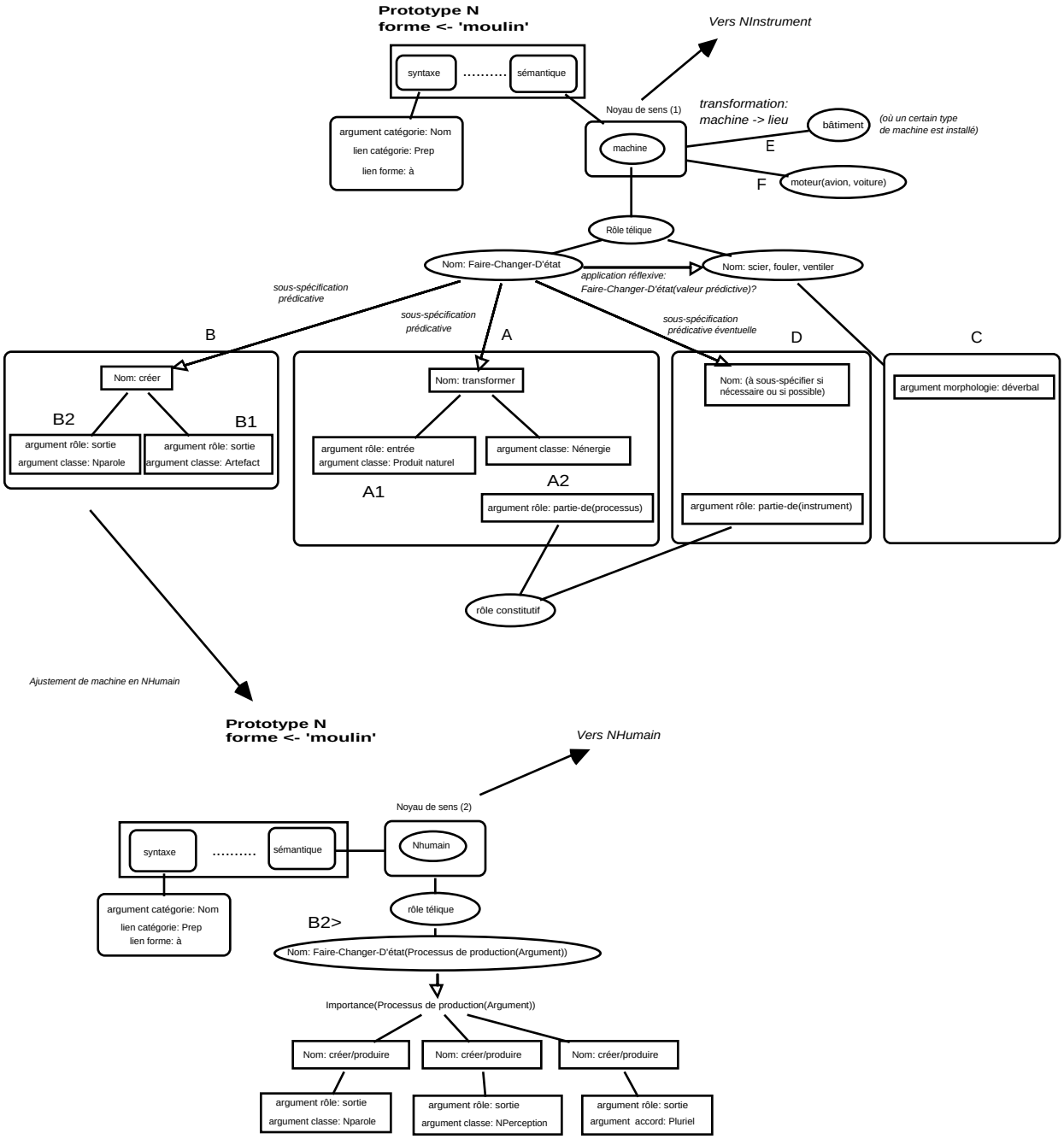
- par extension un bâtiment où un certain type de moulin (décrit supra) est installé ou l'a été ($moulin_E$) (sauf pour `moulin à prières` et pour les emplois métaphoriques);

Les emplois métaphoriques de `moulin` posent semble-t-il un autre type de problème (qui dépasse d'ailleurs le cadre de notre travail). En effet, on peut considérer que les métaphores de moulin envahissent le langage au moment où d'autres machines plus perfectionnées sont sur le point de détrôner le prototype de machine que constituent les moulins. On peut d'ailleurs tracer un parallèle entre ce développement des emplois métaphoriques de moulin au moment du développement industriel et le développement des emplois métaphoriques de l'électricité que l'on rencontre aujourd'hui. L'électricité est elle aussi amenée à être supplantée par d'autres sources d'énergie. Ce passage vers d'autres sources d'énergie s'accompagne dans le même temps d'une multiplication là aussi d'emplois métaphoriques : *"Pourquoi, alors que le courant passait de moins en moins bien, on pétait les plombs de plus en plus chez ceux qui avaient deux fils qui font masse. Fallait-il à ce point court-circuiter le langage officiel afin de créer un électrochoc dans la société? Manière, en quelque sorte, de se trouver quelques atomes crochus entres électrons libres. Histoire d'en finir avec les*

---

[64] Les indices attachés aux définitions possibles pour `moulin` font référence aux "pistes de sens" que la figure qui suit présente

branchés et leurs passions cathodiques. Le langage électrique stigmatisait avant tout le dérèglement, le dérangement, la panne dans la continuité du circuit. Il vieillissait vite, mais son emploi en rajeunissait certains. Par contraste, il était remarquable de constater que l'informatique n'avait produit jusqu'alors qu'un vocabulaire, mais pas encore de langage, malgré les efforts de novlangue" (Gilles Tordjman in *Les Inrockuptibles* n°10 à propos du livre *Atlas occidental* de Daniele del Giudice). La principale difficulté que met en lumière cette nouvelle description sémantique de `moulin` est de faire appel à des savoirs de différentes natures pour fournir une interprétation possible ou des éléments pour un début d'interprétation. Le problème de l'articulation de ces savoirs ne se laisse pas appréhender sans difficulté. Les aspects sémantiques ne sont pas pertinents de manière régulière dans la construction des différentes interprétations possibles. La fonction, la constitution de l'objet décrit, la sémantique de l'argument etc. sont autant de facettes dont il faut tenir compte pour reconstruire une signification possible, d'autant plus que ces savoirs peuvent eux aussi agir en retour sur la représentation sémantique. Il reste malgré tout possible de tracer une esquisse des directions de sens qui semblent se dessiner dans cet ensemble d'interprétations. Si `moulin` est une machine, le type de machine en question sera explicité ou esquissé par les informations que l'on pourra récupérer dans la structure argumentale associée. La sémantique résultante n'est d'ailleurs pas figée : certaines informations sémantiques extérieures peuvent affiner le savoir attaché à `moulin`. Tous les moulins n'ont pas de rôle télique prédéfini dans cette représentation (ils peuvent malgré tout hériter d'une valeur prédicative générique comme dans les figures présentées supra); il n'est d'ailleurs pas sûr qu'ils en aient tous un; on peut cependant penser que `moulin à cylindres`, par exemple, peut recevoir des informations sémantiques supplémentaires dans un contexte donné qui précisent la représentation sémantique initiale que décrit la figure 4.23.



(1) traduit en particulier que moulin est un instrument concret, comptable, inanimé...
(2) traduit quant à lui que moulin est devenu +humain...

Figure 4.24 : `Moulin`, des pistes de sens.

Cette mise en ordre de l'interprétation ne limite pas les ajustements qu'il reste possible de faire : soit en précisant plus avant les aspects sémantiques décrits, soit en examinant de nouvelles séquences que les présentes hypothèses ne prennent pas encore en compte. Il reste en effet possible de dégrossir cette cartographie des directions de sens possibles en y greffant de nouveaux points d'entrée sur les noeuds existants ou en précisant les chemins déjà tracés.

Interprétations de `moulin` et représentations logiques :

En reprenant les notations de Fradin (Fradin 93) :

`Qtél` *"concerne les propriétés téliques au sens large : la fonctionnalité de l'objet aussi bien que son origine"* .
`Qmod` *"concerne le mode d'existence de l'objet : essentiellement sa catégorisation par rapport aux enchaînements de causalité"* .
`Qasp` *"concerne les propriétés aspectales et constitutives : rapports tout/partie, figure/site, et propriétés servant à identifier l'objet en question"* .

On peut construire les représentations sémantiques suivantes pour `moulin` :

- Interprétation A1 : `moulin à blé`
  ```
  (moulin x)
  Qtél = (Transformer y x) et (ProduitNaturel y)
  Qmod = (instrument x)
  ```

- Interprétation A2 : `moulin à vent`
  ```
  (moulin x)
  Qtél = (Transformer y x) et (Energie y)
  Qmod = (instrument x)
  ```

- Interprétation B1 : `moulin à farine`
  ```
  (moulin x)
  Qtél = (Créer y x) et (Artefact y)
  Qmod = (instrument x)
  ```

- Interprétation B2 : `moulin à prières`
  ```
  (moulin x)
  Qtél = (Créer y x) et (NParole y)
  Qmod = (instrument x)
  ```

- Interprétation B2> : `moulin à paroles`
  ```
  (moulin x)
  Qtél = (Créer y x) et (Production(NParole) y) et (Important y)
  Qmod = (NHumain x)
  ```

- Interprétation C : `moulin à ventilateur`
  ```
  (moulin x)
  Qasp = (Partie y x) et (Instrument y)
  Qtél = (Faire-Changer-D'état)
  Qmod = (PartieDe[Instrument] x)
  ```

`(Partie x y)` se lit `(y a pour partie x)` i.e Instrument avec moulin.

- Interprétation D : `moulin à cylindres`
  ```
  (moulin x)
  Qasp = (Partie y x) et (PartieDe[Instrument] y)
  Qtél = (Faire-Changer-D'état)
  Qmod = (instrument x)
  ```

- Interprétation E :

```
(moulin x)
Qtél = (Faire-Changer-D'état)
Qmod = (instrument x)
```

- Interprétation F :

```
(moulin x)
Qmod = (bâtiment x)
```

Une représentation sémantique pour `moulin` peut donner la structure suivante :

```
(moulin x)
Qasp1 = (Partie x z) et (PartieDe[Instrument] x)
Qasp2 = (Partie z x) et (PartieDe[Instrument] z)
Qtél0 = (Faire-Changer-D'état x y)
Qtél1 = (Faire-Changer-D'état x y) et (Artefact y)
Qtél2 = (Faire-Changer-D'état x y) et (ProduitNaturel y)
Qtél3 = (Faire-Changer-D'état x y) et (ProductionDeParoles y) et (Importance y)
Qmod1 = (instrument x)
Qmod2 = (instrument z)
Qmod3 =(NHumain x)
Qmod4 =(Bâtiment x)
```

4.5.2.6. Préférences d'interprétation et prototypie

Si la notion de saillance est fondamentale dans une approche prototypique, une réflexion sur l'organisation des connaissances en termes de préférence d'interprétation reste à faire: on peut ainsi envisager de mettre en avant les interprétations "centrales". Toutes les interprétations n'ont pas le même "poids"; il semble donc cohérent d'articuler les savoirs représentés en tenant compte des réalités de production rencontrées.

Les résultats rassemblés ci-dessous permettent d'avancer que les séquences `moulin à N2` sont avant tout des instruments de transformation puis des instruments de création (aux sens réels de production d'une entité du monde physique ou sonore, comme aux sens figurés où l'activité humaine induite produit un excès; plus loin, ils peuvent être des instruments de transformation d'une source d'énergie ou des instruments dont l'énergie est apportée par le moulin pour réaliser une action; et enfin ils peuvent être des instruments dont la dénomination n'est pas explicite mais dont on sait qu'ils sont constitués par un élément donné (ces derniers relèvent plus en effet de la description). Il apparaît que le rôle télique et la qualité sémantique de l'argument sont déterminants et suffisants dans une majorité de réalisations `[transformer(entrée : produit naturel], créer(sortie : artefact, NParole(B2), Nparole(B2>)]`, et que le rôle constitutif soit nécessaire pour compléter le filtrage pour les autres réalisations.

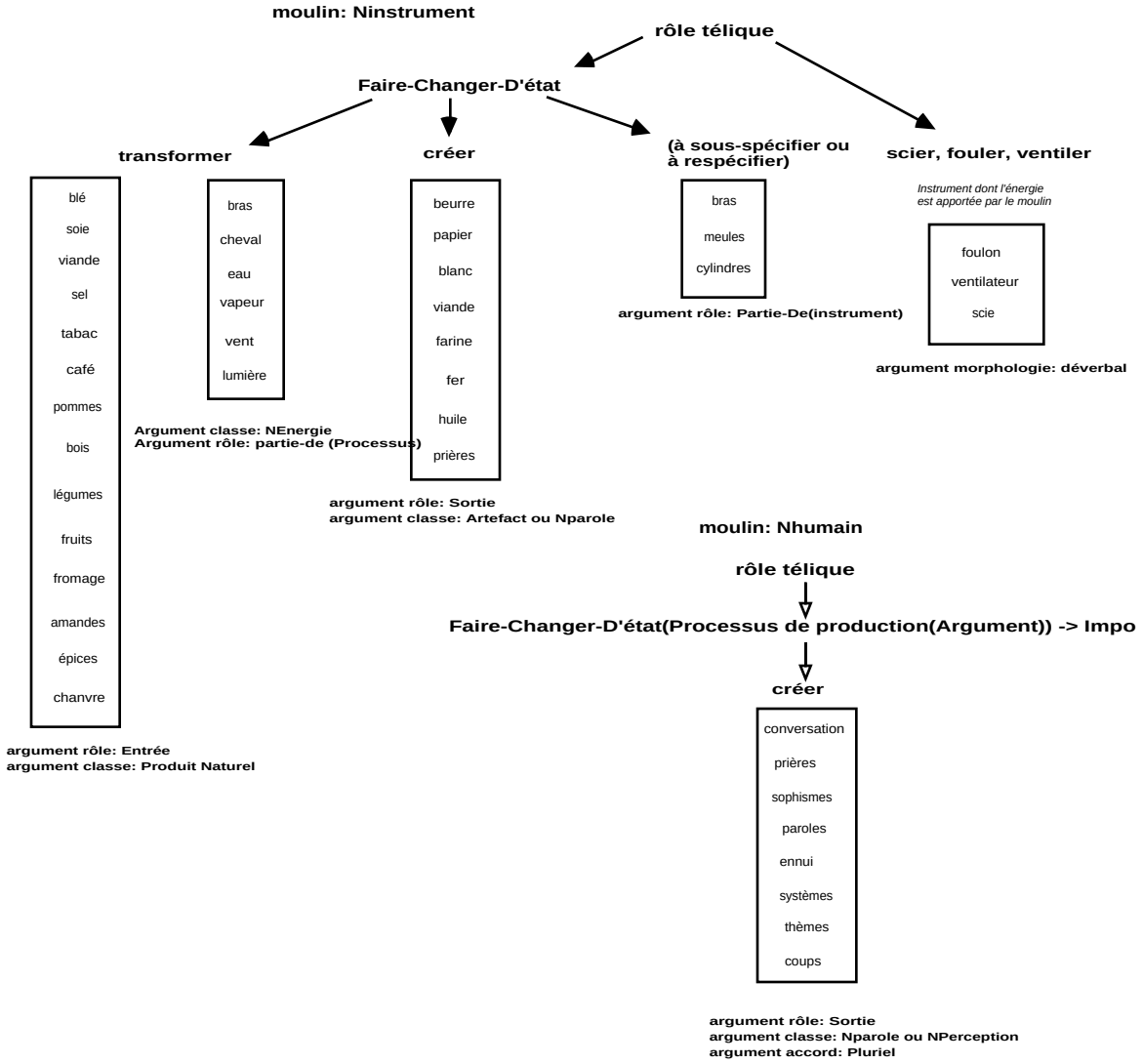


Figure 4.24 : `Moulin`, des préférences d'interprétation.

Peut-on y voir une préférence d'interprétation qui se développe en parallèle de la plus vers la moins conceptuellement définie et de la moins vers la plus descriptivement définie ? Le problème de la préférence d'interprétation peut d'ailleurs être envisagé au cours de l'analyse par une confrontation dynamique des savoirs représentés.

Si `moulin à bras` appartient à deux classes sémantiques dont l'une est plus productive en termes de réalisations attestées, il ne faut pas pour autant laisser de côté l'interprétation "la plus faiblement" réalisée. Si telle interprétation est plus saillante, il semble cohérent de la privilégier mais il est impératif de récupérer les autres interprétations possibles. On peut d'ailleurs considérer que la représentation sémantique attachée à `bras` s'accorde avec cette volonté de représenter le flexible de la langue qui se matérialise dans ce cas par une conversion de type dans une des interprétations possibles pour `moulin à bras`. On peut ainsi considérer que l'interprétation `A2 [rôle télique : transformer, argument rôle : Nénergie]` soit initiée par la représentation sémantique attachée à `moulin` puis résolue par la représentation sémantique attachée à `bras` qui permet une conversion de type. On construit ainsi les deux interprétations possibles : des informations supplémentaires devraient permettre de choisir l'une ou l'autre des interprétations. De la même manière, on peut associer à `cheval` un rôle constitutif qui exprime le fait qu'un `cheval` peut être un composant d'un processus de traction, de poussée, etc.; `cheval` devient dans ce cas par conversion un `Nénergie`.

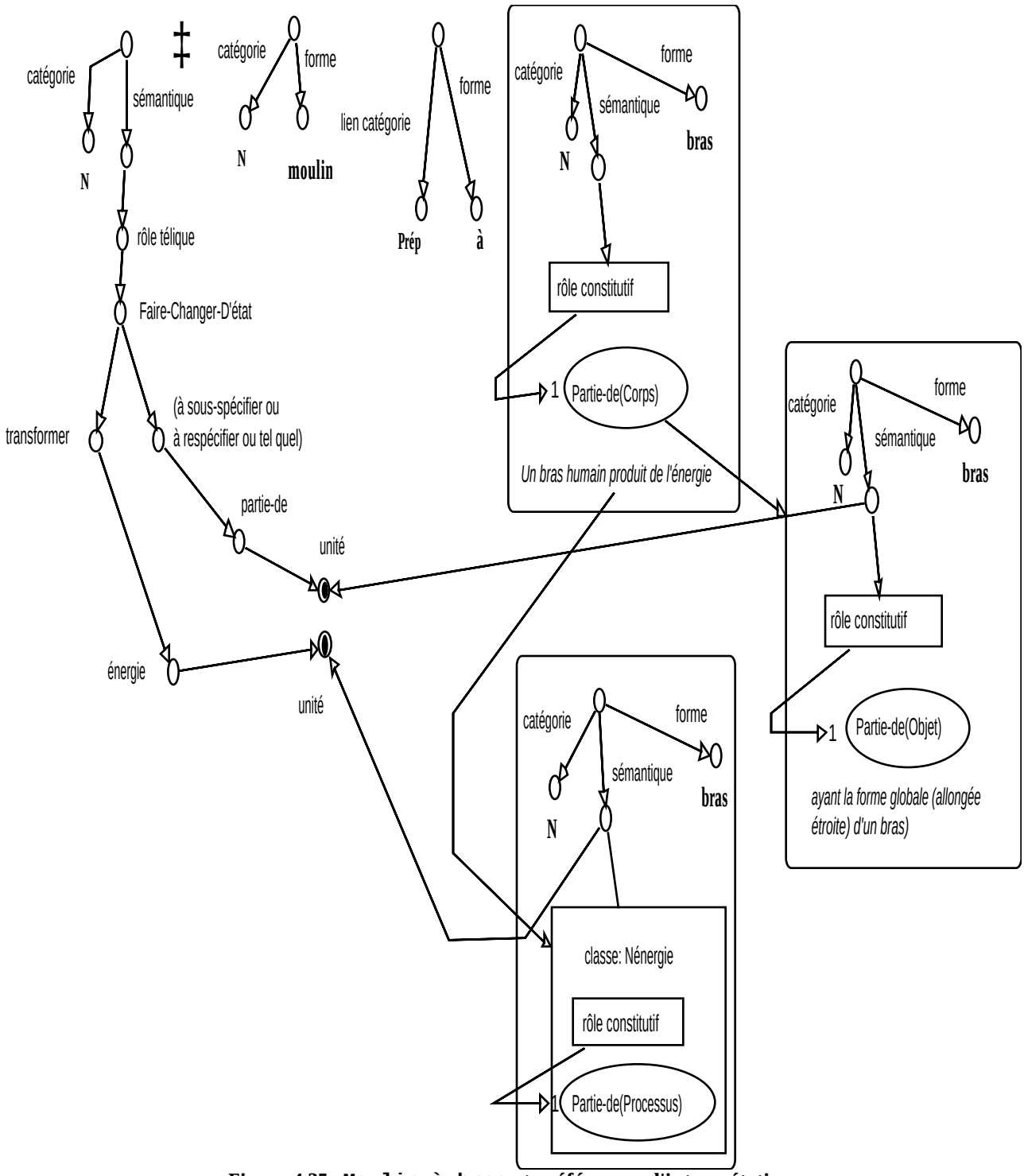


Figure 4.25 : `Moulin à bras` et préférences d'interprétation.

4.5.2.7. `Moulin` et les noms d'instruments

Etendre la représentation sémantique aux noms d'instruments : comment mettre en oeuvre une catégorisation plus générale tout en préservant une évolution potentielle de cette représentation initiale? Cette représentation sémantique tend à identifier (à isoler) les comportements sémantiques que l'on peut attacher à `moulin` et qui permettent de construire des (débuts d') interprétations possibles qu'il reste possible d'affiner. Ces comportements traduisent-ils des opérations sémantiques plus générales qu'il serait possible

de factoriser pour une classe de noms particuliers (en particulier pour les noms d'instrument). La figure suivante illustre ce que pourrait être cette recatégorisation.

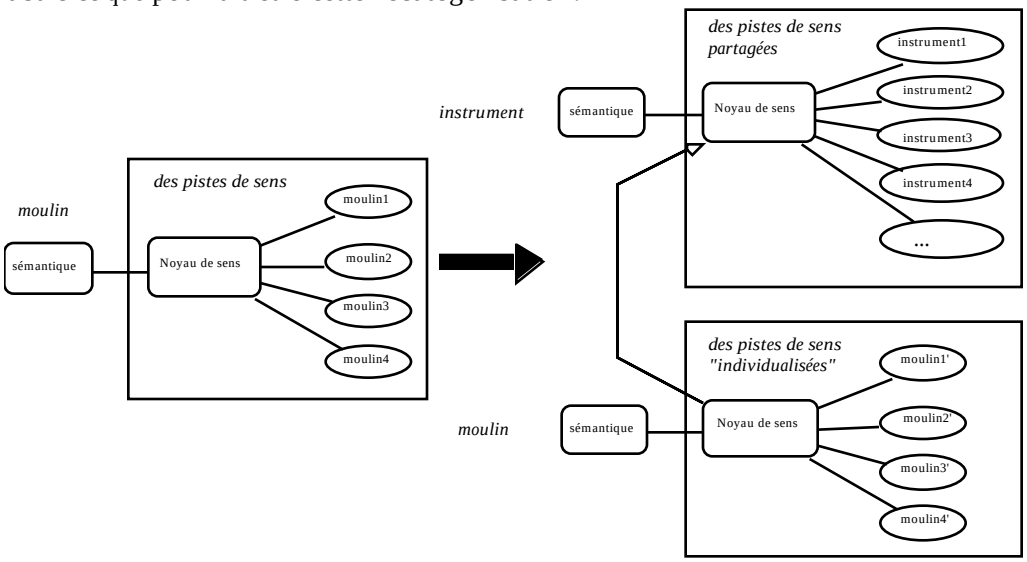


Figure 4.26 : `Moulin` et noms d'instrument, des comportements sémantiques partagés.

Si une approche prototypique se révèle adéquate pour rendre compte d'un micro-phénomène particulier, il convient de ne pas perdre de vue la place de ce micro-phénomène dans un système plus complexe, ou tout au moins de dégager de l'étude particulière de celui-ci, les éléments qui le dépassent.

4.5.2.8. Construire un espace de concepts

Si les concepts n'existent qu'au travers de nos activités de langue, l'approche retenue s'accorde, semble-t-il, assez bien avec la construction d'un espace de concepts qui eux-aussi se définiraient par touches successives. L'approche montante poursuivie à partir de représentations prototypes sous-déterminées vise à construire une représentation sémantique des unités linguistiques manipulées par affinements successifs; l'espace conceptuel à venir doit donc se mettre en place à mesure que la construction de l'espace sémantique se développe. Ce que montre le travail de D. Kayser est qu'il est impossible d'anticiper toutes les significations que l'on peut attacher à un mot, et qu'au contraire celles-ci se distinguent au cours de la construction dynamique d'une interprétation. Il est cependant nécessaire de disposer de moyens efficaces de pouvoir ajuster les représentations qui sont faites dans une modélisation des activités d'interprétation. De la même manière que la figure précédente tend à organiser, pas à pas, le savoir sémantique (en organisant une généralisation minimale et un savoir particulier modifiable), il faut rendre possible la mise en place d'un espace conceptuel qui s'articule au mieux à ce qui est en cours de construction.

4.6. Comment nomme-t-on les choses ?

"Si tu vois quelque chose de loin et ne comprends pas de quoi il retourne, tu te contenteras de le définir comme un corps en extension. Quand il se sera approché de toi, tu le définiras alors comme un animal, même si tu ne sais pas encore s'il s'agit d'un cheval ou d'un âne. Et enfin, quand il sera plus près, tu pourras dire que c'est un cheval, même si tu ne sais pas encore si c'est Brunel ou Favel. Et seulement quand tu seras à la bonne distance, tu verras que c'est Brunel (autrement dit ce cheval et pas un autre, quelle que soit la façon dont tu décides de l'appeler)." (Umberto Eco, Le nom de la rose, Livre de Poche 1982).

Quand on ne sait pas quelle désignation précise donner aux choses auxquelles nous sommes confrontées, appréhende-t-on ces éléments dans une globalité indéfinie, quitte à préciser ultérieurement ce flou initial? Ou bien tente-t-on de repérer les spécificités de ces éléments qui nous permettront de l'identifier? Pour nommer, classifier, il faut voir les choses de près, identifier les éléments, composants de celles-ci, pour les comparer puis les apparier ou ranger dans les classes déjà connues. Si la spécificité d'une chose qui permet de la désigner précisément n'est pas directement perceptible, on est dans la situation d'un observateur éloigné de la chose à identifier et qui ne peut reconnaître celle-ci de manière certaine. De plus, suivant les savoir-faire, les points de vue, les domaines d'expérience, les choses à nommer ne seront pas appréhendées de manière identique d'un individu à un autre : les hommes (mâles), les kangourous, les opossums et les chauves-souris, l'arc-en-ciel sont désignés par une même expression par les arborigènes d'Australie, de même qu'une fleur, le maïs et le bison chez les Osages. L'étude de la catégorisation devrait contribuer à éclairer le questionnement constant sur ce *"qui permet à la pensée d'opérer sur les êtres une mise en ordre, un partage en classes, un groupement nominal par quoi sont désignées leur similitudes et leurs différences,..., là où depuis le fond des temps le langage s'entrecroise avec l'espace"* (Michel Foucault 1966).

Partie 2 : Du possible

Chapitre 5

"Au début, on ne peut qu'essayer de nommer les choses, une à une, platement, les énumérer, de la manière la plus banale possible, de la manière la plus précise possible, en essayant de ne rien oublier".
Récits d'Ellis Island de G.Perec avec R. Bobert P.O.L

Classifier en terrain mouvant : Approche prototypique pour la construction du sens de moulin à N2

5.1. "Parti de rien, virgule..."

La linguistique informatique a depuis longtemps[65] adopté les mécanismes basés sur l'héritage (taxonomie de classes) pour la représentation des connaissances. Ce type de mécanisme se révèle efficace pour modéliser des domaines de connaissances stabilisées. Dans les domaines de représentation des connaissances où on ne dispose pas encore de savoirs figés, une approche objet à taxonomie de classes nécessite, dans l'état actuel de nos connaissances sur les méta-représentations ou sur les connaissances capables de créer de nouvelles connaissances, des choix initiaux de représentations qu'il peut être coûteux de modifier : les modifications superficielles pouvant entraîner une refonte complète de l'existant. Si la représentation hiérarchique souvent adoptée en linguistique informatique n'est pas adaptée au problème de représentation des connaissances mouvantes, évolutives ou instables, il est légitime de la remettre en question. L'idée importante dans la notion de prototype est celle de la non-rigidité du système qui utilise ce type d'objets : dans ce cadre les rapports entre objets n'ont aucune part de prédéfinition. Une autre notion importante dans les langages à prototypes (qui rejoint d'ailleurs la précédente) est celle de l'éclatement des connaissances au sein d'une quantité aléatoire d'objets. Ce type de langages permet d'envisager le développement de programmes utilisant aisément des données modulaires.

[65] La dizaine d'années qui nous sépare des textes fondateurs de Gazdar (1985-1990) est une quasi éternité en temps informatique.

5.2. Prototypes et Analyse Automatique

5.2.1. Préliminaires

Le travail présenté ici amorce la mise au point d'une analyse automatique qui utilise les potentialités des langages à prototypes (-> incise 1) : au niveau de la représentation des connaissances de la langue et au niveau des éléments pertinents pour l'analyse. Les choix faits ci-dessous pour la représentation des entités linguistiques pour cette phase expérimentale d'utilisation des prototypes à la Self dans une analyse automatique ne visent absolument pas à donner une réponse ou une illustration de choix théoriques pré-supposés. Il s'agit d'établir tout d'abord un parallèle avec un travail en cours de réalisation dans un modèle objet à taxonomie de classes. Les choix faits s'inspirent d'une part de la tradition (récente) en linguistique de traiter de la typicalité (en privilégiant le champ d'étude autour des propriétés du système lexical) et d'autre part de la prise en compte des travaux existants sur l'analyse automatique dirigée par le lexique (WEP en particulier dont nous reparlerons (-> présentation de WEP)).

5.2.2. Cadre linguistique

Le point d'ancrage linguistique pour l'expérimentation d'une analyse automatique prototypique est celui de l'étude syntaxique des dénominations complexes de la forme `N1 à N2` définie par (Habert & Fabre 1993). Ce travail présente *"une approche hiérarchique de la construction du sens de dénominations complexes de la forme N1 à N2"* qui utilise les mécanismes mis en place par DATR. Le modèle proposé permet de formuler des hypothèses sur l'occupabilité des places argumentales dans le patron `moulin à N2`. Une partition des possibilités peut se matérialiser par les ensembles suivants :

moulin POUR/sortie moulin POUR/entrée moulin AVEC/énergie moulin AVEC/partie-de

L'ordre d'écriture des ensembles ainsi défini de réalisations des structures possibles sur le patron `moulin à N2` correspond à la réalisation quantitative de telles structures : les occurrences attestées et regroupées dans les ensembles ci-dessous diminuent quand on lit les ensembles dans l'ordre donné. Cette analyse permet d'une part d'éliminer des séquences données si celles-ci ne rentrent pas dans le cadre ainsi défini, d'autre part de classer les séquences `moulin à N2` en fonction des compatibilités observées. De plus, on peut ainsi définir des bases pour des heuristiques évaluant la classification des lexèmes à entrées multiples dans le cadre défini par la classification supra. Le travail présenté par (Habert & Fabre 1993) s'inscrit dans un cadre d'analyse tel que la construction du sens s'établit non pas par la juxtaposition et la combinaison des sens des entrées lexicales mais par la construction progressive en précisant pas à pas les hypothèses initiales, quitte à rester dans l'indétermination quand les informations disponibles ne permettent d'aller plus loin. Il s'agit de mettre en place un travail d'affinage des constructions développées par l'analyse qui progresse du plus général vers le particulier. Il s'agit donc aussi d'évaluer en parallèle les résultats obtenus dans une approche de représentation des connaissances qui utilise les notions d'objets et de structuration hiérarchique au sens de la PàO (c'est le cas du travail présenté par (Habert & Fabre 1993)) et les résultats que l'on va construire ici dans une approche de représentation des connaissances avec des prototypes.

5.2.3. Approches pour une Analyse automatique prototypique avec Self

5.2.3.1. Phase0 : Filtrage Catégoriel

Tous les éléments constitutifs du système mis en place pour cette phase introductive à une analyse automatique prototypique ont été définis comme des prototypes suivant les règles de programmation définies par Self 2.0 puis Self 3.0 (-> incises 1, 2, 3). Cette phase introductive peut être considérée comme une phase de familiarisation avec l'environnement informatique proposé par Self via une analyse syntaxique simplifiée à l'extrême.

Existe-t-il des éléments prototypiques dans la langue ?

Il ne s'agit évidemment pas de fournir une réponse définitive à cette question d'entrée de jeu. Cette question n'était d'ailleurs pas l'objet principal de ce travail, même si l'ombre de cette question revient souvent au cours des réflexions sur les développements successifs de ce travail. Le choix fait de donner la primauté à une approche prototypique lexicalisée s'est révélée la plus évidente, mais ce choix n'est évidemment pas une réponse définitive à la question initiale : c'est simplement la vision la plus élémentaire et la plus immédiate qui a permis de démarrer.

Prototype et analyse de mots

Il s'agit dans un premier temps de fournir une chaîne de caractères représentant une dénomination complexe de la forme `N1 à N2` du type `moulin à paroles` et de construire à partir de cette chaîne un objet prototypique associé à celle-ci regroupant en son sein trois prototypes distincts : un prototype de type `N` associé à la chaîne "moulin" , un prototype de type `Prep` associé à la chaîne "à" et un prototype de type `N` associé à la chaîne "paroles".

Représentation du lexique et des catégories grammaticales

La démarche retenue dans un premier temps consiste à associer à chaque entrée lexicale un prototype en utilisant pour cela les catégories grammaticales classiques. Les choix de représentation des connaissances lexicales dans cette phase de démarrage n'ont pas été formalisés de manière définitive. Il s'agit de disposer de représentation minimale qui privilégie l'association d'une entrée lexicale donnée et de la catégorie grammaticale associée à cette entrée lexicale (i.e sans prendre en compte les ambiguïtés possibles) : les catégories retenues ici sont (`Nom, Préposition`). De plus chacun des prototypes construits porte des attributs[66] qui correspondent aux valeurs catégorielles habituellement associées à ces entrées lexicales : `genre`, `nombre`, `semantique` et un attribut `forme` qui portera la valeur de l'entrée lexicale associée au prototype. On associe aux entrées lexicales de type `Nom` un prototype dont la définition reprend celle des noms communs. Le prototype `catN` possède les attributs précités. Il possède aussi un attribut `parent` qui lui permet de déléguer certains comportements (en l'occurrence, le clonage et l'impression des valeurs attribuées aux attributs donnés) au `traits catN`. Ces attributs ont pour valeur des prototypes nommés respectivement `genre, nombre, semantique et forme` (qui délèguent eux aussi les mêmes types de comportement à des `traits` respectivement de même nom). On définit de même les prototypes `catPrep, catNaN`. On donne en annexe (-> annexe partie 2 chapitre 5) le code correspondant à la création des `traits` et `prototypes catN`.

Contrôle et analyse dans un monde de prototypes

Dès cette première phase de notre implémentation, la **notion de contrôle** sur les entités construites s'est révélée cruciale. En effet si l'on veut pouvoir évaluer, tracer et contrôler les activités des prototypes construits, il est impératif de pouvoir les retrouver aisément. La difficulté étant que la PàP tend à définir des unités a priori peu connectées entre elles. La première difficulté a consisté à pouvoir "mettre la main" sur les prototypes ainsi définis. Une première approche de parsage a défini un prototype de parseur chargé de "détecter" parmi tous les prototypes construits dans une session de travail ceux qui possèdent un attribut `forme`. Ce prototype de parseur doit vérifier si la valeur associée au champ `forme` du prototype lexical rencontré correspond à la chaîne de caractère donnée en entrée. Il doit ensuite pouvoir "stocker" les prototypes qui répondent à cette recherche. La même chose est ensuite mise au point avec une suite de chaînes de caractères (une phrase). On obtient ainsi en sortie une liste regroupant les entrées de type `string` donnée en entrée puis la liste des vecteurs composés des `miroirs`[67] pointant sur les prototypes

---

66 Ces éléments seront explicités dans les phases suivantes.

67 La notion de `miroir` fait appel au mécanisme de réflexivité de structure disponible en Self (Cointe & Mulet 1993). Les `miroirs` sont des entités qui rendent visibles au niveau du langage les informations contenues dans la représentation interne des objets. Les `miroirs` sont donc des objets, qui en fait ne contiennent aucune information en eux-mêmes, mais qui permettent de donner accès en lecture aux informations des objets du système via l'utilisation d'un ensemble de primitives prédéfinies.

associés à chacune des chaînes de caractères lues. La solution retenue pour exercer un premier type de contrôle sur les prototypes construits a donc consisté à utiliser très rapidement des processus de **méta-représentation** pour garder la main sur les prototypes construits et ainsi interroger leur structure. Nous reviendrons infra sur la présentation de tels processus et nous verrons aussi que l'utilisation de ces processus va s'avérer nécessaire tout au long de ce travail d'implémentation.

La deuxième étape a consisté à construire les prototypes capables d'effectuer une analyse sur un morceau de phrase et de produire un résultat plus facilement exploitable. Un prototype `regle` est introduit, il regroupe en fin d'analyse tous les prototypes construits suivant le modèle syntaxique associé à ce prototype (-> annexe partie 2 chapitre 5). Dans notre cadre de travail, le modèle syntaxique retenu peut être représenté par la règle hors contexte : `N1àN2 -> N1 Prep N2 (où N1 = moulin)`. Le démarrage de chaque analyse initialise d'une part les attributs du prototype associé à la règle à reconnaître/construire et d'autre part enregistre la suite de mots à analyser (-> annexe partie 2 chapitre 5).

L'analyse commence et se déroule donc de la manière suivante :

- 1. Lecture de la suite de mots à analyser.

- 2. On prend les entrées lexicales enregistrées une par une et on recherche les prototypes lexicaux associés qui permettent de mettre à jour la règle à construire.

Le déroulement de l'analyse ainsi développée est représenté dans la figure 5.1 :

- 1. Lecture de la suite de mots à analyser de longueur `InputTextSize :=3`.

- 2. Pour chaque mot, recherche de tous les prototypes existants ayant un attribut `forme`, sélection de celui ayant un attribut `forme` dont la valeur est identique au mot examiné.

- 3. Si le prototype retenu (via le prototype `category` contenu dans l'attribut `categorie`) est de même catégorie grammaticale que celle attendue par la règle à construire, on ajoute celui-ci dans la règle via l'attribut `RhsContent` qui contiendra tous les prototypes associés aux mots lus.

- 4. En fin de lecture des entrées lexicales, (5a) on vérifie si les tailles respectives de la règle à construire (`rhs`) et du contenu prototypique de la règle à construire coïncident. Si tel est la cas (5b), on peut construire la partie gauche (`N1 à N2`) : on récupère pour cela, chacun des prototypes reconnus précédemment et on les affecte à leur position respective dans le prototype `N1 à N2` à construire.

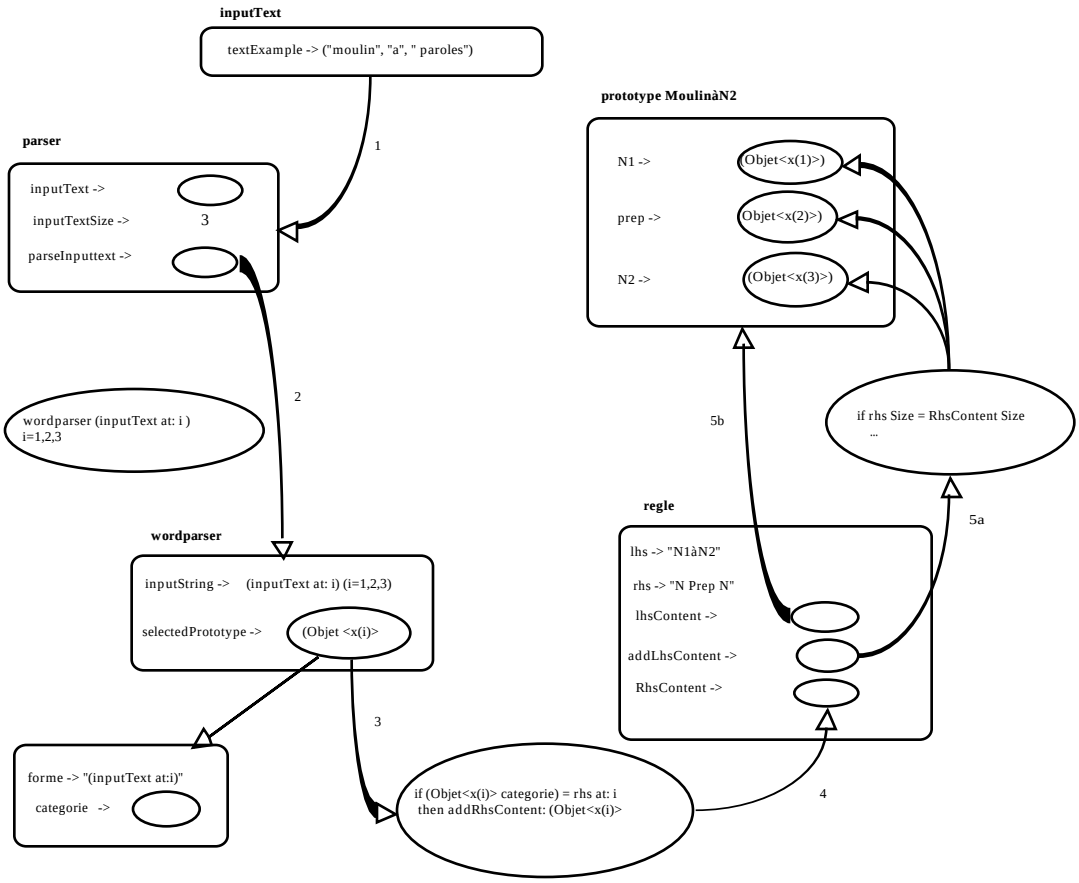


Figure 5.1 : Analyse prototypique Phase0.

L'utilisation de la version Self 3.0 a permis d'utiliser un environnement de programmation avec une interface graphique adéquate pour la mise au point et pour la visualisation des résultats construits par l'analyse. On présente infra (-> annexe partie 2 chapitres 6, 7, 8, 9) une version graphique des résultats obtenus par la suite en utilisant donc une version du code précédent modifié pour tenir compte de cette nouvelle version.

5.2.3.2. Phase1 : Affinement du filtrage

La phase d'expérimentation précédente s'est développée afin d'établir un parallèle avec le travail de (Habert & Fabre 1993). On a retenu le cadre sémantique défini pour la classification des dénominations complexes de la forme `N1 à N2` à savoir la partition des ces formes suivant les quatre classes suivantes :

moulin POUR/sortie moulin POUR/entrée moulins AVEC/énergie moulin AVEC/partie-de

Quatre prototypes sémantiques correspondant chacun à une des classes données ci-dessus sont définis. Ces prototypes comportent un attribut contenant la valeur de la classe à laquelle ils appartiennent sous la forme d'une chaîne de texte. De même, tous les éléments lexicaux de type `Nom` qui peuvent entrer dans le cadre syntaxique `N1 à N2` (en position `N2`) comportent un attribut `semantique` qui recevra comme valeur un des prototypes `semantique` ainsi définis.

La figure suivante reprend la présentation des objets lexicaux faite ci-dessus :

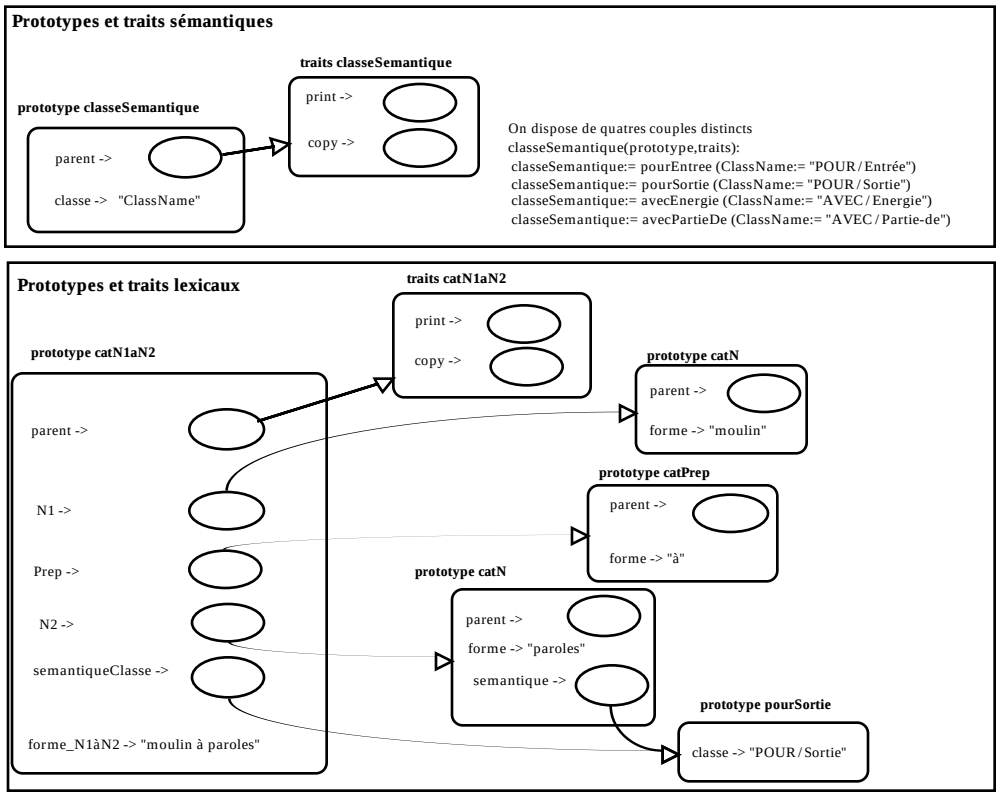


Figure 5.2 : Prototypes et traits pour analyse prototypique Phase1.

Analyse et reconnaissance/construction des prototypes `N1 à N2`

Le mécanisme général de parsage décrit supra est maintenu, seule la règle construite au cours de l'analyse est modifiée pour prendre en compte l'approche sémantique choisie. L'analyse se déroule donc ainsi :

- 1. On lit la forme complexe `N1 à N2` donnée, par exemple `moulin à paroles.`

- 2. Le parseur lit les entrées lexicales une à une, cherche les prototypes lexicaux associés et enregistre les résultats temporaires dans la règle à construire.

- 3. La fin de lecture correspond à la phase de création du prototype `N1 à N2` dont les attributs pointeront sur les différents prototypes reconnus au cours de l'analyse. La classe sémantique du `N1 à N2` construit sera déterminée par la valeur sémantique contenue dans le prototype lexical `N2`, dans notre exemple le prototype `paroles` associé à la chaîne "paroles" possède un attribut `semantique` pointant sur un prototype `semantique` du type `pourSortie` dont l'attribut `classe` donne la valeur de la classe sémantique associée au prototype `N1 à N2` construit à savoir la classe `POUR/Sortie`. De même, l'attribut `forme` du prototype résultant sera construit par la lecture des différents attributs `forme` puis par leur concaténation.

La figure 5.3 illustre toutes ces opérations :

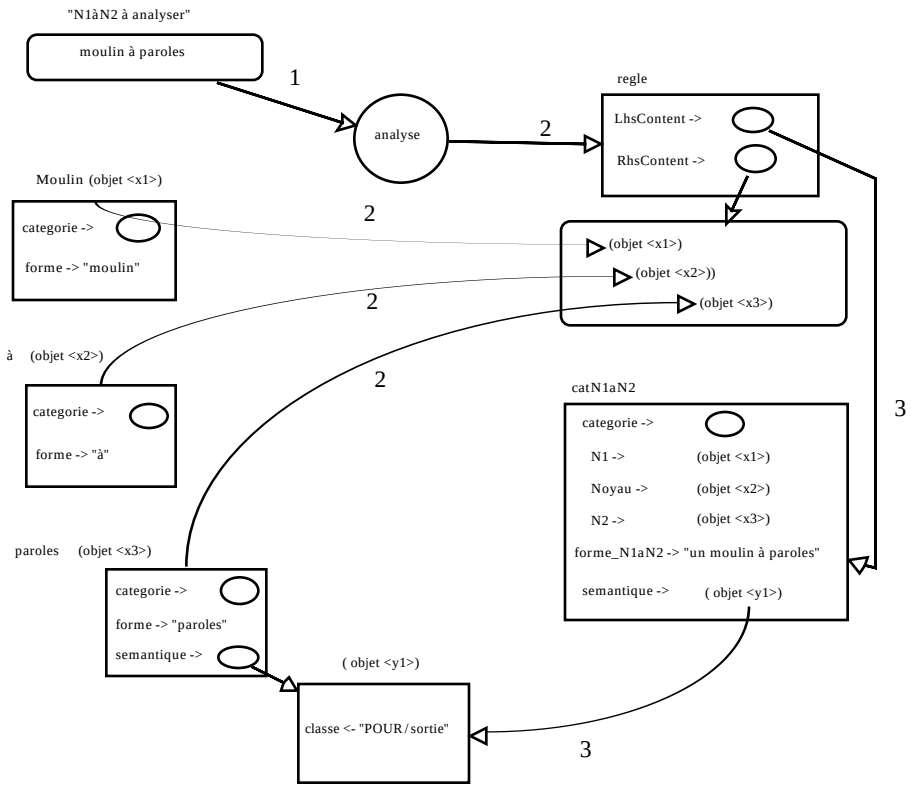


Figure 5.3 : Analyse prototypique Phase1.

L'attribut `semantique` associé aux éléments lexicaux manipulés ici ne concerne que la part de sémantique dont ils sont porteurs dans le cadre défini plus haut : la construction de formes `N1 à N2` suivant les quatre classes sémantiques retenues. Le prototype `N2` est porteur de la valeur sémantique attribuée au prototype `N1 à N2` résultant de l'analyse, et non pas d'une valeur sémantique caractérisant l'entrée lexicale attachée au prototype `N2`. Il est évidemment possible et nécessaire d'enrichir la sémantique de ces éléments par des spécifications supplémentaires : cette phase de travail sera faite infra. Le fait de partager la valeur de l'attribut `semantique` entre l'élément prototypique en position `N2` et le prototype construit `N1 à N2` n'est pas la solution définitive qui sera adoptée : la sémantique du `N1 à N2` sera déterminée par la valeur sémantique attachée au prototype `N2 initial` (non pas en termes de résultat pré-construit comme ci-dessus, mais en fonction d'un calcul fondé sur une classification sémantique donnée). D'une manière générale, il faut donner aux objets manipulés dans une analyse automatique la possibilité de leur propre auto-modification en fonction d'évolutions dynamiques de leurs états successifs. L'ajout de spécification sémantique au prototype en position `N2` doit entraîner une spécification du prototype sémantique associé au prototype `N1 à N2`. Dans notre exemple, toute modification des objets `x1, x2, x3` et `y1` et donc de tous les prototypes attachés à ces derniers sera implicitement connue du prototype `N1 à N2` construit. Si ces modifications affectent les valeurs des attributs de celui-ci, il faut mettre en place des processus de réactualisation de ces attributs pour que ces spécialisations deviennent explicites au sein du prototype `N1 à N2`.

5.2.3.3. Phase2 : Vers une représentation prototypique des connaissances

La phase précédente d'analyse consiste simplement à construire le sens d'une séquence `N1 à N2` par appariement de la classe sémantique attachée à ce prototype : le `N2` est porteur de la classe sémantique du prototype `N1 à N2` dont il est un des composants. Pour mener à bien une tentative de construction du sens des dénominations complexes `N1 à N2` (où `N1` est fixé), nous avons essayé d'adapter les principes de classification syntaxiques et sémantiques retenus par (Habert & Fabre 1993). Le "calcul" de la sémantique attachée au prototype `N1 à N2` construit est déterminé par la valeur sémantique contenue dans le

prototype N2 (cette valeur sémantique caractérisant des facettes sémantiques de N2). Le principe de classification sémantique présenté dans (Habert & Fabre 1993) a été repris : il utilise des classes sémantiques telles que aliment, noté ali, synthétique (pour des référents synthétiques, non naturels), noté synth, permettant de résumer un ensemble de traits élémentaires tels que concret, comptable :

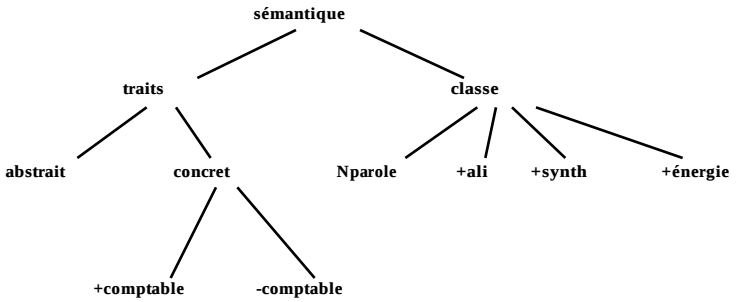


Figure 5.4 : Classification sémantique.

(Habert & Fabre 1993) associe à moulin un prédicat à quatre arguments :

PRODUIRE(Entrée,Sortie,Energie,Partie-De)

Le modèle sémantique proposé (fondé sur un modèle de représentation à base d'héritage) a permis d'affiner les résultats existants sur la classification des N1 à N2 (Poncet 1991) en intégrant notamment la classe jusqu'alors résiduelle des séquences du type moulin à paroles, celle-ci s'intégrant au modèle proposé suivant les contraintes sémantiques précisées dans la figure 5.5.

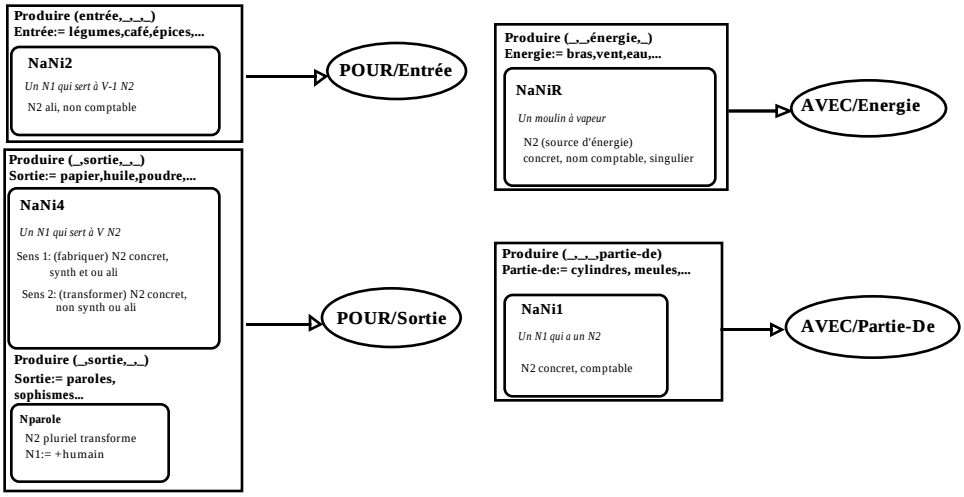


Figure 5.5 : Modèle sémantique pour les séquences N1 à N2.

L'extension de notre analyseur prototypique adapte ce mécanisme de construction du sens des formes moulin à N2.

- Le prototype moulin à N2 construit résulte de l'association syntaxique des prototypes attachés aux trois chaînes lexicales données suivant le mécanisme décrit supra.
- La sémantique de ce prototype résulte de l'identication de la définition sémantique du prototype N2 retenu avec la valeur sémantique attachée aux classes sémantiques définies ci-dessus.

Le calcul de la sémantique du prototype N1 à N2 résultant s'effectue suivant les valeurs sémantiques du N2 initial :

Valeurs sémantiques N2

concret	abstrait	Nparole	ali	énergie	synth	
+nc	-	-	+	-	-	pourEntree
+	-	+	-	-	-	pourSortie (Ajout du champ Nhumain := '+')
+	-	-	+	-	+	pourSortie
+	-	-	-	-	+	pourSortie
+	-	-	-	-	-	pourSortie
+nc	-	-	-	+	-	avecEnergie
+c	-	-	-	-	-	avecPartieDe

Valeur sémantique pour N1àN2 résultar

Figure 5.6 : Sémantiques N2 initial et N1 à N2 résultant.

Les prototypes de type `CatN` ont donc été redéfinis de telle sorte que l'attribut `semantique` dont ils sont porteurs, soit valué sur un prototype `semantique` qui prend en compte le principe de classification choisi. On donne ci-dessous une illustration de la définition sémantique sur des éléments du lexique :

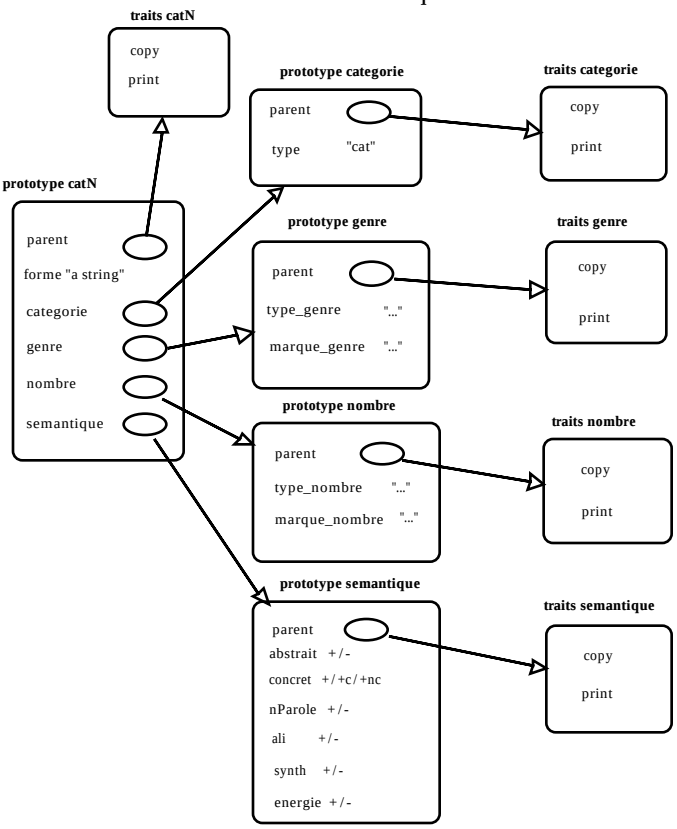


Figure 5.7 : Traits et prototypes pour analyse prototypique Phase2.

L'analyse et la construction du sens se déroule de la manière suivante :

- 1. Lecture des entrées lexicales.
- 2. Recherche des prototypes lexicaux associés et mise à jour de la règle.
- 3. Création du prototype `moulin à N2` :

3a. Calcul de la sémantique : début.
3b. Lecture de la sémantique de `<x3> (N2)` et des sémantiques définies dans le modèle.
3c. Après identification, la valeur sémantique retenue est enregistrée.

- 4. Test `NParole` : si `(<x3> semantique nParole) = '+'`, ajout de `Nhumain <- '+'` au prototype sémantique attaché au prototype `moulin à N2` résultant.

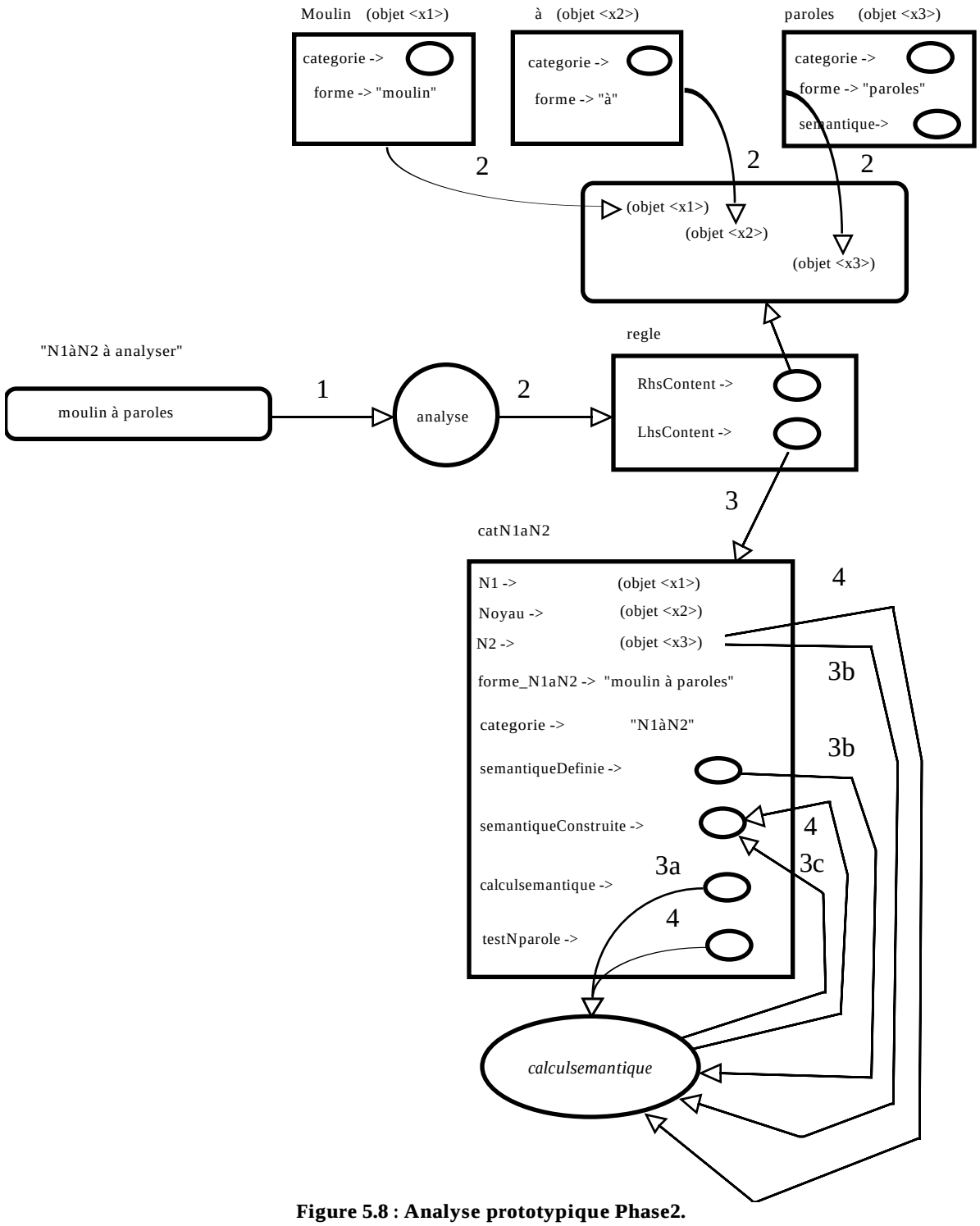


Figure 5.8 : Analyse prototypique Phase2.

5.2.3.4. Phase 0-2 : Premiers résultats acquis, perspectives

Les prototypes construits peuvent être ajustés automatiquement (i.e. recatégorisés)

Ces premières phases de travail donnent en effet une illustration simple de l'ajustement dynamique qu'il est possible de mettre en place sur les structures de représentations construites dans le cadre de la construction du sens des composés `moulin à N2`. Dans la figure 5.8, la sémantique du prototype `moulin à N2` construit est modifiée dynamiquement par l'ajout d'un attribut `Nhumain<-'+'`. C'est ce genre de mécanisme de construction du sens qu'il faut mettre en place, si l'on veut affiner progressivement les connaissances sémantiques des objets construits. Il s'agit désormais de progresser dans la représentation des éléments définis pour une analyse automatique expérimentale avec des prototypes : du point de vue du code développé mais aussi du point de vue de la représentation des connaissances proposée. On peut ainsi envisager de définir, pour les prototypes `moulin à N2`, des comportements généraux de mise à jour du sens construit qui tiennent compte de la sémantique des composants initiaux (au niveau des `traits catN1àN2`). En effet, la mise à jour proposée (via les attributs `calculsemantique` et `testNparole`) est définie en termes de méthodes construites au niveau du prototype `catN1àN2`. S'il possible de "stabiliser" / généraliser ce type de comportement, il serait souhaitable pour le développement d'un code cohérent (pour la lisibilité selon Self, et pour d'éventuelles mises à jours "économiques") de définir ces comportements au niveau des `traits` associés. On peut ainsi considérer que le calcul d'identification de la sémantique du `N2` avec une des classes sémantiques définies est un mécanisme général de construction du sens des séquences `moulin à N2`. De ce fait, ce comportement commun devrait être redéfini au niveau de l'objet `traits catN1àN2`. Une construction progressive de stabilisation des résultats doit entraîner une classification[68] nouvelle des connaissances sur les objets prédéfinis. Cette réorganisation des représentations initiales introduit une nouvelle fois dans ce travail la nécessité de définir un "méta-regard" sur les constructions en cours pour évaluer les résultats et prendre en compte les régularités ou généralisations qui se dessinent afin de modifier les représentations initiales. Nous y reviendrons infra.

Elargir la couverture sémantique du modèle

Il s'agit aussi de mettre à l'épreuve les mécanismes de calcul et de représentation choisis sur le corpus des formes `moulin à N2` pour évaluer les évolutions possibles. On sait en effet que certains exemplaires de `moulin à N2` ne rentrent pas dans le cadre du prédicat à quatre places retenu : `scie, foulon, ventilateur`. Il s'agit pour ces derniers, non pas de ce que produit le moulin, mais de l'instrument constitué par le moulin : l'énergie du moulin sert à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation : `instrument dont l'énergie est apportée par le moulin et non Partie-De`.

De même, l'étude des emplois métaphoriques peut élargir la limitation faite ici au `NParole`. On peut ainsi avoir des `moulin à N2` métaphoriques avec (`N2 pluriel` : "coups, thèmes, images, sophismes" ou avec (`N2 massif singulier` : "ennui"), dans les deux cas, on a une production massive. La présentation faite ici doit donc prendre en compte ces exemplaires de `moulin à N2` qui apportent des précisions au modèle initial.

---

[68] Il en va d'ailleurs de même dans toute activité d'intelligence humaine : même si les individus manipulent plutôt des entités individualisées dans des conditions empiriques données, le recours à une classification est nécessaire à terme pour structurer les informations acquises par apprentissage ou par expérience.

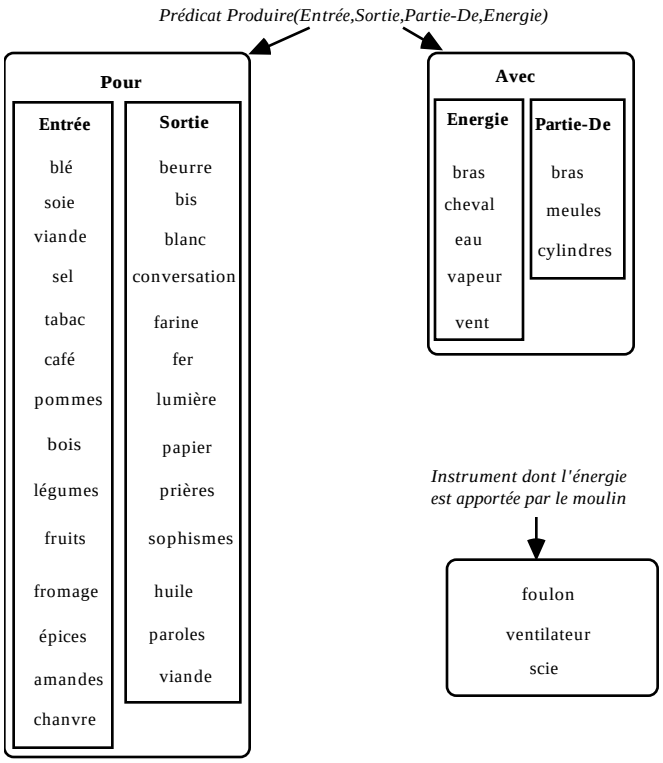


Figure 5.9 : Répartition lexicale suivant le modèle sémantique retenu.

5.2.4. Analyse automatique et construction des prototypes `N1 à N2`

5.2.4.1. Préliminaires

Cette phase d'extension de l'analyse automatique présentée supra accentue le rôle des prototypes dans la conduite et le déroulement de l'analyse. Les prototypes mis en oeuvre ne sont plus uniquement des entités passives, en particulier les prototypes résultants de l'analyse seront porteurs de savoirs lexicaux ou syntaxiques de plus en plus précis afin de leur permettre d'infléchir le déroulement de l'analyse. Un prototype `manager` a pour fonction de superviser les activités prototypiques mises en oeuvre par l'analyse, de mémoriser et de stocker les résultats construits. Cette solution permet d'accroître notre **contrôle** sur les entités manipulées et construites lors de l'analyse. Le `manager` constitue en effet un point d'entrée central pour interroger les prototypes qu'il gère au cours de l'analyse. Comme précédemment, le manager utilise des **méta-processus** pour interroger l'ensemble des prototypes définis. Il est de plus chargé de "garder la main" sur les prototypes qui répondent positivement à ses requêtes.

5.2.4.2. Phase3.0 : Filtrage pour la construction du prototype `moulin à N2`

Cette première étape restreint le modèle syntaxique initial à la règle hors contexte suivante :

```
moulinàN2 -> moulin Prep N2 (où Prep = à).
```

Le prototype `moulin à N2` construit connaît déjà deux de ses constituants lexicaux (`N1` et `Prep`) ainsi qu'un comportement du calcul sémantique. L'analyse qui suit opère un filtrage sur le `N2`. Les propriétés sémantiques du `N2` détermineront la classe sémantique du prototype `moulin à N2` résultant suivant le modèle présenté supra qui associe à `moulin` le prédicat `PRODUIRE(Pour/Avec)`.

L'analyse se déroule de la manière suivante :

Le prototype `manager` lit une entrée lexicale et déclenche le processus de filtrage de cette entrée lexicale dans le but de construire un prototype `moulin à N2` qui entre dans la classification prédéfinie.

- 1. Création d'un prototype `moulin à N2` (l'objet à construire) par clonage du prototype `moulin à N2` prédéfini. La forme lexicale lue est affectée à l'attribut `forme` du prototype en position `N2` dans le prototype `moulin à N2` en cours de construction.

- 2.(a) Le comportement qui effectue le calcul sémantique du prototype résultant opère une identification de la sémantique du prototype en position `N2` avec le modèle sémantique défini par les prototypes `moulin à N2`, i.e `PRODUIRE(Avec,Pour)` (b).

Le résultat produit un prototype `moulin à N2` qui soit vérifie le prédicat proposé (sa sémantique est alors définie par l'une des quatre classes sémantiques) soit ne vérifie pas le modèle proposé (i.e. les traits sémantiques associés au `N2` ne sont pas reconnus par les filtres mis en place), dans ce cas sa sémantique reste indéfinie (-> figure 5.10).

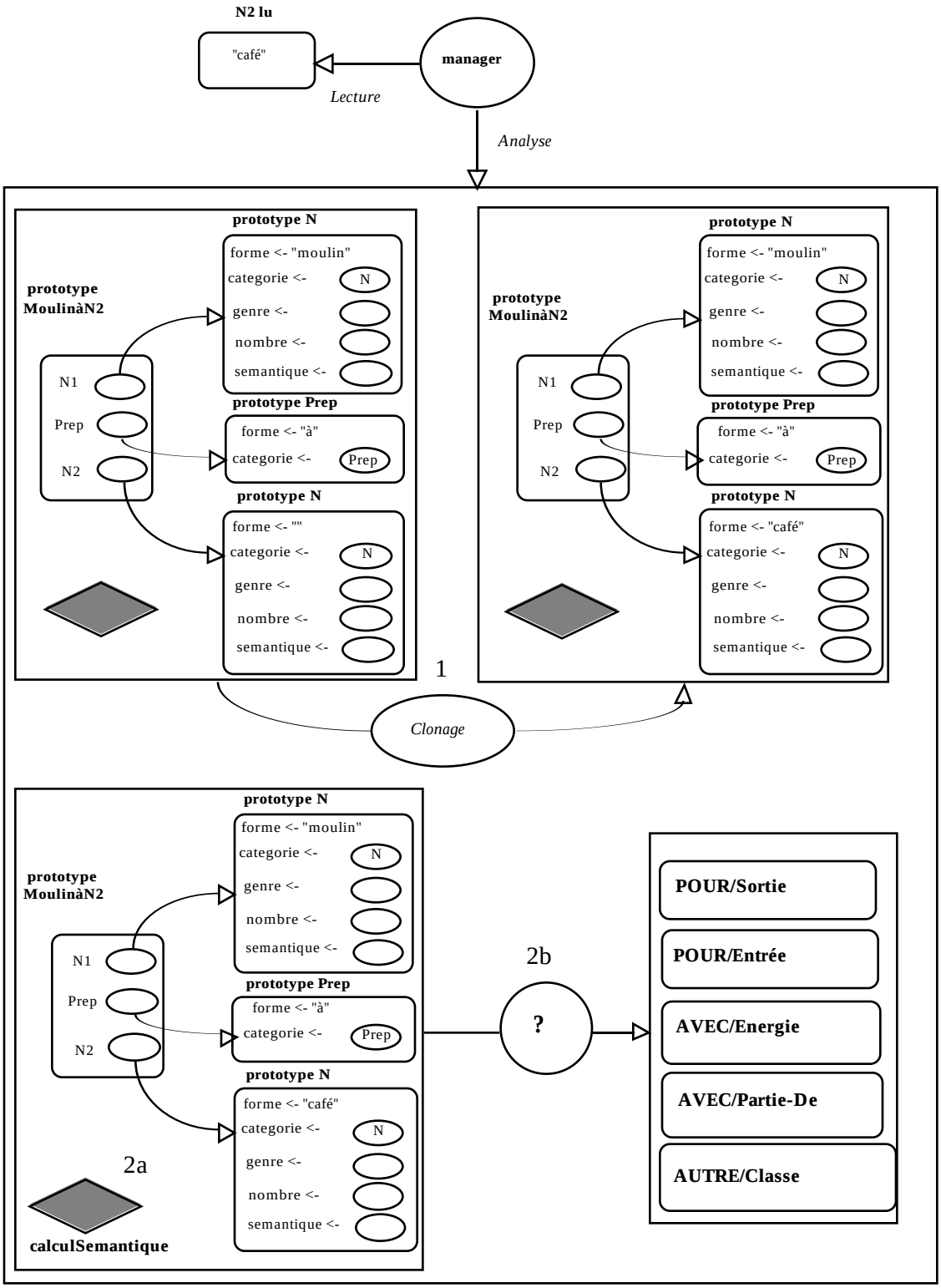


Figure 5.10 : Analyse prototypique Phase3.0.

5.2.4.3. Phase3.1 : Filtrage pour la construction du prototype N1 à N2

Cette deuxième étape reprend le modèle syntaxique initial : N1àN2 -> N1 Prep N2, et ne restreint plus la forme lexicale du N1. Il s'agit de construire des prototypes N1 à N2 en tenant compte des résultats obtenus supra. Si la forme lexicale lue est "moulin", la phase précédente nous permet d'envisager la construction éventuelle d'un prototype moulin à N2 suivant les contraintes définies. Si ces contraintes ne sont pas vérifiées, l'analyse construit un prototype N1 à N2 pour lequel aucune construction de sens n'est envisagée pour le moment (Si la forme lexicale lue est de type "boite", par exemple). Le prototype N1 à N2 initialement construit est donc porteur de différents niveaux de savoirs. Le savoir lexical qui porte sur la forme lexicale du N1 permet soit de déclencher une analyse sémantique du type de la phase 3.0, soit de rester dans un cadre plus général d'analyse pour lequel aucun traitement sémantique précis n'est encore défini.

L'analyse se déroule de la manière suivante :

Le prototype manager lit les entrées lexicales et déclenche le processus d'analyse. Le premier mot lu est porteur d'un certain type de savoir syntaxique : il peut entrer dans la construction de prototype N1 à N2 en position N1. Le manager créé donc un prototype N1 à N2 (l'objet <x1>) par clonage du prototype N1 à N2 prédéfini. Le prototype N1 à N2 est quant à lui porteur d'un savoir lexical lui indiquant que pour certains prototypes N en position N1, leur forme lexicale peut spécialiser l'analyse vers un autre type de prototype à construire. C'est le cas des prototypes N en position N1 dans un prototype N1 à N2 dont l'attribut forme est égal à "moulin".

A. Le prototype N en position N1 permet de construire un prototype moulin à N2 :

- 1. Dans ce cas, le prototype N1 à N2 active la création d'un prototype moulin à N2, (l'objet <x2>) par clonage du prototype moulin à N2 prédéfini, et l'analyse se poursuit comme dans la phase3.0 pour la construction de ce prototype. Le prototype manager distribue, par filtrage catégoriel, les entrées lexicales restantes aux prototypes en position Prep et N2 dans le prototype moulin à N2 en cours de construction ainsi que dans le prototype N1 à N2 initial. On garde "sous la main" ce prototype au cas où la construction sémantique du prototype moulin à N2 ne serait pas définie.

La construction du prototype moulin à N2 se déroule en suivant les étapes suivantes :

- Ajout d'un prototype N(forme := 'moulin') en position N1.
- Ajout d'un prototype Prep en position Prep.
- Ajout d'un prototype N en position N2.
- Si les étapes précédentes se sont déroulées, le calcul sémantique est effectué, et le manager "considère" que la construction du prototype moulin à N2 est effective, i.e le prototype N1 à N2 initial n'est pas conservé par le manager.

De même, la construction du prototype N1 à N2 se déroule en suivant les étapes suivantes :

- Ajout d'un prototype N(forme := 'moulin') en position N1.
- Ajout d'un prototype Prep en position Prep.
- Ajout d'un prototype N en position N2.

- Si le prototype `moulin à N2` créé par le prototype `N1 à N2` n'est pas effectivement construit (i.e sémantique non définie), le `manager` considère que la construction du prototype `N1 à N2` est effective.

La validation de construction du prototype `N1 à N2` effectue en effet au préalable la vérification de construction du prototype `moulin à N2` qu'il a déclenchée.

- 2. Le comportement sémantique, spécifique aux prototypes `moulin à N2`, effectue ensuite le calcul sémantique du prototype résultant. Il opère donc une identification de la sémantique du prototype associé à l'attribut `N2` avec le modèle sémantique défini par les prototypes `moulin à N2`.

- 3. Le résultat produit est soit un prototype `moulin à N2` qui vérifie le prédicat proposé (sa sémantique est l'une des quatre classes sémantiques) soit un prototype `N1 à N2` dont la sémantique n'est pas encore identifiée.

La figure 5.11 donne une présentation détaillée du déroulement de l'analyse ainsi décrite.

B. Le prototype `N` en position `N1` ne permet pas de construire un prototype `moulin à N2` :

Dans ce cas, (1) l'analyse effectue, par filtrage catégoriel, un simple appariement des prototypes associés aux entrées lexicales restantes vers les attributs `Prep` et `N2` du prototype `N1 à N2` construit, (2) la sémantique de ce dernier reste non identifiée (-> figure 5.12).

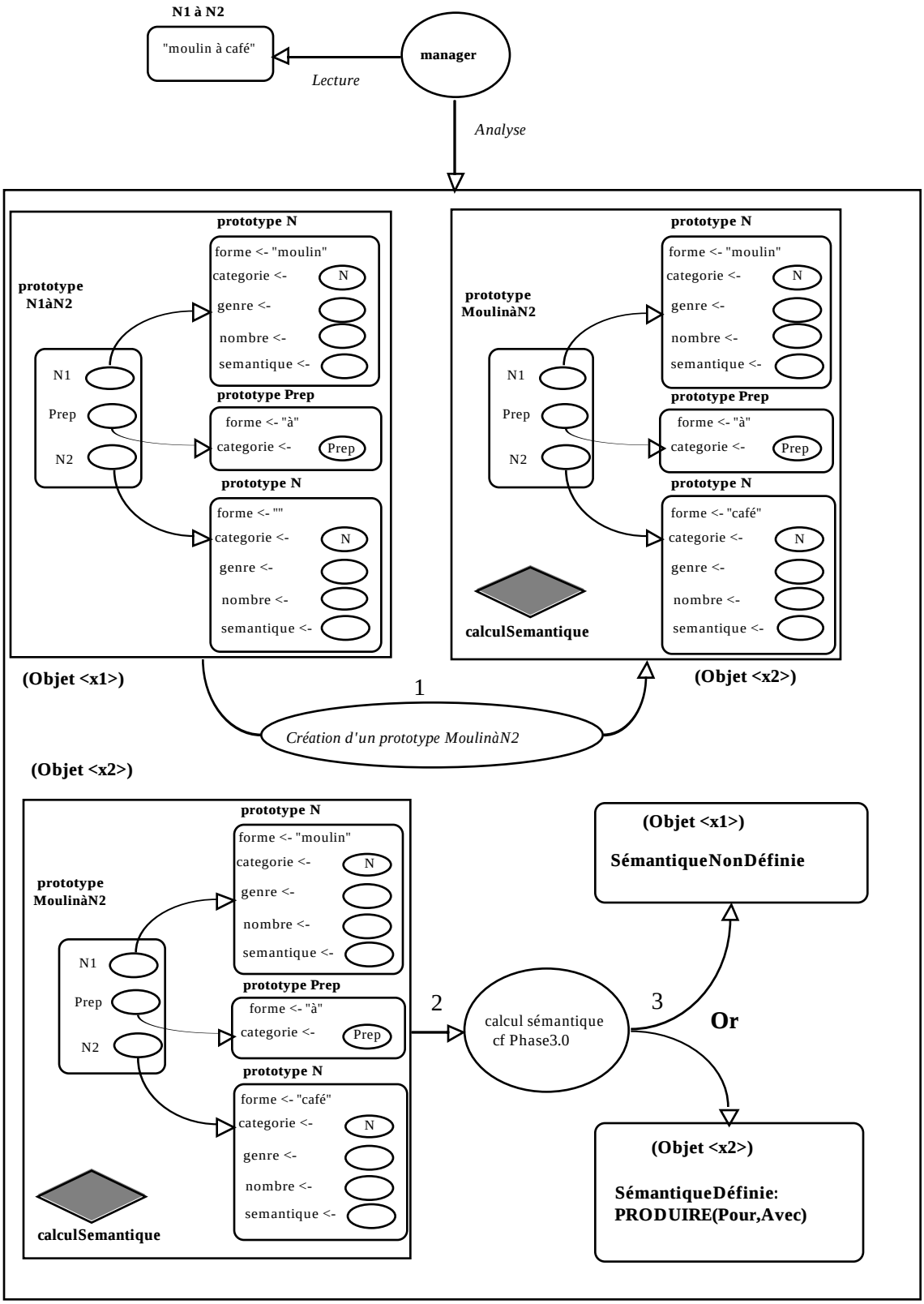


Figure 5.11 : Analyse prototypique Phase3.1(A).

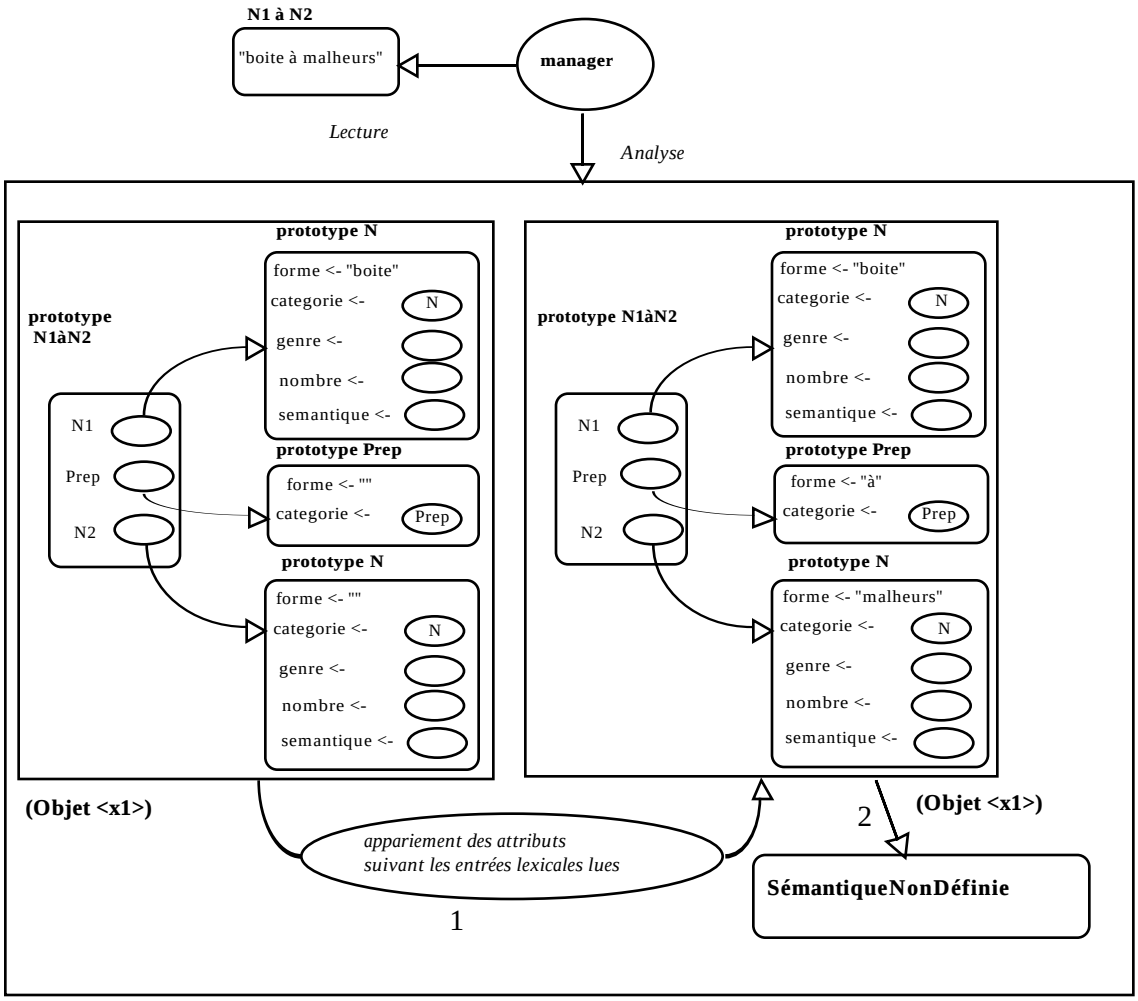


Figure 5.12 : Analyse prototypique Phase3.1(B).

La figure 5.13 détaille les activités du `manager` lors de l'analyse. La numérotation qui suit suit fait référence aux indications données dans cette figure :

- 1 Le `manager` lit la phase à analyser et pour chaque entrée lexicale.
- 2 Il déclencle une recherche de tous les prototypes lexicaux ayant un attribut `forme` de même valeur que l'entrée lue.
- 3 Cette recherche permet de sélectionner un certain nombre de prototypes.
- 4(0) Les prototypes lexicaux dont l'attribut `forme` est 'moulin' et en particulier un prototype `N` (`categorie type := 'nom'`),
- 4(1) Les prototypes lexicaux dont l'attribut `forme` est 'a' et en particulier un prototype `Prep` (`categorie type := 'Prep'`),
- 4(2) Les prototypes lexicaux dont l'attribut `forme` est 'paroles' et en particulier un prototype `N`, (`categorie type := 'nom'`).

- 5(0) Pour chaque prototype, on fait une recherche des constructions éventuelles à déclencher à partir de ces prototypes. Dans notre cas, seule la construction des prototypes `N1 à N2` est envisagée. Certains prototypes possèdent donc un attribut indiquant le type de prototype "complexe" qu'il est possible de construire partir de ce prototype lexical initial . Dans notre exemple, le prototype `N` associé à 'moulin' déclenche la création d'un prototype `N1 à N2` (par clonage).

- 5(1) De plus les prototypes `N1 à N2` permettent la création d'un prototype spécialisé si le prototype en position `N1` a un attribut `forme` de valeur égale à 'moulin' : dans ce cas le prototype `N1 à N2` génère un prototype `moulin à N2` et le manager poursuivra l'analyse en construisant en parallèle ces deux prototypes.

- 6 Les prototypes en cours de construction sont stockés, les opérations 2, 3, 4, 5 sont répétées.

- 7 Pour chacune des entrées lexicales, si le filtrage catégoriel est vérifié, les prototypes en cours de construction sont mis à jour (on leur adjoint les prototypes lexicaux correspondants).

- 8 En fin de lecture, le `manager` ne contient que les prototypes qui ont vérifié le filtrage catégoriel (dans notre cas `Nom Prep Nom`), i.e soit un prototype `N1 à N2`, soit un couple de prototypes (`N1 à N2`, `moulin à N2`).

- 9 Dans ce dernier cas, si le prototype `N1 à N2` a permis la construction d'un prototype `moulin à N2` (c'est le cas dans l'exemple donné), le prototype `N1 à N2` vérifie si le prototype `moulin à N2` qu'il a créé est effectivement construit.

- 10 Si le calcul sémantique qu'il permet produit un résultat cohérent avec le modèle sémantique choisi, si tel est le cas le prototype `N1 à N2` "considère" qu'il n'est pas construit puisqu'un prototype plus précis peut lui être associé.

- 11(0) C'est le prototype `moulin à N2` qui est retourné au `manager`.

- 11(1) Ou le prototype `N1 à N2` qui est effectivement construit et retourné au `manager`.

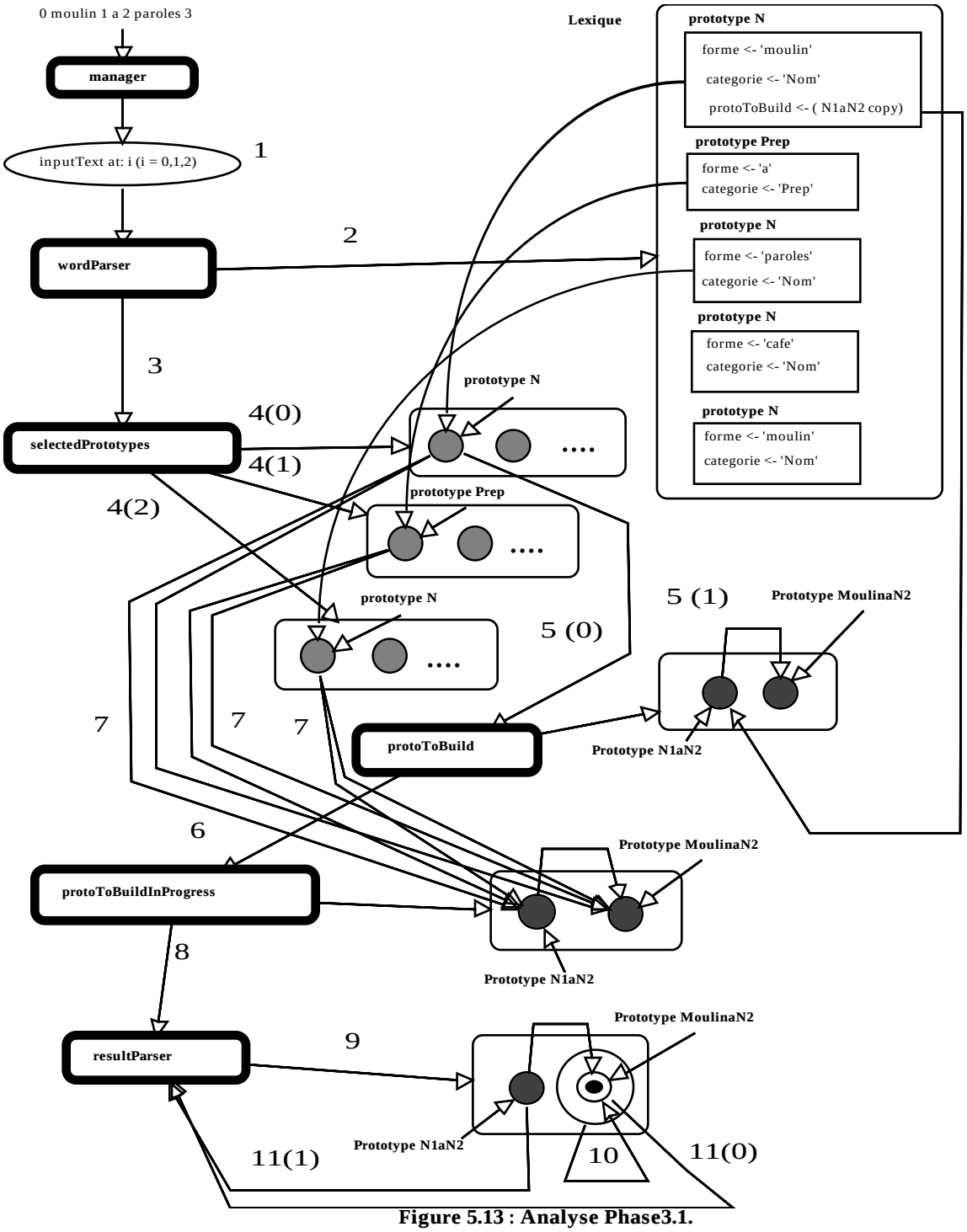


Figure 5.13 : Analyse Phase3.1.

5.2.4.4. Phase3.2 : Filtrage pour une construction spécialisée des prototypes N1 à N2

Cette troisième phase doit enrichir le modèle de construction du sens présenté dans la phrase précédente. Si les restrictions faites pour la construction du sens des prototypes N1 à N2 ne permettent pas de prendre en compte un grand nombre de ces formes, il faut pouvoir enrichir les règles existantes. Les deux phases précédentes permettent d'effectuer un première approche de construction spécialisée des séquences N1 à N2 : pour de telles formes où N1 = moulin et pour lesquelles le calcul sémantique reconnaît une forme sémantique qui entre dans la classification prédéfinie, l'analyse construit une forme prototypique de N1 à N2 plus précise que si les contraintes précitées n'étaient pas vérifiées. Cette première approche demande a être précisée dans la mesure où, évidemment, un grand nombre de séquences N1 à N2 ne sont pas prises en compte par l'analyse ainsi présentée. C'est justement grâce à cette approche prototypique de représentation des éléments de l'analyse qu'il nous semble possible de poursuivre cette étude. Puisque la classification de telles formes n'est pas encore figée, les connaissances "individuelles" sur de telles formes doivent nous permettre de progresser en affinant les résultats construits. Il faut pour cela prolonger les règles existantes et ajouter à celles-ci des éléments qui permettent de préciser plus avant les filtrages opérés. On a déjà vu par exemple que certaines séquences de moulin à N2 ne vérifient pas le prédicat Produire(Avec/Pour), à l'image de"scie, foulon, ventilateur". Il s'agit pour ces derniers, non pas de ce que produit le moulin, mais de l'instrument constitué par le moulin : l'énergie du moulin sert à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation. Parmi les prototypes N1 à N2 de sémantique non définie construits à l'issue de la Phase3.1(A), on peut donc opérér un filtrage suppléméntaire pour prendre en compte ce type de construction sémantique dans le cas où le N2 initial est un déverbal.

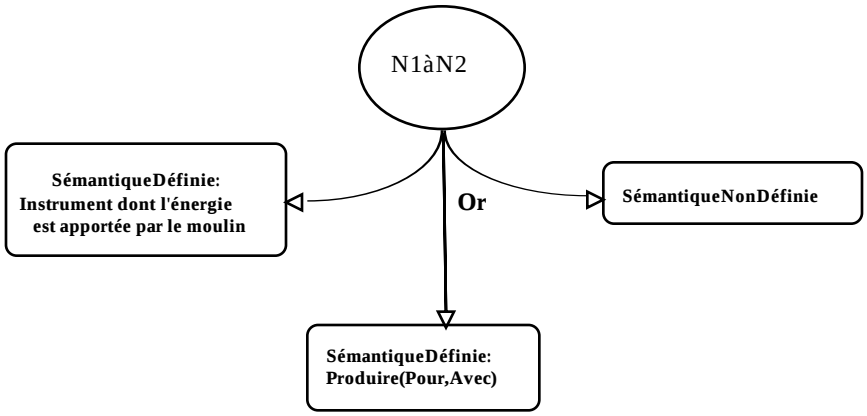


Figure 5.14 : Analyse prototypique et construction progressive du sens des séquences N1 à N2.

5.2.5. Une Analyse automatique prototypique étendue

5.2.5.1. Analyse par un réseau de prototypes

Cette étape vise à développer le mécanisme de l'analyse pour disposer d'un analyseur plus "puissant". Celui-ci n'est plus restreint à l'analyse des formes N1 à N2, il est capable d'analyser des phrases complexes. De plus un traçage des opérations effectuées par l'analyseur est disponible.

Mise en place d'un réseau de prototypes

L'analyseur ne dispose pas de grammaire initiale. Ce sont les prototypes définis pour l'analyse qui portent le savoir syntaxique.

Déroulement de l'analyse

Comme précédemment, un prototype nommé `manager` est chargé de conduire l'analyse puis de fournir les résultats obtenus.

- 1. Le `manager` lit la phrase à analyser : une liste de mots.

- 2. Pour chaque entrée lue, on recherche tous les prototypes lexicaux dont le champ `forme` a une valeur identique à cette chaîne. Le `manager` enregistre ces prototypes et leur position dans la phrase.

- 3. Pour chacun des prototypes initiaux enregistrés, on vérifie si leur champ `protoToBuild` contient un prototype ou plusieurs qu'il est possible de construire à partir de ce prototype initial. Dans les figures qui suivent, un prototype lexical de type `catDet` permet la création d'un prototype `catSn`, représentant un groupe nominal. Le prototype `catDet` sait donc implicitement qu'il peut permettre la création d'un prototype `catSn` regroupant un prototype `catDet` et d'un prototype `catN`, le prototype `CatSn` se construisant de "droite à gauche" : il attend tout d'abord un prototype de catégorie `Det` puis un prototype de catégorie `Nom` (`Sn -> Det N`). Le prototype `catSn` pouvant lui aussi être porteur de savoir sur le type de prototype dont il peut déclencher la création (en l'occurrence un prototype `catP` dans les exemples donnés représentant un phrase). Ces prototypes à construire enregistrent donc les prototypes dont ils sont issus et leur position dans la phrase. Le `manager` stockent ces prototypes à construire.

- 4. Le prototype `manager` possède à ce stade d'une liste de prototypes construits (les prototypes initiaux) et d'une liste de prototypes à construire (partiellement construite à l'aide des prototypes initiaux). Pour chacun des prototypes à construire, le `manager` vérifie s'il existe un prototype qui peut entrer dans la construction de celui-ci parmi les prototypes disponibles construits (au départ il ne dispose que des prototypes construits au point (2) puis des prototypes construits par l'analyse). La sélection d'un prototype à adjoindre au prototype à construire doit s'assurer d'une part de la cohérence des places respectives du prototype à construire et du prototype à adjoindre dans la phrase, et d'autre part de la concordance du type de catégorie attendue par le prototype à construire avec celle du prototype disponible sélectionné. Si ces deux contraintes sont vérifiées, le prototype sélectionné est ajouté au prototype à construire dans la position qui lui convient.

- 5. Le `manager` répète les opérations décrites au point (4) (sélection d'un prototype à construire et recherche d'un prototype à adjoindre) tant que l'analyse créé de nouveaux prototypes à construire ou tant qu'elle adjoint un prototype à un autre.

Une trace complète de ces mécanismes est donnée en annexe (-> annexe partie 2 chapitre 5). Et les figures 5.15, 5.16, 5.17 illustrent le réseau de prototypes que l'analyse met en place.

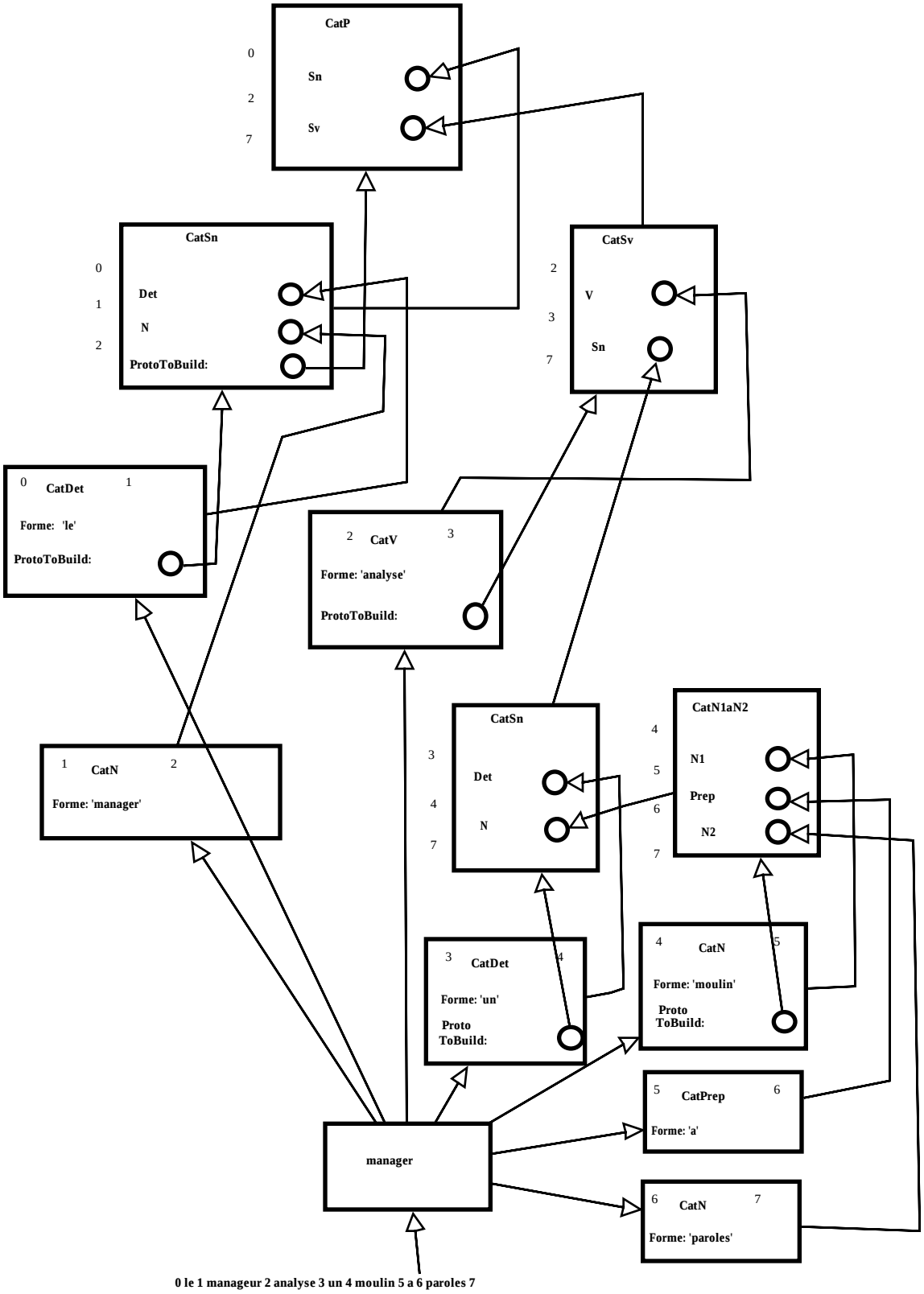


Figure 5.15 : Analyse par un réseau de prototypes.

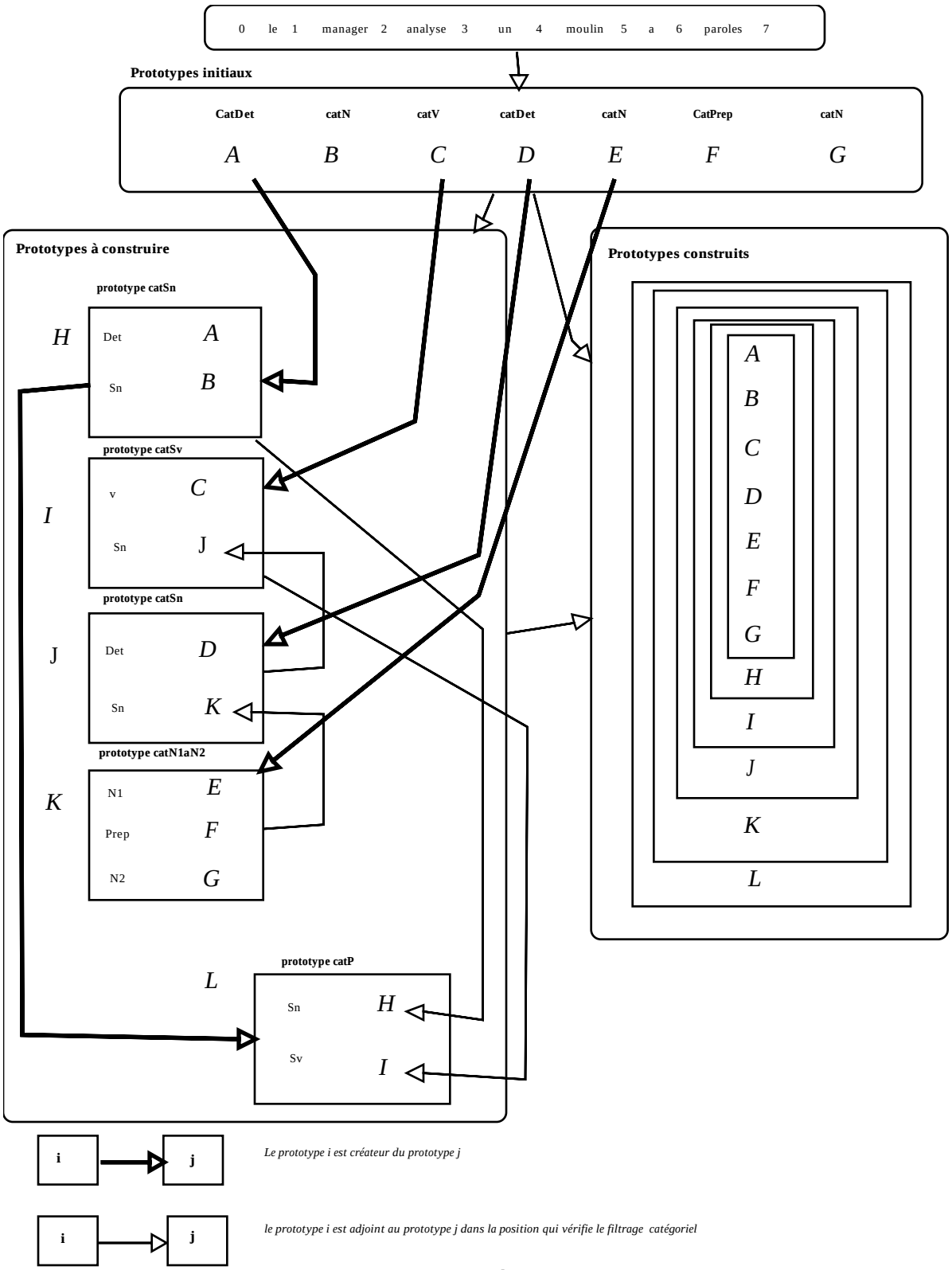


Figure 5.16 : Mise en place du réseau de prototypes pour l'analyse.

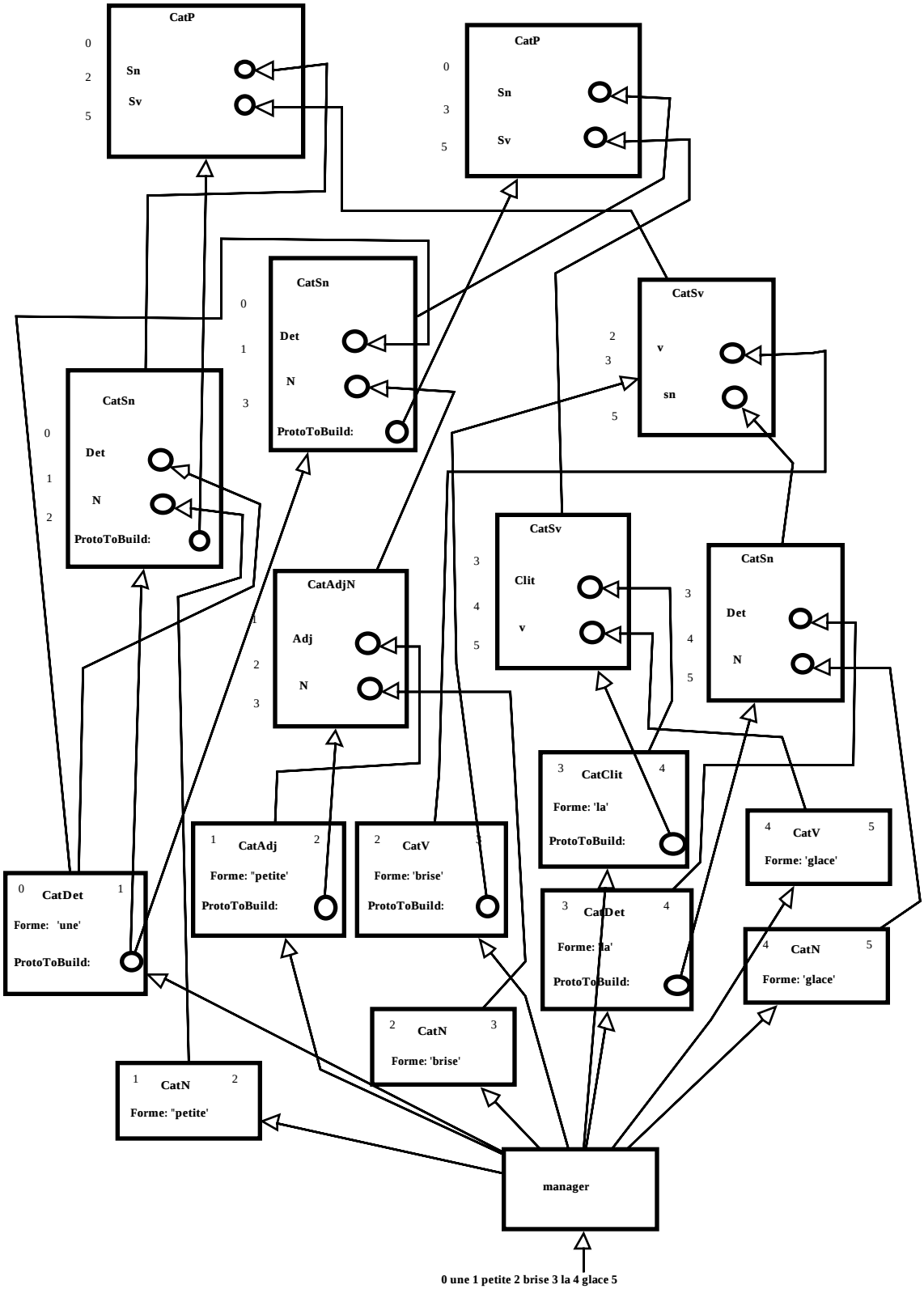


Figure 5.17 : Une petite brise la glace (1)

Cette figure est une illustration du résultat d'analyse obtenu via l'interface graphique de Self-3.0. Cette phase d'analyse nous a permis en effet d'intégrer aux processus d'analyse des mécanismes qui activent une **trace graphique des activités du `manager`**. L'utilisateur peut ainsi suivre dans cette fenêtre une "version animée" de l'analyse réalisée. Les prototypes construits apparaissent au rythme de leur construction et en fin d'analyse l'utilisateur peut ausculter les objets, les ajuster ou les remodeler à sa guise. Dans l'implémentation construite, on a choisi de ne présenter (de manière prédéfinie) via cette interface graphique qu'une partie des attributs associés aux objets construits, l'utilisateur peut évidemment étendre la présentation des objets présents dans la fenêtre graphique en suivant les protocoles[69] définis par Self pour l'utilisation de cette interface.

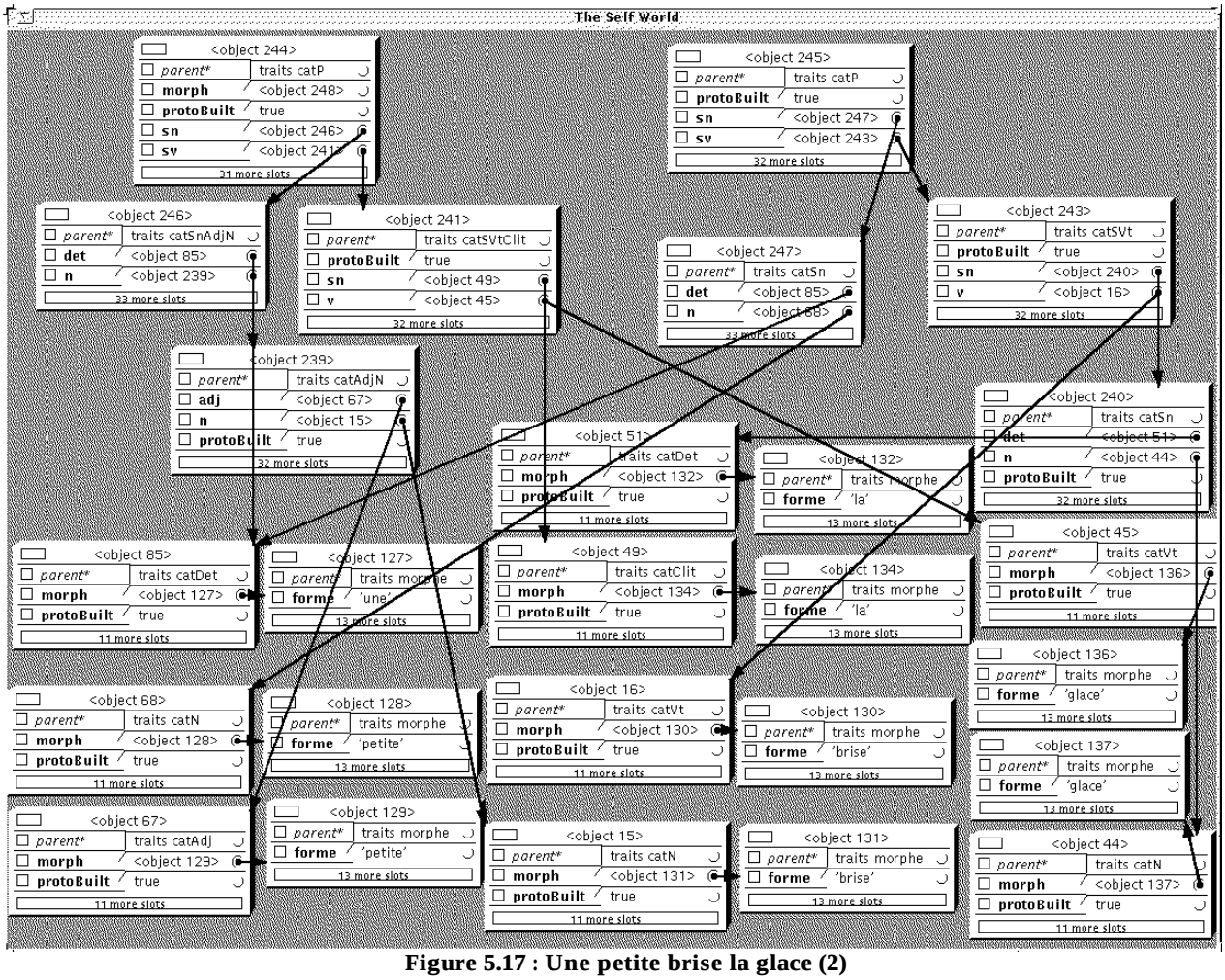


Figure 5.17 : Une petite brise la glace (2)

5.2.5.2. L'utilisateur peut contrôler la construction du sens des séquences en `moulin à N2`

Le travail présenté jusque là a consisté à développer des outils pour une analyse syntaxique prototypique dans le but de disposer d'outils fiables mais aussi "agréables" pour la présentation des résultats. Le détour fait supra vers une analyse syntaxique de phrases complexes a ainsi permis de mettre au point un prototype de parseur qui répond partiellement à ces contraintes (notamment pour une présentation lisible des résultats). Cette phase de travail a été poursuivie pour affiner les traces produites lors d'une analyse, en particulier pour évaluer les opérations effectuées lors d'une analyse. Le point d'ancrage linguistique de ce

69 Tous les objets sont en fait réactifs à la souris (-> tutorial Self).

travail, à savoir l'étude des formes complexes N1 à N2, va de nouveau être le centre d'intérêt principal dans les phases de travail présentées ci-dessous. L'étude de la construction de ces formes complexes par affinage progressif des connaissances représentées a nécessité la mise en place d'outils qui permettent de réaliser ce travail d'affinage des connaissances quand celles-ci s'avèrent insuffisantes pour construire une forme complexe dans le cadre sémantique déjà présenté. Lorsqu'une analyse d'une forme moulin à N2 ne construit pas un représentant d'une classe sémantique déjà connue, il paraît souhaitable de rendre la main à l'utilisateur pour que celui-ci affine les représentations initiales, si cela est possible afin de construire une forme dont le sens peut être reconnu par le calcul sémantique mis en place. Plus précisément, si une forme complexe du type N1 à N2 permet de construire un prototype moulin à N2 (<- spécialisation d'un prototype N1 à N2), i.e si N1='moulin', le calcul sémantique attaché au prototype moulin à N2 est effectuée. Si ce calcul produit un résultat cohérent avec le modèle Produire(Pour,Avec), l'analyse ira à son terme sans intervention de l'utilisateur. Par contre, si ce résultat n'est pas le représentant d'une classe sémantique définie l'analyseur propose à l'utilisateur d'agir sur la sémantique des prototypes attachés aux composants de la forme complexe en cours de construction. Cette intervention au cours de la construction du prototype résultant permet ou non d'infléchir l'analyse. Elle est aussi guidé par le manager qui propose à l'utilisateur un protocole pour réaliser une éventuelle mise à jour des prototypes qui interviennent dans l'analyse.

Dans un premier temps le calcul présenté précédemment pour le calcul de la sémantique d'un prototype moulin à N2 a été maintenu : celui-ci est déterminé uniquement par les valeurs sémantiques (exprimées sous forme de traits) associées au prototype lexical en position N2. Ce choix peut être remis en question. En effet il peut être souhaitable de donner un plus grand poids à chacun des composants de la forme résultante au lieu de privilégier le prototype en position N2. Ce point sera repris infra. On donne en annexe (-> annexe partie 2 chapitre 5) une illustration d'une analyse qui propose à l'utilisateur de modifier la sémantique du prototype lexical en position N2 dont la sémantique ne permet de reconnaitre un des représentants des classes sémantiques définies. Toute l'analyse est gérée via l'utilisation de prototypes qui proposent un menu de choix, aussi bien pour charger les données de l'analyse que pour présenter les éléments lexicaux ou encore pour permettre au cours de l'analyse d'intervenir si le résultat produit n'est pas sémantiquement satisfaisant. On introduit ici le phénomène crucial du contrôle dans des processus de représentation des connaissances. L'analyse permet ici de rendre la main à l'utilisateur pour que son intervention modifie les valeurs sémantiques initiales afin de construire un résultat sémantique cohérent.

5.3. Approche prototypique de la construction du sens des séquences moulin à N2 : résumés et perspectives

Pour le moment on opère un appariement de traits sur un nombre restreint de prototypes sémantiques de référence. Le risque est grand de devoir utiliser un très grand nombre de prototypes de référence si le nombre de traits sémantiques utilisés augmente. Les prototypes de référence pourraient être des points d'entrée très généraux qui se préciseraient petit à petit si l'inférence le nécessite. Dans le cas où l'inférence ne conduit pas à un résultat satisfaisant, on pourrait ainsi disposer d'un début de chemin "à travers" un prototype de référence (du point d'entrée vers le point critique où l'inférence échoue). Il serait ainsi possible de conserver en mémoire ce parcours puis de le modifier en ajoutant les informations nécessaires pour aller plus loin : définir un chemin (i.e une nouvelle classe). Les problèmes à résoudre sont :

- Comment pouvoir mémoriser un nouveau chemin (i.e une nouvelle classe) ?

- Si ce qui permet de reconnaitre une classe est un mécanisme d'appariement, comment figer et inclure un appariement sur un nombre croissant de valeurs sémantiques? La fonction Sem définie plus haut opère un appariement sur un nombre restreint et fixe de traits sémantiques. Il peut être nécessaire de faire évoluer ce nombre de traits sémantiques. Il serait donc souhaitable de pouvoir ajuster la fonction Sem ainsi que les représentations initiales des prototypes sémantiques de référence

Il faut donc pouvoir mettre en place un formatage récurrent, une rétroaction sur les représentations construites et des fonctions nouvelles sur ces représentations. Si on dispose d'un grand nombre de formes moulin à N2 non reconnues par le modèle initial, il faut pouvoir évaluer les valeurs des traits et les comparer à toutes celles qui sont déjà stockées et essayer de les classer. On atteint là une étape cruciale dans notre démarche de construction des processus de représentation avec la PàP qui nous oblige à revoir le type de représentation à définir pour réaliser les objectifs annoncés.

5.3.1. Lexique et prototypes

Que sont les comportements (au sens donné par Self de ce terme) communs à tous les exemplaires d'une catégorie ou, pour rester dans le domaine de la linguistique, d'une catégorie grammaticale? Cette question a-t-elle un sens? Dans ce qui a été présenté supra, les seuls comportements communs définis sur les "catégories" manipulées ne concernent pas des propriétés comportementales purement linguistiques, il s'agit simplement d'"artifices" informatiques qui permettent d'agir sur les objets utilisés soit pour permettre de dupliquer ces objets soit pour permettre des opérations d'impression des facettes déterminées sur ces objets. Peut-on envisager que les comportements (s'ils existent) des différentes catégories grammaticales ne soient pas dépendants de ces mêmes catégories? Le partage de certains comportements pourrait exister de manière transversale par rapport aux catégories. Il aurait pu en être ainsi dans ce qui est présenté supra puisque tous les prototypes utilisés déléguent au moins le comportement de clonage à un trait associé : une factorisation de ce comportement est donc envisageable. Là encore, il ne s'agit pas du partage de comportement purement linguistique : la mise en place de telles factorisations sur des comportements linguistiques communs à plusieurs catégories linguistiques est à faire. On atteint là le problème général de l'organisation des connaissances linguistiques à penser pour l'analyse dans un environnement informatique à la Self. La notion de comportement est fondamentale dans celui-ci (*"An object is what he does"*) et ne recouvre pas complètement les notions de catégories "linguistiques" traditionnelles (privilégiant la représentation). On peut d'ailleurs modifier les prototypes N1 à N2 utilisés précédemment et répartir différemment les attributs définis sur ces prototypes.

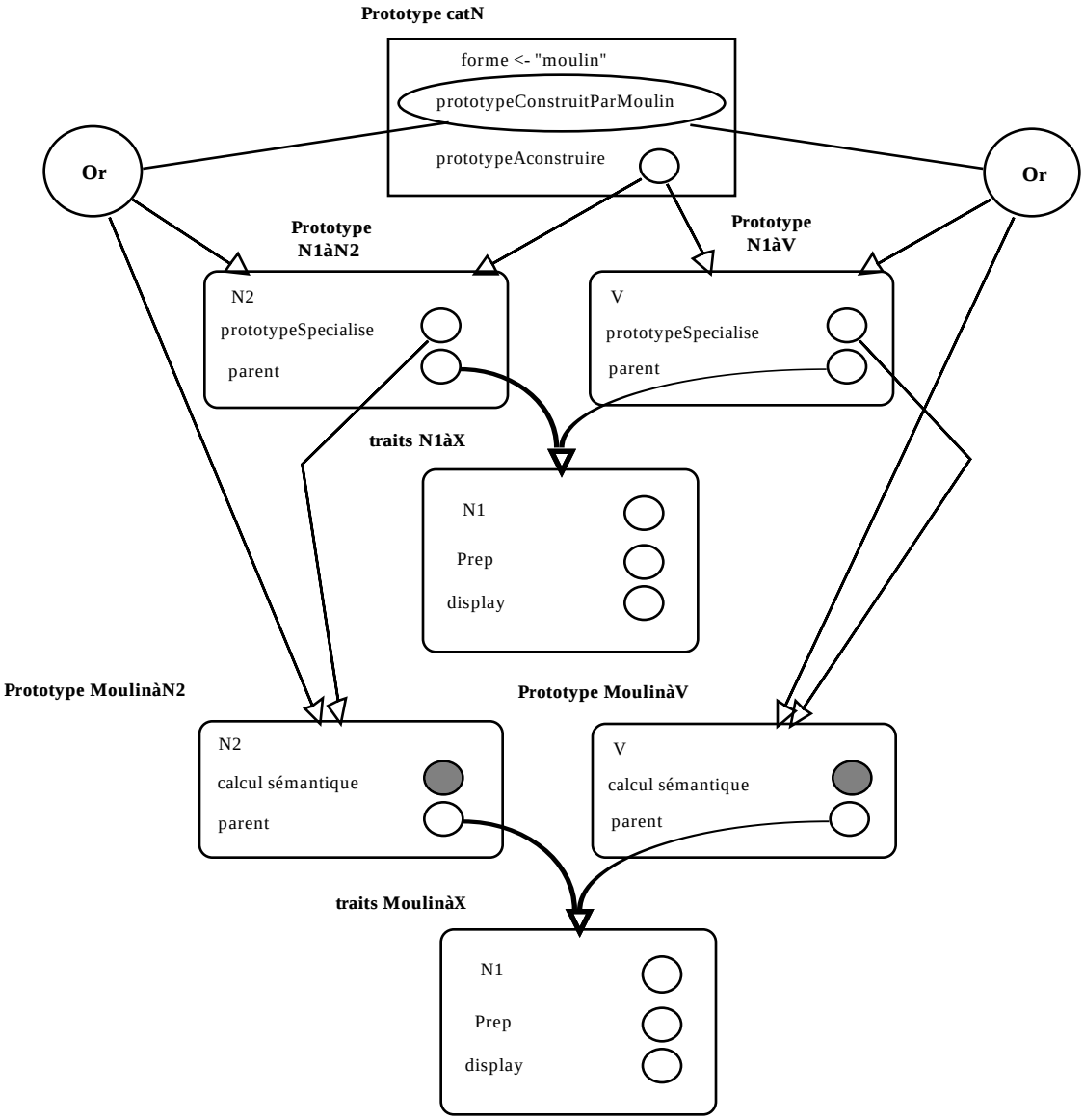


Figure 5.18 : Prototypes et représentation des composés `N1 à X`.

Puisque notre modèle privilégie le rôle du `N2`, on peut adopter une construction de ces prototypes en suivant le modèle que décrit la figure précédente. L'objet `traits CatN1aX` est destiné à recevoir les "comportements" communs aux prototypes `N1 à N2`, `N1 à V`. Notre modèle donnant aux constituants N2 et V un statut particulier (c'est lui qui permet une classification sémantique du composé), les attributs non spécifiques aux prototypes (`N1, Prep, display`) sont pris en charge par l'objet qui réalise le partage des connaissances entre ces prototypes. Les attributs spécifiques (`N2` et `V`, et les attributs réalisant le calcul sémantique) sont quant à eux maintenus dans la définition des prototypes `N1 à N2` et `N1 à V`. Chacun de ces prototypes est potentiellement créateur d'un prototype spécialisé, respectivement `moulin à N2` et `moulin à V`, si `N1 = moulin`.

Pour les mêmes raisons, les objets `traits` et les prototypes associés perpétuent cette évolution de représentation. Une catégorie ainsi définie serait donc un ensemble composé par :

- Un ensemble de comportements à tous les prototypes regroupé au sein d'un objet,

- Un ensemble de prototypes dont les attributs pointent sur des valeurs spécifiques et qui délèguent des comportements commmuns à l'objet précédemment défini.

La construction du prototype N1 à X (ou celle du prototype moulin à X) reste "un modèle de calcul particulier" pour tous les objets que nous cherchons à construire. Le prototype défini n'est pas un phénomène de surface[70], il est le filtre définitoire que les constituants éventuels devront traverser pour s'inscrire à leurs places respectives. Il s'agit bien sûr d'un filtre syntaxique qui ne dit rien explicitement sur la sémantique du prototype. Cette sémantique est en fait le résultat du comportement qui sera associé à chacun des prototypes construits c'est-à-dire qui auront déjà réussi à "traverser" ce filtre syntaxique. Cette approche de représentation s'éloigne assez nettement de la notion de prototype à la Rosch, en ce qu'elle met en avant une définition comportementale des prototypes : ce ne sont pas des traits distinctifs qui les distinguent mais plutôt des comportement différents dans un contexte donné. Notre approche construit des entités prototypiques qui ont une **ressemblance de famille structurelle** entre eux et dont la sémantique est définie individuellement : le prototype initial définit un calcul de cette sémantique qui est effectué et qui produit un résultat particulier en fonction des éléments du contexte rencontrés.

5.3.2. Syntaxe et prototypes

Une analyse syntaxique, telle que celle que réalise OLMES (Habert 1991c), dispose d'un savoir initial représenté par une grammaire donnée sous forme de règles et il s'agit au cours de l'analyse d'identifier les éléments lus en entrée aux différentes règles de cette grammaire. Notre approche prototypique n'a pas utilisé le même type de représentation du pré-savoir que constitue la grammaire. On a essayé au contraire de construire des prototypes qui encapsulent ce savoir de manière prototypique. Nous sommes restés pour le moment à une solution intermédiaire entre un savoir sous forme de règles et donc assez hiérarchique (et d'une portée assez longue sur une structure phrastique) et un savoir éclaté sur chaque élément d'une structure phrastique. On peut d'ailleurs voir le développement de notre travail comme une tentative de dissoudre un cadre syntaxique traditionnel (en l'occurrence celui représenté sous forme de règle) et de disséminer le savoir qui y est habituellement attaché sur plusieurs entités de l'analyse : **un effacement progressif de la notion de règle**.

Le but de l'analyse est de construire des prototypes déclenchés par les entrées lexicales lues sur la phrase à analyser. Et l'analyse est considérée comme terminée, quand aucune opération de création de nouveaux prototypes, ou d'adjonction de prototypes n'a lieu. L' opération de création de prototypes fait office de partie gauche d'une règle dans une grammaire traditionnelle : moulin peut déclencher la création d'un prototype N1 à N2 ce que l'on peut traduire par N1àN2 -> N1 Prep N2; tandis que l'opération d'appariement de prototypes peut être assimilée à une lecture gauche-droite de la règle que le prototype déclenchée décrit structurellement : le prototype associé à "paroles" peut être intégré au prototype N1 à N2 en position N2, si les prototypes en position N1 et Prep l'ont déjà été à leur place respective. Le savoir syntaxique porté par les éléments prototypiques est resté restreint à des constructions de "courte portée" (ne dépassant la portée d'une phrase). Il semble malgré tout possible de poursuivre cette intuition de représentation qui se situe dans la perspective de construire une analyse automatique qui se ferait en l'absence d'une grammaire à vérifier mais dans une perspective d'une grammaire à construire. On part de rien "mais à chaque passage" on construit la grammaire et si ça marche on analyse le texte donné et on a construit parallèlement la grammaire de ce texte : en fait un prototype de grammaire!

5.3.3. Sémantique et prototypes

La sémantique mise en place dans les différentes phases présentées est restée assez peu déterminée. On a adapté le modèle défini dans (Habert & Fabre 1993). Cette adaptation a consisté à associer aux éléments lexicaux représentés sous forme prototypique, un prototype sémantique regroupant un ensemble de traits donnant une définition minimale de cet élément lexical. Il est apparu très vite que ce choix initial n'est pas adéquat pour atteindre pleinement le but recherché. L'écriture du calcul sémantique défini pour la construction du sens des configurations nominales du type moulin à N2 est restée très statique. Cela est dû en grande partie à l'absence de reflexion sur ce que pourrait être une représentation de la sémantique des

[70] On s'éloigne ici de la version étendue du prototype des cognitivistes.

éléments prototypiques lexicaux dans un environnement à la Self. La difficulté principale est de construire une représentation sémantique des éléments manipulés cohérente avec la notion de prototype. Self permet une structuration hybride des représentations : le partage via les traits, et une spécialisation des connaissances via les prototypes. Une modélisation de la sémantique qui utilise ces caractéristiques définitoires du langage est à faire. En particulier, l'utilisation d'une hiérarchie de concepts est à mettre en place si on veut pouvoir évaluer l'articulation qui est à l'oeuvre entre cette hiérarchie des concepts et la contextualisation des entrées lexicales.

De plus, suivant le choix fait de privilégier une représentation prototypique qui porte à la fois des éléments syntaxiques définitoires et sémantiques, c'est toute l'articulation de ces deux ordres qui est à penser. On en présente une illustration plus haut dans la création du prototype moulin à X. En effet, celui-ci porte un savoir syntaxique qui est "réparti" sur deux objets (via la délégation) et qui indique la valeur des composants qu'il peut recevoir dans telle position. Il porte aussi un savoir sémantique qui lui est attaché et qui dépend du composant en position N2. L'articulation forme-sens est ici restreinte à la portée du composé étudié, il n'est pas sûr qu'on puisse faire évoluer à plus longue distance cette articulation, tout au moins dans notre cadre de travail, cette étude est à faire. Il serait pourtant souhaitable de pouvoir dire par exemple que si le prototype N1 à N2 de forme "moulin à cheval" (porteur d'un prototype spécialisé moulin à N2 "moulin à cheval") dans un texte "prototypique" dont le thème général est de parler de phénomènes énergétiques, on puisse dans ce cas récupérer ce savoir contextuel et le transmettre au prototype moulin à N2 (via le prototype N1 à N2) dont le calcul sémantique serait ainsi facilité. Si l'on pu mettre en évidence la possibilité de modifier les représentations initiales d'une entrée lexicale en fonction d'informations sémantiques relevées par l'analyse (ajout du trait +humain à moulin dans moulin à paroles), les insuffisances de modélisation des représentations mentionnées ci-dessus ne permettent pas de mesurer la portée réelle d'une telle potentialité. Même si l'on peut déjà considérer qu'il s'agit, a priori, d'un élément déterminant en faveur de ce type d'approche en comparaison avec une hiérarchie lexicale prédéterminée qui supporte mal l'affinage en fonction de résultats produits par une analyse.

5.3.4. Ajuster les mécanismes statiques/dynamiques de re/ classification en/pour ajustant/ajuster les représentations initiales

Le dispositif que nous souhaitons construire doit mettre en place des opérations de catégorisation, c'est-à-dire qu'il doit définir les opérations qui permettent de classifier les séquences moulin à N2 en les examinant une à une pour construire ce que Milner[71] décrit comme "*la taxinomie la plus complète, la plus fine et la plus économique possible*" (Milner 1989). On a déjà précisé plus haut les opérations qui permettent d'identifier certaines configurations nominales dans un modèle prédicatif. Le développement de notre travail doit répondre à la question :

Comment à partir de représentations statiques du lexique,
mettre en place des opérations dynamiques pour l'interprétation ?

5.3.4.1. Ajuster les mécanismes de classification en ajustant les représentations initiales

Certains éléments de réponse ont déjà pu éclairer un affinage des représentations initiales. Avec par exemple l'ajout du trait +humain à moulin dans certaines configurations nominales du type moulin à

---

[71] *"La littéralisation est d'autant meilleure qu'elle induit une taxonomie plus naturelle. Les difficultés empiriques ne doivent pas affecter les propositions générales : dans une science empirique littéralisée, une lettre vaut par l'ensemble des propositions empiriques qu'elle rend accessible; dans la science du langage, le noyau dur des propositions empiriques est de nature taxonomique; par conséquent, la science du langage doit se proposer au minimum de construire une littéralisation qui permette la taxinomie la plus complète, la plus fine et la plus économique possible"* (Milner 1989).

paroles. Cela étant, les problèmes déjà mentionnés concernant le travail encore à faire pour une représentation des connaissances compatible et adéquate dans un environnement prototypique ne permettent pas encore de donner une illustration plus précise de phénomènes relevant "d'opérations dynamiques pour construire l'interprétation". Pour reprendre notre exemple d'ajout de trait à un élément lexical dans un contexte donné, on peut néammoins être amené à poursuivre notre réflexion suivant au moins deux axes.

Ne pas maintenir des représentations contraignantes et non éclairantes

Tout d'abord, si l'on maintient la notion de traits, une prise en compte d'un plus grand nombre de traits dans la définition sémantique des éléments lexicaux ainsi qu'une évaluation plus précise de la construction sémantique des composés, pourraient conduire à la mise en place d'opérations plus fines dans la construction sémantique des composés construits et notamment dans l'affinage de cette construction. Le maintien d'une représentation sémantique sous la forme d'une famille de traits sémantiques pose malgré tout, comme on l'a déjà souligné, de nombreux problèmes dans le développement du dispositif à construire. Aussi bien pour l'implémentation et l'évolution de cette implémentation, que pour une description fine de ce qui est représenté (la construction du sens en particulier). On peut en effet difficilement restreindre la construction du sens à un "simple" calcul sur un ensemble de valeurs discrètes, comme s'il s'agissait de distribuer une série de valeurs binaires dans une équation pour produire un résultat adéquat. On a d'ailleurs déjà évoqué le fait que la construction du sens met en place des phénomènes complexes qui s'appuie sur des mécanismes cognitifs qu'il est difficile d'assimiler à de simples équations.

Isoler les mécanismes qui participent à la construction du sens et les articuler hiérarchiquement

On peut aussi mettre l'accent sur la prise en compte des résultats produits pour affiner les représentations lexicales initiales avant de poursuivre une prise en compte d'autres phénomènes. Le modèle sémantique initial permet de traiter de nouvelles configurations nominales du type "moulin à paroles" (qui ajoutent un trait sémantique à la représentation initiale d'un élément lexical) ou "moulin à ventilateur" (dans ce cas c'est la prise en compte d"une "déviance" par rapport au modèle sémantique global qui a été initialisé). On peut donc envisager de considérer ces résultats comme des éléments définitoires dans la représentation du composé (et exécutables dans certains contextes donnés). Cette description de l'ajustement dans la construction du sens du prototype résultant ne doit donc plus être partie intégrante dans la définition du prototype et notamment dans le calcul sémantique qui lui est attaché : d'autant que ce calcul a été construit comme une globalité peu manipulable dont il est difficile d'isoler les différents actions effectuées. Il s'agit au contraire de l'extraire de cette représentation initiale. Puis de définir une description de cet ajustement sémantique comme un comportement sémantique définitoire de l'ensemble des prototypes qui partagent ce type de comportements. On peut ensuite inscrire ce comportement dans l'objet qui centralise les comportements des prototypes concernés. La prise en compte de résultats temporaires doit donc s'accompagner d'un ajustement dans les représentations initialement développées. Ce type d'approche s'inscrit d'ailleurs parfaitement dans un environnement de programmation comme le met en place Self. Il est possible de jouer dynamiquement[72] sur les représentations initialement construites. On peut d'ailleurs procéder de la manière suivante :

- 1. Retrait de l'attribut `addSemantique` au prototype `moulin à N2` via la primitive `_RemoveSlot`;
- 2. Redéfinition de celui-ci , via la primitive `_AddSlots`, en ayant supprimé par exemple le test `Nparole` ou le test `Déverbal`;
- 3. Ajout d'un attribut particulier dans l'objet `traits moulin à N2` qui effectue le même type d'opérations.

---

[72] L'utilisation des primitives `_AddSlots:`, `_AddSlotsIfAbsent:`, `_RemoveSlot:` permet de modifier dynamiquement les prototypes créés.

En procédant de la sorte, ce ne sont plus les valeurs sémantiques qu'il s'agit de préciser, mais il s'agit plutôt d'identifier puis d'articuler les mécanismes qui permettent la construction du sens.

C'est en ce sens que l'on peut caractériser les langages à prototypes comme des langages de programmation contextuelle et les langages à taxonomies de classes comme des langages de représentation. Dans ces derniers, toutes les choses du monde peuvent être décrites sous la forme d'un objet (au sens figé du terme) attaché à une classe définitoire : il s'agit de représenter le savoir que l'objet décrit, la structure définitoire auquel il est attaché est suffisante pour définir cet objet. A l'inverse une approche prototypique ne présume pas de toutes les facettes de ce qui est à représenter : un prototype n'est jamais complètement déterminé quand on le créé. Les circonstances peuvent infléchir la représentation initialement retenue : une redéfinition partielle voir totale de la structure définitoire d'un objet-prototype est toujours possible. A l'inverse, une modification de la définition structurelle d'un objet dans une taxonomie de classes implique le passage obligé par le filtre définitoire que constitue la classe[73] dont il est une instance. Et c'est donc toute la famille d'objets instanciées qui se trouve affectée par cette évolution de représentation.

5.3.4.2. Ajuster les mécanismes de classification pour ajuster les représentations initiales

Il s'agit ici de laisser les représentations ouvertes aux éléments contextuels ou autres qui éclairent leur construction. On a déjà mentionné précédemment un exemple de situation où des éléments contextuels pourraient infléchir la construction du sens d'un composé. C'est le cas de configurations du type "moulin à bras" et "moulin à cheval" pour lesquelles il n'est pas juste d'adapter la représentation des éléments lexicaux en ajoutant un sens `Nénergie` aux entrées lexicales "bras" et "cheval". Nous n'en sommes pas encore à une phase de développement de notre travail qui permette une prise en compte globale des phénomènes contextuels. Malgré tout, il ne faut pas présupposer les résultats à construire, la réflexion sur les mécanismes de catégorisation serait ainsi vite réglée. Il faut au contraire élargir les mécanismes de transferts d'informations entre les agents de l'analyse, même si certains d'entre eux sont considérés dans un premier temps comme vituellement présents même s'ils sont provisoirement incomplètement représentés. On peut ainsi envisager que telle configuration nominale `moulin à N2` ne soit pas effectivement construite sémantiquement car **en attente** d'informations supplémentaires que pourraient par exemple lui fournir des prototypes déjà construits ou à construire, le contexte sous une forme à déterminer, etc. On peut en effet déjà anticiper un modèle de représentation du type d'informations qui seraient pertinentes pour construire des configurations nominales prototypiques et des mécanismes d'envoi de messages qui leur seraient appliqués. Une évolution de ces représentations serait à faire à mesure que les éléments de construction d'une analyse globale se mettraient en place. On rejoint là certaines approches à l'oeuvre dans des formalismes par expert de mots qu'il convient de présenter pour mesurer la distance qui existe entre les approches retenues dans ce type de modèle.

5.3.4.3. Un analyseur qui manipule des mots experts : WEP

De même que notre approche privilégie le mot dans la conduite de l'analyse, WEP (Small 1987) considère chaque mot comme la source essentielle des connaissances pour la compréhension, qu'il s'agisse des connaissances que chaque mot a sur lui-même ou des connaissances que chaque mot a sur ses relations avec les autres mots ou les concepts. C'est ainsi que dans ce modèle on attache à chaque morphème linguistiquement intéressant un expert qui est un graphe de décision dont les noeuds correspondent à des actions ou à des questions. Un noeud action utilise des primitives de Langage d'Interaction Lexicale (LIL) ou des primitives du Langage de Différentiation du Sens (LDS) pour :

- Construire ou compléter un concept.

---

[73] On peut aussi penser à la spécialisation de classe puis à l'instanciation de cette nouvelle classe pour créer un nouvel objet.

- Envoyer des messages.
- Modifier un état.
- Suspendre l'activité d'un expert.

Accès d'un expert aux informations

Un expert dispose d'un certain nombre d'informations qui concernent la compréhension des phrases précédentes (cette mise en oeuvre est théorique et est définie par des questions soumises à l'utilisateur), la liste des experts en attente d'information, l'état global de l'analyse, la situation des démons postés par les experts de mots. Un démon est un agent qui est mis en place par l'expert déclenché si celui-ci ne peut pas dans l'état présent de son activation répondre positivement par manque d'informations : *"Any word expert that desires some piece of information through interaction with one of its neighbors specifies the nature of that information in a data structure called a restart demon"* (Small 1987).

Activation d'un expert

- Par questionnement d'un autre expert.
- Par le biais d'un de ses démons de réactivation qui a pu construire un résultat.
- Par la prise en compte du mot suivant dans la phrase.

Représentation de la compréhension de la phrase

Un expert est capable de représenter les compréhension de la phrase sous la forme d'objets appelés *concepts*. Un *concept* possède des attributs qui décrivent :

- Une liste des éléments de la mémoire dont le concept est une occurrence.
- Une liste des éléments de la mémoire dont le concept peut-être une occurrence de l'un d'eux.
- Une liste des éléments de la mémoire dont le concept n'est pas une occurrence.
- Une liste de couples indiquant les rôles que peut jouer le concept pour d'autres concepts.
- Son rôle particulier dans la phrase.
- La forme lexicale associée à ce concept.

Comportement de l'expert

Un expert peut exécuter les actions suivantes :

1. Lecture du mot suivant de la phrase.
2. Demande d'informations à d'autres experts.
3. Créer un concept, en complèter un autre.
4. Changer l'état global de l'analyse.
5. Questionner le contexte ou solliciter des connaissances plus générales.
6. Suspendre son exécution.
7. Terminer son exécution.

Mise en oeuvre des actions d'un expert

Les langages LIL et LDS définissent des primitives qui décrivent les actions qu'un expert est susceptible de réaliser. Les primitives du langage LIL permettent de déclencher trois types d'action :

- Mise en place de démons.
- Regard en avant.
- Envoi de messages.

Les primitives du langage LDS permettent quant à elles aux experts :

- D'infléchir leur décision en questionnant le contexte.
- De construire des concepts.
- De placer des démons.

WEP et PàP : confrontations

S'il s'agit pour nous d'essayer de décrire les régularités de certaines configurations linguistiques, l'approche de WEP se situe au contraire dans une volonté de tout décrire en partant des irrégularités de la langue : les régularités de la langue ne décrivant qu'une partie des phénomènes à l'intérieur de l'ensemble des irrégularités. Si une telle approche est éclairante sur la mise en place de contrôle local sur les activités d'une analyse, elle pose malgré tout des problèmes fondamentaux dans la gestion et la cohérence des informations qui sont manipulées au cours de cette analyse.

Autant il est intéressant de pouvoir disposer d'éléments qui réalisent une articulation entre les différents domaines de connaissance sur des endroits critiques d'une analyse (c'est d'ailleurs le point qui nous interesse) et qui mettent en place les opérations de récupération de l'information manquant, autant il paraît difficile de concevoir dans le cadre de la construction du sens, un contrôle dans le déroulement de l'analyse qui ne serait dispersé que sur les mots. D'une part, on perd toute idée d'un méta-regard sur l'ensemble de l'énoncé à construire, d'autre part, il ne permet pas de prendre en compte certaines régularités de la langue d'une portée plus longue et de ce fait de construire des méta-connaissances de plus grande portée sur les faits de langue. Les limites que l'on peut concéder à WEP sont en fait contenues dans l'approche retenue. En effet si on limite les problèmes de construction du sens d'un énoncé au niveau des mots, une telle approche peut se révéler cohérente mais reste condamnée à ne rendre compte que des phénomènes locaux, alors qu'on sait que les phénomènes à l'oeuvre dans une construction du sens d'un énoncé appréhendé dans sa globalité ne restreignent pas les interactions possibles au niveau du mot.

WEP s'inscrit dans une approche de la représentation des connaissances dans des domaines de savoir où la classification est floue ou mouvante. Si une telle approche priviligie la modélisation des irrégularités des faits de langue, il faut lui adjoindre un module qui "mette de l'ordre" là où il est possible de le faire. Celui-ci doit modéliser les régularités de la langue (et elles ne manquent pas). On dépasserait ainsi le mot et on pourrait prendre en compte des phénomènes plus importants. C'est d'ailleurs ce que nous souhaitons mettre en place dans notre approche prototypique. Plus précisément, les opérations mises en place par WEP peuvent éclairer notre approche prototypique pour résoudre la mise en oeuvre d'une remise à jour automatique des représentations construites. Si nous disposions d'éléments qui gèrent le questionnement, l'attente d'informations..., nous pourrions ne pas rejeter certaines configurations du fait simplement qu'elles n'ont pas encore reçu les informations pertinentes permettant leur construction.

Incise Chapitre 5

Incise 1. Environnement de programmation avec Self

Les éléments présentés infra font référence à la version 3.0 de Self. Pour certains résultats, il s'agit de calculs faits avec la version 2.0, des indications seront données dans ce cas pour identifier les différences minimes qui n'altèrent en rien la présentation générale de Self.

`Lobby`

Il s'agit de l'objet grâce auquel on accède au monde Self. Quand on veut un créer un nouvel objet, on adresse une requête qui doit être comprise dans le contexte défini par cet objet. Les attributs `traits, globals` (dans la version Self-3.0, ou `prototypes` dans les précédentes), `mixins` sont les racines d'objets accessibles à partir du `lobby`.

`Traits`

Cet attribut regroupe les objets qui définissent les comportements partagés par un ensemble de prototypes. Les prototypes se distinguent principalement en ce qu'ils héritent de deux types possibles de comportement : les `traits clonable` définissent la possibilité de multiplier les prototypes par clonage et les `traits oddball` qui spécifie que le prototype qui en hérite est défini comme unique. C'est le cas par exemple de l'objet `true`. A l'inverse, une liste, un vecteur sont des prototypes qu'il est possible de cloner.

`Globals`

Cet attribut regroupe les prototypes (pré-) definis.

`Mixins`

Cet attribut regroupe certains types de comportement qu'il est possible d'ajouter aux objets sans pour autant leur faire partager l'intégralité d'un comportement.

`Shell`

"In Order for a Self expression to be evaluated, the meaning of "self" must be defined. When an expression is evaluated by a read-eval-print loop, the shell object plays the role of self" (Self Group 1993).

On donne ci-contre une présentation graphique du monde Self qui présente l'organisation initiale disponible.

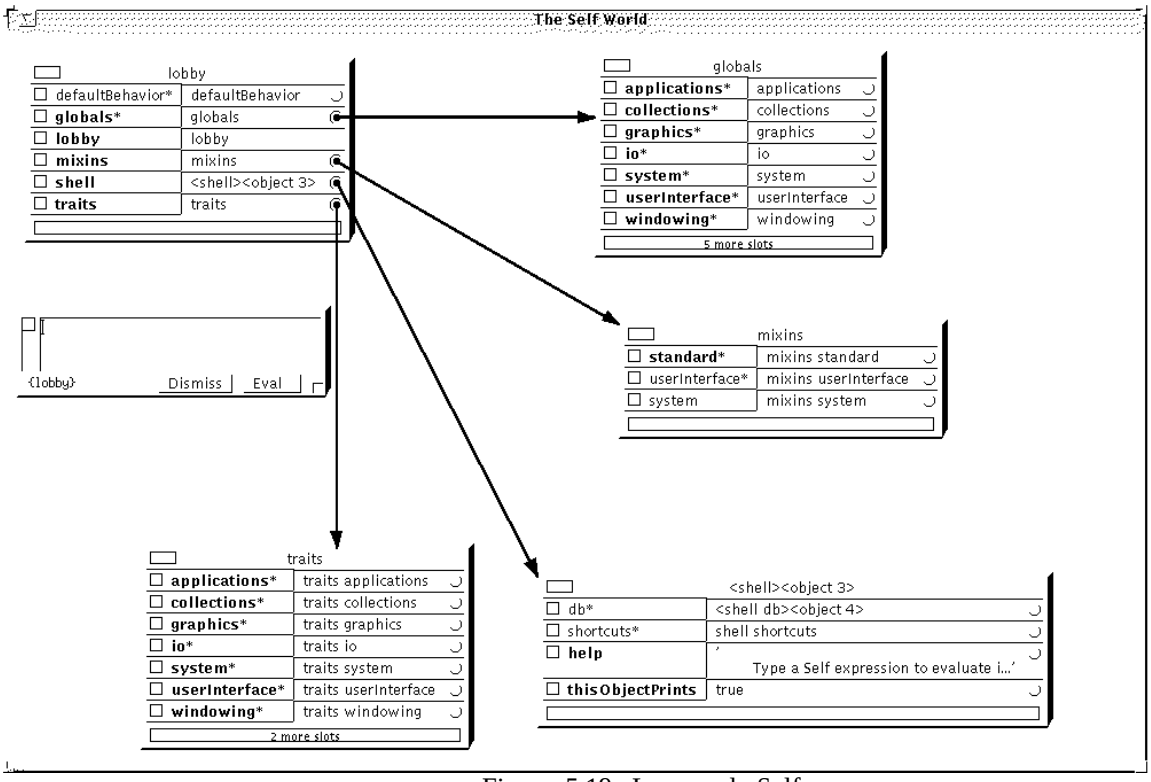


Figure 5.19 : Le monde Self.

"The lobby is a common parent of many object in the system and therefore provides a kind of linguistic common ground for the language. In particular, many useful objects such as true and all the prototype objects are accessible from the lobby. When the virtual machine evaluates the expression you type, messages implicitely sent to self get looked up first in the shell object, then in the lobby, then in the lobby's parents, and so on" (Self Group 1993).

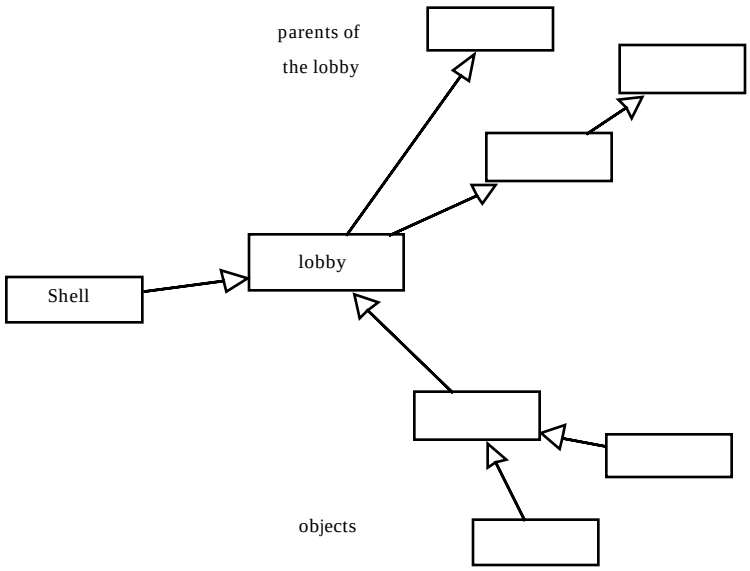


Figure 5.20 : Le `lobby` est au centre du monde Self.

I.1.1. Inspecter les prototypes

La méthode `inspect:` imprime le contenu de chaque attribut de l'objet donné en argument.

I.1.2. Ajouter, supprimer des attributs

Deux primitives permettent d'ajouter des attributs dans l'environnement Self : `_AddSlots`, `_AddSlotsIfAbsent`. Ces deux primitives ajoutent les attributs donnés en argument au receveur du message.

```
_AddSlotsIfAbsent: (| aPrototype = () | )
```

Dans cet exemple, le message d'ajout d'un attribut (si celui-ci n'existe pas) est adressé dans le contexte du `lobby` (par défaut).

```
aPrototype: _AddSlots (| x = 'valeur de l'attribut x du prototype aPrototype'. ^_ y <- 'valeur de l'attribut y du prototype aPrototype'. | )
```

Ici on adresse au receveur `aPrototype` un message d'ajout de deux attributs notés `x` et `y`. Le premier attribut est constant, le second est un attribut statique public (^_)[74] et assignable. La primitive `_Print` permet de visualiser le prototype ainsi défini :

```
aPrototype _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de l'attribut y du prototype aPrototype'. _ y: <-. | )
```

Chaque objet est numéroté, notre prototype est donc associé au nombre 55. Ce nombre permet de pouvoir faire appel à ce prototype en utilisant la primitive `_AsObject`:

```
(55 _AsObject) _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de l'attribut y du prototype aPrototype'. _ y: <-. | )
```

La primitive `_RemoveSlots` supprime un attribut au receveur du message :

```
aPrototype _RemoveSlot: 'y:'
```

Le résultat de cette suppression de l'attribut d'assignation est le suivant :

```
aPrototype _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de l'attribut y du prototype aPrototype'. | )
```

Les deux primitives `_AddSlots` et `_RemoveSlots` permettent de modifier ponctuellement les prototypes auxquels elles s'adressent. Il est possible de construire ou de redéfinir complètement un prototype en utilisant la primitive `_Define`. Celle-ci supprime tous les attributs du receveur, puis ajoute tous les attributs donnés en argument à ce même receveur.

```
aPrototype _Define: ( |  x = 'valeur de l'attribut x du prototype aPrototype'. y <- 'valeur de l'attribut y du prototype aPrototype'. | )
```

I.1.3. Définir des prototypes

---

[74] Voir infra pour une présentation complète des restrictions d'accès aux attributs.

Le code ci dessous permet la définition d'un prototype représentant un point :

1.Définition des comportements des points :

```
traits applications _AddSlotsIfAbsent: (| pointTraits = () | )
```

La primitive `_AddSlotsIfAbsent` permet d'ajouter un attribut pour porter les comportements communs des points :

```
traits pointTraits _Define:  (| clone = ( _Clone).
                                print = (x print.String,
                                        '@',y printString. ). |)
```

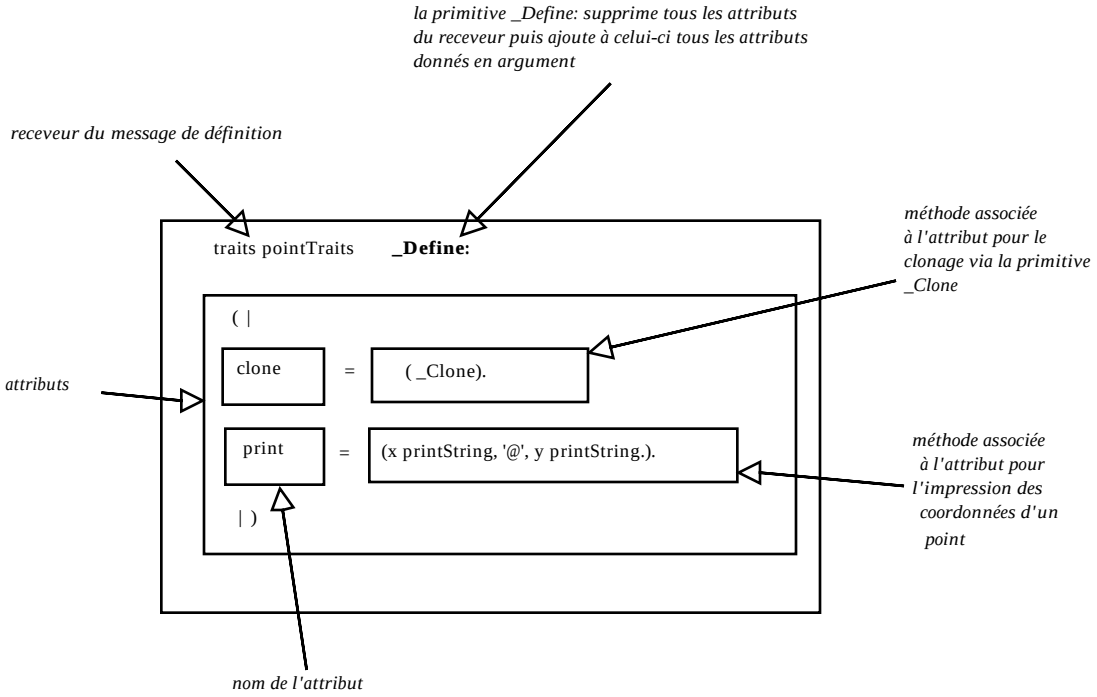


Figure 5.21 : Définition de l'objet regroupant les comportements des prototypes `point`.

2.Définition des prototypes `point` :

```
globals applications _AddSlotsIfAbsent: ( | pointProto = () | )

pointProto _Define: ( | parent* =  traits pointTraits.
                                x <- 0.
                                y <- 0.
                      | )
```

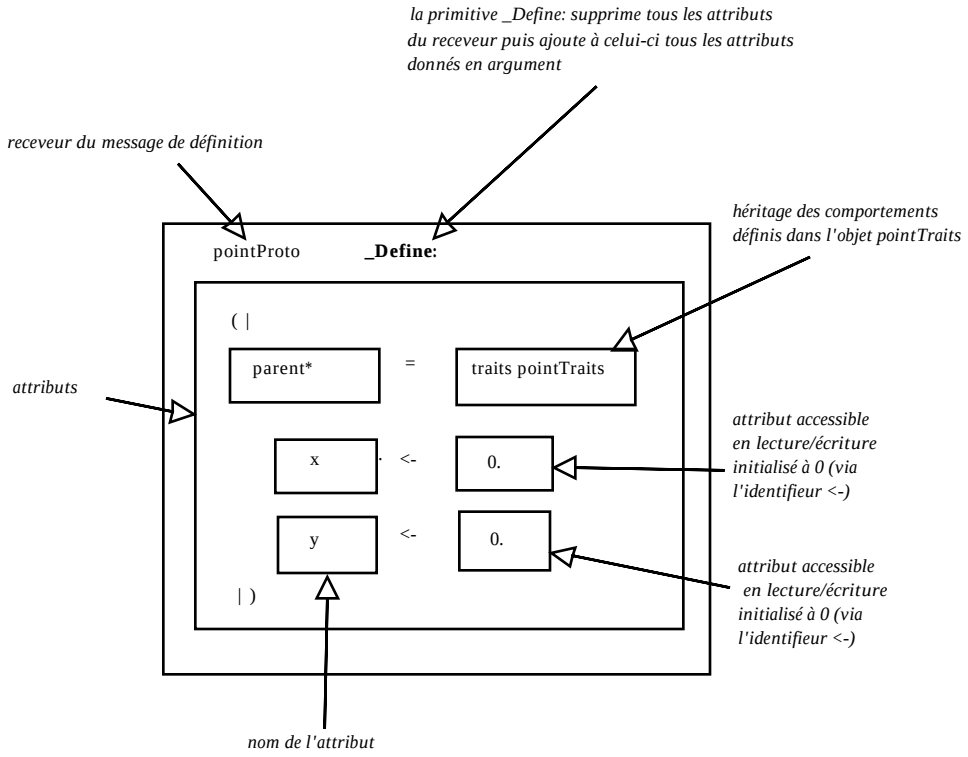


Figure 5.22 : Définition d'un prototype de `point`.

I.1.4. Créer des prototypes

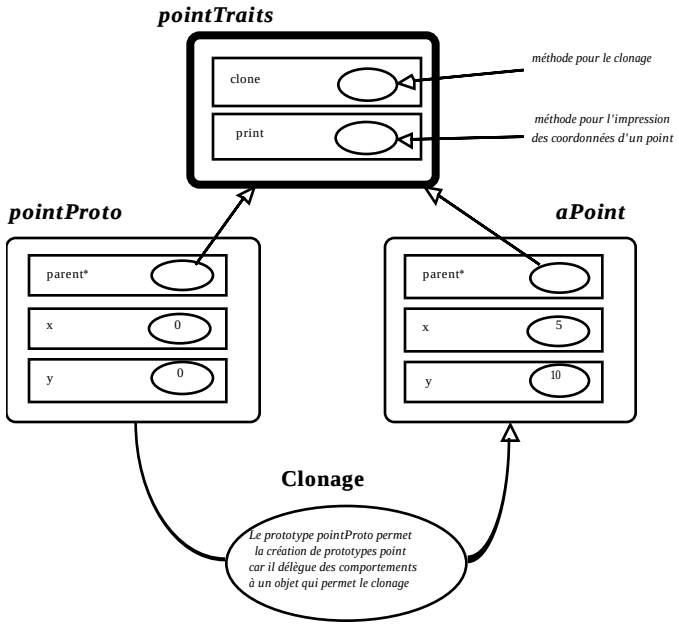


Figure 5.23 : Création d'un prototype de `point`.

La primitive `_AddSlots` ajoute un attribut dans le monde Self :

```
Self> _AddSlots: ( | aPoint | )
shell
```

On peut ensuite adresser à cet attribut les messages combinés de clonage du prototype `point` avec assignation des attributs `x` et `y` aux valeurs données en arguments :

```
Self> aPoint: ((pointProto clone) x: 5) y: 10)
<object 23>
```

Les comportements associés aux prototypes `point` sont décrits ci-dessous :

```
Self> traits pointTraits _Print
<5>: ( |  clone = <a method>.  print = <a method>. | )
nil
ou
Self> (5 _AsObject) _Print
<5>: ( |  clone = <a method>.  print = <a method>. | )
nil
```

La primitive `inspect` permet d'obtenir une visualisation précise de l'implémentation des objets qu'elle décrit :

```
Self> inspect: traits pointTraits
( |  clone = ( _Clone).
      print = (x printString, '@', y printString.). | )
<object 5>
Self> inspect: (5 _AsObject)
( |  clone = ( _Clone).
      print = (x printString, '@', y printString.). | )
<object 5>
```

Le prototype `point`, créé ci-dessus à partir du prototype initialement défini, délègue ses comportements à l'objet `traits pointTraits` (numéroté 5) et possède deux attributs de données `x` et `y`, accessibles en lecture et en écriture (décrit par l'identifieur <-) :

```
Self> aPoint _Print     (ou (23 _AsObject) _Print )
<7>: ( | parent* = <5>. x = 5. x: <-.  y = 10. y: <-. | )
nil
Self> inspect: aPoint   (ou  inspect: (23 _AsObject) )
( | parent* = traits pointTraits.
         x  <- 5.
         y <- 10. | )
<object 23>
```

I.1.5. Accès aux attributs des prototypes

```
Self> aPoint x (lecture)
5
Self> aPoint y (lecture)
10
Self> aPoint x: 6 (écriture)
<object 23>
Self> aPoint x
6
Self> inspect: aPoint
( | parent* = traits pointTraits.
            x  <- 6.
            y <- 10. | )
<object 23>
```

La méthode d'impression des coordoonnées d'un point est définie dans l'objet `traits pointTraits` (porteur des comportements communs aux prototypes `point`); on peut donc adresser aux prototypes représentant des points le message suivant :

```
Self> aPoint print
'6@10'
```

I.1.6. Documenter le code

Il est aussi possible de fournir des commentaires sur les objets manipulés :

```
{ ' Cet objet possède un attribut'
( | x = 0. | ) }
```

Incise 2. Illustration par la modélisation des points avec Self (Ungar & al 1987)

I.2.1. Définition d'un objet attaché aux comportements des objets

```
_DefineSlots: ( | traits =
              ( |clonable = ( |
                 copy = (_Clone).
                 |).
                |).
               |)
```

I.2.2. Définition d'un objet attaché aux prototypes

```
_DefineSlots: ( | prototypes = () |)
```

I.2.3. Définition des comportements attachés aux points

Dans le code qui suit on définit des opérations d'affichage et d'addition de point(s) (via les coordonnées de ces points), alors que la définition des prototypes `point` n'a pas été précisée. Cette définition des `traits point` tient compte d'une connaissance implicite de la définition des prototypes `point` qui délègueront les comportements définis ici aux `traits point`. Il est important de noter que l'utilisation des objets `traits` doit être considérée comme un mécanisme qui comble le manque d'organisation d'un langage à prototypes. L'utilisation des objets `traits` permet de stocker au sein de ces objets des comportements généraux utilisables par les prototypes qui permettent de tels comportements. Ce mécanisme permet d'une part d'alléger la définition structurelle des prototypes en ne conservant au sein de cette définition que les éléments propres aux prototypes et d'autre part de définir des comportements généraux accessibles par tous les prototypes définis par l'utilisateur.

```
traits _DefineSlots: (|
point = (|
1 parent* = traits clonable.
2 print = (x printString, '@', y PrintString).
3  +  = ( { méthode pour l'addition des points } ).
  |).
       |)
```

Le code précédent permet:
1 d'hériter d'un comportement,
2 de définir une méthode pour l'impression d'un point,
3 de définir une méthode pour l'addition des points,

I.2.4. Définition d'un prototype de point

```
prototypes _DefineSlots: (| point =
                            (| parent* = traits point.
                               x <- 0.
                               y <- 0.
                            |).
                          |)
```

On donne ci-dessous une illustration graphique de ce modèle de points en Self.

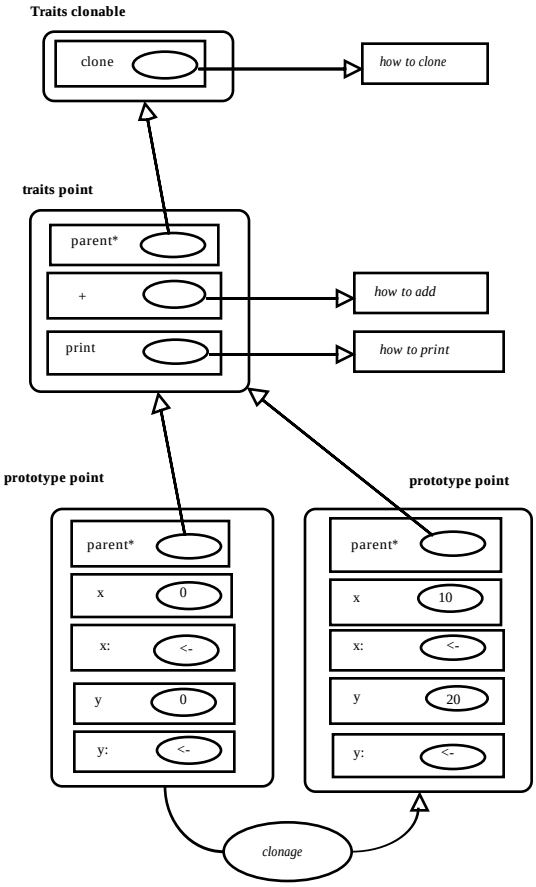


Figure 5.24 : Modèle de points en Self.

Dans ce type de modèle de prototypes, les points sont donc des objets possèdant deux attributs mutables `x` et `y`. Ils délèguent des comportements à l'objet `traits point` qui définit la somme de ces points ainsi que l'impression des coordonnées d'un point. Cet objet parent délégue le comportement de clonage à l'objet `traits clonable`.

Incise 3. Illustration par la modélisation des Noms avec Self

I.3.1. Définitions des objets attachés aux comportements des prototypes

```
1 traits  applications _AddSlotsIfAbsent: ( | catN = () | )
2 traits catN _Define: ( |
3                       parent* = traits clonable.
4                       copy    = (clone).
5                       categorie <- 'Nom'.
6                       display = ( ('<Catégorie: ', categorie printString,'>') print.
                                    morph display.
                                    accord display.
                                ).
                         | )
```

Le code ci dessus permet:
1 d'adresser à l'objet `traits applications` un message de création d'un attribut si celui-ci n'existe pas,
2 d'adresser à l'attribut créé un message de définition,
3 de définir un héritage,
4 de définir une méthode de duplication,
5 de définir un attribut qui portera la valeur catégorielle,
6 de définir un attribut pour l'affichage des attributs des prototypes créés.

La définition de ce réservoir de comportements qui seront accessibles par les prototypes déléguant ce type de comportements à l'objet ainsi créé constitue en fait un regard en avant sur ce que sera une partie de la définition des prototypes non encore définis. On pré-suppose en effet que les prototypes déléguant les comportements définis possèderont certains attributs dont les noms sont utilisés avant même qu'ils ne soient inscrits dans la définition des prototypes. Le comportement `display` s'applique sur des attributs `morphologie` et `accord` qui n'ont pas encore été définis. On procède de même avec les objets suivants:

```
traits applications _AddSlotsIfAbsent: ( | morphologie = () | )
traits morphologie _Define: ( |
    parent* = traits clonable.
    copy    = (clone).
    display = (
                ( ' <Lemme: ', lemme  printString,'>
                    <Forme: ', forme printString,'>
                    <Forme Plur: 'formePlur printString,'>
                    <Forme Sing: ', formeSing printString, '>') print.
                ( ' <Classe: ', classe printString, '>') print.
               ).
| )
traits applications _AddSlotsIfAbsent: ( | accord = () | )
traits accord _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ display = (
      (' <Genre: ', genre type_genre printString,'>
         <Nombre: ', nombre type_nombre printString, '>') print.
               ).
| )
traits applications _AddSlotsIfAbsent: ( | genre = () | )
traits genre _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
| )
traits applications _AddSlotsIfAbsent: ( | nombre = () | )
traits nombre _Define: ( |
  _ parent* = traits clonable.
```

```
  ^ copy    = (clone).
| )
```

I.3.2. Définitions des prototypes

```
globals applications _AddSlotsIfAbsent: ( | catN = () | )
catN _Define: ( |
    parent* = traits catN.
    morph <- ''.
    accord <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | accord = () | )
accord _Define: ( |
  _ parent* = traits accord.
  ^ genre <- ''.
  ^ nombre <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | morphologie = () | )
morphologie _Define: ( |
  _ parent* = traits morphe.
  ^ lemme <- ''.
  ^ forme <- ''.
  ^ formeSing <- ''.
  ^ formePlur <- ''.
  ^ classe <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | genre = () | )
genre _Define: ( |
  _ parent* = traits genre.
  ^ type_genre <- ''.
  ^ marque_genre <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | nombre = () | )
nombre _Define: ( |
  _ parent* = traits nombre.
  ^ type_nombre <- ''.
  ^ marque_nombre <- ''.
| )
```

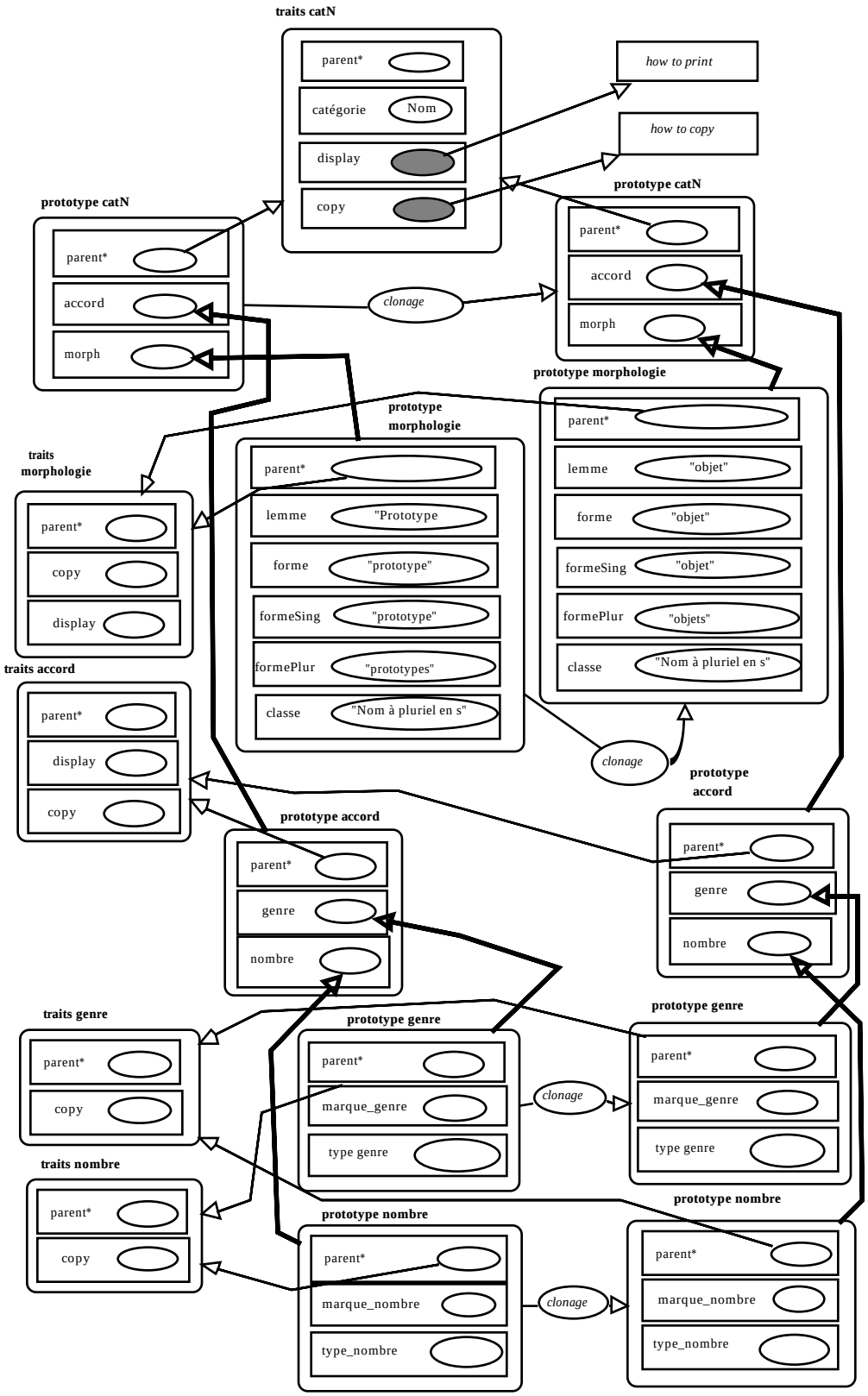


Figure 5.25 :Un modèle de représentation des Noms en Self.

Chapitre 6

Le prototype veut enfin représenter : un nouveau type de moulin

6.1. Représentation dynamique de la construction du sens des séquences en moulin à N2

6.1.1. "Moulin" est avant tout une machine

Le noyau de sens attaché à "moulin" doit tendre à tracer une direction "sémantique" adéquate pour la description de "moulin" comme instrument. Il s'agira ensuite de détacher dans les comportements sémantiques définis ceux qui relèvent plus généralement de la classe des noms d'instrument, tout en permettant de retrouver ces informations particulières dans cette classe de noms et notamment celles qui concernent le comportement sémantique de "moulin" (mise en oeuvre d'héritage de comportement sémantique). On peut d'ailleurs s'interroger sur le fait de savoir si ce processus d'abstraction est cohérent avec une approche prototypique.

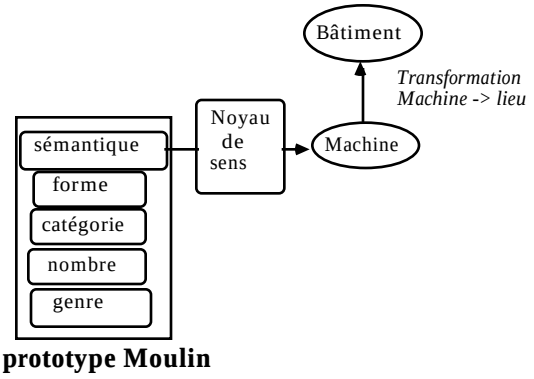


Figure 6.1 : Moulin est avant tout une machine[75].

6.1.2. "Moulin" fonctionne aussi comme un opérateur syntaxique qui prend en arguments une préposition et un nom

[75] La transformation machine -> lieu, une opération générale sur bon nombre de noms de machine (dépendant en partie de la taille de la machine visée : "Je suis entré dans le réacteur" vs "Je suis entré dans le concasseur").

"moulin à blé" "moulin à farine"...

On associe à `moulin` un prédicat dont la valeur prédicative dépend du `N2`.

On se place donc ici dans une approche qui élargit la notion de prédicativité nominale à des constituants autres que des déverbaux : *"C'est dire que certains noms, bien que n'étant pas des déverbaux, n'en disposent pas moins d'un cadre argumental qui restreint les interprétations possibles des autres constituants du composé"* (Habert & Jacquemin 1993). On peut en effet considérer que "moulin" implique un changement d'état qui recouvre plusieurs réalisations lexicales suivant que l'argument soit un objet fabriqué, soit un produit naturel ou un nom d'énergie. On peut ainsi définir un prédicat générique unique `faire-changer-d'état` dont les manifestations lexicales dépendent de l'argument.

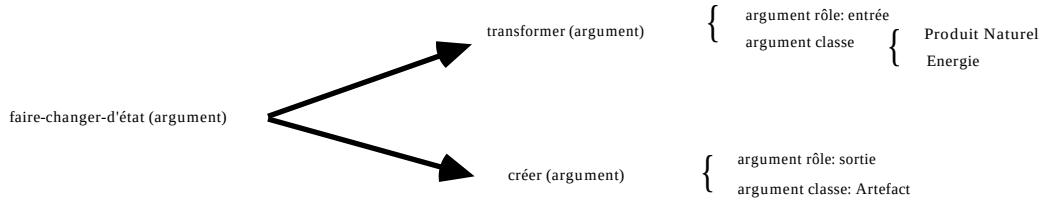


Figure 6.2 : Sous-spécification du prédicat associé à `moulin`.

Ce choix permet tout d'abord de maintenir une représentation qui s'accorde avec bon nombre de noms d'instrument; ensuite, la sous-spécification du prédicat en transformer ou créer peut être vue soit comme un problème de spécialisation lexicale, soit comme un phénomène de co-compositionalité.

Si `N2` est un produit naturel, la valeur prédicative traduit une transformation.

un moulin à blé transforme du blé en farine

Si par contre, le `N2` est un artefact, celle-ci traduit plutôt une création.

un moulin à farine est un moulin qui crée de la farine

6.1.3. `Moulin` et les prototypes

`Moulin` est un opérateur : notre prototype `moulin` initial peut déclencher la construction d'un prototype `moulin à N2` qui sera construit s'il est possible de lui adjoindre un prototype `Prep` de forme "à" et un prototype de catégorie `Nom` en position `N2`.

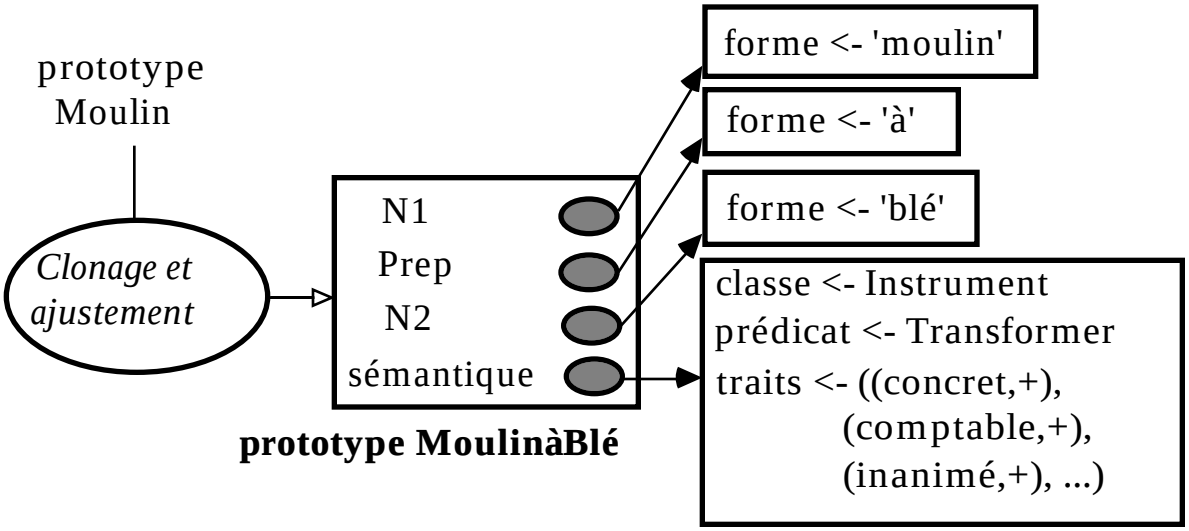


Figure 6.3 : Un prototype pour `moulin à blé`

`Moulin à farine` peut être associé à un prototype obtenu par clonage et ajustement du prototype associé à `moulin à blé`.

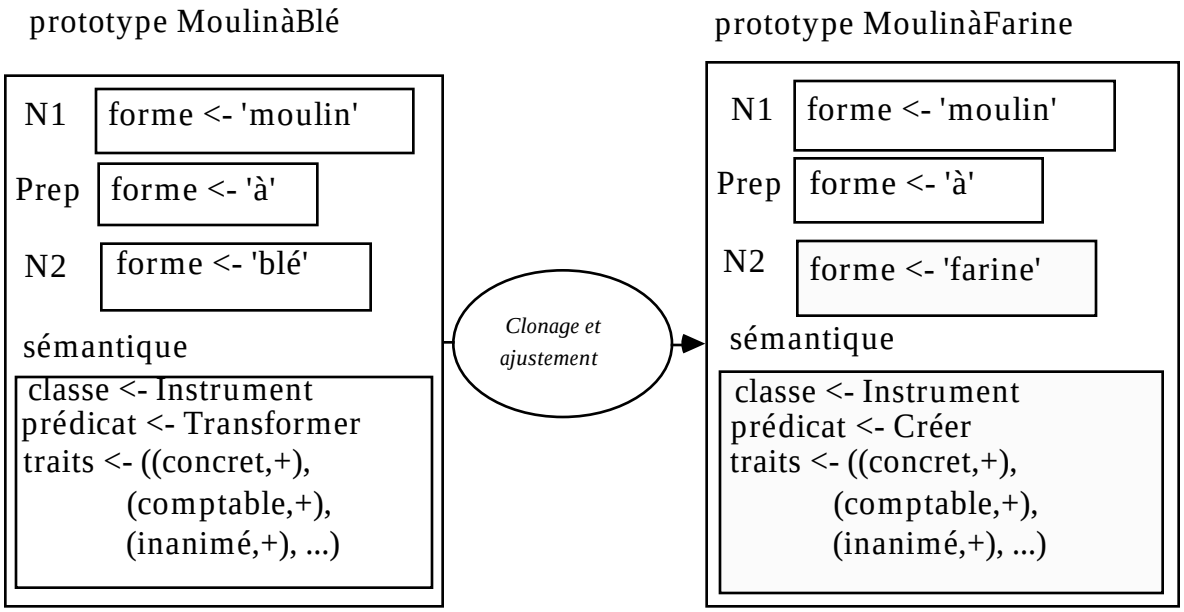


Figure 6.4 : Un prototype pour `moulin à farine`

6.1.4. `Moulin à paroles` peut être vu comme une spécialisation

- `Moulin à paroles` peut être associé au prédicat `créer` : création d'un excès de paroles.
- `Moulin à Paroles` désigne une entité de type `+humain` : une personne qui telle une machine produit une grande quantité de paroles.
- Création du prototype `moulin à paroles` par différenciation du prototype `moulin à farine`.

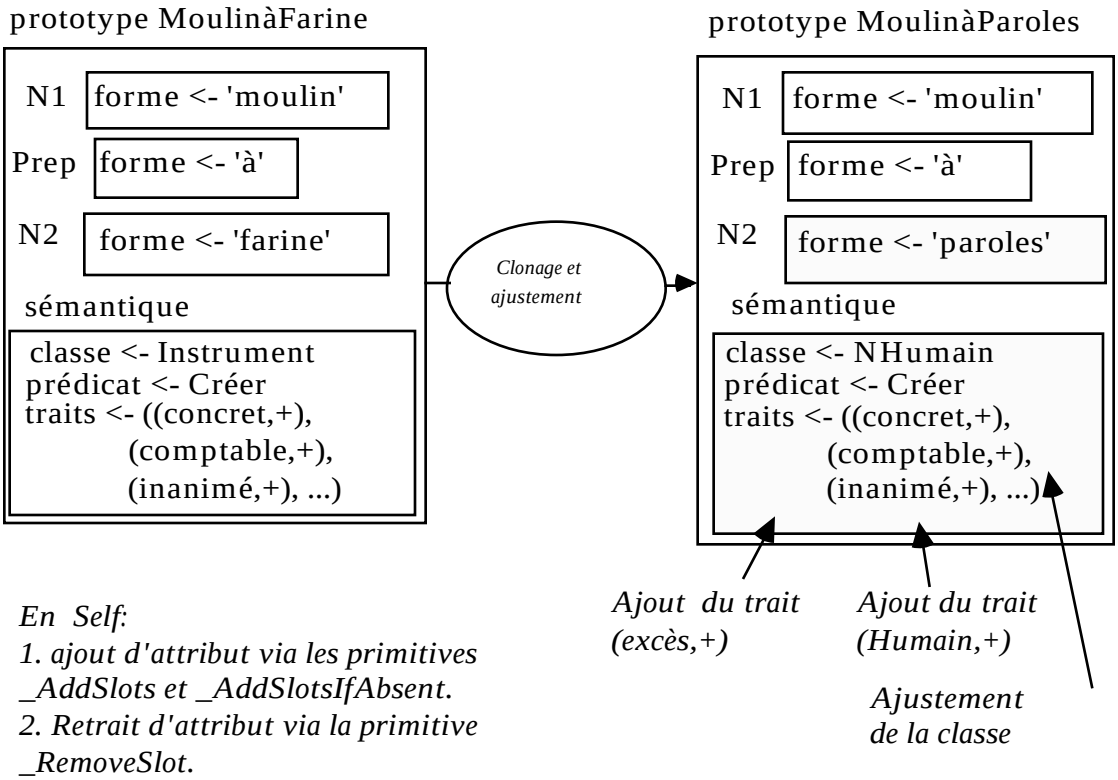


Figure 6.5 : Un prototype pour `moulin à paroles`

6.1.5. `Moulin à paroles` peut être vu comme une généralisation : un nouveau type de moulin

Les emplois métaphoriques permettent d'élargir la prise en compte des `N2` de type `Nparole`.

`moulin à coups, moulin à systèmes, moulin à thèmes anglais, moulin à images, moulin à ennui ...`

- Changement d'état du processus de création de l'argument donné : production massive, importante...
- Reconfiguration de la représentation initiale.
- Création d'un prototype `moulin à N2` : une machine abstraite.
- Le prototype `moulin à N2` est, par défaut, un instrument de transformation ou de création.

La sémantique initiale attachée au prototype `moulin à N2` est sous-déterminée : `moulin` est un instrument et la valeur prédicative associée à ce prototype est `Faire-Changer-D'état`.

- Le prototype `moulin à N2` peut décrire un `Nhumain`.

Si les valeurs lexicales et/ou sémantiques attachées au `N2` vérifient les filtres mis en place dans le calcul sémantique développé par le prototype `moulin à N2`, celui-ci peut déclencher la construction de nouveaux prototypes `moulin à N2` "spécialisés" qui ajustent la sémantique initiale ou qui la redéfinissent.

- Ajustement des valeurs sémantiques et ajout de valeurs sémantiques supplémentaires.

Le prototype moulin à N2 est donc défini prototypiquement de manière sous-déterminée, i.e. la représentation initiale construite lui associe la sémantique suivante : {classe : Instrument, valeur prédicative : Faire-Changer-D'état, traits : Artefact, Concret, Inanimé,...}. L'analyse d'une réalisation donnée pourra étendre cette représentation et construire, par clonage et ajustement, un prototype moulin à N2 dont la sémantique précisera ces valeurs initiales. La figure 6.6 présente une illustration de la représentation initiale du prototype moulin à N2 prédéfini et des ajustements que ce dernier peut générer dans la construction de nouveaux prototypes moulin à N2.

La construction sémantique des composés en moulin à N2 propose une interprétation privilégiée : un moulin est avant tout un nom de la classe instrument, la sémantique "de base" pour moulin porte la valeur prédicative Faire-Changer-D'état. C'est le "poids" de moulin (du N1 dans le composé en N1 à N2) qui déclenche la construction sémantique du composé visé; celle-ci peut à son tour être modifiée en tenant compte des informations co-textuelles rencontrées : en l'occurrence, le "poids" sémantique du N2 peut sous-spécifier cette sémantique initiale soit la respécifier.

- Si l'on rencontre une séquence nouvelle, il reste possible à partir des interprétations existantes, de construire une nouvelle interprétation.

- Clonage du prototype adéquat puis ajout ou retrait dynamique d'informations.

Ces ajustements peuvent être réalisés "à la main" : mise à jour dynamique des attributs des prototypes visés. Si l'on dispose de méta-connaissances qui permettent de mettre en lumière les informations pertinentes qui éclairent une nouvelle interprétation, ces ajustements peuvent être réalisés via l'utilisation de méthodes idoines : mise en place de comportements généraux partagés dans les traits, ou de comportements spécifiques au niveau des prototypes rencontrés.

Ce prototype de référence constitue un point d'entrée très général qui se précise petit à petit si l'inférence le nécessite. Cette solution permet à la fois de tenir compte des savoirs déjà connus (une interprétation privilégiée est mise en avant), elle permet aussi de créer de toutes pièces de nouvelles interprétations par spécialisation ou remodelage. Elle permet enfin de laisser ouverte le modèle de représentation construit. Rien n'empêche de construire une nouvelle interprétation en modifiant les prototypes construits par ce modèle et en ajustant dynamiquement leur représentation. On peut par exemple cloner le prototype moulin à N2 générique et modifier dynamiquement ces attributs pour produire une interprétation différente de celles qui sont présentées dans les figures précédentes.

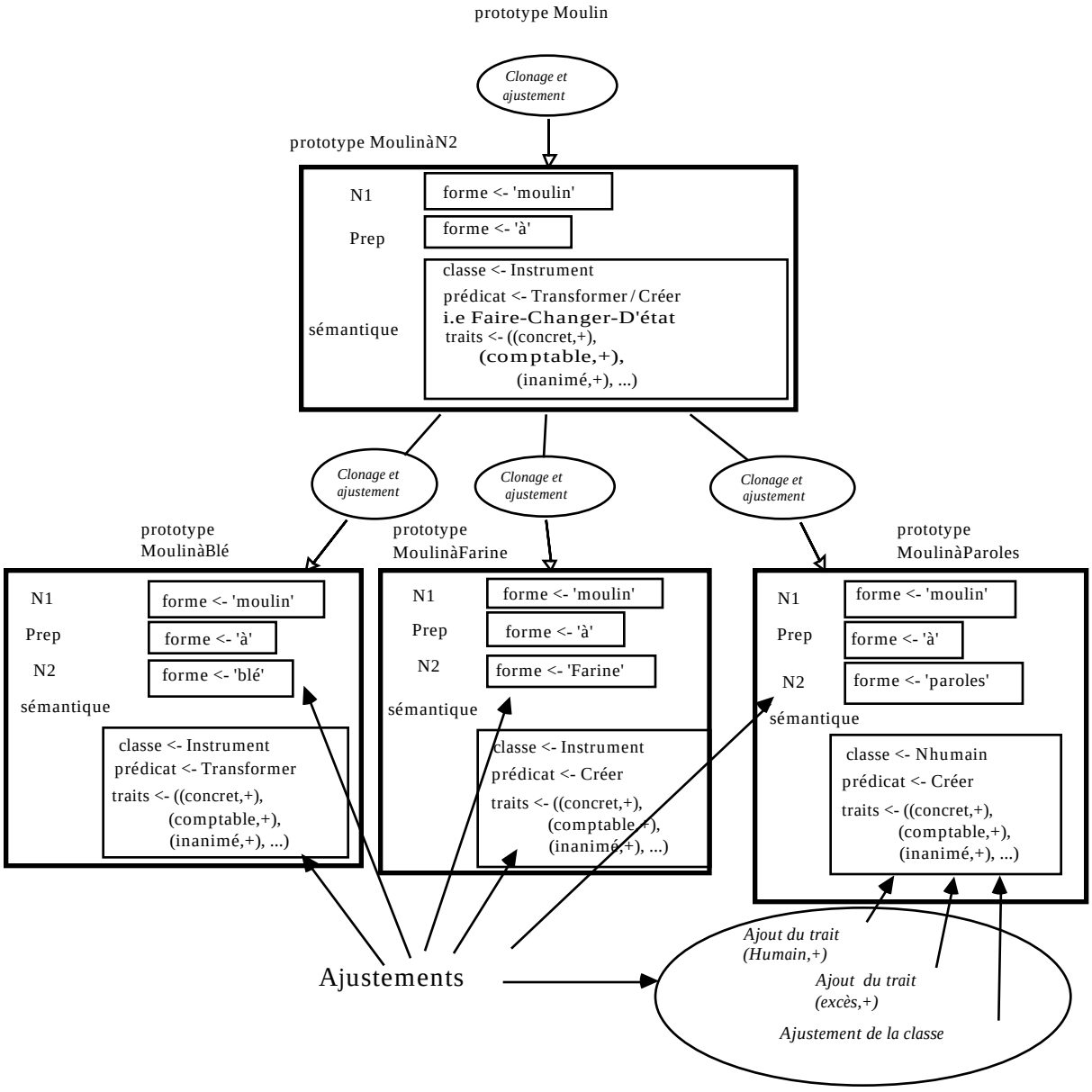


Figure 6.6 : Un nouveau type de `moulin`.

6.2. Extension du domaine de l'analyse

Cette phase de développement d'une implémentation d'un analyseur syntaxique (guidé par des prototypes) poursuit le travail initié dans le chapitre précédent pour ce qui concerne la mise au point de l'analyse et s'appuie sur le "travail théorique" de la construction du sens des formes moulin à N2 initié au départ et résumé en partie ci-dessus.

6.2.1. Du côté de l'analyse

L'utilisateur peut choisir de tracer (ou non) l'analyse, d'afficher (ou non) les prototypes construits à l'issue de l'analyse. De plus, si l'on travaille sur une station couleur (un moniteur 8-bit supportant la version Self 3.0), il est aussi possible d'activer (ou non) une trace graphique de l'évolution de l'analyse. On peut ainsi activer l'affichage des prototypes extraits, sélectionnés, construits au cours de l'analyse (en parallèle au traçage dans le shell). En fin d'analyse, la fenêtre de la trace graphique présente les prototypes construits à l'issue de l'analyse : la présentation est limitée à certains attributs : pour un affichage complet des objets, il convient d'utiliser les outils mis en place par Self pour effectuer la présentation des attributs souhaités. On donne infra une présentation du résultat graphique obtenu.

Déroulement de l'analyse

Le prototype `manager` gère toutes les activités de l'analyse.

- a. Lecture de la phrase ou suite de mots donnée en entrée.
- b. Recherche pour chacun de ces mots s'il existe des prototypes ayant un attribut `forme` qui porte la même valeur que le mot donné.
- c. Le prototype `manager` enregistre les prototypes qui répondent correctement à l'étape précédente.
- d. Recherche des prototypes à construire :

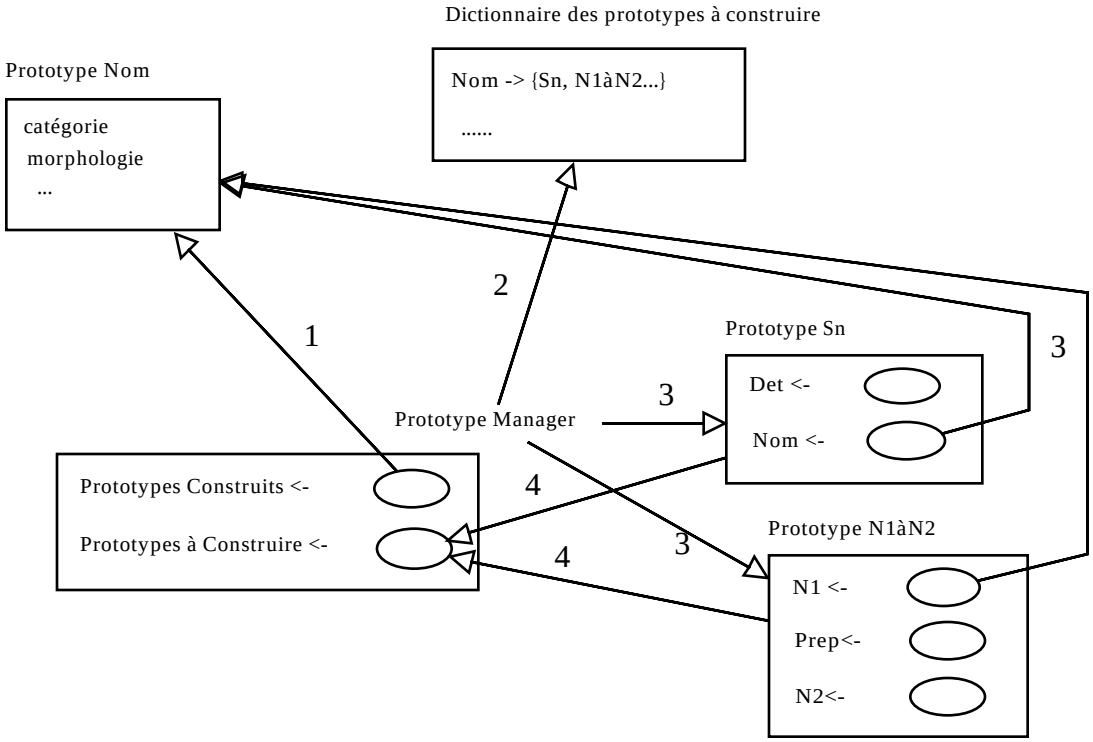
Pour chacun des prototypes initiaux, le `manager` vérifie si ces prototypes sont syntaxiquement créateurs de prototypes de plus grande portée (syntaxique) : par exemple, un prototype dont l'attribut catégorie porte la valeur `Nom` peut déclencher la construction de prototypes représentant des syntagmes nominaux, un "prototype verbe" peut déclencher la construction d'un "prototype syntagme verbal" etc. Le prototype `manager` effectue cette "vérification" en lisant les informations adéquates dans un dictionnaire qui, à chaque entrée (une catégorie du type `det, Nom, V, Sn`...) associe la liste des prototypes qu'il est possible de construire à partir d'un prototype qui porterait une valeur catégorielle identique à celle de l'entrée. Si le `manager` peut associer à un prototype donné un nouveau prototype à construire, le prototype initial est adjoint au nouveau prototype à construire à la place que ce dernier lui réserve et le nouveau prototype à construire est adjoint[76] à la liste des prototypes à construire au cours de l'analyse. Les prototypes initiaux constituent en fait les prototypes "majeurs" pour la construction de ces prototypes de plus longue portée : un prototype de valeur catégorielle Nom peut être qualifié de prototype majeur pour la construction éventuelle d'un prototype de valeur catégorielle `Sn` ou `N1 à N2`. L'analyse se déroulera donc, par la suite, de manière bi-directionnelle : sur la droite pour la construction du `prototype N1 à N2` (ajout d'un `prototype Nom`, en position `N1`, puis d'un `prototype Prep` et enfin d'un `prototype Nom`, en position `N2`) et sur la gauche pour le `prototype Sn` (ajout d'un `prototype Nom` puis d'un `prototype Det`). Le dictionnaire des prototypes à construire est défini de la manière suivante :

76 L'ajout des prototypes à construire est de plus restreint par la position du prototype initial dans la phrase: un prototype initial en position 0 ne peut construire des prototypes que s'il est en position initiale dans un éventuel prototype à construire et si ce dernier se construit sur la droite; de même un prototype initial en position terminale dans la phrase ne peut construire des prototypes que s'il est en position terminale dans ce prototype à construire et si ce dernier se construit sur la gauche.

Ce dictionnaire est créé par clonage de la structure définie par Self pour représenter un dictionnaire. Les entrées de ce dictionnaire sont les catégories qui déclenchent la construction d'un prototype et les valeurs associées à ces entrées sont les listes des prototypes à construire pour ces entrées: `dictOfProtoToBuild at: 'catégorie' Put: {prototype 1, prototype 2...}`.

```
_AddSlotsIfAbsent: ( | dictOfProtoToBuild | )
dictOfProtoToBuild: dictionary copy
dictOfProtoToBuild at: 'Vt' Put: (vector copy) asList
(dictOfProtoToBuild at: 'Vt' ) addLast: (catSVtSn copy)
(dictOfProtoToBuild at: 'Vt' ) addLast: (catSVtClit copy)
dictOfProtoToBuild at: 'Nom' Put: (vector copy) asList
(dictOfProtoToBuild at: 'Nom' ) addLast: (catN1aN2 copy )
(dictOfProtoToBuild at: 'Nom') addLast: (catN1aV copy)
(dictOfProtoToBuild at: 'Nom' ) addLast: (catSn copy)
(dictOfProtoToBuild at: 'Nom' ) addLast: (catAdjN copy)
dictOfProtoToBuild at: 'AdjN' Put: (vector copy) asList
(dictOfProtoToBuild at: 'AdjN' ) addLast: (catSnAdjN copy)
dictOfProtoToBuild at: 'N1aN2' Put: (vector copy) asList
(dictOfProtoToBuild at: 'N1aN2' ) addLast: (catSnN1aN2 copy)
dictOfProtoToBuild at: 'MoulinaN2' Put: (vector copy) asList
(dictOfProtoToBuild at: 'MoulinaN2' ) addLast: (catSnMoulinaN2 copy)
dictOfProtoToBuild at: 'MoulinaVinf' Put: (vector copy) asList
(dictOfProtoToBuild at: 'MoulinaVinf' ) addLast: (catSnMoulinaVinf
copy)
dictOfProtoToBuild at: 'Sn' Put: (vector copy) asList
(dictOfProtoToBuild at: 'Sn' ) addLast: (catP copy)
```

Le contenu du dictionnaire des prototypes à construire est restreint aux entités nécessaires pour les analyses présentées. Il convient donc d'ajuster ce dictionnaire pour élargir le champ de l'analyse.



1. Le prototype `manager` a construit un prototype `Nom`;
2. Le prototype `manager` vérifie dans le dictionnaire des prototypes à construire la liste de ces prototypes dont ce prototype est potentiellement créateur (si elle existe);
3. Si cette liste n'est pas vide, le prototype `manager` créé les prototypes adéquats et leur adjoint le protype `Nom` à la position adéquate : en position `N1` dans le prototype `N1 à N2` et en position `Nom` dans le prototype `Sn`

4. Le prototype `manager` enrichit sa liste des prototypes à construire.

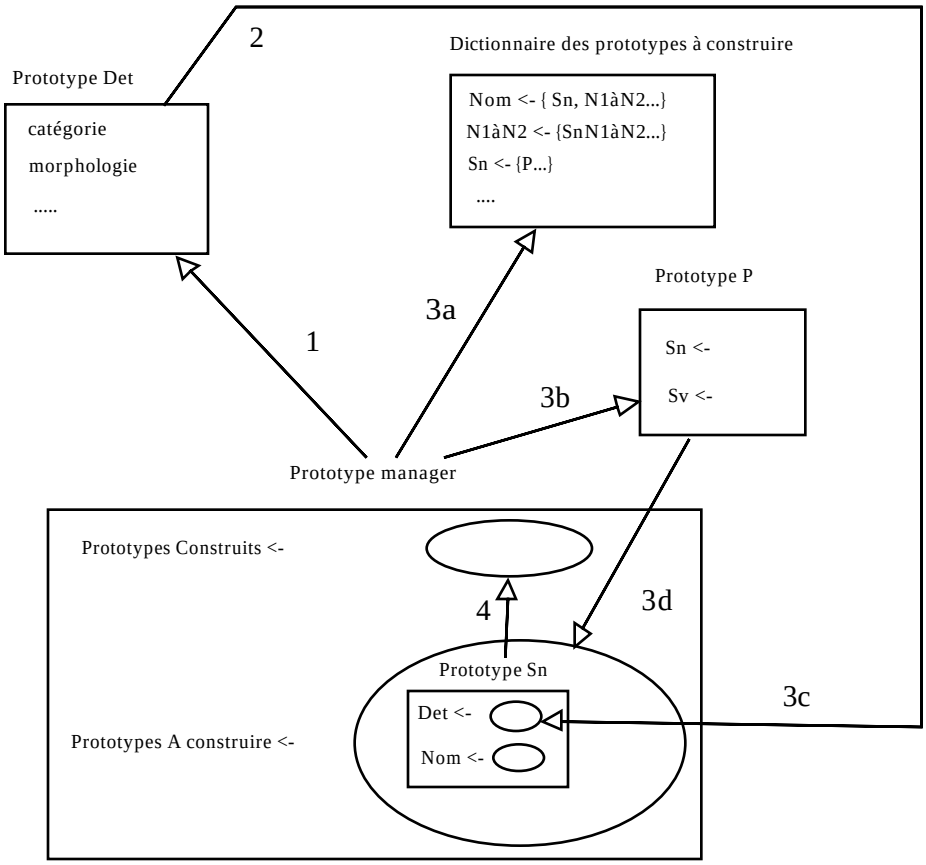
Figure 6.7 : Construction ascendante de prototypes pour l'analyse.

Nous nous plaçons dans une approche de représentation qui manipule a priori des structures de représentation sous-déterminées que les processus mis en place doivent pouvoir affiner dès que de nouvelles informations sont disponibles. Le choix d'une analyse montante s'inscrit dans cette approche : on construit les choses de "haut niveau" à partir d'éléments de "bas niveau".

- e. Construction de nouveaux prototypes :

L'analyse se poursuit donc en prenant appui sur la liste des prototypes à construire. Pour chacun d'eux le manager essaie de lui adjoindre un des prototypes construits disponibles. Le prototype `manager` qui gère les activités de l'analyse va chercher à adjoindre les prototypes disponibles à droite ou à gauche des prototypes sélectionnés[77]. Cette adjonction de prototypes tient compte des contraintes suivantes :

77 L'ajout des prototypes à construire au cours de l'analyse est elle aussi restreinte par la position du prototype déclencheur dans la phrase: un prototype en position terminale dans la phrase ne pourra déclencher la construction éventuelle d'une nouveau prototype que si ce dernier se construit sur la gauche du prototype déclencheur; de même un prototype en position initiale dans la phrase ne pourra déclencher de nouveau prototype que si ce dernier se construit sur la droite du prototype déclencheur.



1. Le manager doit construire un prototype Sn (déclenché dans la figure précédente) auquel il convient d'adjoindre un prototype de valeur catégorielle Det. Il trouve un prototype qui satisfait les conditions imposées.
2. Le prototype Det est adjoint au prototype Sn.
3. Le prototype Sn résultant est construit: il n'impose plus le recherche de constituants supplémentaires.
(3.a) Le prototype manager recherche donc dans le dictionnaire des prototypes à construire si ce nouveau prototype construit est déclencheur de la création d'un nouveau prototype.
(3.b) Dans notre cas, le prototype Sn déclenche la création d'un prototype P.
(3.c) Le manager crée donc ce prototype et lui adjoint le prototype Sn à la position qui lui revient.
(3.d) Le manager place ensuite ce prototype dans la liste des prototypes à construire.
4. Dans le même temps, le prototype Sn construit est extrait de la liste des prototypes à construire et rejoint la liste des prototypes construits disponibles pour la suite de l'analyse.

Figure 6.8 : Les prototypes se multiplient au cours de l'analyse.

1. A chaque fois que l'analyse construit un prototype, le point d est activé : recherche des nouveaux prototypes à construire à partir des prototypes construits. La figure 6.8 reprend l'exemple présenté ci-dessus.

2. La construction d'un prototype peut aussi s'accompagner, en préalable à la construction effective de ce prototype, du clonage du prototype sélectionné : on maintient le prototype sélectionné "actif" (susceptible de recevoir une adjonction de composants) pour ne pas restreindre la construction d'un seul prototype. Dans la figure 6.9, la construction du `prototype Sv (V Sn)` s'accompagne, avant l'adjonction du `prototype Sn` au `prototype Sv`, du clonage du `prototype Sv (V *)` qui attend une adjonction éventuelle d'un prototype de valeur catégorielle `Sn`. Cette adjonction sera réalisée par la suite dès que le `prototype Sn (Det N1àN2)` sera disponible.

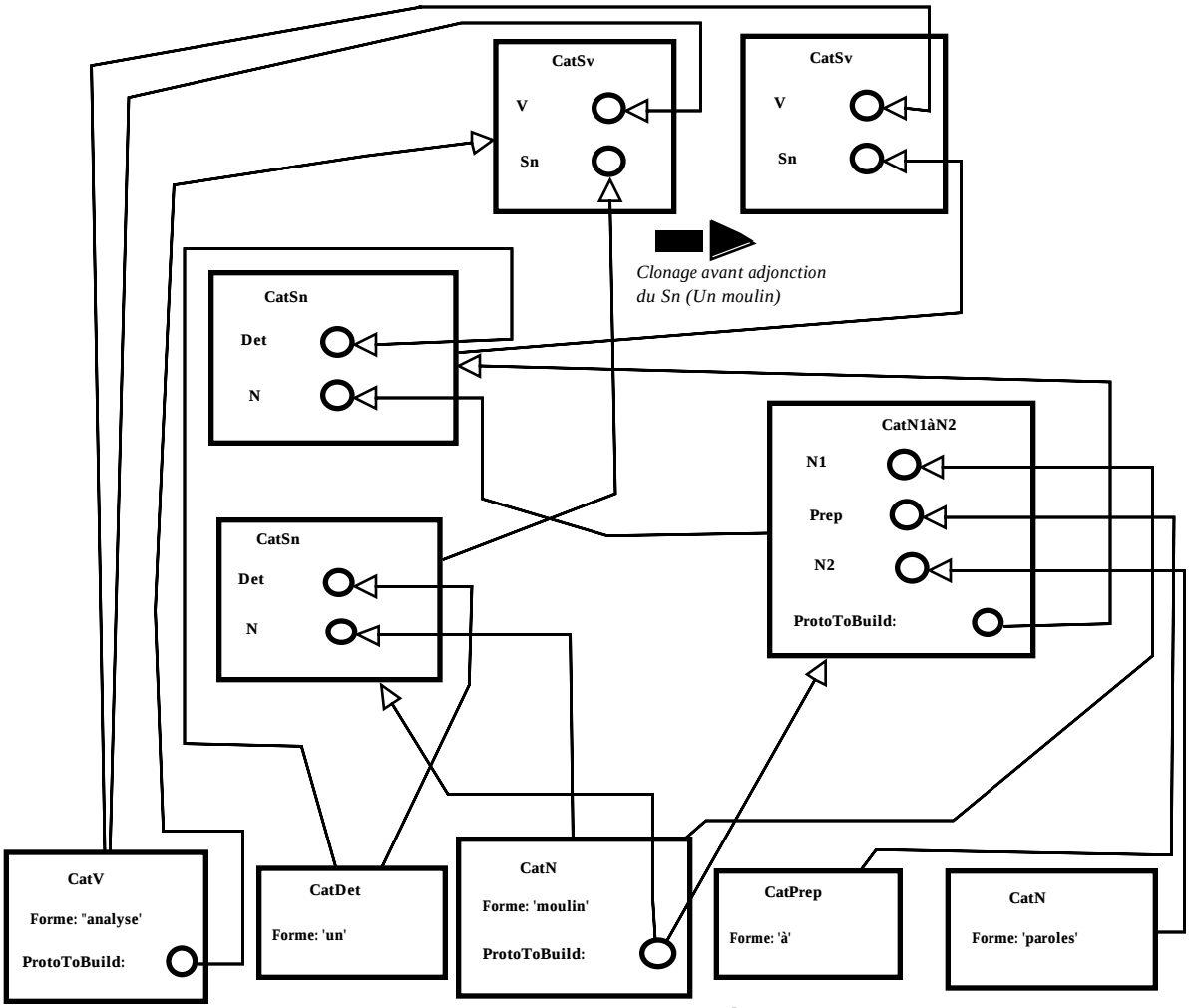


Figure 6.9 : Clonage d'un prototype à construire.

- f. L'analyse s'arrête dès que l'on ne construit plus de prototypes et que l'on ne trouve pas de nouveaux prototypes à construire (Points 1 et 2 précédents).

La figure suivante présente le déroulement de l'analyse sur "Une petite brise la glace". On donne aussi une trace du résultat graphique issu de l'analyse sur cette même phrase.

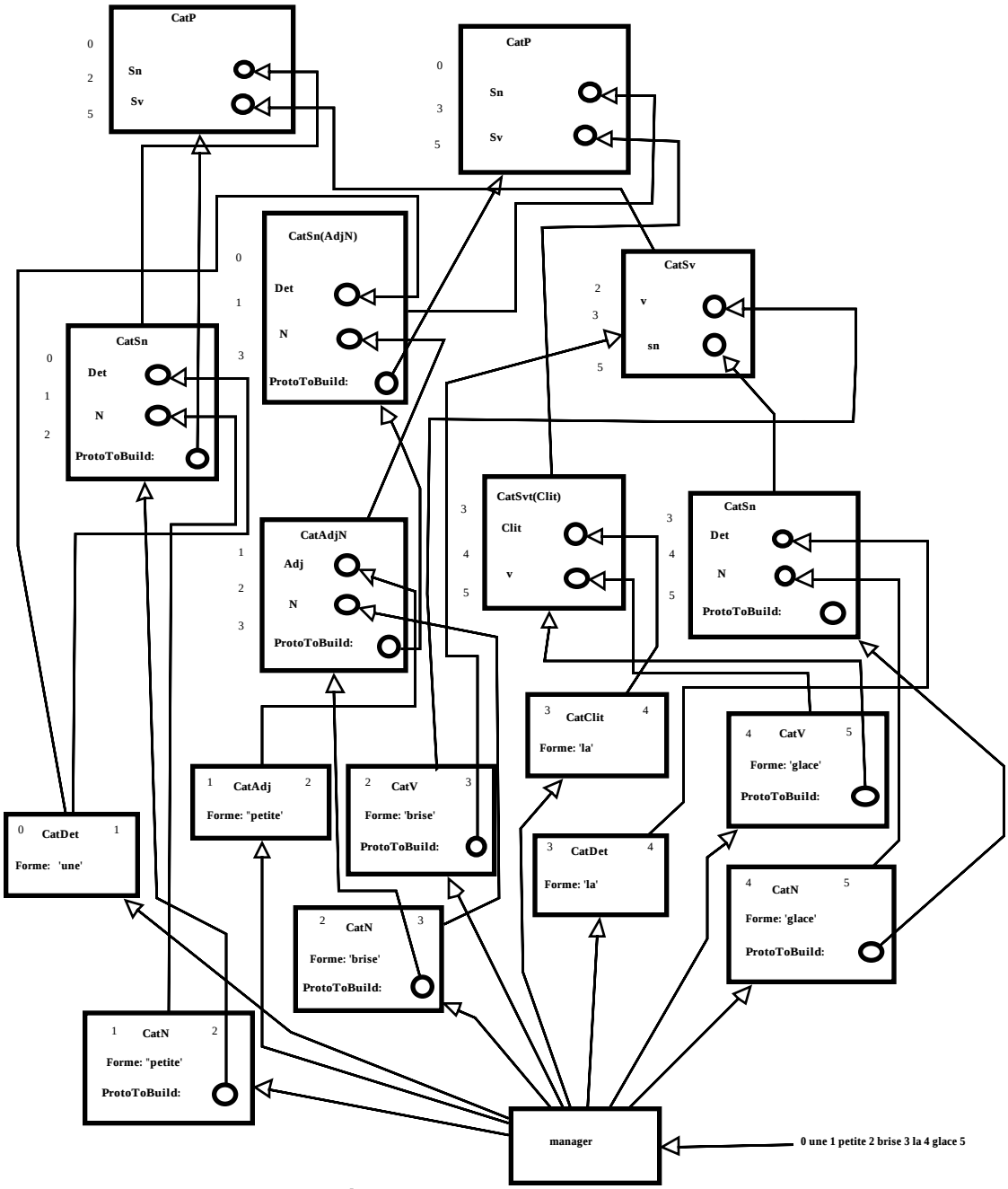


Figure 6.10 : Déroulement d'une analyse sur : "Une petite brise la glace" (1)

Dans la trace graphique qui suit, on présente le résultat de l'analyse obtenu sur "une petite brise la glace". Les prototypes <object 244> et <object 245> représentent cette phrase : <object 244>porte les attributs sn et sv associés aux objets <object 246>, <object 241>qui représentent respectivement "une petite brise" et "la glace", <object 245>porte les attributs sn et sv associés aux objets <object 247>, <object 243>qui représentent respectivement "une petite " et "brise la glace"

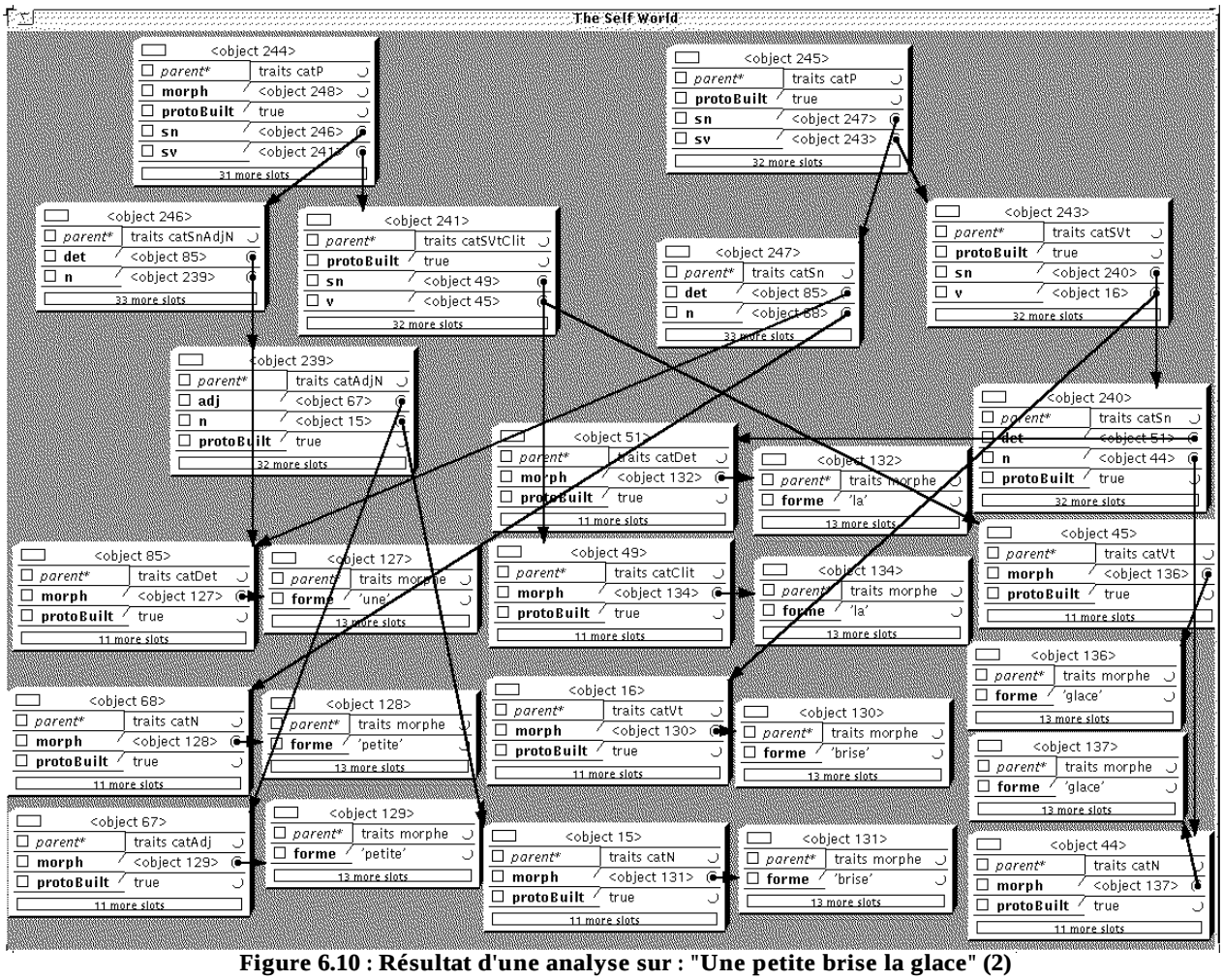


Figure 6.10 : Résultat d'une analyse sur : "Une petite brise la glace" (2)

6.2.2. Du côté des "moulin à N2"

Le prototype moulin peut, à l'image de tous les prototypes représentant des Noms, construire un prototype N1 à N2 (<- dictionnaire des prototypes à construire). Ce dernier peut sous-spécifier sa construction suivant la valeur du N1 qu'on lui adjoint. La construction d'un composé en N1 à N2 peut ainsi déclencher la construction de prototypes spécialisés suivant la forme lexicale du N1 rencontré. Le prototype N1 à N2 se construit donc s'il est possible de lui adjoindre un prototype Prep de forme à puis un prototype Nom en position N2. Il déclenche ensuite la construction d'un prototype spécialisé moulin à N2 qui effectue une (tentative de) construction sémantique du composé ainsi formé. Le déclenchement de la construction de ce prototype spécialisé est activé par le prototype N1 à N2 via la lecture d'un dictionnaire des prototypes spécialisés qui a l'allure suivante :

```
_AddSlotsIfAbsent: ( | dictOfSpecializedProto | )
dictOfSpecializedProto: dictionary copy

dictOfSpecializedProto at: 'moulin' Put: (vector copy) asList
(dictOfSpecializedProto at: 'moulin' ) addLast: (catMoulinaN2 copy)
(dictOfSpecializedProto at: 'moulin' ) addLast: (catMoulinaVinf copy)
```

Là encore le contenu de ce dictionnaire est restreint à la construction particulière des séquences en moulin à N2. Il convient donc d'ajuster ce dictionnaire pour élargir le champ de la construction sémantique des composés N1 à N2. Si l'on étudie, par exemple, plus avant la construction des séquences en Boîte à N2, il suffit de complèter le dictionnaire ainsi défini.

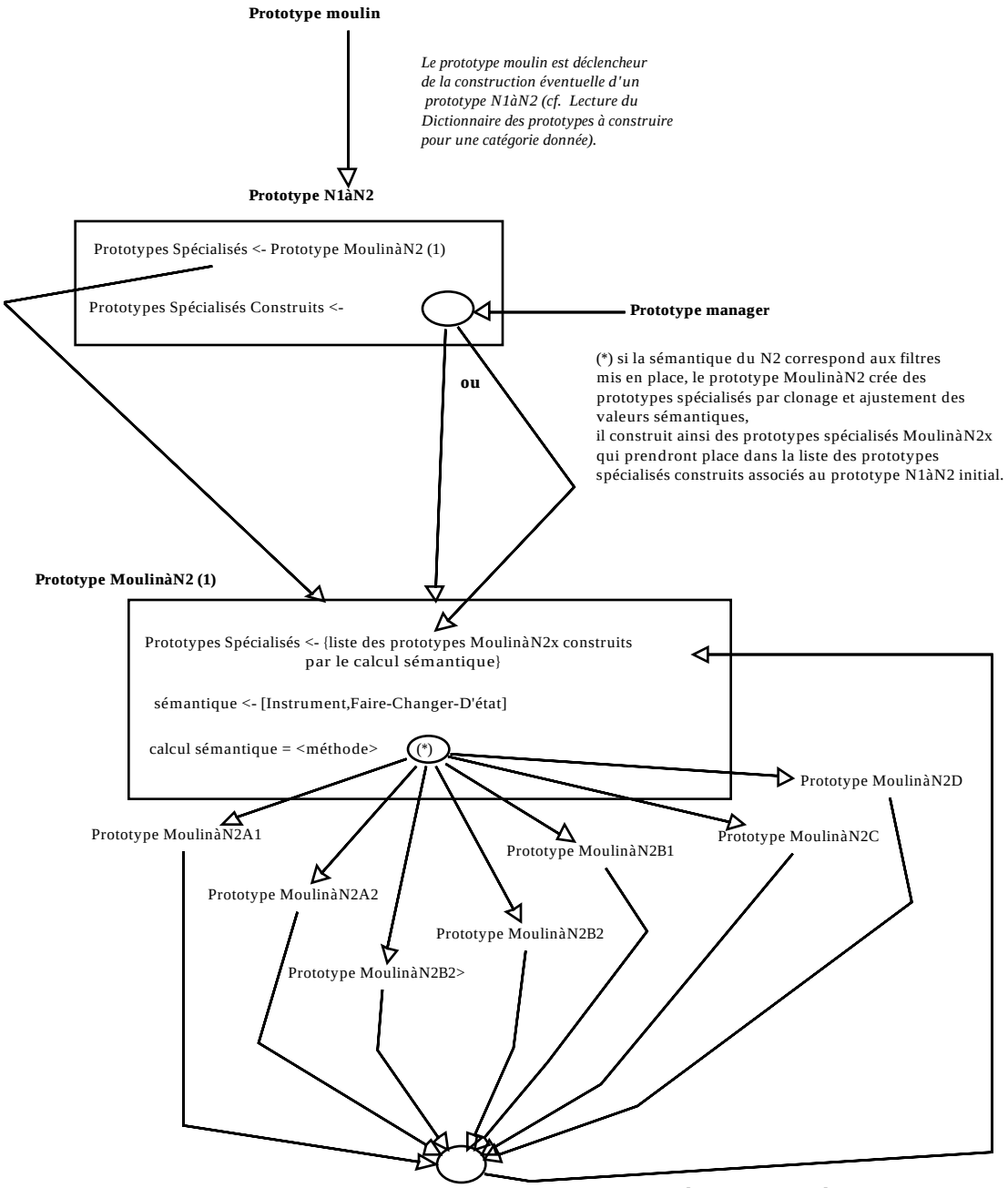


Figure 6.11 : Construction du sens des composés en moulin à N2 (Phase10).

Le prototype `moulin à N2` est défini prototypiquement de manière sous-déterminée, i.e. la représentation initiale construite lui associe la sémantique suivante : {`classe : Instrument, valeur prédicative : Faire-Changer-D'état, traits : Artefact, Concret, Inanimé,...`}. L'analyse d'une réalisation donnée pourra étendre cette représentation et construire, par clonage et ajustement, un prototype `moulin à N2` dont la sémantique précise ces valeurs initiales. La figure précédente présente une illustration de la représentation initiale du prototype `moulin à N2` prédéfini et des ajustements que ce dernier peut générer dans la construction de nouveaux prototypes `moulin à N2`. Si les valeurs lexicales et/ou sémantiques attachées au `N2` vérifient les filtres mis en place dans le calcul sémantique développé par le prototype `moulin à N2`, celui-ci peut déclencher la construction de nouveaux prototypes `moulin à N2` "spécialisés" qui ajustent la sémantique initiale ou qui la redéfinissent. La construction sémantique des composés en `moulin à N2` propose une interprétation privilégiée : un moulin est avant tout un nom de la classe `instrument`, la sémantique "de base" pour `moulin` porte la valeur prédicative `Faire-Changer-D'état`. C'est donc le "poids" de `moulin` (du `N1` dans le composé en `N1 à N2`) qui déclenche la construction

sémantique du composé visé; celle-ci peut à son tour être modifiée en tenant compte des informations co-textuelles rencontrées : en l'occurrence, le "poids" sémantique du N2 peut sous-spécifier cette sémantique initiale ou la remodeler.

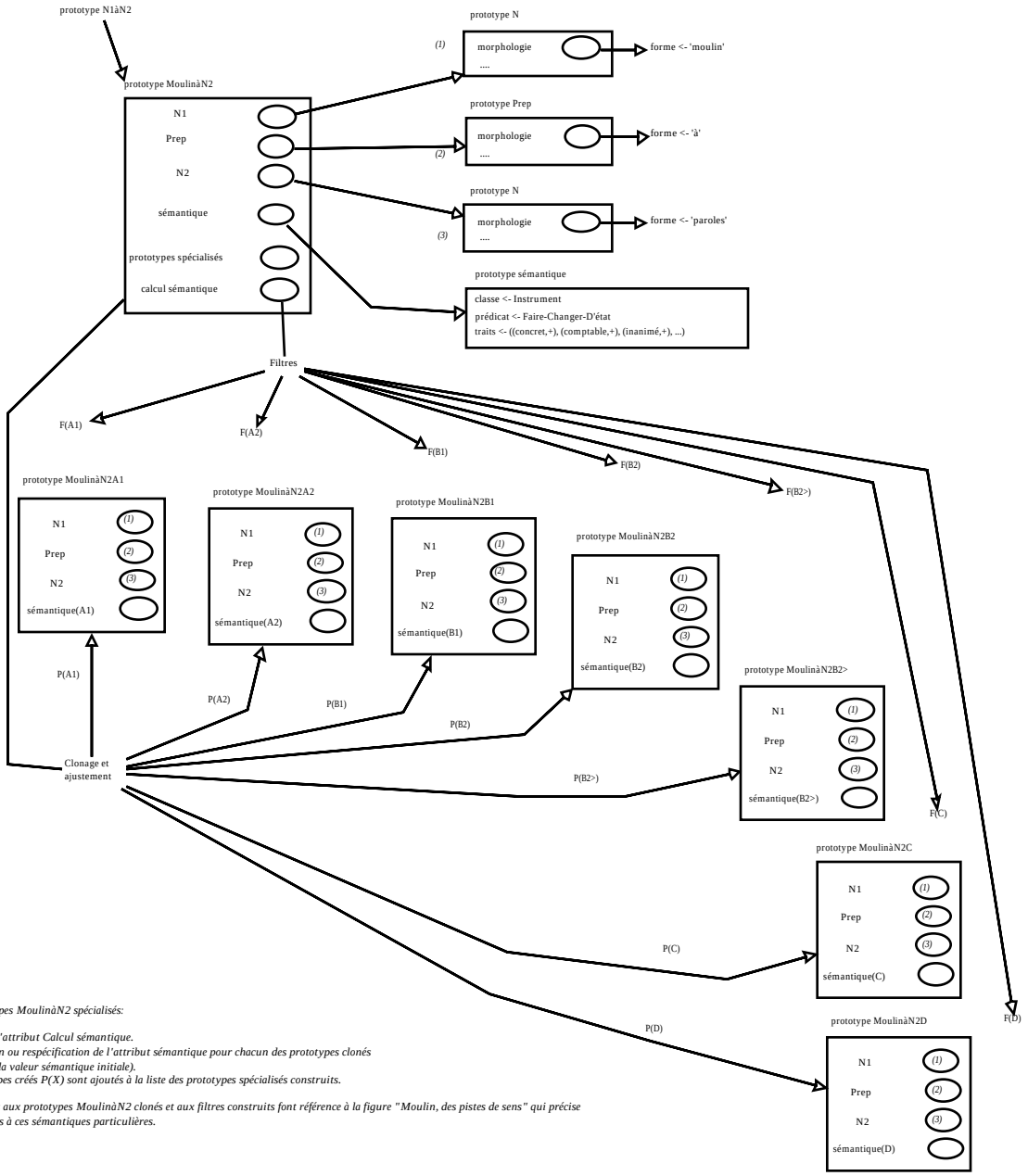


Figure 6.12 : Clonage et ajustement de prototype `moulin à N2` au cours de l'analyse.

Les prototypes `P(X)` éventuellement définis au cours de l'analyse ne sont pas préconstruits comme des prototypes mais existent potentiellement dans la représentation définie par le prototype `moulin à N2`[78] (Création paresseuse). C'est la validation des filtres associés qui induit leur création. Ce qui revient à dire qu'il est nécessaire d'ajuster ces filtres si d'autres prototypes `P(X)` sont nécessaires; il est aussi possible d'ajuster la pré-représentation de ces prototypes dans la représentation qui est faite du prototype `moulin`

[78] Les filtres définis sont en fait associés à l'objet `traits MoulinàN2`: on considère en effet que ces filtres définissent des comportements communs à tous les prototypes `MoulinàN2`.

à N2 initial : on peut de nouveau ajuster la représentation des prototypes construits P(X), il convient pour cela de définir les conditions de cet ajustement et la réalisation de cette spécialisation.

Les figures qui suivent présentent une illustration graphique des prototypes construits à l'issue d'analyses de séquences en moulin à N2.

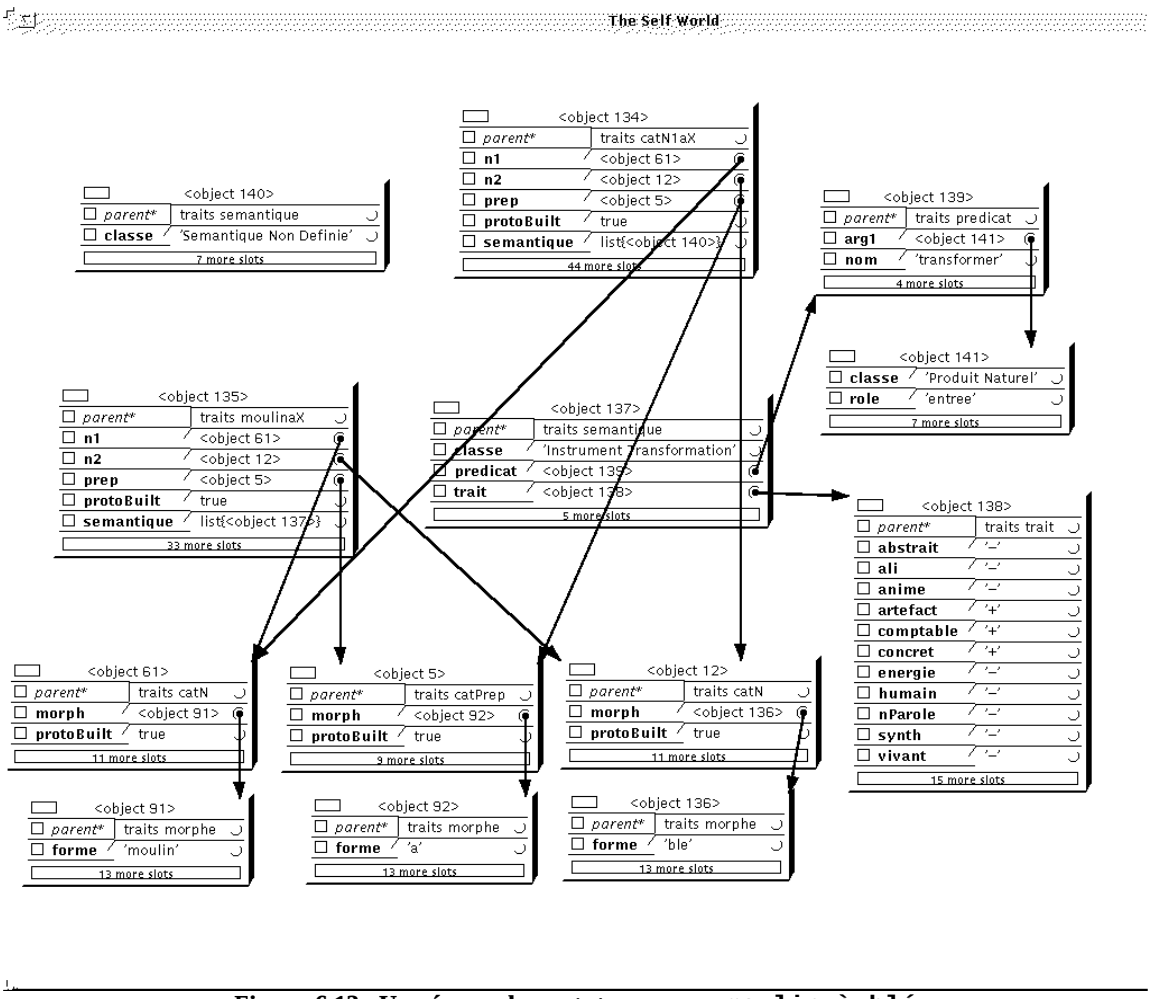


Figure 6.13 : Un réseau de prototypes pour moulin à blé

Comme précédemment, la trace graphique d'une analyse propose le cas échéant un résultat de l'analyse via l'interface graphique de Self. La figure précédente présente les objets issus d'une analyse de la séquence "moulin à ble". Le prototype <object 134> représente le prototype N1 à N2 construit à partir de moulin (à partir du prototype <object 61> représentant moulin), ce prototype a pour composants les objets <object 61>, <object 12>, <object 5> (représentant respectivement moulin, ble, a) portés par les attributs n1, n2, prep. Le prototype <object 135> représente un prototype moulin à N2 représentant moulin à blé construit à partir du précédent et avec les mêmes composants : la sémantique de <object 135> est portée par <object 137> dont les attributs portent les valeurs associées à cette sémantique via les objets <object 138>, <object 139>, <object 141>.

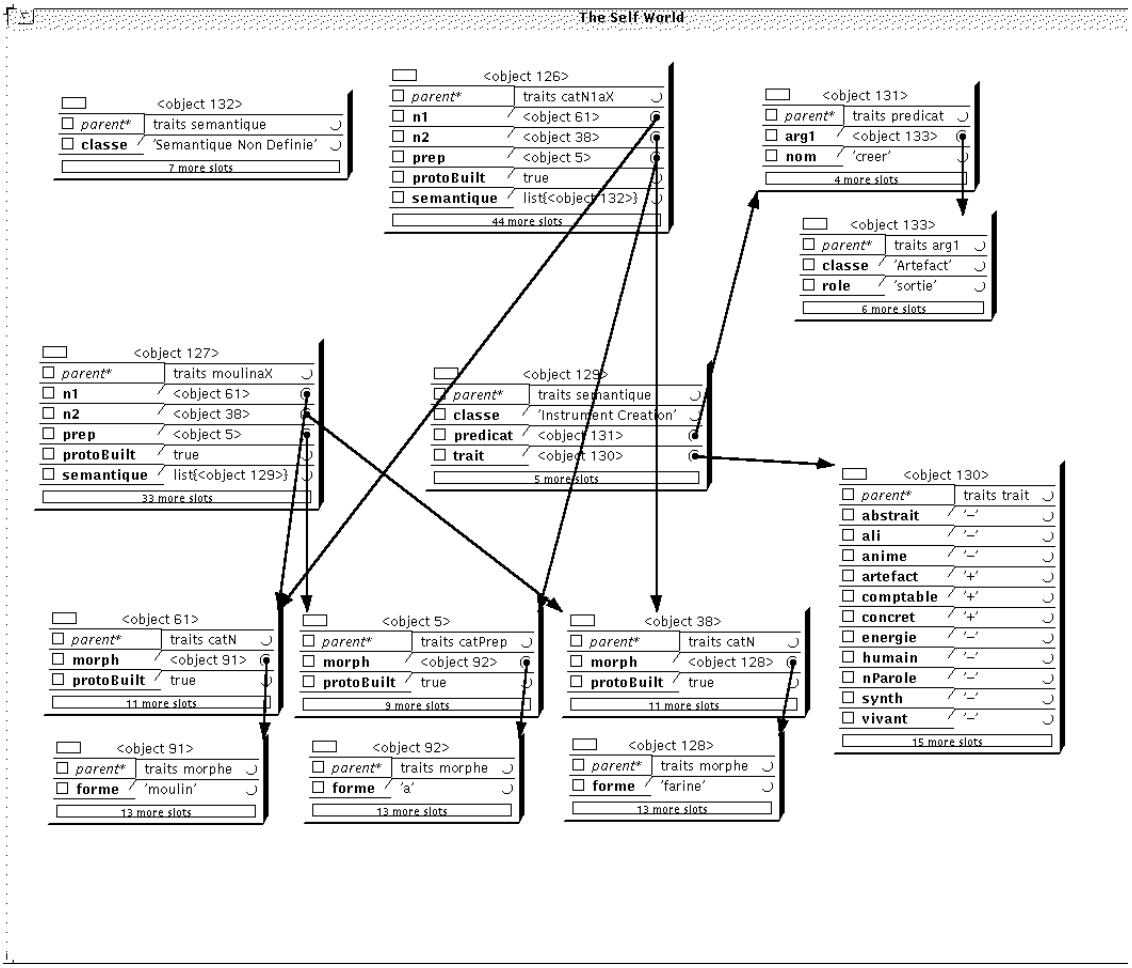


Figure 6.14 : Un réseau de prototypes pour `moulin à farine`

La figure précédente présente les objets issus d'une analyse de la séquence "`moulin à farine`". Le prototype `<object 126>` représente le prototype `N1 à N2` construit à partir de `moulin` (à partir du prototype `<object 61>` représentant `moulin`), ce prototype a pour composants les objets `<object 61>`, `<object 38>`, `<object 5>` (représentant respectivement `moulin`, `farine`, `a`) portés par les attributs `n1`, `n2`, `prep`. Le prototype `<object 127>` représente un prototype `moulin à N2` représentant `moulin à blé` construit à partir du précédent et avec les mêmes composants : la sémantique de `<object 127>` est portée par `<object 129>` dont les attributs portent les valeurs associées à cette sémantique via les objets `<object 130>`, `<object 131>`, `<object 133>`.

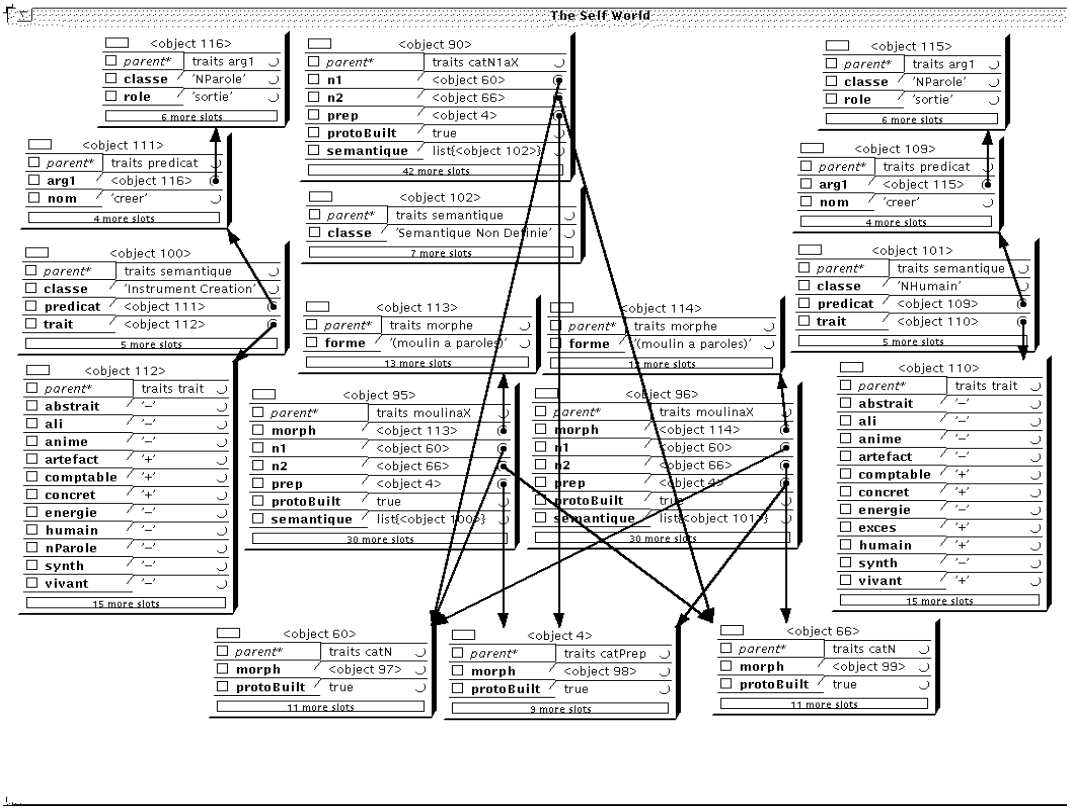


Figure 6.15 : Un réseau de prototypes pour `moulin à paroles`

La figure précédente présente les objets issus d'une analyse de la séquence "`moulin à paroles`". A la différence des deux figures précédentes, l'analyse produit ici deux prototypes `moulin à N2` porteurs de deux sémantiques différentes : `<object 100>` décrit un `instrument de création`, <object 100> décrit un `NHumain`.

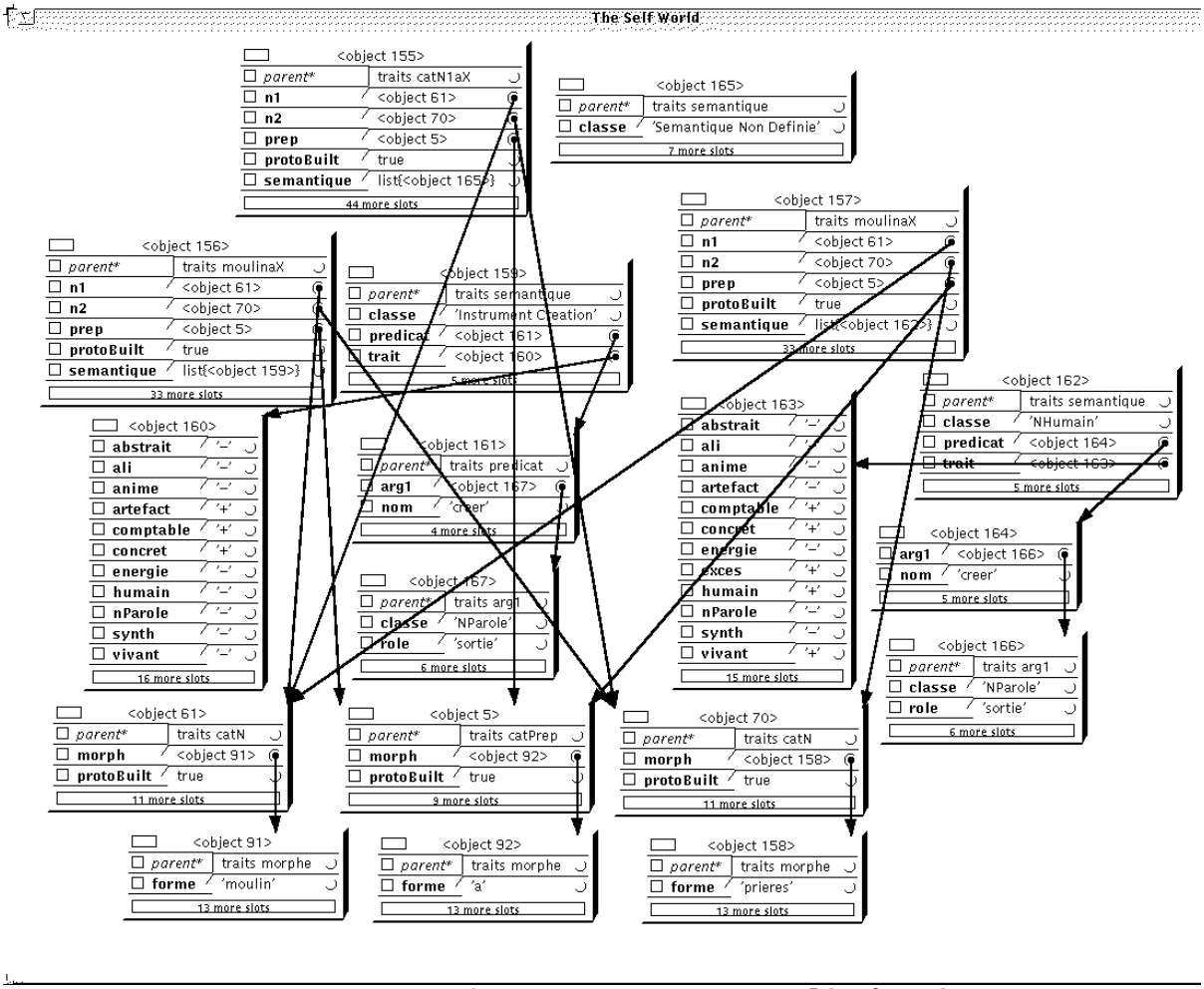


Figure 6.16 : Un réseau de prototypes pour `moulin à prieres`

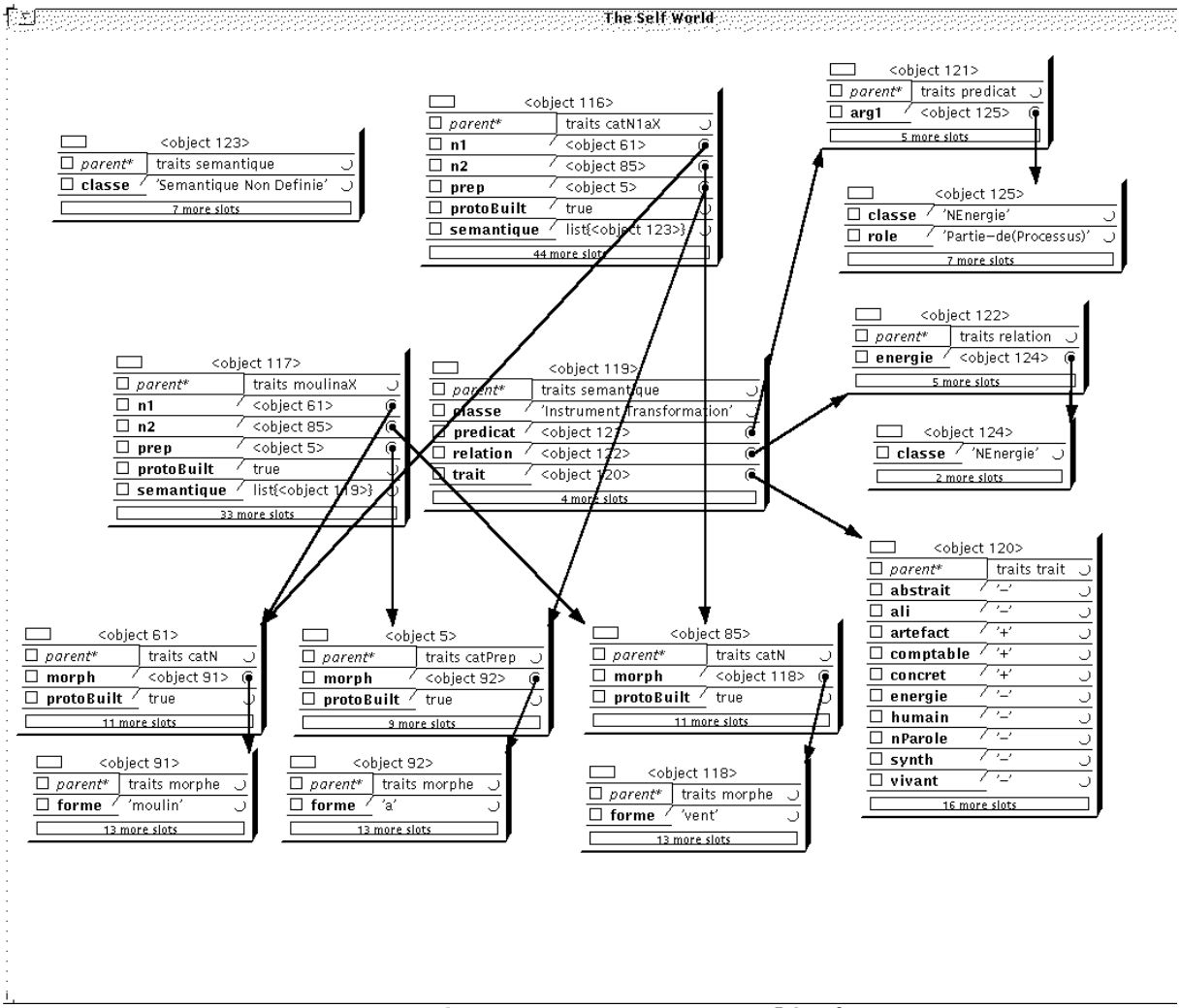


Figure 6.17 : Un réseau de prototypes pour `moulin à vent`

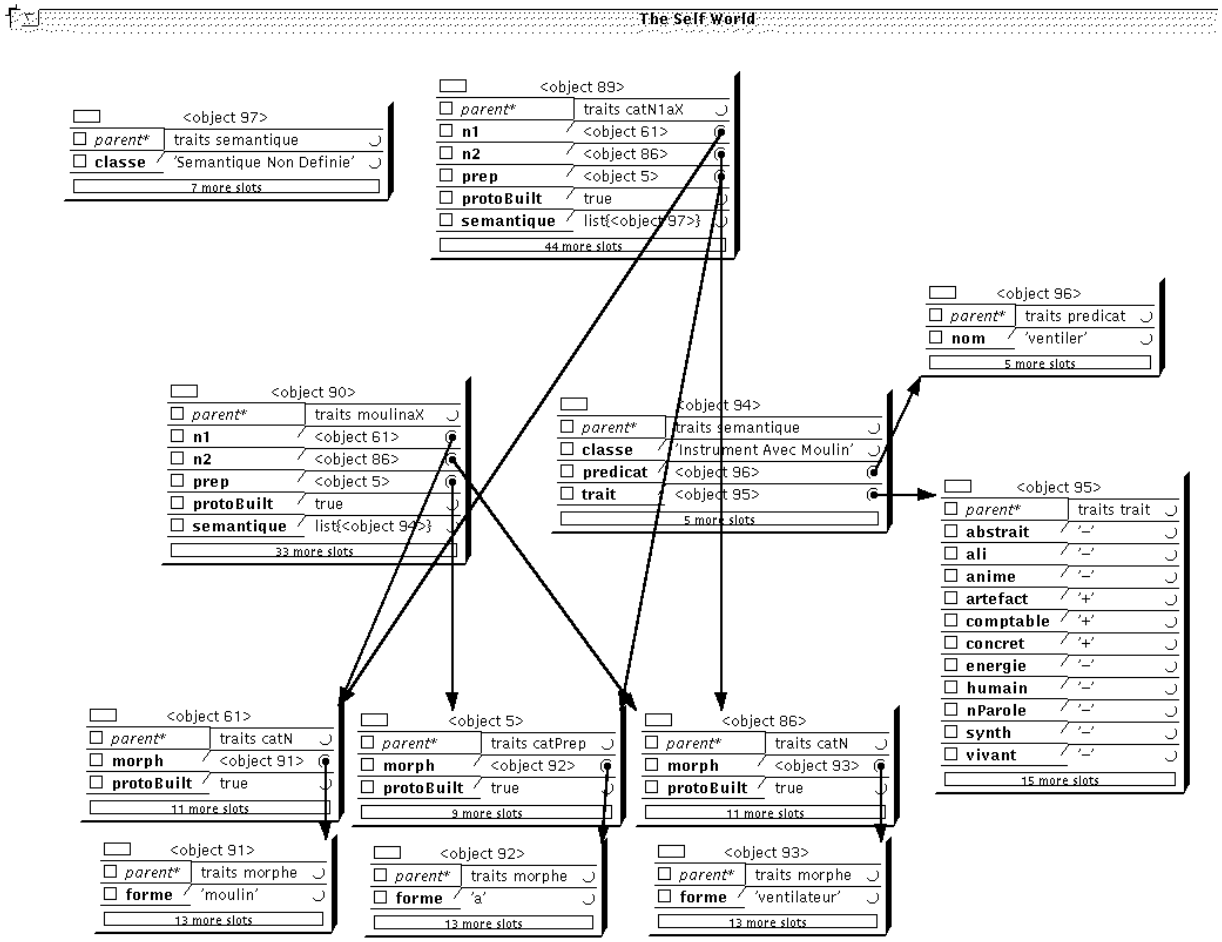


Figure 6.18 : Un réseau de prototypes pour `moulin à ventilateur`

On donne aussi une illustration de l'analyse de la phrase "`Le manager analyse un moulin à paroles`". Le résultat de l'analyse présenté pour cette phrase dans la figure 6.19 propose la construction de quatre prototypes disctincts couvrant la phrase initiale; en effet l'analyse du syntagme verbal n'impose pas encore de contraintes sur la valeur sémantique du composé qui est adjoint au syntagme nominal composant ce groupe verbal. Comme notre analyse propose de ne pas restreindre les interprétations des composés (on construit tous les possibles, tout au moins on essaie), chacun de ces composés construits permet de poursuivre une analyse particulière. Les prototypes `P` obtenus se distinguent donc par la valeur sémantique attachée au prototype associé à la séquence `moulin à N2` présente dans la phrase initiale.

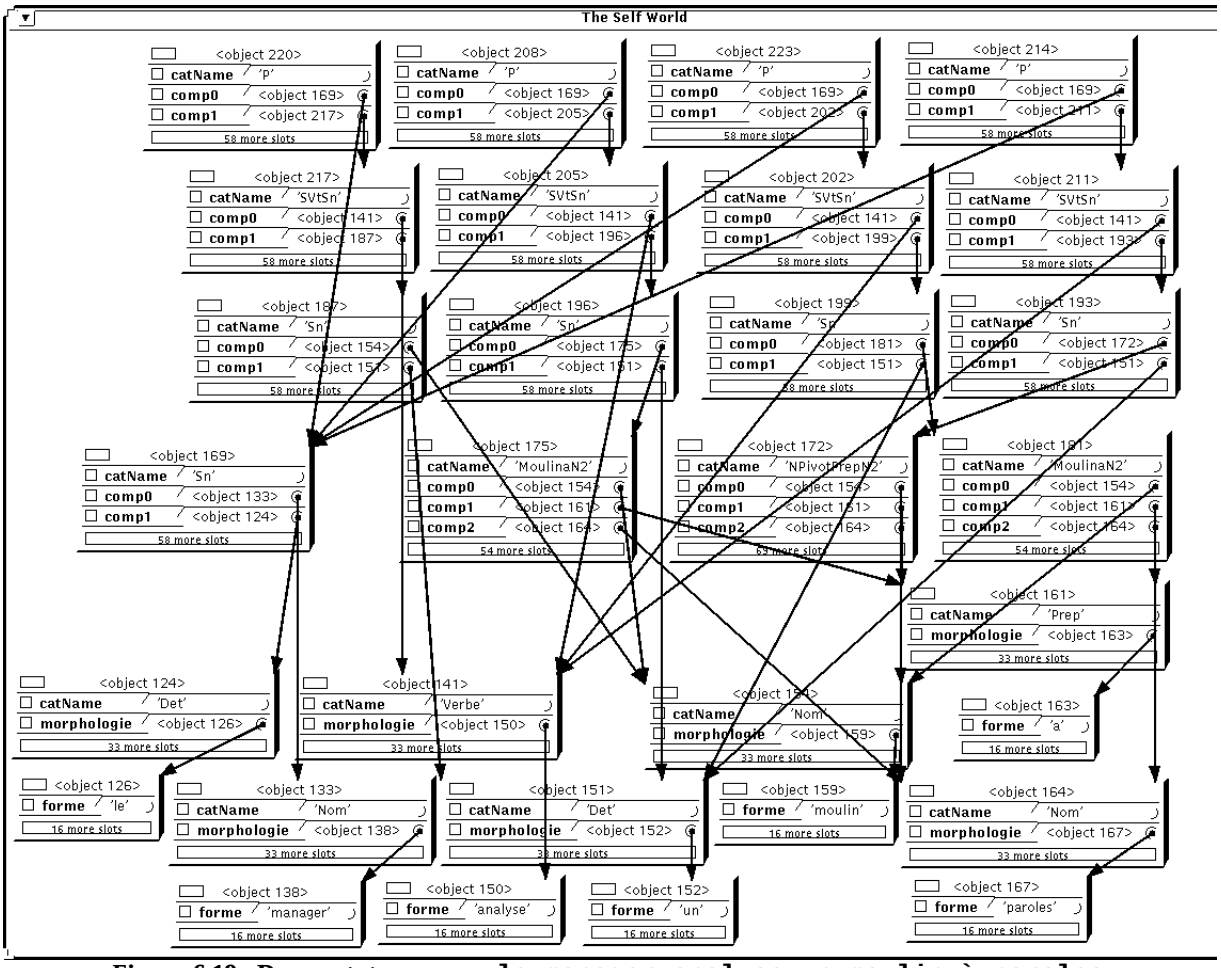


Figure 6.19 : Des prototypes pour `le manager analyse un moulin à paroles`

6.3. Traitement Automatique du Langage Naturel et Prototypes

6.3.1. Représentation prototypique et stabilisation des savoirs

L'implémentation présentée supra propose une représentation de tous les éléments définis pour l'analyse sous la forme de prototypes. En particulier la sémantique d'une entrée lexicale est associée à un objet prototypique. La valeur sémantique attachée à cet objet est initialement définie de manière sous-déterminée : il convient ensuite de construire ou de définir les opérations qui permettent d'affiner cette valeur sémantique initiale. Dans notre approche, il s'agit donc de partir de représentations individuelles peu précises puis de fournir, dans un premier temps, les moyens pour affiner ces représentations. Ce travail s'appuie donc au départ sur des représentations conceptuelles individuelles : les objets prototypiques définis sont associés à des entrées lexicales données. Il semble raisonnable, dans un deuxième temps, de vouloir mettre un peu d'ordre dans les représentations adoptées ou tout au moins de vérifier s'il est possible de dégager des régularités dans les représentations ou dans les processus définis. On peut ainsi vouloir passer de représentations conceptuelles individuelles à des représentations conceptuelles plus générales. Peut-on construire, par induction, des représentations générales qui synthétisent les informations stables que l'analyse des séquences étudiées a permis de révéler, tout en conservant la possibilité de revenir sur cette stabilisation provisoire si des informations supplémentaires mettent en lumière des ajustements à opérer ?

La représentation par prototypes force à concevoir la représentation comme un processus perpétuellement à ajuster. La représentation pour le TALN doit s'attacher à proposer des modèles qui laissent ouvertes les représentations définies, dans la mesure où ce qui est représenté n'est jamais figé. Une représentation sous la forme de prototypes pour la mise en place d'une analyse automatique semble cohérente avec cette nécessité de représentation évolutive pour le TALN : les prototypes peuvent évoluer dynamiquement, il reste à évaluer les moyens qui réalisent cette évolution dynamique dans un analyse automatique du langage naturel.

6.3.1.1. Du méta-regard sur l'analyse à l'automodification des représentations

Un premier problème à résoudre est donc celui de la réorganisation des représentations initiales si l'analyse sur notre domaine restreint révèle des failles et donc des ajustements à opérer.

Un second problème à résoudre peut se poser de la manière suivante : tel type de comportement sémantique attaché à `moulin` se révèle être un comportement sémantique plus général pour une classe de noms plus large. Est-il possible de réorganiser cette part de savoir plus général? Comment effectuer cette réorganisation du savoir? Quel est l'impact d'une telle réorganisation sur les savoirs représentés? Il conviendrait pour cela de disposer de savoirs capables de détecter que tel comportement attaché à `moulin` se retrouve sous une forme assez proche dans la représentation sémantique attachée à un autre nom d'instrument. Puis d'étendre ce rapprochement de comportements partagés à un ensemble d'entités représentées. La difficulté ne s'arrête évidemment pas là, il s'agit ensuite de définir les modalités d'une réorganisation des savoirs partagés. La question de savoir si un tel aménagement est souhaitable n'est d'ailleurs pas à écarter : ne risque-t-on pas de définir des regroupements aléatoires dont la pertinence n'est pas justifiée? Il peut en effet y avoir des recoupements entre les comportements sémantiques de plusieurs classes de noms, il faut pouvoir y retrouver une cohérence à moins d'y perdre toute volonté de classer les choses "raisonnablement".

6.3.1.2. Comment définir des "démons" capables de redéfinir les comportements généraux à factoriser

Pour reprendre ce que nous disions au départ sur la (re)construction d'une représentation sémantique que l'on peut attacher à moulin, il s'agit aussi d'évaluer comment il serait possible d'étendre cette représentation sémantique aux noms d'instruments : comment mettre en oeuvre une catégorisation plus générale tout en préservant une évolution potentielle de cette représentation initiale? La représentation sémantique proposée dans la figure 2.25 tend à identifier (à isoler) les comportements sémantiques que l'on peut attacher à moulin et qui permettent de construire des (débuts d') interprétations qu'il reste possible d'affiner. Si ces comportements traduisent des opérations sémantiques plus générales qu'il est possible de factoriser pour une classe de noms particuliers (en particulier pour les noms d'instrument), il convient de restructurer l'organisation des représentations sémantiques attachées aux entités représentées pour disposer d'une représentation ouverte et cohérente du domaine visé. Si une approche prototypique se révèle adéquate pour rendre compte d'un micro-phénomène particulier, il convient de ne pas perdre de vue la place de ce micro-phénomène dans un système plus complexe, ou tout au moins de dégager de l'étude particulière de celui-ci, les éléments qui le dépassent. Si le TALN a du mal à classer l'ensemble des savoirs manipulés et leurs interactions, il convient de réfléchir à des processus qui organisent les savoirs et leurs interactions sur des domaines restreints en procédant par couches successives : dégager les strates "stables" et les comportements réguliers quitte à revenir sur ces choix si des éléments nouveaux viennent les mettre en doute.

6.3.2. Quels outils pour une évolution dynamique des représentations?

Le problème de toute forme de représentation pour un système de TALN se heurte à la difficulté de construire des méta-connaissances. Si l'on veut faire évoluer des objets représentés de manière sous-déterminée, il est possible d'ajuster le code de manière à répondre à ces évolutions de représentations. Une solution provisoire présentée précédemment redonnait la main à l'utilisateur pour qu'il modifie certaines valeurs sémantiques des entrées rencontrées si celles-ci ne permettaient pas de construire un composé en moulin à N2 satisfaisant pour le modèle défini. Ces ajustements restaient malgré tout limités aux représentations pré-définies : il était possible de modifier les valeurs des attributs des objets représentés mais ces ajustements ne permettaient pas de modifier les représentations construites. On peut ainsi vouloir que les représentations initialement construites puissent évoluer dynamiquement sans intervention directe de l'implémenteur dans les boites noires définies , ou sans intervention de l'utilisateur. Cette deuxième perspective consistait en fait à "forcer" une représentation pour qu'elle s'accorde avec un modèle préconstruit. Il ne s'agit donc pas de considérer le modèle initial comme un ensemble figé de représentations et de processus mais de permettre d'ajuster dynamiquement ce modèle (dans ses représentations et dans les processus qu'il définit).

L'exemple de la construction du sens des séquences en moulin à N2 illustre cette problématique. Dans la construction sémantique présentée supra, on part d'une représentation sémantique sous-déterminée pour une séquence en moulin à N2 qui peut être ajustée suivant la nature des composants de la séquence visée. Cet ajustement est malheureusement prédéterminé par le processus défini pour le calcul sémantique des composés étudiés. Le calcul sémantique permet d'ajuster les représentations et les valeurs attachées aux champs définis dans la limite fixée par ce calcul. Le corpus examiné a permis de construire un classement provisoire de séquences rencontrées. On a donc pu définir un filtrage dans l'implémentation du calcul sémantique défini qui s'accorde avec ce classement. Pour chaque séquence rencontrée dans notre corpus, l'analyse va produire au moins la réponse attendue. Il reste donc à définir les moyens pour ajuster les représentations et les processus de calcul pour éventuellement traiter des séquences qui n'entrent pas dans le modèle sous-jacent aux filtres sémantiques construits.

Le modèle pour la construction du sens esquissé ici construit par défaut une interprétation sous déterminée. Si une séquence donnée ne vérifie pas les filtres qui permettent de préciser cette interprétation "prototypique", c'est la solution "minimale" qui sera proposée. A partir de là, on peut vouloir :

- 1. Soit ajuster la solution fournie, c'est-à-dire disposer de règles sémantiques (de processus informatiques) qui permettent ces ajustements (qui les réalisent).

- 2. Soit modifier le modèle défini pour prendre en compte cette nouvelle réalisation d'autant plus si cette solution se révèle pertinente pour d'autres séquences. Cette approche revient à établir un nouveau classement au sein de notre modèle.

Le premier point semble délicat à envisager d'un point de vue dynamique. Il semble en effet difficile de définir, puis d'utiliser une nouvelle règle sans passer, au préalable, par une analyse linguistique. A moins de disposer d'une somme considérable de méta-connaissances sur la langue, il ne semble pas possible de pouvoir mettre à jour dynamiquement cette base de savoirs sémantiques représentées par les règles définies. Il semble plus raisonnable de travailler sur un modèle qui accepte de manière peu coûteuse des ajustements dans la base de savoirs qu'il manipule.

6.4. De la linguistique à l'informatique : retour sur un examen des besoins propres à la représentation des connaissances pour le TALN

6.4.1. Paradigmes courants en TALN et environnements informatiques

En examinant quelques uns des posés et présupposés de la sémantique différentielle et ceux couramment mis en oeuvre dans les travaux actuels en TALN, on mesure les tensions qui peuvent exister entre ces deux approches.

6.4.1.1. Posés et présupposés dans les travaux courants actuels en TALN

Une large famille de traitements automatiques actuels se sont développés sur la base des formalismes d'unification. Les présupposés de ces formalismes sont nombreux :

- 1. La détermination du local sur le global.

Le principe de projection lexicale à partir des têtes lexicales est central. Il peut mener jusqu'à des grammaires entièrement lexicalisées dans lesquelles des règles non lexicalisées sont difficilement pensables. Dans les formalismes d'unification les traits sont utilisés de facon essentialiste, à la manière de primitives sémantiques. Les entrées lexicales sont créées indépendamment de considérations de genres et de domaines, ce qui conduit en général à des traitements homonymiques, sans prendre en compte le fait qu'une partie des ambiguïtés est artificielle, dans la mesure où elle naît de la réunion de significations qui ne se réalisent pas dans les mêmes textes.

- 2. La primauté en fait, voire en droit, de la syntaxe.

- 3. L'horizon indépassé de la phrase.

- 4. La visée compositionnelle et la construction en parallèle de représentations syntaxiques et sémantiques.

Les formalismes d'unification se veulent monotones. L'usage des défauts est reservé a la représentation du lexique, mais n'intervient pas au cours de l'analyse.

- 5. La langue et non les textes.

6.4.1.2. Représentation par Objets et TALN : la PàO et la PàP

Les travaux menés en TALN ont longtemps privilégié les modèles de représentation fondées sur les taxonomies de classes. Les domaines de savoir où la catégorisation est mouvante (non figée) ne peuvent malheureusement pas se contenter de mode apriorique de représentation (les taxonomies de classes). La construction du sens met en avant la nécessité de disposer de mode de représentation inductif (à l'image des prototypes) capable de faire évoluer les structures de représentation construites.

6.4.2. La sémantique différentielle bouscule les paradigmes courants du TALN : problèmes théoriques et implications pour une mise en oeuvre informatique

6.4.2.1. Posés et présupposés de la sémantique différentielle

Dans l'approche sémantique suivie par Rastier, c'est le global (i.e. les textes) qui détermine le local. Un mot ne se définit pas par rapport à des états de chose ou à des états mentaux mais par rapport à ses contextes : un mot (occurrence) ne se définit que dans et par un contexte et reçoit des déterminations par le texte. Le sens d'un mot ne lui est donc pas permanent, il est toujours le produit d'une interprétation fondée ou non sur des inférences par défaut qui héritent ou non de la sémantique d'un type lexical préconstruit et si le sens résulte d'une interprétation, il convient pour en rendre compte de décrire au mieux les parcours interprétatifs et les contraintes linguistiques sur ces parcours. De fait, placer les mots sous l'autorité d'un type revient à les décontextualiser et à les détextualiser et le rapport du type à l'occurrence s'éclaire si l'on considère que les types ne préexistent pas aux occurrences mais sont reconstruits à partir d'elles (Rastier 1995).

La sémantique différentielle met au centre de sa démarche les textes (et non le signe) et les genres, discours et pratiques sociales qui les régissent. Dans cette démarche les significations ne sont pas définies par référence à des choses, mais par des relations au sein de classes lexicales dans un corpus donné, la définition compositionnelle du sens, tout comme la notion d'un sens premier, modifié par le contexte, s'avèrent invalides. En fait c'est le contexte global (le genre du texte, ses conditions de production) qui conduit à l'interprétation des signes : *"l'interprétation ne s'appuie pas sur des signes déjà donnés, elle reconstitue les signes en identifiant leurs signifiants et en leur associant des signifiés. L'identification des signes comme tels résulte donc de parcours interprétatifs"* (Rastier in (Rastier & al. 1994)). Ces parcours interprétatifs remanient dynamiquement les ensembles de sèmes associés aux données textuelles, actualisant certains, virtualisant d'autres. Cette approche renverse la conception la plus courante de la référence : *"Alors que la tradition métaphysique occidentale en philosophie du langage pose que les mots ont une signification parce que les choses ont un Etre /.../, nous concluons à l'inverse que les choses nous paraissent douées de qualités substantielles comme l'identité à soi dès lors qu'elles sont réifiées par les processus de référenciation. /.../ En d'autres termes, les "objets" nous sont acquis au sein de pratiques, et non par une saillance physique qui nous les imposerait; la référence va donc des mots aux choses, isolées et specifiées par leur sémiotisation, qui est un processus hautement culturel"* (Rastier in (Rastier & al. 1994)).

Nous reprenons ci-dessous les points principaux qui fondent la problématique de la sémantique différentielle et qui, en regard de ce qui a été dit sur les supposés des formalismes d'unification, éclairent les écarts théoriques entre les deux approches.

- 1. Primauté du global sur le local.

Dans cette approche, il faut *"admettre l'incidence déterminante du palier textuel sur les paliers inférieurs"* (Rastier in (Rastier & al. 1994)). En simplifiant on peut dire que l'ambiguïté dans une situation de communication donnée n'existe pas : il n'y a en général que de la polysémie artificielle, la mise en contexte des signes linguistiques lève toute ambiguïté potentiellemment active sur les signes hors contexte. Dans le cadre des prototypes, on peut envisager de mettre en place cette incidence du global sur le local sous la forme de processus qui agissent sur les entités de bas niveau représentées en ajustant les structures de ces entités suivant les nécessités rencontrées.

- 2. Remise en cause de la syntaxe.

"Quand /./ l'ordre syntaxique domine l'ordre herméneutique, l'identification des signes est considérée comme allant de soi : il suffirait de leur appliquer les règles syntaxiques pour procéder ensuite à leur interprétation. Ce n'est pas si simple, car les conditions de l'identification des signes et du choix des règles ne résident pas dans le texte, mais dans la situation où il est produit et interprété" (Rastier in (Rastier & al. 1994)). De plus, Rastier indique qu'*"en pratique beaucoup d'applications n'exigent pas d'analyse syntaxique préalable"* (Rastier in (Rastier & al. 1994)). Malgré tout, l'absence de précision sur des réalisations, concrètes et de grande ampleur, de systèmes dans une telle approche empêche de savoir si la sémantique différentielle pourrait réellement se passer d'une analyse syntaxique préalable.

- 3. Dépasser les cadres syntaxiques traditionnels.

Les domaines pour la sémantique ne correspondent pas aux constituants de la syntaxe. *"Toute "barrière" syntaxique inhibe les relations sémantiques contextuelles"* (Rastier in (Rastier & al. 1994)). Et même si le syntagme demeure, *"l'intégration du syntagme résulte des relations de concordance et de rection qui s'y établissent. Elle ne doit pas être confondue avec une composition, ni même une combinaison"*. La phrase (pleine de ronces syntaxiques) est remplacée par la période. Cependant, la critique de la conception hiérarchique de la phrase qui oublie les relations diagonales et les attachements multiples doit être précisée.

- 4. Une dynamique de l'interprétation.

"La sémantique différentielle est dynamique/./; au cours de l'interprétation, le contenu lexical est susceptible de diverses tranformations à l'intérieur des sémèmes[79] *eux-mêmes, parallèlement à l'utilisation de ce contenu pour la construction de structures interprétatives comme les isotopies et les molécules sémiques"* (Rastier in (Rastier & al. 1994)). On retrouve avec la sémantique différentielle l'idée que le sens n'est pas une donnée : *"Les signes linguistiques ne sont que le support de l'interprétation, ils n'en sont pas l'objet. Seuls les signifiants, sons ou caractères, sont transmis : tout le reste est à reconstruire"* (Rastier in (Rastier & al. 1994)). Aucune énumération, aucune décomposition des unités qui composent un énoncé ne peut donner une somme de sens correspondante : *"Le principe de compositionnalité est invalide en sémantique linguistique et /./ l'interprétation échappe ainsi de droit au paradigme du calcul /./"* (Rastier in (Rastier & al. 1994)). Chaque texte donne lieu à la construction d'un ensemble de mondes. L'interprétation consiste généralement à décrire les inférences qui permettent de passer de la linéarité du texte à une compréhension plus précise : le découpage du texte, l'inventaire et l'articulation des relations entre mondes jouent un rôle essentiel pour exprimer un des sens possibles du texte visé : *"Le sens d'un texte ne lui est pas immanent, il est immanent à la situation de communication à laquelle appartient le texte. Il varie donc avec les paramètres de cette situation"* (Rastier in (Rastier & al. 1994)). Comme on l'a déjà vu avec `moulin`, les mécanismes à l'oeuvre dans la construction du sens peuvent faire évoluer les savoirs et modifier leurs valeurs : *"Si l'on définit les significations non par référence à des choses, mais pas leurs interrelations au sein de classes lexicales, les significations se modifient quand les classes changent : ainsi la définition de 'métro' a changé quand au sein de la classe des transports collectifs urbains est apparu 'RER'; et cependant le métro lui-même ne changeait pas. En outre, les discours remanient sans cesse les classes lexicales, qui ne sont fixées que par des normes révisables; et par là il modifie la signification des mots"* (Rastier in (Rastier & al. 1994)).

Représentation du lexique.
Si la langue est une matière organique, si ces composants évoluent individuellement ou dans leurs interrelations, il est illusoire de tenir leur classement ou leur représentation comme définitifs et figés :*"Le lexique d'une langue n'est pas une nomenclature qui quadrillerait uniformément une réalité supposée. En outre, il ne se laisse pas organiser en taxonomies arborescentes (sauf précisément les parties du lexique structurées par de disciplines comme la zoologie)"* (Rastier in (Rastier & al. 1994)). Les savoirs que l'on peut associer aux mots peuvent être remis en question, le sens des mots peut être altéré ou modifié dans des situations particulières. De plus les savoirs établis à un moment donné peuvent évoluer et être étendus quand de nouvelles activités, de nouveaux comportements, de nouveaux outils... sont créés.

- 5. Les textes et non la langue.

La sémantique différentielle met en avant le travail sur le rôle des textes comme objet de la linguistique (en lieu et place des langues), comme points de départ des parcours interprétatifs; elle met aussi en avant la médiation des genres. On est donc très loin de la tradition courante du TALN qui consiste à travailler sur la grammaire et le lexique d'une langue donnée : "*Une langue n'est pas composée d'un et d'un seul système, mais de plusieurs instances régulatrices en interaction constante, et qui évoluent selon des temporalités différenciées"* (Rastier in (Rastier & al. 1994)).

---

[79] *"Un sémème est un ensemble structuré de traits pertinents, les sèmes"*(Rastier 1994).

6.4.2.2. Quels outils informatiques pour la sémantique différentielle ?

"Les théories linguistiques qui ont la meilleure capacité descriptive ne sont pas nécessairement celles qui se prêtent le mieux à la mise en oeuvre informatique, dont les conditions sont au demeurant contingentes (état de l'art, habitudes de pensée, etc..)" ((Rastier in (Rastier & al. 1994))).

Quels devraient être les environnements informatiques[80] dont il faudrait disposer pour la sémantique différentielle? La place du travail sur corpus suppose de préciser les outils d'exploration et de structuration de ces corpus qui s'avèrent nécessaires. Quels sont d'ailleurs les paradigmes informatiques qui permettent de rendre compte clairement de phénomènes tels que :

- L'articulation du quantitatif et du symbolique.
- La négociation entre informations de différentes natures et de différents niveaux.
- L'organisation hiérarchique / hétérarchique des mécanismes d'analyse à employer.
- La place de la non-monotonie (le poids du global peut amener à virtualiser certains traits inhérents).

6.4.3. Prototypie (au sens de la PàP) et sémantique différentielle

On rappelle que notre travail vise à interroger les formalismes informatiques disponibles (en particulier ceux qui tournent autour de la représentation par Objet) afin de déterminer si ces formalismes répondent clairement aux exigences impératives que nos choix dans les traditions linguistiques imposent pour la construction du sens : quitte à envisager une articulation de ces formalismes si ceux-ci se révèlent cohérents pour résoudre certains phénomènes particuliers (on peut par exemple organiser les savoirs stabilisés dans un mode de représentation hiérarchique (via une approche `Objet (Instance de Classe)`), et les savoirs flous dans un mode de représentation inductif (via une approche `Objet (Prototype)`).

La représentation pour le TALN doit s'attacher à proposer des modèles qui laissent ouvertes les représentations définies, dans la mesure où ce qui est représenté n'est jamais figé. Une représentation sous la forme de prototypes pour la mise en place d'une analyse automatique semble cohérente avec cette nécessité de représentation évolutive pour le TALN : si les prototypes peuvent évoluer dynamiquement, la notion d'""évolution dynamique" propre aux prototypes s'accorde-t-elle avec la dynamique de l'interprétation que la sémantique différentielle met en avant?

(In)Adéquation entre le paradigme "Représentation par Prototype" et les besoins propres à la sémantique différentielle ?

Nous limiterons nos remarques aux points qui cristallisent les tensions exprimées précédemment.

- 1. Sur notre domaine restreint des séquences en `moulin à N2`, c'est la syntaxe qui guide la construction sémantique des composés.
- 2. Les représentations lexico-sémantiques adoptées restent encore peu précises tout en maintenant une définition sémantique essentialiste à la manière des formalismes d'unification. Sur ce point, il convient de repréciser que le développement de notre travail est encore loin des exigences que nous souhaitons mettre en oeuvre et notamment dans la représentation lexico-sémantique des éléments manipulés. Notre travail vise à la mise en oeuvre de processus dynamiques qui agissent au cours de la construction du sens

---

80 *"Seules certaines approches connexionnistes se sont jusqu'ici révélées capables de reproduire un certain déterminisme global/local au cours du fonctionnement du système"* (Rastier in (Rastier & al. 1994)).

sur les représentations définies : ce travail vise donc à considérer l'interprétation comme une reconstruction du sens à partir des éléments présents dans la séquence linguistique donnée en tenant compte des conditions de production de cette séquence; il ne s'agit pas donc pas de réaliser un simple calcul qui combine les valeurs sémantiques individuelles pour produire un résultat global.

- 3. L'état actuel de notre travail sur la construction du sens des séquences en `moulin à N2` peut être assimilée à une vision compositionnelle de la construction du sens. Ce sont les valeurs locales des composants qui déterminent les valeurs globables des composés. Nous avons déjà mentionné supra que nous envisagions la nécessité de pouvoir fournir des informations qui dépassent le cadre du composé et qui permettent d'ajuster la construction sémantique visée si nécessaire; soit en récupérant des contraintes sémantiques liées au développement de l'analyse, soit en tenant compte des contraintes générales que les textes (contenant ce type de séquences) révèlent (genre ou domaine d'un texte pour reprendre les termes de Rastier) : ce travail reste à faire. Dans le cadre particulier étudié ici, il semble possible de faire coïncider en parallèle une détermination du local sur le global et une détermination du global sur le local. L'absence (provisoire) de telles contraintes globales sur la production des séquences analysées force pour le moment l'analyse à donner plus de poids aux informations sémantiques locales (en particulier celles attachées au `N2` dans le composé) et donc à orienter l'analyse vers une version compositionnelle : il ne s'agit en tout cas pas d'un choix théorique. Encore une fois nous souhaitons à terme pouvoir confronter les informations sémantiques disponibles, ainsi que contraindre les opérations que le global impose sur le local et réciproquement. Cette dernière exigence pose d'ailleurs le problème de savoir de quelle manière représenter les contraintes globales sur un texte dans un formalisme qui manipule des prototypes. Cette question reste ouverte.

- 4. Une première approche pour mettre en oeuvre des contraintes globales sur la construction du sens des séquences en `moulin à N2` passe à coup sûr par une étude précise des conditions de réalisation des séquences visées. Et donc à un retour sur une analyse plus précise de notre corpus : en particulier pour y mettre en lumière les éléments sémantiques (extérieures à la séquence) qui agissent sur l'interprétation des séquences en `moulin à N2`. Là encore ce sont les besoins linguistiques qui doivent interroger les outils disponibles pour déterminer si ces derniers peuvent répondre clairement à ces besoins : si des contraintes linguistiques globales permettent de résoudre l'interprétation de certaines séquences, il reste à définir les mécanismes qui permettent d'ajuster les structures de représentation construites pour rendre compte de ce genre de phénomène.

- 5. De la même manière que Rastier définit les significations par les interrelations au sein de classes lexicales, la notion de prototype, telle que nous l'utilisons, permet de définir les significations des prototypes manipulés par un jeu de relations qu'il est possible de construire sur les comportements que l'on peut attacher à ces prototypes : les prototypes sont ce qu'ils font i.e. ils se différencient par leurs comportements. On peut ainsi concevoir l'interprétation comme l'activation potentielle de comportements particuliers : ces comportements sémantiques n'étant pas le reflet d'une norme sémantique, mais un état particulier qu'il reste possible d'ajuster, d'ordonner, de réifier... suivant les conditions rencontrées.

- 6. La notion de comportement semble rejoindre la volonté de Rastier de mettre en avant le travail sur le rôle des textes en lieu et place de la langue. Il semble en effet assez clair de parler de comportements contextuels particuliers lorsqu'on parle d'unités de langue alors que vouloir définir un comportement général pour les mêmes unités peut vite devenir une illusion.

6.5. Conclusion et perspectives

- Dans cette première partie de notre travail, nous avons étudié des séquences nominales du type moulin à N2 dont l'interprétation est principalement déterminée par le local (i.e. les composants de ces séquences) : ce sont les valeurs locales des composants qui déterminent les valeurs globables des composés étudiés. Cependant dans le cadre particulier étudié ici, il semble possible de faire converger une détermination du local sur le global et une détermination du global sur le local. L'absence (provisoire) de telles contraintes globales sur la production des séquences analysées force pour le moment l'analyse à donner plus de poids aux informations sémantiques locales (en particulier celles attachées au N2 dans le composé) et donc à orienter l'analyse dans une version compositionnelle. Les savoirs lexicaux représentés étant ajustables dynamiquement, il reste possible d'affiner ces interprétations si des processus décrivant le rôle des textes sur ce type de séquence sont disponibles : soit en récupérant des contraintes sémantiques liées au développement de l'analyse, soit en tenant compte des contraintes générales disponibles ou à venir. Ce dernier point n'est pas traité dans cette première phase de notre travail mais la question reste ouverte. Nous souhaitons à terme pouvoir confronter les informations sémantiques disponibles en tenant compte en particulier des opérations que le global impose sur le local et réciproquement.

- Par la suite, on travaille sur des savoirs extraits à partir de corpus. Notre travail va se poursuivre en intégrant cette nécessité de travailler sur des réalisations textuelles pour y dégager les comportements particuliers à l'oeuvre. Les savoirs linguistiques prennent leurs réelles dimensions dans leur mise en action au sein de discours. Leur représentation passe donc par un examen plus précis de leurs réalisations.

Chapitre 7

"Cet écran où mon regard bute, tout en persistant à y voir de l'air, ne serait-ce pas plutôt l'enceinte d'une densité de plombagine ? Pour tirer cette question au clair j'aurais besoin d'un bâton ainsi que des moyens de m'en servir (...). Mais l'époque des bâtons est révolue."
"L'innommable", Samuel Beckett, Ed. de Minuit.

Des arbres aux prototypes : vers une classification dynamique

7.1. Acquisition de connaissances à partir d'un corpus

"Le point de départ de toute description est la reconstruction des classes sémantiques au cours de la lecture du corpus."
(Rastier 1994)

Nous nous plaçons ici dans le cadre de travail mis en oeuvre par (Habert & Nazarenko 1996). Dans ce cadre, il s'agit de simplifier des groupes nominaux complexes fournis par un outil d'extraction terminologique, LEXTER (Bourigault 1993), pour obtenir les arbres élémentaires montrant les relations syntaxiques fondamentales d'un corpus. Ce processus d'acquisition de connaissances est testé sur un corpus médical. Les savoirs extraits sont des arbres d'analyse fournis par un outil d'extraction terminologique (Lexter[81], Bourigault 1993), ces arbres étant ensuite simplifiés dans le but de déterminer les arbres de dépendance élémentaire qu'il est possible d'associer aux entrées lexicales (Cyclade[82], Habert & Nazarenko 1996). Ce travail vise à établir les fonctionnements d'une entrée lexicale dans le corpus, et en particulier à mettre en avant ce que cette entrée peut tisser comme liens avec les différentes entrées lexicales présentes dans le corpus. Il vise aussi à établir des classes de comportements syntactico-sémantiques que l'analyse du corpus peut révéler. Notre travail s'inscrit dans une reprise de l'approche harrissienne (Sager & al. 1987, Habert & Nazarenko 1996) : en fait la détermination de classes d'opérateurs et d'opérandes par le fonctionnement linguistique. Notre travail s'articule autour de trois éléments : un problème, un corpus et un outil informatique.

7.1.1. Un Problème

Le problème à résoudre consiste à définir les fonctionnements lexicaux qu'il est possible d'attacher aux mots dans le cadre d'un analyseur à mots. Il s'agit d'une part de déterminer ces comportements lexicaux et d'autre

[81] Lexter est un outil d'extraction terminologique conçu et réalisé dans un environnement industriel: la Direction des Etudes et Recherches de EDF, pour aider à la mise au point de terminologie. Il prend en entrée des textes arbitrairement longs et produit des arbres d'analyses de séquences nominales en décomposant ces séquences en Tête et Expansion, de manière récursive.

[82] Le programme Cyclade (Habert & al. 1995) a pour tâche de simplifier les arbres d'analyse fournis par un analyse, ici par Lexter, et de mettre en évidence les arbres qui présentent des relations de dépendance entre mots pleins (vs. mots vides: prépositions, déterminants, conjonctions...).

part de les représenter. Notre démarche vise à mettre en lumière dans des domaines restreints les comportements des mots ainsi que les corrélations multiples qui existent entre les mots dans un flot continu de discours. Elle consiste en quelque sorte à "faire-émerger" ces comportements des mots puis à les représenter ou affiner les représentations existantes. L'hypothèse suivie ici étant qu'il y a peu de sens à vouloir faire de l'acquisition sémantique en dehors d'un sous-langage.

7.1.2. Un Corpus

La langue présente trop de variations pour pouvoir être appréhendée globalement. Ces variations se retrouvent le plus souvent au niveau des mots où elles sont plus directement accessibles. Notre travail de représentation et de classement s'appuie sur une recherche initiale au niveau des mots des régularités et des redondances d'utilisation dans des corpus donnés. Les processus de représentation et de classement commencent leur travail à partir de ces informations en sachant que des ajustements seront toujours possibles sur les résultats construits. Notre tâche vise donc à s'élever du particulier au général et à procéder à des regroupements pour y découvrir les faces cachées derrière les détails. Puisque la hiérarchie est un artifice, utile d'ailleurs pour classer les choses, nous ne nous imposons pas un classement prédéfini des savoirs à représenter, mais nous choisissons une démarche qui tente de reconstruire ce classement en tenant compte des informations délivrées par le travail d'extraction de savoirs à partir de corpus. La phase de représentation utilise des savoirs extraits à partir du corpus constitué dans le cadre du projet MENELAS (Zweigenbaum 1994, Zweigenbaum & al. 1995) pour la compréhension de textes médicaux. Ce corpus est utilisé par le Groupe de Travail Terminologique et Intelligence Artificielle (PRC-GDR Intelligence Artificielle, CNRS). L'unité thématique de ce corpus a trait aux maladies coronariennes. L'apprentissage sur corpus réalisé vise à repérer les arbres de dépendances élémentaires entre mots (relations opérateur-opérande) et les contraintes sur les combinaisons de ces arbres.

7.1.3. Un Outil Informatique

On a déjà vu que la PàP peut être considérée comme un outil de représentation privilégiée pour la représentation de domaines de savoirs pour lesquels on ne dispose pas de catégories bien précises. Si les savoirs à représenter ne sont pas connus de manière définitive, il est possible de commencer le processus de représentation en utilisant les savoirs déjà recensés puis d'affiner dynamiquement les objets construits dès que de nouveaux savoirs sont disponibles. A la limite, cette démarche de représentation n'a pas à se préoccuper de la forme terminale des objets à construire. Elle peut démarrer à partir d'une structure vide qui sera mise à jour dès que des informations sont disponibles. La représentation avec des prototypes n'est pas un processus qui capte globalement les savoirs sur un domaine donné et qui construit une structure définitive pour représenter ces savoirs. Il s'agit au contraire de mettre à jour progressivement tous les savoirs pertinents pour affiner le domaine à représenter. Cette redéfinition continuelle des objets à construire ne ressemble donc en rien à un programme qui serait stocké dans un répertoire d'alternatives potentielles, mais dépend des informations reçues "au fil de l'eau". Cette flexibilité de la PàP rend possible une mise en oeuvre du processus de représentation sur des domaines de connaissances qui peuvent évoluer. La démarche qui va être suivie vise à construire des structures de représentations d'unités lexicales initialement sous-déterminées et qu'il conviendra ensuite d'enrichir.

7.2. Situation effective par rapport au travail de Harris

Une approche à la Harris débouche sur la mise en évidence de classes d'opérateurs et de classes d'arguments, qui sont supposés fortement liées aux notions et relations du domaine considéré : peut-on y voir là une solution pour l'acquisition automatique de connaissances à partir de corpus et à termes pour la construction d'ontologies? Nous verrons infra qu'il convient de rester prudent sur ce point (Habert & al. 1996). Notre travail se situe dans un courant de linguistique expérimentale et utilise une approche symbolique, celle des sous-langages. Ce travail vise à caractériser syntaxiquement et sémantiquement certaines configurations nominales dans un domaine donné. En effet, il est souvent impossible de savoir si une configuration est acceptable ou non si on se place hors du domaine en cause (Habert & al. 1996).

Mise en évidence des comportements des mots d'un corpus pour une automatisation de leur représentation et de leur classement dans un programme de TALN manipulant des prototypes

Notre travail s'inscrit dans une reprise de l'approche harrisienne et vise à automatiser les traitements (de représentation des mots et de leur classement) et à souligner les limites de cette induction de savoirs (un travail d'interprétation manuel semble indispensable).

- 1. Nous limitons notre étude à certaines configurations nominales et plus particulièrement aux séquences du type `NAdj` (Justeson & Katz 1995). Il s'agit en particulier de rechercher au niveau des mots des régularités et des redondances d'utilisation puis d'évaluer les regroupements obtenus (-> partie 2 chapitre 8 : expérimentations sur corpus).

- 2. Notre tâche vise à automatiser les traitements de représentation des mots (sous la forme de prototypes) puis leur classement : ces traitements sont conçus comme des processus évolutifs i.e. les résultats construits peuvent être affinés dès que de nouveaux savoirs permettent d'améliorer la qualité de ces résultats. Ces traitements sont précédés d'une phase d'extraction de savoirs à partir de corpus. La représentation inductive des mots s'appuie sur les résultats d'analyse fournis par les analyseurs Lexter (Bourigault 1993) et Cyclade (Habert 1995). Le but de ces outils est d'une part d'extraire des savoirs à partir de corpus (Lexter) et d'autre part de simplifier ces savoirs puis de caractériser leurs fonctionnements (Cyclade).

Il est important de souligner que les résultats construits ne sont pertinents que pour les corpus étudiés. Les relations syntactico-sémantiques mises en avant par le travail d'extraction de savoirs sont propres aux corpus examinés[83], le travail de représentation et de classement des mots sur la base des savoirs reçus est donc lui aussi complètement dépendant de ces corpus. Si on peut envisager d'élaborer une ontologie à partir des résultats construits (un travail d'interprétation sur les résultats construits reste à définir et à faire), celle-ci ne sera elle aussi pertinente que pour le corpus étudié (Zweigenbaum & al. 1997).

---

83 *"Un corpus de spécialité ne permet pas de repérer toutes les relations, bon nombre d'entre elles font partie de l'implicite partagé par les locuteurs du sous-langage et n'effleurent pas"* (Habert & al. 1995).

7.3. Extraction des fonctionnements lexicaux : LEXTER et ZELLIG

7.3.1. Buts : retrouver les fonctionnements lexicaux

Pour construire une représentation des savoirs lexicaux, on utilise des résultats d'analyse fournis par des outils d'analyse : Lexter et Cyclade. Le but de ces outils est d'une part d'extraire des savoirs à partir de corpus (Lexter) et d'autre part de simplifier ces savoirs puis de caractériser leurs fonctionnements (Zellig).

7.3.2. LEXTER (Bourigault 1993)

Lexter prend en entrée des textes longs et produit des arbres d'analyses de séquences nominales en décomposant ces séquences en Tête (T) et Expansion (E), de manière récursive. Sur la séquence "stenose de le tronc commun gauche", on obtient l'analyse suivante.

[T [T stenose] [E severe]][E de le [T tronc][E commun]][E gauche]

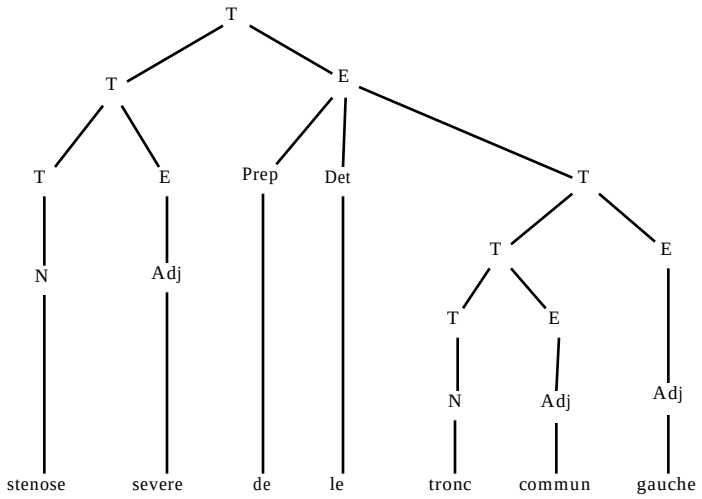


Figure 7.1 : "stenose severe de la tronc commun gauche".

Il est important de souligner que la démarche de Lexter est endogène c'est à dire que le travail d'analyse s'appuie uniquement sur les résultats d'analyse déjà construits pour analyser de nouvelles séquences.

Si le corpus contient la séquence "angine de poitrine instable", on a deux analyses possibles pour cette séquence :

(1) [angine de poitrine] instable
(2) angine de [poitrine instable]

Lexter prend appui sur les séquences déjà analysées pour produire une analyse de cette séquence. Si on a la séquence "angine de poitrine" et que l'on n'a pas "poitrine instable" :

angine de poitrine existe
poitrine instable n'existe pas

Lexter produira l'analyse (1).

7.3.3. ZELLIG (Habert & al. 1996)

7.3.3.1. Buts

- Simplifier les arbres d'analyses fournis ici par LEXTER
- Mettre en évidence les relations élémentaires de dépendance entre mots pleins

7.3.3.2. Simplification d'arbres d'analyse

Le logiciel ZELLIG (Habert & al. 1996) a pour tâche de simplifier les arbres d'analyse fournis ici par Lexter et de mettre en évidence les arbres qui présentent des relations élémentaires de dépendance entre mots pleins. Pour le travail d'extraction des arbres élémentaires, un certain nombre d'arbres abstraits (sans terminaux) sont déclarés comme élémentaires. On crée une table d'arbres de ce type. Sont considérés comme élémentaires les arbres mettant en évidence une relation binaire entre deux mots pleins, nom ou adjectif, dans des schémas comme, par exemple, `N Prep N` ou `N Adj`. Ces dépendances associent à un élément gouverneur (nommé `tête`) soit un `argument` soit un `circonstant`. *"Les dépendances élémentaires ainsi définies n'ont pas forcément de réalisation effective dans le corpus mais ils correspondent à des relations de dépendance vérifiées dans les arbres d'analyse, si l'on passe par une représentation logique de ces arbres et de ces dépendances élémentaires"* (Habert & al. 1996).

Sur notre exemple d'arbre fourni par Lexter, celui-ci est d'abord transformé en un arbre syntagmatique via le transducteur FRT (Habert & Bourigault 1996). Les Symboles non-terminaux sont numérotés pour pouvoir y faire référence.

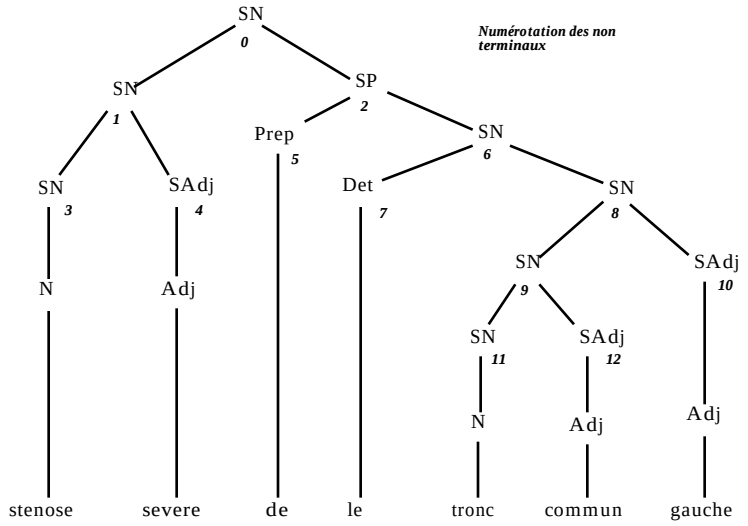


Figure 7.2 : Un arbre syntagmatique.

Le programme Cyclade (Habert & Nazarenko 1996) est ensuite chargé de déterminer les arbres élémentaires via un filtrage de quasi-arbres (Habert & Folch 1996). Il met à jour des arbres élémentaires qui ne sont pas directement présents dans l'arbre de départ. Au total, les dépendances élémentaires mises à jour sont les suivantes :

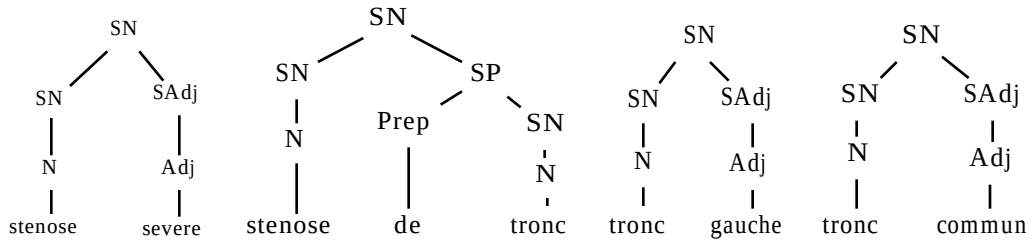


Figure 7.3 : Cyclade : simplifications des arbres d'analyse.

7.3.4. Difficultés pour extraire des arbres "élémentaires"

L'apprentissage à partir de savoirs extraits sur corpus et le travail de caractérisation de ces savoirs posent des problèmes majeurs pour la représentation des fonctionnements lexicaux.

7.3.4.1. Les arbres d'analyse produisent des arbres "élémentaires" non pertinents

Si on dispose dans le corpus des séquences,

`N coronaire droite` et `N coronaire`
(où N n'est pas un nom d'artère)

l'analyse proposée par Lexter est la suivante :

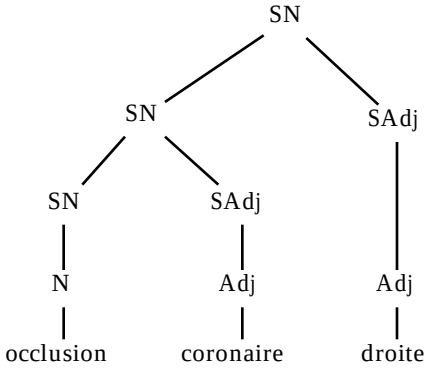


Figure 7.4 : [occlusion coronaire] droite

On obtient donc par élimination du modifieur l'arbre élémentaire :

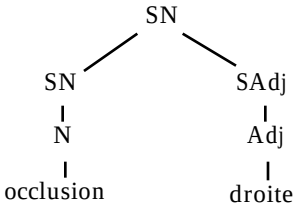


Figure 7.5 : occlusion droite

Or "`droite`" dans la séquence précédente est un modifieur de l'adjectif dénotant une artère. L'analyse à considérer est donc la suivante :

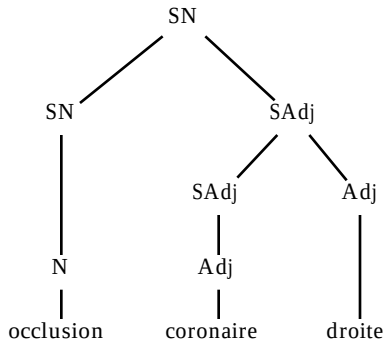


Figure 7.6 : occlusion [coronaire droite]

7.3.4.2. Imprévisibilité des contraintes sur les combinaisons d'arbres

- `artère` et `infarctus` entrent dans des relations de localisation qui se réalisent de manières distinctes :

  `artere {coronaire circonflexe diagonal...}, infarctus {anterieur inferieur apical}`

- Proximités de contextes entre adjectifs...

  `coronaire` et `coronarien` partagent des contextes :

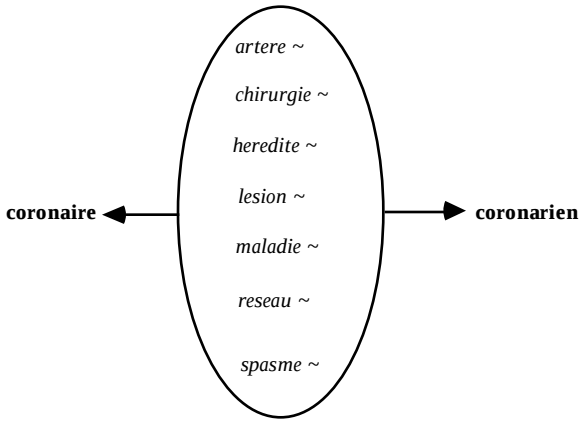


Figure 7.7 : coronaire-coronarien

- Mais ces deux mots ont des modes de combinaison divergents

  `coronarien` est associé à des adjectifs évaluatifs `{severe, significatif, important}`, `coronaire` ne l'est pas.

La figure 7.8 met en avant la diversité des combinaisons d'arbres réalisées sur une même (sous) famille de mots.

7.3.4.3. Imprévisibilité des comportements lexicaux

Le travail d'extraction confirme le fait que les comportements lexicaux ne suivent pas des parcours uniformes sur des entités de même catégorie. Et surtout qu'il est difficile de les prévoir. D'une part les mêmes comportements lexicaux ne se réalisent sur tous les mots d'une même catégorie, ils se distribuent sur des sous-familles particulières. D'autre part, ces comportements ne se réalisent de manière uniforme sur les différentes familles de mots d'une même catégorie. Sur telle famille de mots, un comportement se réalise d'une manière qui est différente de sa réalisation sur une autre famille.

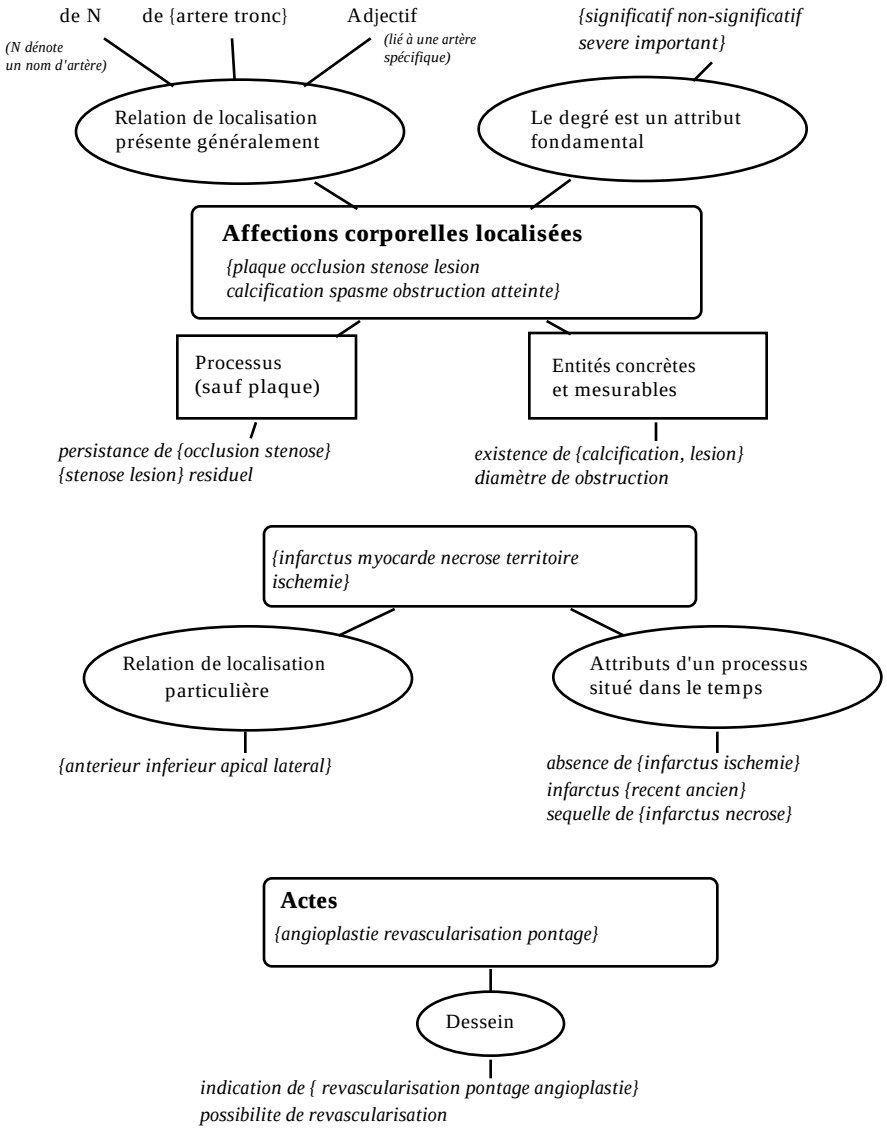


Figure 7.8 : Les comportements des mots sont imprévisibles

La figure précédente présente les comportements qui se réalisent sur certaines familles de mot (des noms) de notre corpus. Les relations rencontrées ne se réalisent pas de manière uniforme sur ces mots : c'est le cas de la relation de localisation qui est soit généralement présente, soit présente sous une forme particulière, soit non présente.

Le travail d'extraction de savoirs à partir de corpus permet d'identifier les valeurs particulières des fonctions lexicales (Mel'cuk 1988) (rencontrées généralement dans la langue) pour un domaine de discours donné. Ces réalisations varient semble-t-il selon les domaines. Ce qu'il est important de souligner ici, c'est que le processus d'extraction puis de simplification des savoirs est conçu comme un processus évolutif. Si sur un état particulier du corpus on obtient des savoirs non pertinents, on peut penser que si le corpus est élargi,

les nouvelles séquences analysées permettront d'affiner les savoirs déjà recensés et donc d'éliminer les savoirs non pertinents retenus à la phase précédente (-> partie 2 chapitre 8 : un système qui module les flux d'information).

7.4. Du lexique jaillit un réseau de prototypes

"On remarquera que la convention, partagée par les linguistes, les informaticiens et les quelques mathématiciens qui s'y intéressent, amène à dessiner les arbres comme orientés vers le bas, racine en l'air, à l'envers en somme. Etrange!"
Jacques Roubaud, Mathématique:, Ed. du Seuil 1997.

7.4.1. Définitions préliminaires

Dans la suite nous utiliserons les notations suivantes :

- On appellera **arbre élémentaire** un arbre associé à une entrée lexicale et qui traduit une relation binaire de dépendance entre mots pleins. Cette unité lexicale étant présente dans l'arbre au niveau de l'une des feuilles.

- On appellera **arbre dérivé** ou **arbre d'analyse** un arbre associé à un arbre élémentaire : cet arbre d'analyse est obtenu par des opérations effectuées à partir de l'arbre élémentaire (adjonction, substitution...).

- On appellera catégorie syntaxique ou mineure une des catégories grammaticales traditionnelles telles que `Nom, Verbe, Adjectif...`

- On appellera prototype d'**arbre élémentaire** l'objet informatique (le prototype) défini pour représenter un arbre élémentaire.

- On appellera prototype d'**arbre d'analyse** l'objet informatique (le prototype) défini pour représenter un arbre d'analyse.

7.4.2. Dégager les comportements syntaxiques d'une entrée lexicale

Nous travaillons ici à partir des résultats obtenus par (Habert & Fabre 1995).

On donne donc ci-dessous certains résultats issus du travail réalisé sur le corpus MENELAS et dont on a pu extraire les informations suivantes :

#LESION#

1#(SN (SN NOM) (SPREP PREP (SN NOM)))#

- **LESION DE TRONC simplification**
  arbre complexe lié: LESION DE LE TRONC GAUCHE

- **LESION SUR CIRCONFLEXE**
  arbre complexe lié: LESION SUR LA CIRCONFLEXE DISTALE

2#(SN (SN NOM) (SADJ ADJ))#

- **LESION NON-SIGNIFICATIF**

- **LESION RESIDUEL**

- **LESION CORONARIEN**
  arbre complexe lié: LESION CORONARIEN SEVERE

- **LESION SIGNIFICATIF**
  arbre complexe lié: PAS DE LESION SIGNIFICATIF

- **LESION SEVERE**
  arbres complexes liés:
  MONTRE DE LES LESION TRITRONCULAIRES SEVERE
  LESION ATHEROMATEUX SEVERE
  LESION CORONARIEN SEVERE
  LESION TRITONCULAIRES SEVERE

- **LESION TRITRONCULAIRES**
  arbres complexes liés:
  MONTRE DE LES LESION TRITRONCULAIRES SEVERE
  LESION TRITONCULAIRES SEVERE

- **LESION COMPLEXE**

- **LESION DE DIFFUS**

- **LESION ATHEROMATEUX**
  arbres complexes liés:
  EXISTENCE DE LESION ATHEROMATEUX
  EXISTENCE DE LESION ATHEROMATEUX SEVERE
  LESION ATHEROMATEUX SEVERE

- **LESION IMPORTANT**

- **LESION MINIME**

- **LESION DIAGONAL**
  arbre complexe lié: LESION CORONAIRE DIAGONAL

- **LESION CORONAIRE**
  arbres complexes liés:
  EXISTENCE DE UNE LESION CORONAIRE
  LESION CORONAIRE DIAGONAL

- **LESION CIRCONFLEXE**
  arbre complexe lié: LESION CIRCONFLEXE DISTALE

- **LESION BITRONCULAIRES**
  arbre complexe lié: EXISTENCE DE LES LESION BITRONCULAIRES

- **LESION DISTALE**
  arbre complexe lié: LESION CIRCONFLEXE DISTALE

- **LESION VERTEBRAL**

Ces premiers résultats vont permettre une première phase de représentation des informations ainsi révélées : le savoir attaché à cette première entité lexicale va servir de base de travail. Il est clair que nous devons garder à l'esprit qu'il conviendra à coup sûr de revenir sur cette première ébauche de représentation. Notons ACL la famille lexicale regroupant les noms désignant une affection corporelle localisée. Lésion appartient à cette famille. On décide donc de construire un prototype associé à lésion qui porte les attributs habituellement associés à cette entrée lexicale (<- représentations antérieures). On associe de plus à cette entrée la liste des comportements syntaxiques que le corpus a révélés, i.e. les arbres élémentaires et leurs arbres complexes liés. Ces comportements sont attachés à notre prototype via la délégation, l'attribut parent indiquant l'objet qui porte ces comportements. Pour ne pas figer cette représentation, cette assignation est réalisée via un attribut modifiable.

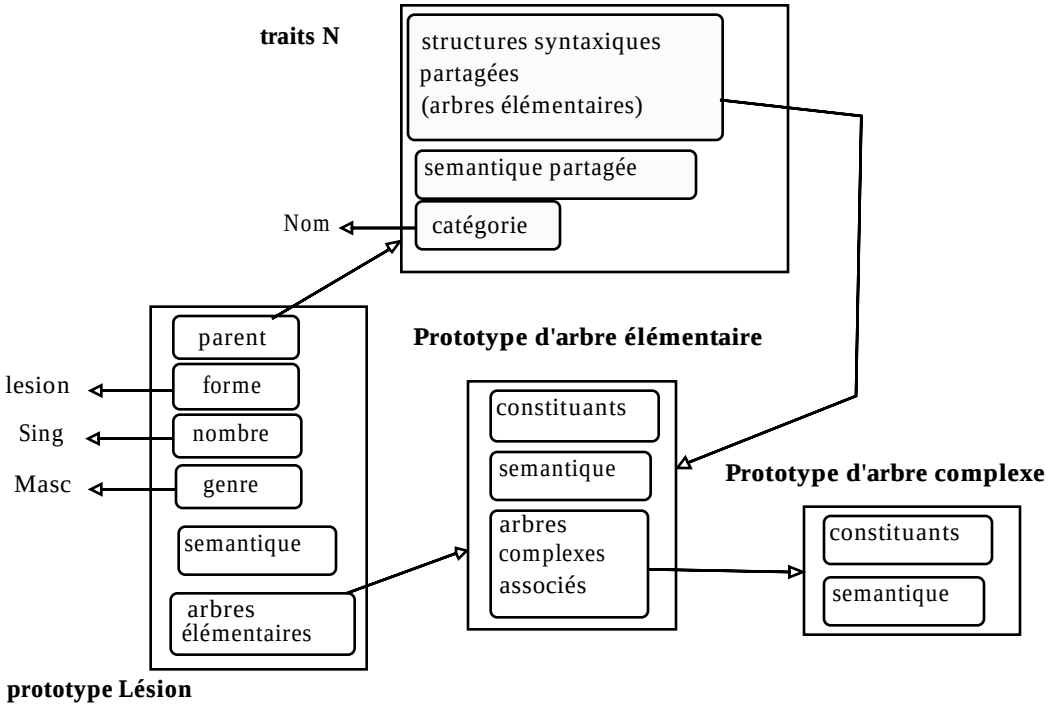


Figure 7.9 : Un prototype pour `lesion`.

7.4.3. Etendre la représentation en ordonnant les comportements

Si l'on considère maintenant une nouvelle entrée appartenant à la même famille lexicale, dont les arbres élémentaires sont de même nature que ceux associés à `lésion`, on peut se servir de cette première phase de représentation pour construire une représentation prototypique de cette nouvelle entité.

Famille des affections localisées :

On peut synthétiser les comportements syntaxiques des éléments de cette famille de la manière suivante :

$Comp(ACL_1)$: N_{ACL} + (1) DE {artere, tronc}
$Comp(ACL_2)$: N_{ACL} + (2) DE {nom d'artère}
$Comp(ACL_3)$: N_{ACL} + (3) SUR {nom d'artère}
$Comp(ACL_4)$: N_{ACL} + (4) ADJ {adjectif lié à un nom d'artère}

Les prototypes représentant des éléments de cette famille lexicale délègueront donc les comportement syntaxiques associés à cette famille dans l'objet (`traits`) que l'on peut noter :

$$\{i = 1,2,3, 4;\ Comp(ACL_i)\}$$

Si un des prototypes représentés ne possède pas tous les types de comportement associés à cette famille, il convient de remodeler cette représentation de la manière suivante :

- On factorise les comportements communs que l'on regroupe dans un objet `traits` donné, et on définit un nouvel objet `traits` qui porte les comportements résiduels.

- On modifie les liens parentaux entre les prototypes définis et leurs parents respectifs : un prototype donné peut déléguer ces comportements aux deux objets `traits` ainsi définis ou vers un seul de ces objets `traits`.

De même, si l'on considère une nouvelle famille d'entités lexicales, on peut, dans un premier temps, entreprendre une première phase de représentation de la même manière que celle que nous avons mise en place pour `lesion`. Si par la suite, la représentation adoptée met en évidence des parallèles comportementaux entre les deux familles, il reste possible de mettre en commun les comportements syntaxico-sémantiques visés. Il s'agit en quelque sorte de définir des catégories abstraites représentant des comportements partagés par un ensemble d'entités lexicales de manière transversale par rapport à leur famille catégorielle initiale.

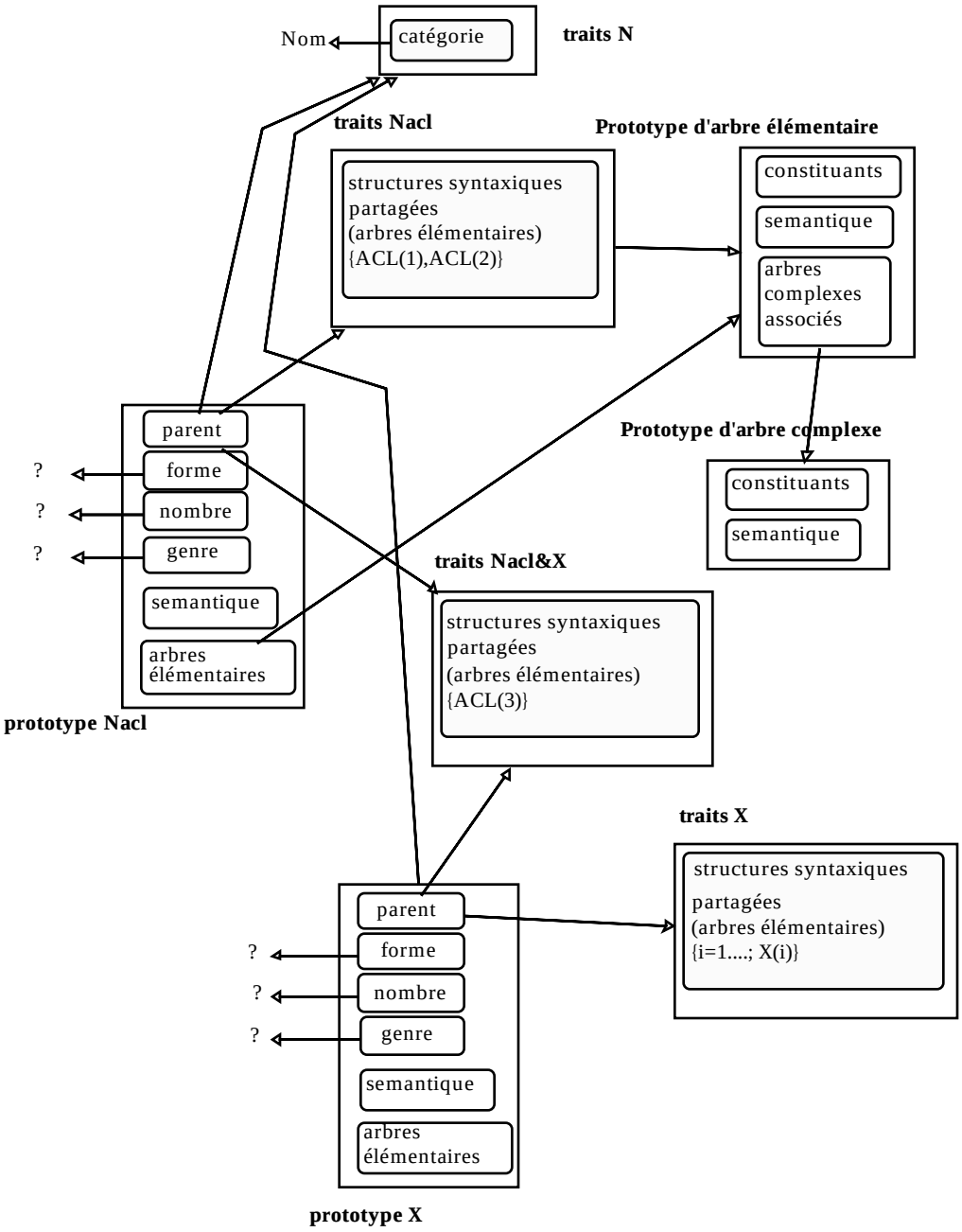


Figure 7.10 : Des Comportements et des Prototypes.

7.5. Esquisse d'un réseau de comportements syntaxico-sémantiques hiérarchisés

Dans le cadre de notre analyse automatique utilisant le langage à prototypes Self, nous allons donc utiliser des résultats d'analyses syntaxiques produits par les analyseurs Lexter et Cyclade sur des corpus de langages spécialisés pour associer aux mots (et donc à leurs représentations prototypiques) des contraintes. L'objectif à atteindre dans notre développement de l'analyse automatique avec des prototypes est de trouver, sur un corpus donné, les éléments de savoir linguistique qui vont permettre de définir un système d'interprétation cohérent et maîtrisable; c'est-à-dire les faits et les règles qui peuvent produire, à partir de ce corpus, des éléments utiles pour une analyse automatique de textes.

7.5.1. Une première phase de représentation des informations recueillies sur un corpus donné

On affecte désormais aux prototypes de mot construits leurs comportements syntaxico-sémantiques. On considère pour commencer que l'on récupère des informations sur certaines entités lexicales sous la forme suivante :

`pontage Nom (Sn(Det Nom) NPrepN(Nom Prep Nom) AdjN(Adj Nom) NAdj(Nom Adj) PrepN(Prep Nom) ...)`

Le travail d'extraction associe au mot `pontage` de catégorie `Nom` les arbres `Sn(Det Nom) NPrepN(Nom Prep Nom) AdjN(Adj Nom) NAdj(Nom Adj) PrepN(Prep Nom)`... Ces informations sont fournies par le traitement d'extraction de connaissances sur un corpus donné suivant le travail présenté supra. Ces informations vont permettre d'enrichir la représentation des prototypes lexicaux définis dans notre cadre de travail : ces derniers ayant à priori peu de savoir sur ce qu'ils sont capables de faire; ils ne disposent pas, par exemple, de savoir sur les structures syntaxiques dans lesquels ils peuvent être insérés. A la différence de ce que nous faisions précédemment, il ne s'agit plus de pré-déterminer et d'encoder "à la main" les savoirs syntaxiques que l'on peut attacher aux mots. Il s'agit au contraire d'utiliser les savoirs recueillis sur des séquences présentes dans un corpus et de les affecter aux entités prototypiques concernées.

Association automatique des comportements syntaxiques aux prototypes lexicaux

Ces informations sont donc affectées aux prototypes visés par ces contraintes de la manière suivante :

- A partir de la description précédente du mot `pontage`, on affecte automatiquement au prototype représentant ce mot les comportements qui lui sont associés : les arbres `Sn, NPrepN`... sont associés au mot `pontage`. On associe donc au prototype lexical `pontage` les prototypes d'arbres `Sn, NPrepN`...

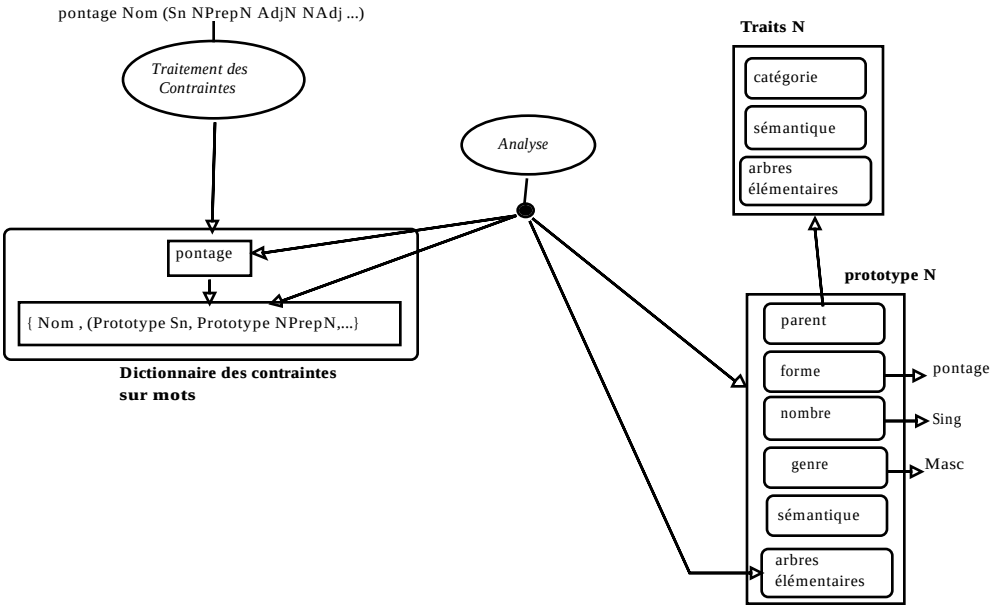


Figure 7.11 : Les prototypes apprennent peu à peu.

Il convient de préciser que cette première phase de représentation maintient pour le moment la prédéfinition des savoirs syntaxico-sémantiques pour des représentations prototypiques de catégories syntaxiques représentant des "groupes de mots" : les prototypes `Sn`, `NPrepN`... disposent d'un savoir syntaxico-sémantique qui a été encodé à l'image de ce que nous avons déjà présenté. Nous verrons infra comment la génération automatique de prototype d'arbre nous permettra de ne pas utiliser de savoirs préconstruits (-> automatisation des traitements de génération de prototypes de mots et d'arbres).

7.5.2. Construction d'une hiérarchie approximative

Le schéma précédent ne tient pas compte de ce que nous avons présenté supra concernant le partage des savoirs syntaxiques communs à un ensemble de prototypes donnés. Il manque pour cela une phase de travail capable d'évaluer les similitudes de comportements entre les entités représentées.

Dans l'exemple qui suit, on dispose au départ des descriptions textuelles des mots `pontage` et `lesion`. On affecte donc aux prototypes lexicaux concernés leurs comportements. On cherche ensuite si ces deux prototypes lexicaux possèdent des comportements communs. Si tel est le cas, les comportements partagés par ces deux prototypes lexicaux sont affectés automatiquement à l'objet existant qui porte déjà leurs comportements partagés (mise à jour de l'attribut `arbre élémentaire` dans l'objet `Traits N`).

- Phase 1 : Lecture des contraintes, vérification de l'existence des prototypes catégoriels ou création de ceux-ci.

- Phase 2 : Attribution des arbres élémentaires aux prototypes de mot.

- Phase 3 : Classement automatique des prototypes de mot en fonction de leurs comportements[84], mise en place d'un héritage local de comportements.

[84] On classe les objets représentant des mots sur la base des comportements réels associés aux mots ainsi modélisés. Les comportements informatiques qui ne décrivent pas les mots dans leur activité langagière ne sont pas pertinents pour ce classement.

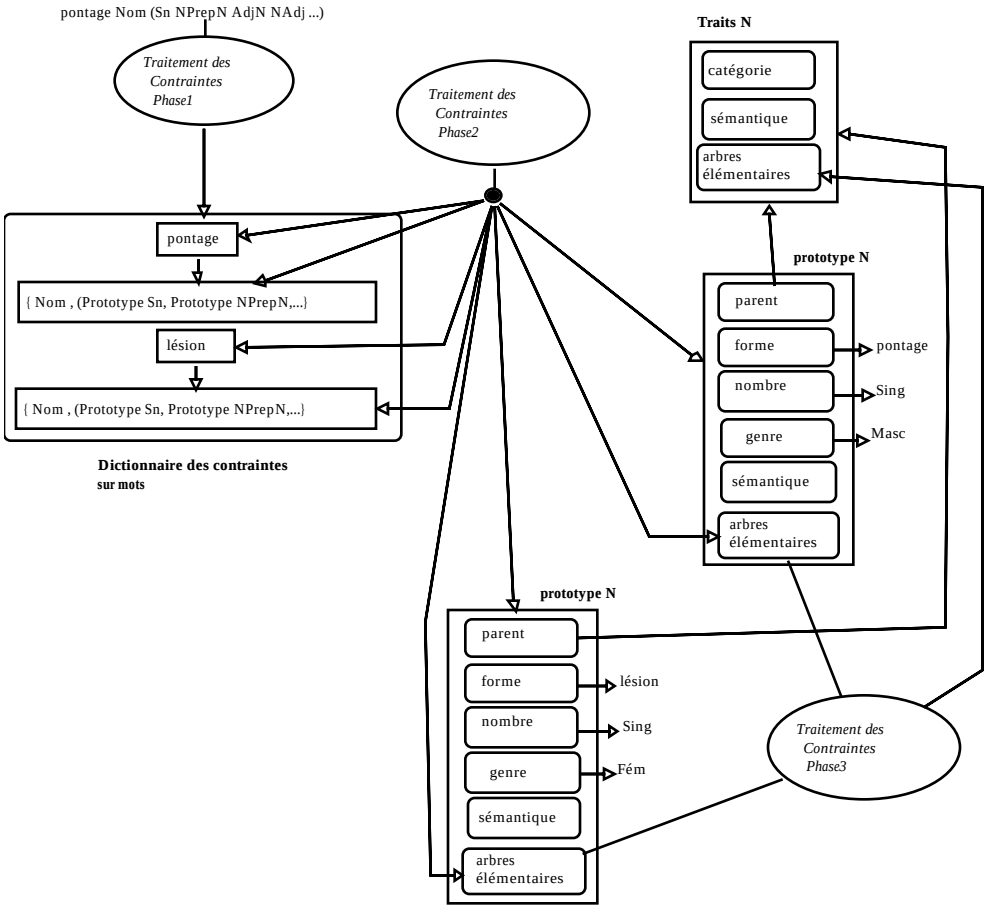


Figure 7.12 : Les prototypes s'organisent peu à peu.

7.5.3. Illustration avec Self

Nous présentons en annexe (-> annexe partie 2 chapitre 7) une trace de l'exécution des opérations décrites ci-dessus. Dans l'exemple donné, on exécute une réorganisation des savoirs associés aux prototypes représentant la catégorie des `Noms`. On fait tout d'abord une recherche de tous ces prototypes et de leurs contraintes respectives. Puis on détermine les comportements communs : une classe de comportements partagés par tous les prototypes qui disposent de contraintes. Cette classe pourra ensuite être associée à l'objet `Traits` commun à tous ces prototypes. Les prototypes qui ne disposent pas de contraintes à l'issue de la définition de celles-ci disposeront ainsi d'un savoir par défaut. Cela ne veut pas dire que ce savoir sera utilisé pour les entités qui ne disposent pas de contraintes particulières. Il s'agira simplement de marquer que sur une famille donnée, certains prototypes possèdent des contraintes et que ces savoirs sont disponibles mais ne sont pas utilisables pour tous les éléments de la famille.

7.5.4. Hiérarchie mouvante et apprentissage

Si la représentation prototypique d'une entrée lexicale n'est pas affectée par le travail d'extraction de connaissances sur le corpus donné, il va être possible de vérifier que les comportements hérités, via le partage défini dans cette phase d'apprentissage de comportements par défaut, sont valides ou non pour cette entité. Dans le premier cas, on peut avoir une confirmation d'un savoir construit par défaut, dans le second cas, on peut envisager que cet apprentissage implique une réorganisation des savoirs représentés : suppression du partage (globalement ou localement) pour cette entité, mise en place d'une hiérarchisation plus fine des comportements qui tienne compte des particularités mises à jour pour cette entité.

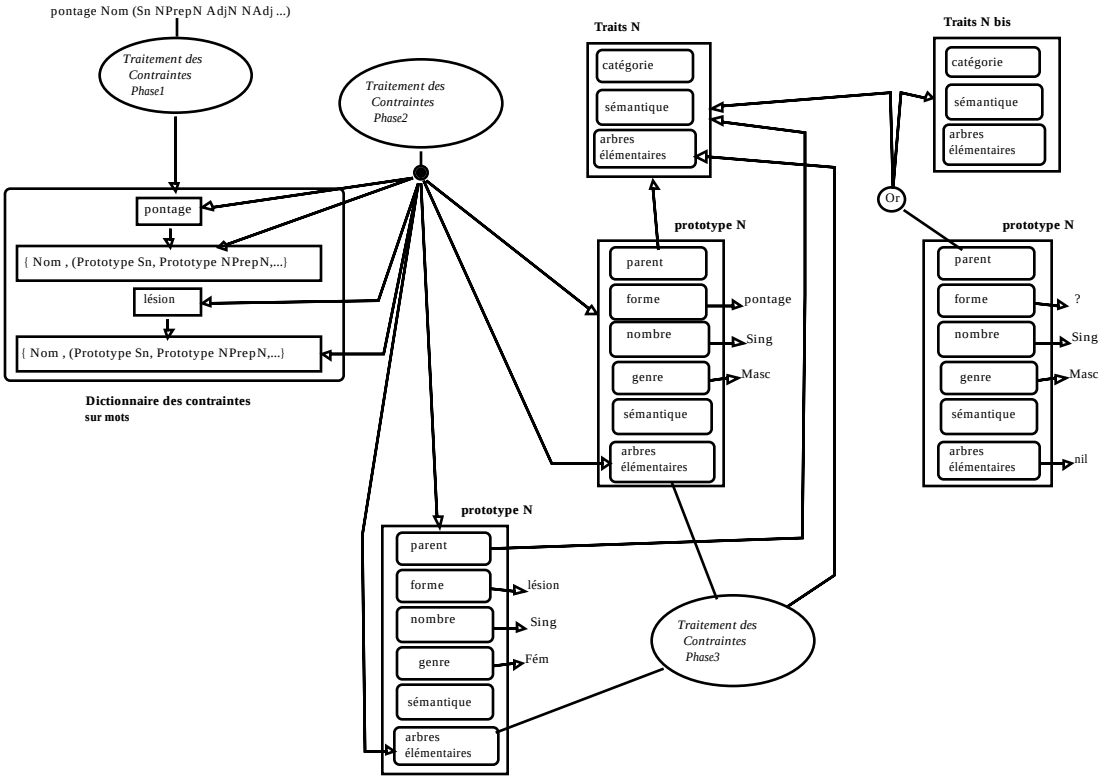


Figure 7.13 : Les représentations se développent par apprentissage et organisation des savoirs.

7.6. Automatisation des traitements de représentation des mots et des arbres

Approches d'un dispositif de génération automatique de prototypes de mot et de prototypes d'arbre à partir de savoirs extraits sur corpus.

7.6.1. Construire les représentations à la demande

Puisqu'il n'est pas possible de prédéterminer ce que peuvent être les informations fournies par la phase d'extraction de savoirs à partir de corpus, le dispositif doit permettre une génération automatique de représentations des informations reçues (mot, catégorie...). Le dispositif doit pouvoir désormais se passer des représentations prédéfinies d'unités de langue, il doit les créer de toutes pièces si nécessaire puis les multiplier.

En reprenant notre exemple précédent :

```
(1) pontage Nom (Sn(Det Nom) NPrepN(Nom Prep Nom) AdjN(Adj Nom)
NAdj(Nom Adj) PrepN(Prep Nom) ...)
```

la phase d'extraction d'informations sur le corpus va produire une suite de contraintes telles que celles associées à `pontage` dans `(1)`; ce qui va conduire ensuite à associer au prototype lexical représentant le mot `pontage` la liste des prototypes représentant les catégories syntaxiques présentes dans la liste des contraintes associées à `pontage` dans `(1)`. Cette association ne posera pas de problèmes si les prototypes associées à ces catégories syntaxiques sont déjà définis. Dans le cas contraire, on ne peut que produire une erreur. Pour pallier ce manque éventuel de prédéfinition de prototypes représentant une catégorie syntaxique donnée, il semble raisonnable de disposer d'un générateur de prototypes capable de produire à la volée un objet capable de représenter une catégorie syntaxique non encore représentée.

Sur ce point, on retrouve un point fondamental déjà énoncé concernant le problème fondamental de l'analyse automatique dans un univers prototypique et plus précisément la nécessité d'aller vers un effacement progressif de la notion habituelle de grammaire telle qu'elle est utilisée dans l'analyse automatique. Notre approche se situe en effet dans une perspective où l'analyse automatique reconstruit une grammaire en fonction des informations rencontrées; et non dans une perspective où l'analyse automatique se contente d'opérer la vérification d'une grammaire donnée. On part de rien (les représentations prototypiques initiales sont minimales) et on reconstruit des représentations ou on les affine en fonction des savoirs mis à jour. En particulier, il s'agit aussi de reconstruire une grammaire, en fait un prototype de grammaire qui, du fait de l'approche retenue ici, n'existera pas au démarrage de l'analyse, mais se révèlera en fin d'analyse.

7.6.2. Génération automatique de prototypes

On a donc mis en place dans les phases de traitements des contraintes telles qu'elles sont présentées dans les figures précédentes, une étape de création "à la volée" (i.e. automatique) de prototypes si on rencontre une catégorie syntaxique non encore représentée. Ces nouveaux objets seront créés soit en utilisant les représentations existantes, soit de toutes pièces.

7.6.2.1. Génération par création explicite d'un nouvel objet

On peut ainsi être contraint de construire de toutes pièces une représentation prototypique d'une catégorie donnée. On dispose pour cela d'un opérateur capable de produire une représentation qui tienne compte des informations fournies (nom de la catégorie, constituants éventuels...).

- Génération à la volée (i.e. automatique) de prototypes par création explicite d'un nouvel objet

Si la catégorie d'une unité lexicale n'est pas encore représentée, on définit les objets pour la représenter avec les attributs adéquats. Dans l'exemple qui suit, si le mot `pontage` ne possède pas encore de représentation prototypique et si la catégorie des noms n'est représentée par aucun prototype, on construit[85] de toutes pièces une nouvelle représentation prototypique de cette nouvelle famille catégorielle en utilisant les informations associées à ce premier élément de cette famille.

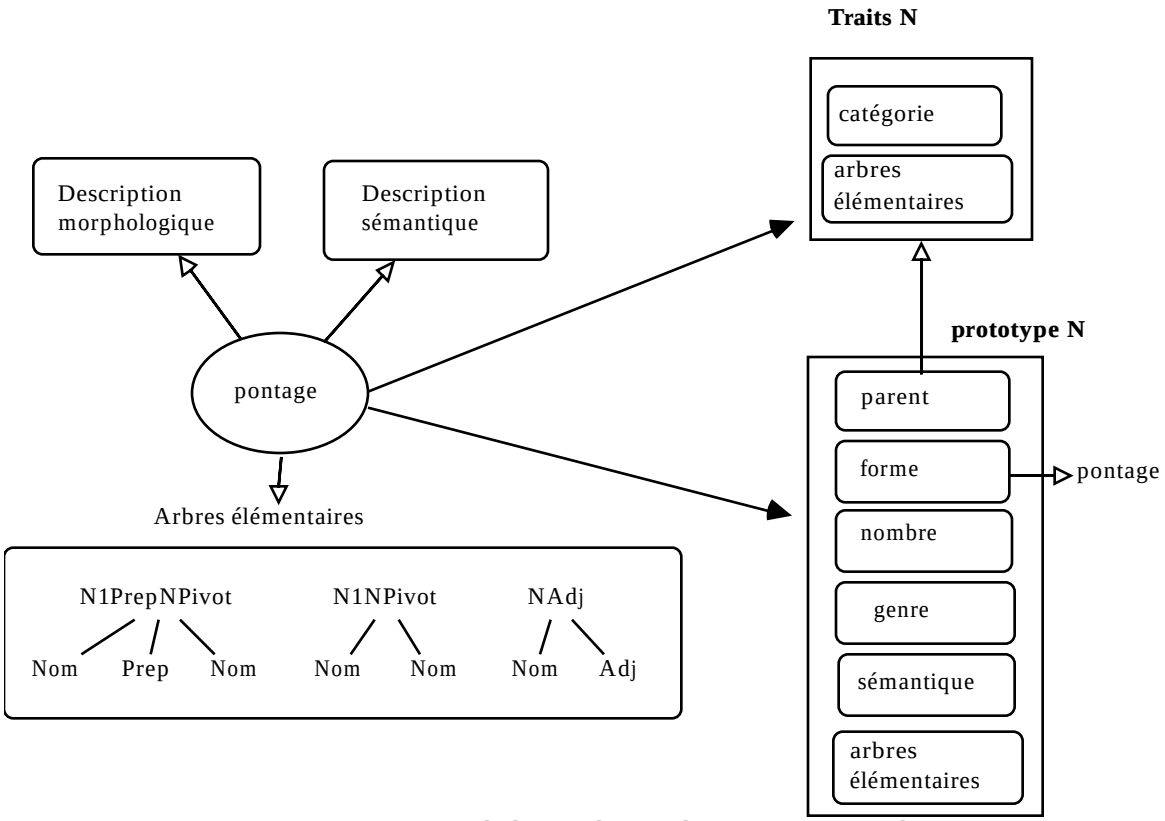


Figure 7.14 : Génération à la volée de prototype catégoriel.

[85] La structure construite pour représenter un mot contient un certain nombre d'attributs qui seront valués dès que les informations pertinentes pour ces attributs seront disponibles.

Si les arbres élémentaires associés à `pontage` ne possèdent pas encore de représentation prototypique (si la catégorie syntaxique tête d'un arbre élémentaire n'est pas encore représentée), le générateur d'arbre construit automatiquement des structures pour représenter ces arbres en tenant compte des informations fournies (nombre de composants, position des composants...).

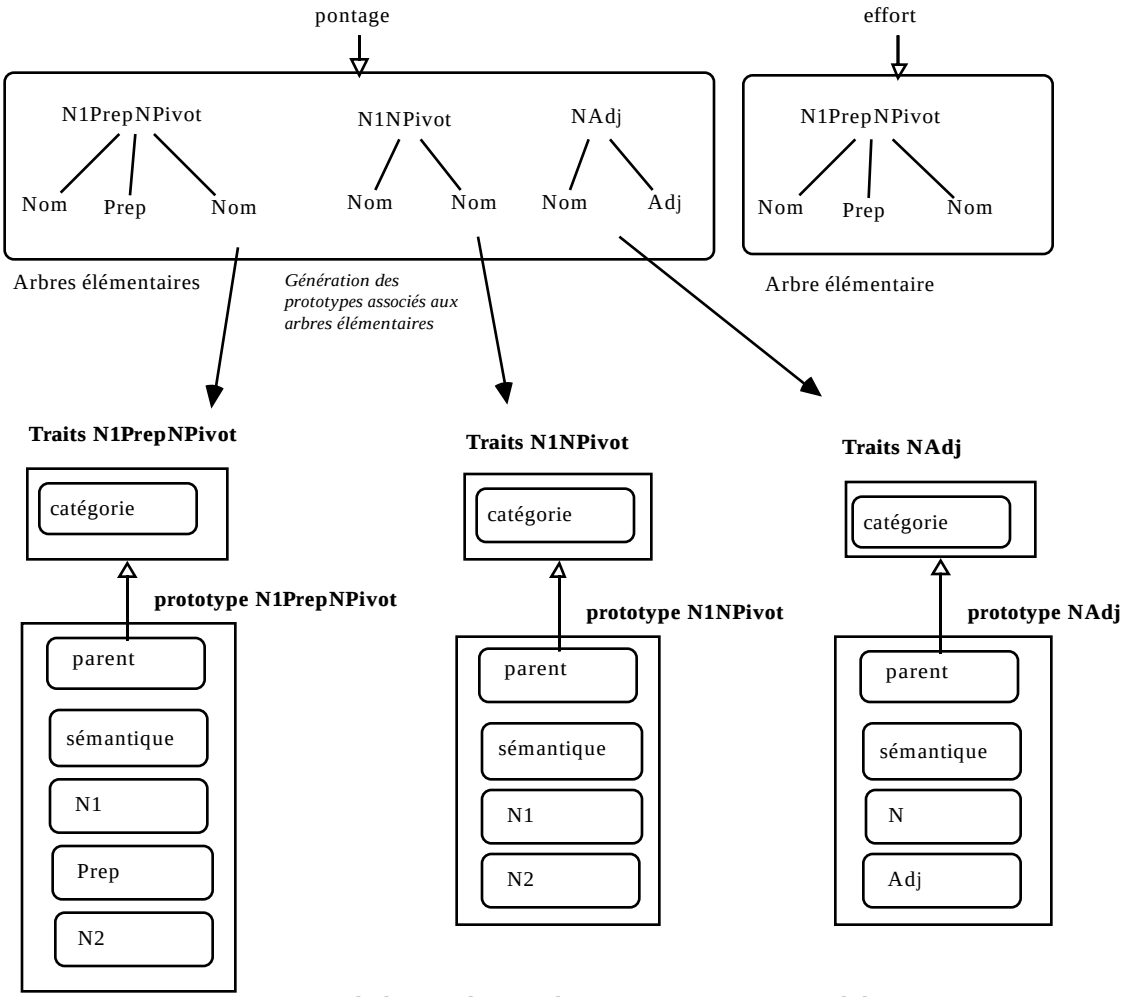


Figure 7.15 : Génération à la volée de prototypes d'arbre élémentaire.

Dans la figure précédente, le prototype `N1PrepNPivot` représente l'arbre `N1 Prep N2` dans lequel le `Nom` en position `N2` est le pivot de l'arbre à construire.

- Génération à la volée (i.e. automatique) de prototypes d'arbres d'analyse par création explicite d'un nouvel objet

De même, si les arbres d'analyse associés à `pontage` ne possèdent pas encore de représentation prototypique (si la catégorie syntaxique tête d'un arbre d'analyse n'est pas encore représentée), le générateur d'arbre construit automatiquement des structures pour représenter ces arbres en tenant compte des informations fournies (nombre de composants, position des composants...).

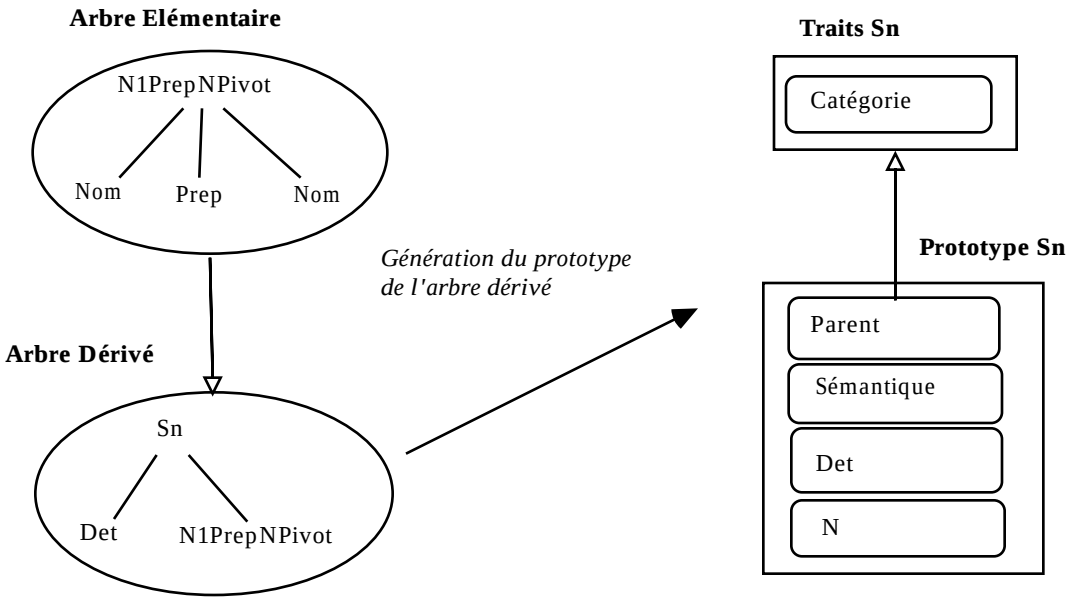


Figure 7.16 : Génération à la volée de prototype d'arbre d'analyse.

7.6.2.2. Génération par clonage et ajustements

On dispose aussi des outils habituels dans un environnement de programmation qui manipule des prototypes pour mettre en oeuvre la génération de nouveaux prototypes, à savoir l'opération de clonage pour dupliquer les objets concernés puis les opérations qui permettent d'ajuster les valeurs des attributs d'un objet donné (ajouter des attributs, en retirer, ajuster leurs valeurs...).

- Génération à la volée (i.e. automatique) de prototypes par clonage et ajustements :

Si le mot "effort" n'a pas encore de représentation prototypique, on utilise le clonage (sur une entité existante et de même catégorie) puis l'ajustement pour le représenter.

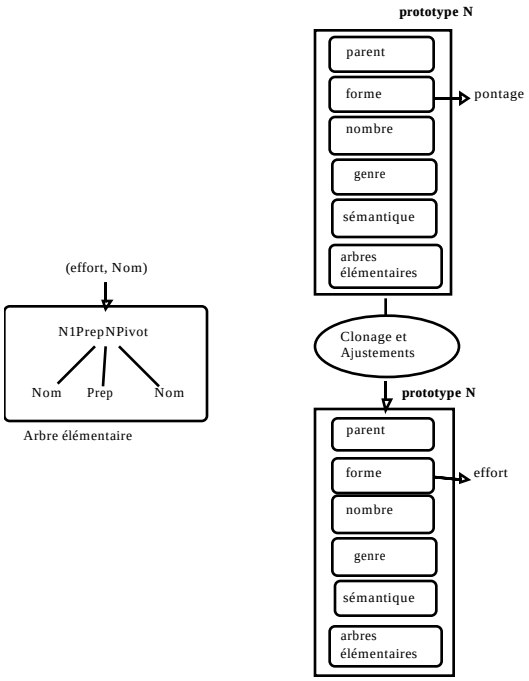


Figure 7.17 : Génération à la volée de prototype par clonage et ajustement.

On donne en annexe (-> annexe partie 2 chapitre 7) une trace de l'exécution de cette nouvelle phase de génération de prototypes.

7.6.3. Une construction inductive des savoirs et une classification dynamique de ces savoirs

La génération de prototypes s'appuie sur les savoirs recueillis pour construire des représentations prototypiques adéquates. Notre prototype de grammaire va donc se construire à partir des informations attachées aux mots. Tout le savoir associé aux formes lexicales va permettre de construire par induction un prototype de grammaire dépendant des informations attachées aux mots et donc au corpus initial qui a servi de base pour l'extraction des informations utilisées ici.

7.6.3.1. Construire un prototype de grammaire lexicalisée

La phase d'extraction de savoirs sur le corpus fournit des éléments d'information sur des entités lexicales déjà catégorisées[86]; ces éléments d'information peuvent d'ailleurs être des formes de savoir abstrait (on ne restreint pas le savoir attaché aux mots à de simples relations entre formes lexicales). Nous souhaitons en fait, grâce à ces éléments reçus en amont de la représentation, pouvoir tenir compte du fait que dans cette représentation, certains éléments peuvent se construire par introspection et d'autres non. Les informations initiales que nous manipulons ont pour le moment l'allure suivante :

```
#Dc
pontage
Nom
Sn
(('Det' & 'Nom') asList)
NPivotprepN2
(('NomPivot' & 'Prep' & 'Nom') asList)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
AdjN
(('Adj' & 'Nom') asList)
NAdj
(('Nom' & 'Adj') asList)
NAdj
(('Prep' & 'Nom') asList)
N1NPivot
(('Nom' & 'NomPivot') asList)
Fc#
```

Dans cet exemple le mot `pontage` de catégorie `Nom` est associé aux arbres élémentaires `Sn`, `NPivotprepN2`, `N1prepV`, `AdjN`, `NAdj`, `NAdj`, `N1NPivot`. A chacun de ces arbres élémentaires, on associe la liste de ses composants: `(('Det' & 'Nom') asList)`, une liste en Self, associée à `Sn`... Il est clair que ce savoir initial demande à être complété par des informations supplémentaires. On peut par exemple y adjoindre des informations morphologiques, sémantiques... sur les mots (<- génération automatique de prototypes) et tout type d'information pertinente pour la description du mot à représenter. On peut ainsi y ajouter les informations qui indiquent que tel arbre élémentaire est associé à tel(s) arbre(s) complexe(s), en indiquant l'historique de la dérivation permettant de passer de l'un à l'autre (Habert & Gaussier 1997).

```
#Dc
pontage
Nom
(Description morphologique)
Masc
Sing
...
(Informations sémantiques...)
...
Sn
(('Det' & 'Nom') asList)
(Arbre complexe lié et dérivation)
NPivotprepN2
(('NomPivot' & 'Prep' & 'Nom') asList)
(Arbre complexe lié et dérivation)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
(Arbre complexe lié et dérivation)
AdjN
(('Adj' & 'Nom') asList)
(Arbre complexe lié et dérivation)
NAdj
(('Nom' & 'Adj') asList)
(Arbre complexe lié et dérivation)
PrepN
```

[86] A la différence de (Bensch & Savitch 1995), notre approche n'a pas à se poser la question de la catégorisation des éléments lexicaux à représenter.

```
(('Prep' & 'Nom') asList)
(Arbre complexe lié et dérivation)
N1NPivot
(('Nom' & 'NomPivot') asList)
(Arbre complexe lié et dérivation)
Fc#
```

D'une part, nous souhaitons construire dynamiquement des représentations informatiques des formes lexicales. D'autre part, en utilisant les savoirs associés aux mots représentés, nous visons à une classification dynamique des mots et de leurs représentations. Nous ne limitons pas le savoir attaché aux prototypes représentant des mots à des associations de formes lexicales (tel mot est fréquement associé a tel(s) mot(s) dans tel(s) contexte(s) (cf. infra)), mais nous souhaitons construire des savoirs comportementaux des formes représentées qui puissent généraliser les savoirs incrémentalement recueillis et construits. On veut par exemple pouvoir repérer que tel groupe de mots est capable de se comporter de manière particulière dans une structure syntaxique donnée (par exemple construction d'une structure de type `N1PrepN2` en position `N1` avec `N2` de type(s) donné(s)). Notre approche de représentation dynamique à partir de savoirs extraits de corpus ne doit pas limiter les savoirs aux seules inter-relations entre les formes lexicales représentées, mais doit pouvoir nous permettre de classer et de prédire les nouvelles représentations à venir.

7.6.3.2. Une amorce de classement guidée par la syntaxe

Notre objectif est de tendre vers la détermination de classes sémantiques, de manière inductive, dans une approche symbolique[87], celle des sous-langages. Tout d'abord, il convient de souligner que le classement automatique opéré jusqu'ici s'appuie principalement sur des contraintes syntaxiques. Dans ce qui a été présenté supra, les comportements associés aux représentations prototypiques des entrées lexicales contraintes sont de simples squelettes syntaxiques dépourvus de contraintes particulières. Il est clair que cette affectation de comportements syntaxiques n'est pas suffisante pour une classification cohérente des éléments représentés : *"/./ des propriétés contextuelles permettent de rapprocher certains mots, mais la syntaxe seule ne permet pas d'identifier les liens sémantiques entre ces unités lexicales. Elle suggère des liens de hiérarchie (`générique` vs. `spécifique`) ou des relations conceptuelles (ex. `a pour partie`) mais des connaissances extérieures sont nécessaires pour valider la nature de ces liens. Par exemple, la similitude des syntagmes `pontage coronarien` et `pontage saphène` n'indique pas que la mention du vaisseau joue dans un cas le rôle de localisateur (pontage **sur** la coronaire) et dans l'autre celui d'un instrument (pontage **à l'aide de** la veine saphène)"* (Habert & Nazarenko 1996).

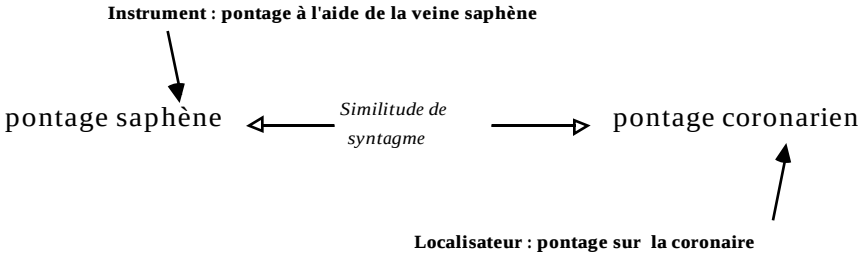


Figure 7.18 : Similitude formelle de syntagmes

La syntaxe doit en quelque sorte dégrossir la représentation mais ne permet pas à elle seule de classer les unités représentées; (Habert & Nazarenko 1996) montrent que la syntaxe est incapable à elle seule de délimiter des classes de mots reflétant une notion. On peut avoir un rapprochement de certaines unités lexicales suivant certains comportements syntaxiques (les noms d'affections), mais aucune classification ne se révèle directement à partir des comportements syntaxiques : *"le fonctionnement syntaxique des unités permet donc non pas de construire une ontologie du domaine mais de dégrossir le travail de définition de*

[87] (Resnik 1993) présente une approche de construction de classes sémantiques de manière inductive dans une optique statistique.

cette ontologie" (Habert & Nazarenko 1996). A l'inverse des approches harrissiennes[88] et statistiques[89], notre approche ne conduit pas à la détermination de classes sémantiques satisfaisantes mais elle constitue une méthode d'amorçage pour l'élaboration de l'ontologie du domaine, nous suivons sur ce point la démarche suivie par (Habert & Nazarenko 1996) : la construction de l'ontologie du domaine étudié nécessite un part d'interprétation, *"il y a, entre le flou notionnel inhérent aux langues naturelles (Kayser 1992), y compris aux langues de spécialités (Dachelet 1994), et la stabilité conceptuelle qui est requise dans les ontologies construites, un seuil qui ne peut être franchi automatiquement"* (Habert & Nazarenko 1996).

On peut aussi envisager de projeter les résultats intermédiaires sur des classes sémantiques construites par ailleurs afin d'affiner les savoirs construits et donc leurs classements. Si les projections de savoir permettent d'affiner les arbres élémentaires en amont, en particulier sur l'arbre NAdj associé à pontage, on va affiner le travail de représentation des arbres élémentaires associés à pontage.

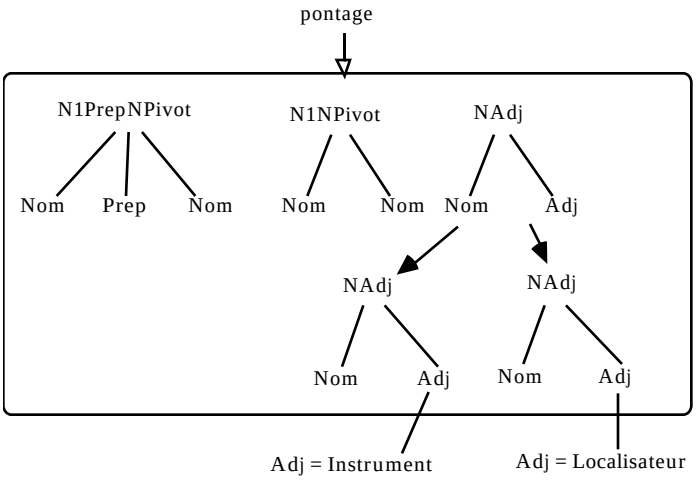


Figure 7.19 : Affiner les savoirs initiaux...

Si on dispose en amont d'informations supplémentaires sur les arbres élémentaires, on peut affiner la construction des représentations prototypiques de ces arbres élémentaires. Sur notre exemple précédent, si la différence dans la nature des liens sémantiques entre les composants de l'arbre élémentaire NAdj est précisée sur les arbres élémentaires recueillis en amont, on utilisera cette précision pour spécialiser les prototypes à construire associés à l'arbre élémentaire NAdj à représenter.

---

[88] Dans l'approche suivie par N. Sager, les classes de mots et les formules syntaxiques d'une grammaire de sous-langage sont données comme en correspondance étroite avec les classes d'objets du monde réel et à leurs relations, le travail d'interprétation des résultats construits semble sous-estimé.

[89] Dans les approches statistiques (Church & Hanks 1990, Hindle 1990), les classes de mots sémantiquement pertinentes sont obtenues statistiquement et induites du corpus à partir des seules mesures de similarités entre mots : *"En réalité, les listes de mots obtenues ne constituent pas de véritables classes de mots cohérentes et homogènes. Il est souvent difficile de leur attribuer une étiquette globale, et ce d'autant plus que les mesures de similarité constituent des résumés statistiques bruts dont les critères de construction sont effacés"* (Habert & Nazarenko 1996).

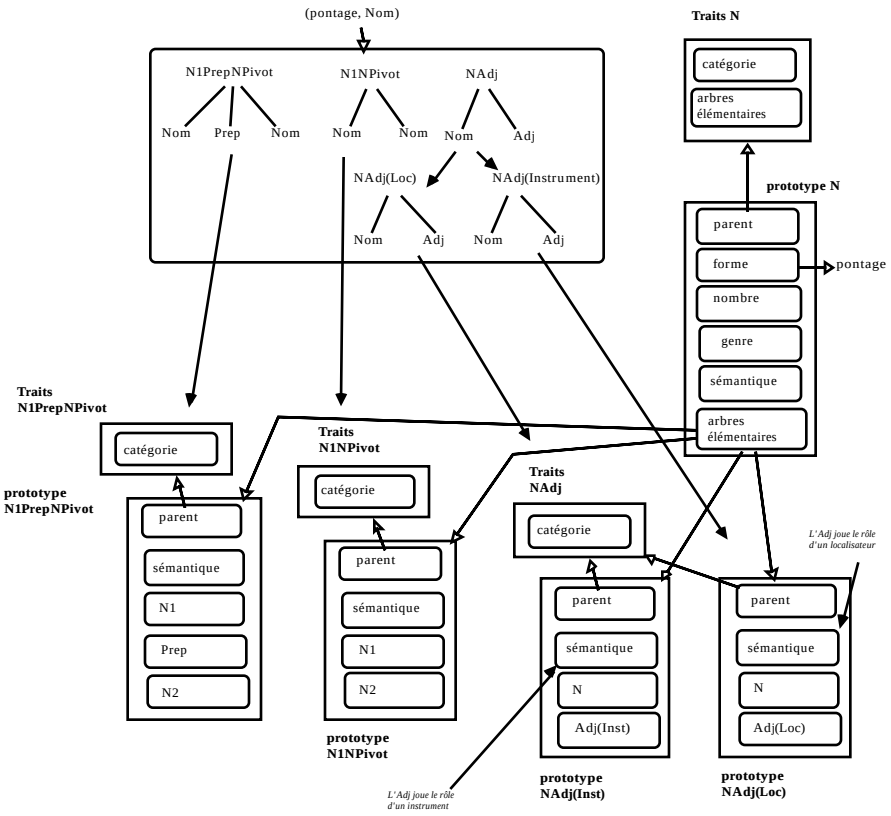


Figure 7.20 : Pour affiner les arbres "élémentaires" à construire.

De plus le classement présenté ne tient pas compte des arbres d'analyse associé aux arbres élémentaires et donc aux mots (-> partie 2 chapitre 8). Or si le classement sur les arbres élémentaires peut rapprocher des mots comme `effort` et `myocarde` sur la base d'un comportement partagé : ces deux mots entrent dans des séquences `N1 Prep N2` en position `N2`. Ce rapprochement ne dit rien de la différence de comportements de ces deux mots dans des arbres d'analyse : dans ces derniers, `effort` ne peut pas y être modifié, alors que `myocarde` l'est toujours par un déterminant.

7.6.3.3. Créer des pôles multiples de regroupements de comportements partagés, les abstraire et les organiser

Notre approche vise à travailler à partir du savoir attaché aux prototypes lexicaux représentés puis à repérer les partages possibles sur ces savoirs parmi tous les objets représentés. Il s'agit en fait de mettre en place et d'affiner l'équilibre qui existe entre des savoirs localement répartis et des savoirs partagés qui établissent des liens entre les éléments représentés. Pour mener à bien cette tâche, il est clair que nous devons pouvoir manipuler et assimiler assez d'informations dans le flot de savoirs extraits du corpus de travail. La difficulté est bien évidemment de construire des outils qui permettent au dispositif d'apprendre et de construire du savoir à partir de ce flot de savoirs initialement indépendants puisque répartis sur chacun des mots. En prenant appui sur un premier classement des unités lexicales sur la base de la micro-syntaxe qui leur est attachée, on va pouvoir construire un réseau de pôles de comportements partagés par des ensembles de prototypes (-> incise 1 : Self et l'héritage). Il reste ensuite à opérer un travail d'interprétation de ces différents pôles de regroupements de comportements. Il est sûr que ces regroupements ne permettent pas de classer les unités lexicales de manière suffisante : les rapprochements syntaxiques peuvent révéler des similitudes purement formelles et masquer la différence de nature entre les types conceptuels en jeu dans ces structures. Dans la figure 7.21, les prototypes construits (représentant des noms) sont regroupés en sous-familles en tenant compte des arbres élémentaires associés à ces prototypes de mots

(par délégation). Les prototypes d'une sous-famille délèguent les mêmes arbres élémentaires à un ou plusieurs pôles de comportements partagés.

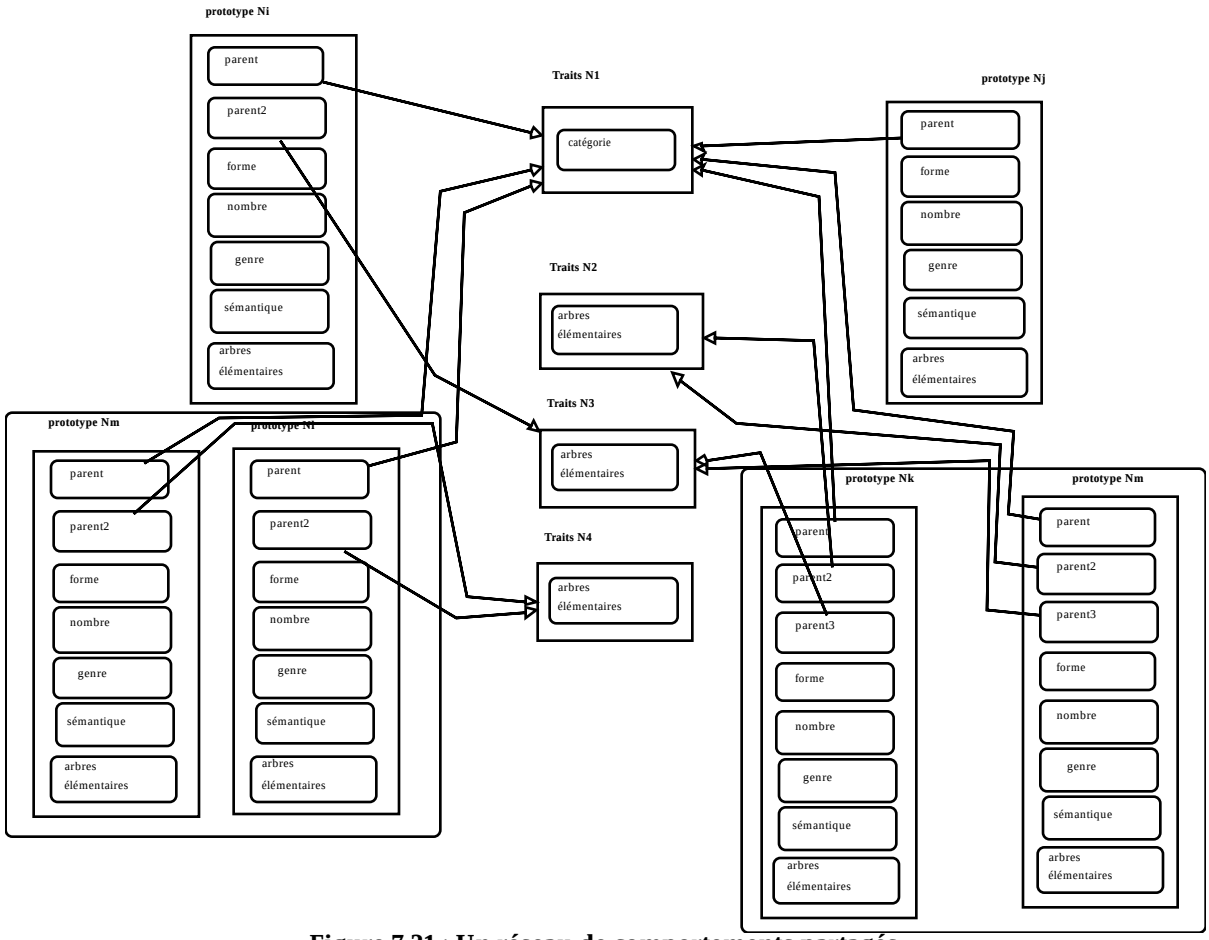


Figure 7.21 : Un réseau de comportements partagés.

Difficultés techniques

La mise en place d'un tel réseau de familles de prototypes qui partagent ensemble des arbres élémentaires ne pose pas de problèmes techniques insurmontables dans la réalisation de ce réseau (-> annexes partie 2 chapitres 7 et 8 : traces complètes de la génération des prototypes et de la construction d'un réseau de pôles de comportements partagés). Il est facile de repérer sur un ensemble de prototypes les comportements communs à certains d'entre eux puis de définir les objets et les liens qui vont unir les objets et leurs pôles de comportements partagés. **L'automatisation du classement et la pertinence de cette automatisation posent davantage des problèmes linguistiques et théoriques (classification automatique) que techniques**. Repérer des similitudes formelles entre les représentations créées est une tâche à la portée de la programmation à prototypes. Le classement obtenu est une amorce de classement sur lequel un travail d'interprétation reste à faire: **Evaluer ces similitudes formelles et interpréter les regroupements restent des tâches auxquelles doit être confronté le linguiste dans la mesure où il est le seul à pouvoir y apporter une réponse précise**.

Difficultés linguistiques

La difficulté est donc principalement d'ordre linguistique. Le classement obtenu va mettre un peu d'ordre là où, au départ, il n'y en avait pas. On va organiser les différentes représentations d'unités lexicales sur la base des comportements syntaxiques communs à certaines d'entre elles. Ces pôles de savoirs partagés restent malgré tout à affiner. Le degré de précision du classement initial des unités lexicales sera directement dépendant de la somme des savoirs qui sera utilisée pour opérer ce classement initial. Si on ne prend appui

que sur des informations syntaxiques, on retrouvera des problèmes déjà soulignés (les similitudes formelles masquent souvent des interprétations de différentes natures).

7.6.3.4. Intervenir manuellement pour interpréter les regroupements

Enfin, si on peut automatiser le classement des prototypes lexicaux sur la base des comportements qui leurs sont associés, les résultats obtenus restent à qualifier, à nommer.

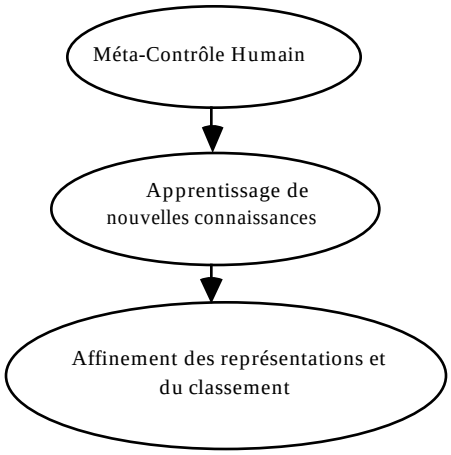


Figure 7.22 : Un méta-contrôle humain pour nommer les choses

Si le classement rapproche des mots comme "absence" ou "ensemble" sur des comportements identiques, cette proximité comportementale n'est pas identifiée comme étant propre à celle des noms prédicatifs. C'est en examinant à la main ce type de rapprochement que l'on pourra leur donner un nom c'est à dire nommer les choses.

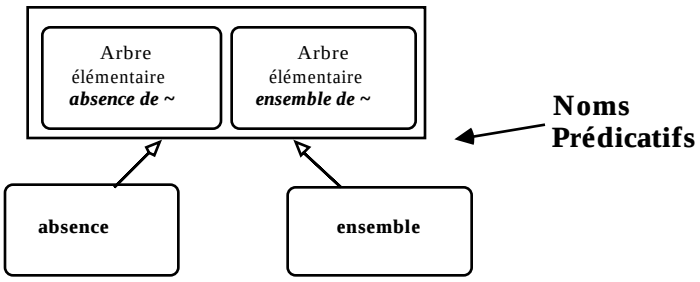


Figure 7.23 : absence-ensemble

D'une part il est nécessaire de définir les informations pertinentes pour amorcer le classement, d'autre part il convient de ne pas perdre de vue ces critères initiaux qui permettent d'obtenir ce premier classement afin de ne pas sur-estimer ce classement. Il s'agit au contraire de considérer que le résultat obtenu reste en deça de ce que l'on souhaite mettre en oeuvre, l'écart résultant des carences inévitables que le traitement automatique met en avant entre la complexité des informations à représenter et notre incapacité à saisir cette complexité dans sa globalité.

7.7. Vers une classification dynamique : bilans et perspectives

7.7.1. La PàP : une nouvelle approche pour le TALN?

La PàP remet en question la notion de représentation vue comme un processus qui ne peut représenter qu'un monde prédéfini. Ce processus de représentation construit progressivement les entités informatiques suivant les connaissances dont on dispose sur un domaine visé : ce processus de représentation peut donc être défini de manière continue (dans le temps) en tenant compte des informations disponibles pour affiner les structures déjà construites sans avoir à reconstruire entièrement de nouvelles structures.

- PàP et Sous-Langages

"J.McNaught (1993) présente une critique sévère de la tendance à vouloir développer des systèmes généralistes, fondés sur des grammaires et des lexiques généraux, que l'on entend spécialiser par ajouts d'éléments lexicaux et de règles" (Péry-Wooodley 1995). A l'image de ce que propose McNaught, notre démarche de représentation est ascendante. Elle vise à construire des niveaux de généralisation à partir de descriptions qui s'appuient sur l'analyse de corpus de textes spécialisés.

- Classement dynamique et généralisation : Mettre à jour les savoirs communs puis construire des généralisations.

Le processus de représentation peut se dérouler en créant "à la volée" (manuellement ou automatiquement) les objets qui vont porter les savoirs disponibles. Il reste possible ensuite de réorganiser les objets construits si des savoirs communs existent.

L'automatisation de la phase de généralisation ou de classement reste malgré tout délicate. Il faut disposer d'un méta-regard sur les entités créées pour y détecter des analogies de structures ou de comportements. Cette méta-analyse d'une domaine de connaissances représentées avec la PàP est potentiellement disponible avec Self qui dispose d'outils qui agissent à un méta-niveau et qui permettent par exemple de selectionner tous les objets construits avec certains types d'attributs.

- Héritage dynamique

La délégation est un mécanisme qui permet de définir des liens d'héritage entre les objets construits : ces liens peuvent être définis dynamiquement; il est possible à tout moment de définir ou de reconfigurer des liens de délégation entre objets (ajout, retrait, modification).

7.7.2. Un dispositif expérimental guidé par le linguiste

- Dans l'état actuel du classement effectué, une intervention manuelle est semble-t-il indispensable pour ce travail d'interprétation en amont et en aval de la phase de représentation des structures représentées : le système construit s'inscrit dans une approche qui vise à établir un dialogue entre le dispositif informatique construit et les problèmes posés par les faits linguistiques étudiés. L'intervention manuelle du linguiste s'inscrit parfaitement dans une telle approche. Dans la mesure où les dispositifs informatiques (et en particulier le nôtre) ne sont pas encore capables de résoudre tous les problèmes posés par le TALN, un dispositif informatique qui permet une interaction dynamique entre l'utilisateur privilégié de ce type d'outils et les résultats produits ou les données traitées correspond au préalable définitoire que nous avions posé pour la mise au point d'un tel dispositif.

- Le classement à mettre en place est fortement dépendant des informations reçues en amont. Il reste donc indispensable de s'interroger sur la nature des informations linguistiques qui seront manipulées pour ce classement. Comme on l'a dit précédemment, les informations recueillies actuellement en amont doivent être renforcées et affinées. Notre travail doit donc préciser quelle est la nature des informations à traiter et suivant les informations manipulées, quel type de classement on obtient, et quel type de classement on veut ou on peut obtenir sur la base de ces informations. Il s'agit donc de mettre en avant les limites du classement automatique obtenu suivant les critères de classement retenus. En précisant les limites de notre travail, d'une part on ne surestime pas les résultats obtenus, d'autre part, on garde en point de mire la trace de ce qu'il reste à faire.

- Les limites de l'automatisation marquent le champ de travail qu'il reste à effectuer manuellement pour mener à bien le classement des unités lexicales. Il semble clair que ce travail d'affinement des représentations obtenues va éclairer et guider la mise au point des éléments de savoirs pertinents que l'on souhaite mettre à jour pour aller vers une automatisation de plus en plus fine du classement escompté. Cette démarche qui vise à définir des outils de classement et les modèles de représentation par touches successives s'accommode assez bien avec le modèle de représentation des connaissances retenu. La PàP encourage une approche de représentation faite de petits sauts successifs qui améliore la qualité de la représentation produite. D'autre part, cette démarche s'accorde aussi avec la nécessaire approche artisanale que constitue le travail du linguiste dans sa volonté de décrire les comportements des faits de langue.

Incise Chapitre 7

Incise 1. Self et l'héritage

Self permet l'héritage multiple et l'héritage dynamique.

On revient ici sur la présentation de l'héritage avec Self et notamment sur la notion de délégation. On construit des hiérarchies locales de contraintes syntaxiques partagées par un ensemble de prototypes lexicaux. Il est donc nécessaire de pouvoir retrouver et utiliser les liens de délégation définis pour adresser les messages pertinents aux objets adéquats. En effet, les processus de génération et de classement mettent automatiquement en place des liens de délégation entre les objets. Ces liens de délégation ne sont pas figés de manière définitive : le processus de classement ajoute par exemple un lien de délégation entre une sous famille de mots et un pôle de comportements partagés. Il est donc important de pouvoir sélectionner le destinataire d'un message si un objet donné délègue des comportements à différents parents : un mot peut déléguer des comportements (des arbres) à plusieurs parents, il convient donc de pouvoir retrouver les différents chemins possibles pour récupérer le comportement adéquat.

I.1.1. Héritage non-ordonné

Self utilise une notion d'héritage multiple désordonné et reconfigurable.

- Il n'est pas possible depuis la version 3.0 de Self d'affecter aux parents d'un objet donné un rang de priorité. Avant cette version, il était en effet possible d'ordonner les liens de délégation. Ce changement de perspective oblige à une attention particulière dans la définition des liens de délégation si l'on ne veut pas produire de l'ambiguïté.

- En effet deux parents directs d'un objet donné ont un même rang de priorité; ils ne sont donc pas ordonnés hiérarchiquement, et sont donc accessibles l'un et l'autre sans préférence. On peut donc facilement générer des messages ambigüs si un ou plusieurs attributs de ces parents portent les mêmes noms. Il peut donc sembler nécessaire de savoir précisement à qui on souhaite adresser un message si l'on veut être sûr de ne pas produire ce type d'erreur.

- On verra cependant infra que les méta-processus disponibles avec Self fournissent des outils pour une description précise des prototypes construits. On peut ainsi sélectionner les objets à qui on adresse un message de manière non ambigue.

I.1.2. Héritage multiple

Self ne restreint pas le nombre de lien de délégation entre les objets. Dans la mesure où il possible d'ajuster dynamiquement les prototypes (ajout ou retrait d'attribut), on peut à tout moment construire des liens de délégation.

I.1.3. Règles de précédence dans l'héritage

L'héritage multiple à la Self s'appuie sur les règles suivantes :

- Si un objet et l'un de ces ancêtres définissent un attribut avec le même nom, c'est l'attribut de l'objet qui sera pris en compte.

- Si deux parents de ce même objet définissent un attribut avec le même nom, Self génère un message d'erreur d'accès à l'attribut (`messageAmbiguous`).

- "Sender Path Tiebreaker Rule" : Les règles d'héritage de Self spécifient que si deux attributs avec le même nom sont définis dans des parents de même rang de priorité, mais un seul de ces parents possède un lien d'héritage avec l'objet qui contient la méthode qui a émis le message, alors c'est l'attribut de ce parent qui est pris en compte.

I.1.4. Renvoi (dirigés) de messages

I.1.4.1. Receveur implicite

On dispose aussi en Self d'un mécanisme de renvoi de message qui permet de diriger un message à un et seul parent d'un objet donné au lieu de l'adresser à l'ensemble. Ce renvoi dirigé peut être utilisé pour résoudre des problèmes d'ambigüité entre les différents parents possibles d'un objet donné. Syntactiquement, ce message de renvoi est spécifié en préfixant le nom du message avec le nom de l'attribut du parent auquel ce renvoi est dirigé. Ce mécanisme est analogue au mécanisme général de renvoi utilisé par Self. Ce dernier permet à une méthode d'invoquer l'appel à un attribut couvert (du fait de la précédence des attributs) par l'attribut auquel est adressé le message : le renvoi de message est écrit avec un receveur implicite (ce mécanisme est similaire au `call-next-method` de CLOS (Habert 1996)).

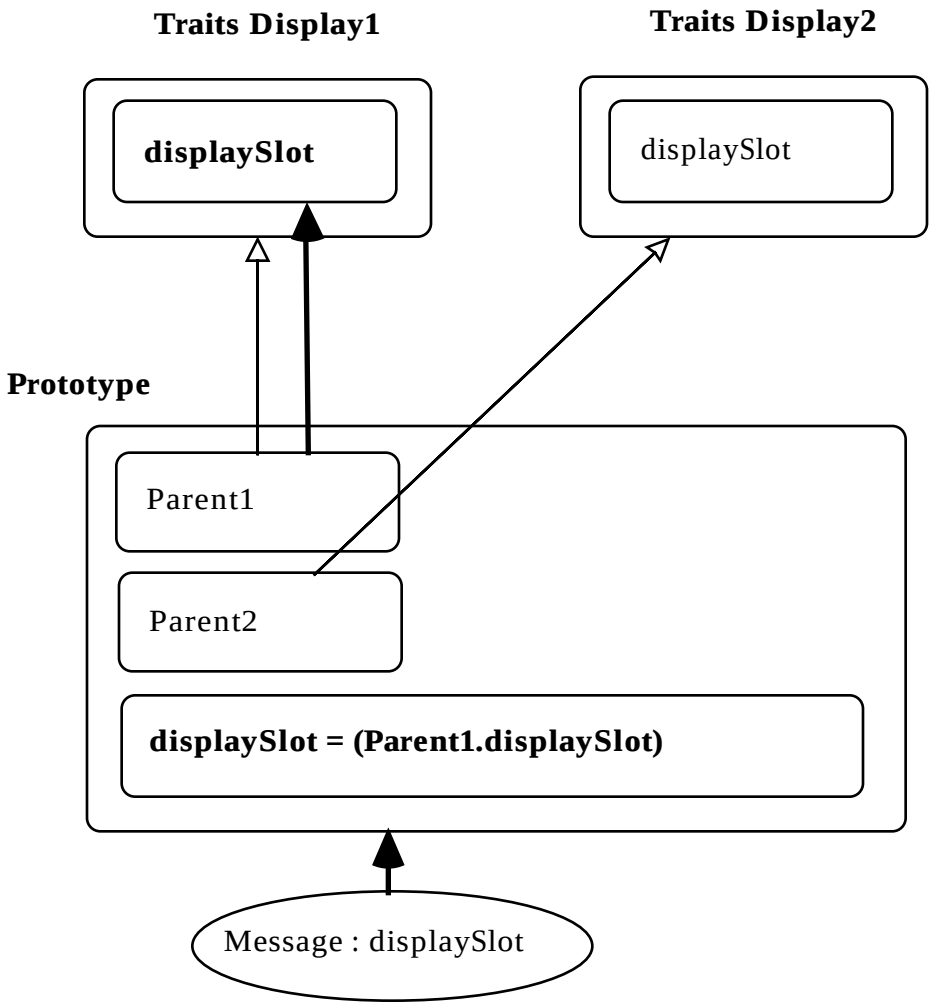


Figure 7.24 : Receveur implicite.

I.1.4.2. Receveur explicite

On dispose aussi avec Self de primitives qui permettent de "forcer" l'envoi d'un message à un parent donné en utilisant par exemple la syntaxe suivante :

```
'message' sendTo: Objet DelegatingTo: parentChoisi
```

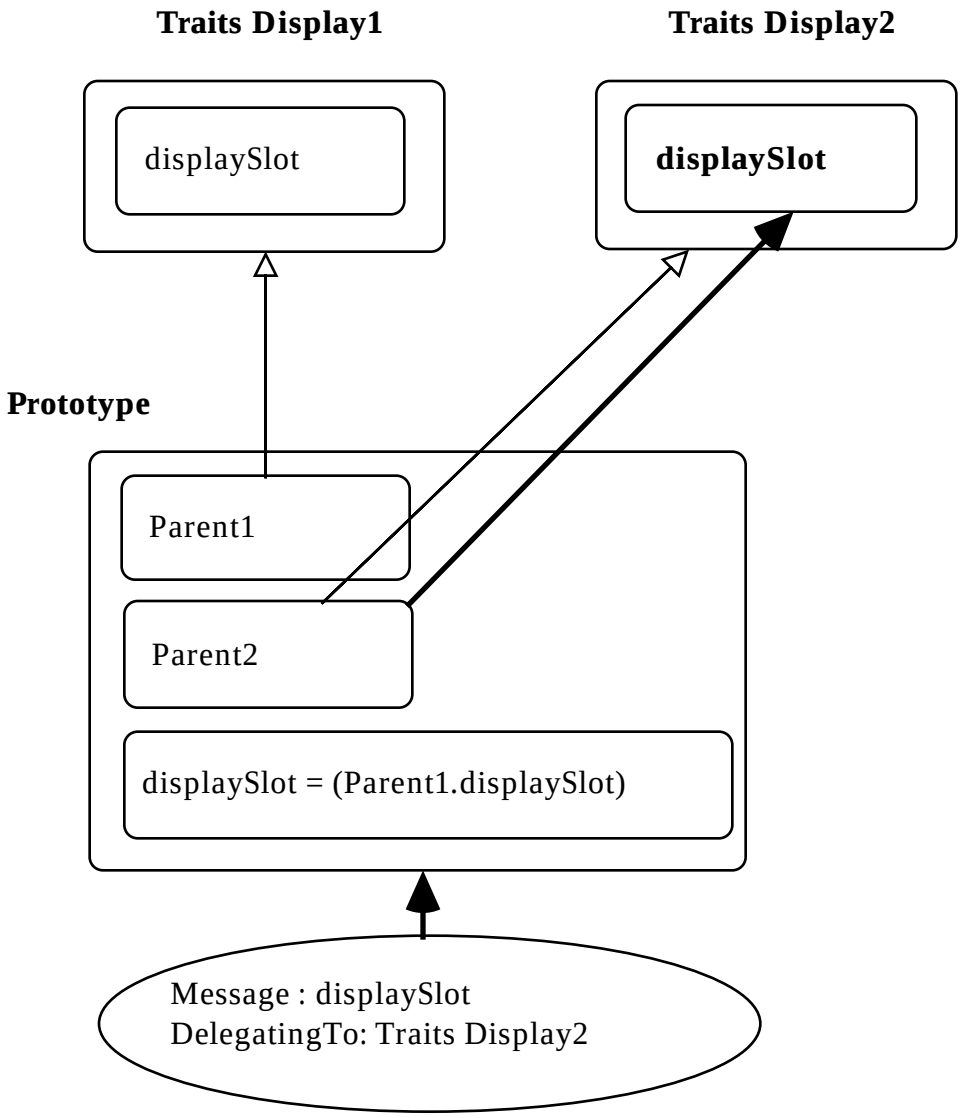


Figure 7.25 : Receveur explicite.

Incise 2. Positionnement par rapport à l'approche probabiliste

"Le pessimisme qu'il pratiquait avait toujours été de cette variété vigoureuse que l'on trouve en Italie : comme Machiavel, il plaçait l'equilibre entre virtus et fortuna aux alentours de fifty/fifty; or voilà que les équations introduisaient un élément aléatoire qui donnait un taux de probabilité si indéterminé, si indicible, qu'il craignait même de le calculer."
Thomas Pynchon, "L'homme qui apprenait lentement (Entropie)", Ed. du Seuil.

I.2.1. Probabilisation des modèles syntaxiques (Rajman 1995)

I.2.1.1. Approche probabiliste pour l'analyse syntaxique

La production de modèles syntaxiques généraux pour les langues naturelles se révèle souvent être une tâche délicate voire difficile à réaliser de manière totalement satisfaisante. La production de ces modèles se heurte, par exemple, à des problèmes de couverture linguistique (permissivité trop grande ou trop restrictive) ou d'ambiguïté d'analyse (ce qui peut conduire à une extension considérable de structures syntaxiques particulières...). Pour pallier ces difficultés, la probabilisation des modèles syntaxiques vise à remplacer lors de la définition de la mise en correspondance entre séquences de mots et structures syntaxiques, la relation d'association binaire définie par tout modèle syntaxique sur l'ensemble des couples (séquences de mots, structure syntaxique) par une probabilité p définie sur ce même ensemble.
L'association entre une séquence W et une structure s n'est plus alors caractérisée en termes booléens mais mesurée par la probabilité conditionnelle suivante :

$$P(s/W)= p(W,s) \ / \ \Re_u \ p(W,u)$$

Cette approche permet de dissocier l'étape de détermination des structures syntaxiques potentiellement associables avec une séquence donnée et l'étape d'évaluation, pour chacune des structures candidates, de la qualité de l'association effective. Cette approche vise aussi à soulager la phase de construction de modèle syntaxique en laissant cette modélisation relativement simple en ce qui concerne la détermination des structures candidates; elle permet aussi réaliser une adéquation fine de la mise en correspondance par une adaptation bien choisie des probabilités. De plus, si l'on travaille sur des corpus, la détermination des probabilité à utiliser peut s'effectuer à partir de décomptes fréquentiels d'occurrences réalisées dans ces corpus.

I.2.1.2. Schéma Global de probabilisation d'un modèle syntaxique

- Phase 1 : détermination des séquences incorrectes.

$$\forall W, \ W \text{ syntaxiquement incorrecte ssi } \forall s, \ P(s/W) = 0$$

i.e la probabilité d'une mise en correspondance avec une structure syntaxique quelconque est strictement nulle.

- Phase 2 : probabilités des séquences acceptables.

Pour obtenir une forme utilisable pour l'expression des probabilités conditionnelles associées aux séquences syntaxiquement correctes, il est nécessaire de postuler des prpriétés supplémentaires pour le modèle syntaxique. Le plus souvent, on utilise des modèles paramétriques, où l'on fait l'hypothèse de l'existence, pour la probabilité P(s,W) d'une expression fonctionnelle de la forme :

$$P_\theta(s,W) = F(\theta(s,W),...,\theta_n(s,W))$$

où : $\theta = (\theta(s,W),..,\theta_n(s,W))$ représente les paramètres du modèle.

La mise en oeuvre du modèle probabiliste passe par la résolution de deux problèmes principaux :

- Apprentissage des paramètres du modèle : Ce qui revient à déterminer, à partir des données et des connaissances disponibles, les valeurs des paramètres du modèle "optimal".

- Evaluation des structures syntaxiques associées à une séquence donnée : Il faut en effet déterminer, pour toute séquence de mots W syntaxiquement correcte, l'ensemble des structures associées qui maximisent la probabilité conditionnelle P(s / W).

I.2.1.3. Prototypes et Comportements vs. Approche Probabiliste

Les techniques probabilistes appliquées à l'analyse syntaxique visent à concentrer l'attention sur la fréquence des mots ou des structures syntaxiques en réduisant l'impact des mots ou des structures à faible fréquence (assimilables à du bruit). Cette approche s'éloigne, sur ce dernier point en tout cas, de ce que nous cherchons à mettre en lumière. Notre travail s'attache à prendre en compte les savoirs de manière inductive et à affiner les différentes étapes de représentation pour tenir compte des évolutions potentielles des choses représentées ou de pouvoir prendre en compte de nouveaux éléments à représenter. Il semble donc délicat voir incongru de vouloir pré-affirmer le poids de savoirs non encore complètement connus et/ou représentés. Nous devons au contraire traiter et enregister les savoirs comme des entités de même "niveau de validité"; on se place ici d'ailleurs dans une approche voisine de celle de (Bensch & Savitch 1995).

Chapitre 8

"C'est ainsi que j'ai pu apprendre que ses navets au jus sont moins bons qu'autrefois mais qu'en revanche ses carottes, au jus egalement, sont meilleures que dans le temps. Le jus n'a pas changé. C'est là un langage que je comprends presque, ce sont là des idées claires et simples sur lesquelles il m'est possible de bâtir, je ne demande pas d'autre nourriture intellectuelle."
"L'innommable", Samuel Beckett, Ed. de Minuit.

Représentation et classification dynamiques

Ce chapitre donne une présentation du dispositif mis en place pour la génération automatique des prototypes de mots et des prototypes d'arbres associés et pour leurs classements. `Gaspar` est le nom générique du dispositif chargé de construire les représentations des mots puis leur classement. L'implémentation s'inscrit dans le cadre défini par la version 4.0 du langage à prototypes Self (-> incise 3) : on dispose ainsi d'une interface graphique pour tracer, évaluer, modifier les résultats construits . Cette implémentation étend donc les outils d'analyse déjà présentés.

8.1. Un dispositif pour la représentation et pour le classement des mots et de leurs comportements

8.1.1. Un dispositif pour l'expérimentation linguistique

Les informations associées aux mots donnent une description des mots à représenter aussi bien dans leur forme que dans les interrelations que ces mots tissent avec d'autres unités de langue, et en particulier les liens qui existent entre ces mots et des structures syntaxiques associées. On donne ci-dessous une présentation du schéma général des opérations mises en place.

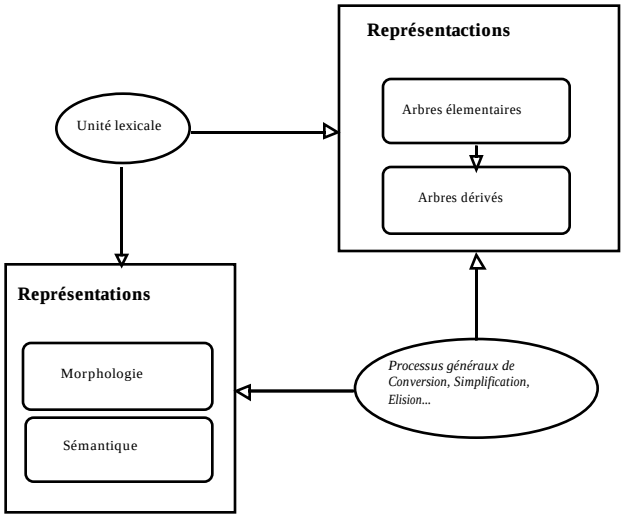


Figure 8.1 : Représentation des mots et "représentactions"[90] sur les mots

Les modifications que peuvent subir les unités lexicales dans certains contextes (glissement de sens, changement de type...) ou réciproquement, les modifications que peuvent déclencher ces mêmes unités lexicales sur d'autres unités dans un contexte donné ne sont pas formalisées de manière précise : il reste possible d'ajuster manuellement les représentations construites pour tenir compte de tels phénomènes (-> incise 1 : présentation de Self 4.0, -> affinements des représentations). De même, la détermination des processus généraux qui agissent sur le corpus donné ne sera pas traitée à ce stade de notre travail. Nous nous intéressons en premier lieu à la représentation du savoir qu'il est possible d'attacher aux unités lexicales. Nous verrons infra que le dispositif construit donne à l'utilisateur la possibilité d'intervenir directement sur les entités construites ou manipulées par le dispositif : il reste donc possible à tout moment d'ajuster la représentation des unités de langue. Le dispositif construit vise en effet à donner à l'utilisateur un rôle actif dans la manipulation des structures définies et dans l'interprétation des résultats à construire. Les résultats construits constituent des amorces d'interprétation qu'il convient d'ajuster. Ce dispositif est avant tout conçu comme un outil d'expérimentation avec lequel il est possible à tout moment d'affiner les représentations définies. Le langage Self et les différentes interfaces mises en place permettent à tout moment d'interroger et de modifier les objets construits (-> annexe partie 2 chapitres 8 et 9 : présentations des traces générées par Gaspar). Si l'examen des représentations construites ou des pôles de comportements partagés calculés révèlent des failles, des manques, des lacunes... l'utilisateur de ce dispositif peut ajuster les informations représentées par touches successives et de manière dynamique.

8.1.2. Description des informations à représenter

Pour commencer nous reprenons l'exemple traité dans le chapitre précédent. On donne ci-dessous une présentation des informations que le travail d'extraction de savoirs sur corpus génère :

```
#Dc
pontage
Nom
      #DM (Description morphologique)
      Masc
      Sing
      ...
      #FM
      #DS(Description sémantique)
      ....
      #FS
#DAE
```

[90] Représentation des actions que les mots permettent.

```
        N1PrepNPivot
        ('Nom' 'Prep' 'NomPivot')
        ->DCAE (Contraintes sur arbre élémentaire)
        <-FCAE
                #DAL
                ↑Sn
                ('Det' 'N1PrepNPivot')
                ->DCAL
                (Contraintes sur arbre lié)
                <-FCAL
                #FAL
                #DAL
                ↓NomPivot
                NAdj
                ('NomPivot' 'Adj')
                ->DCAL
                (Contraintes sur arbre lié)
                <-FCAL
                #FAL
                #DAL
                *N1PrepNPivot
                ('Adj' 'N1PrepNPivot')
                ->DCAL
                (Contraintes sur arbre lié)
                <-FCAL
                #FAL
#FAE
#DAE
        NAdj
        ('Nom' & 'Adj')
        ->DCAE
        (Contraintes sur arbre élémentaire)
        <-FCAE
                #DAL
                ↑Sn
                ('Det' 'NAdj')
                ->DCAL
                (Contraintes sur arbre lié)
                <-FCAL
                #FAL
                #DAL
                ↓Adj
                SAdj
                ('Adj' 'AdjPivot')
                ->DCAL
                (Contraintes sur arbre lié)
                <-FCAL
                #FAL
#FAE
#DAE
....
#FAE
Fc#
```

On peut lire ce qui précède de la manière suivante :

- le mot `pontage` peut entrer dans un arbre élémentaire `N1PrepNPivot` qui lui-même peut générer un arbre d'analyse de type `Sn` et un arbre auxiliaire (<- chapitre 3 : TAGs) `*N1PrepNpivot`;

- il est possible d'adjoindre dans l'arbre élémentaire de type `N1PrepNPivot` un arbre de type `NAdj` au niveau du composant `NomPivot`;

- de même, le mot `pontage` peut entrer dans un arbre élémentaire `NAdj` qui lui-même peut générer un arbre d'analyse de type `Sn`;

- il est possible d'adjoindre dans l'arbre élémentaire de type `NAdj` un arbre de type `SAdj` au niveau du composant `Adj`;

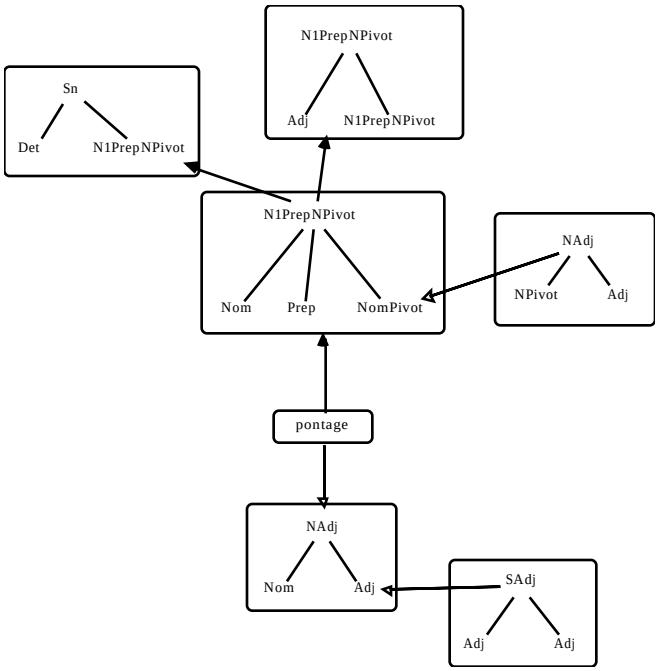


Figure 8.2 : `pontage` et arbres associés

ce qui peut aussi s'écrire :

```
Arbre Elémentaire :

                    pontage => N1PrepNPivot -> Nom Prep NomPivot

Arbre d'analyse montant :

                    N1PrepNPivot => Sn -> Det N1PrepNPivot

Arbre auxiliaire :

                N1PrepNPivot => *N1PrepNPivot -> Adj N1PrepNPivot

Arbre d'analyse par adjonction : (dans N1PrepNPivot sur NomPivot)

                    NomPivot => NAdj -> NomPivot Adj

Arbre Elémentaire :

                    pontage => NAdj -> Nom Adj

Arbre d'analyse montant :

                    NAdj => Sn -> Det NAdj

Arbre d'analyse par adjonction : (dans NAdj sur Adj)

                    Adj => SAdj -> Adj AdjPivot
```

A partir de ces informations, le dispositif peut générer à la volée les arbres élémentaires associés aux mots et les arbres liés aux arbres élémentaires. Il établit ensuite les liens qui existent entre les objets créés.

De manière plus générale, la phase d'extraction de savoirs sur corpus produit un fichier texte qui a l'allure suivante :

```
#Dc
entrée lexicale
Catégorie
   #DM
   (Description morphologique)
   #FM
   #DS
   (Description sémantique)
   #FS
#DAE
   (Arbre élémentaire associé à cette entrée lexicale)
   ->DCAE
   (Contrainte sur arbre élémentaire)
   ->FCAE
          #DAL
          (Arbre d'analyse associé à cet arbre élémentaire)
                 ->DCAL
                 (Contrainte sur arbre d'analyse)
                 ->FCAL
          #FAL
#FAE
#DAE
....
#FAE
Fc#
```

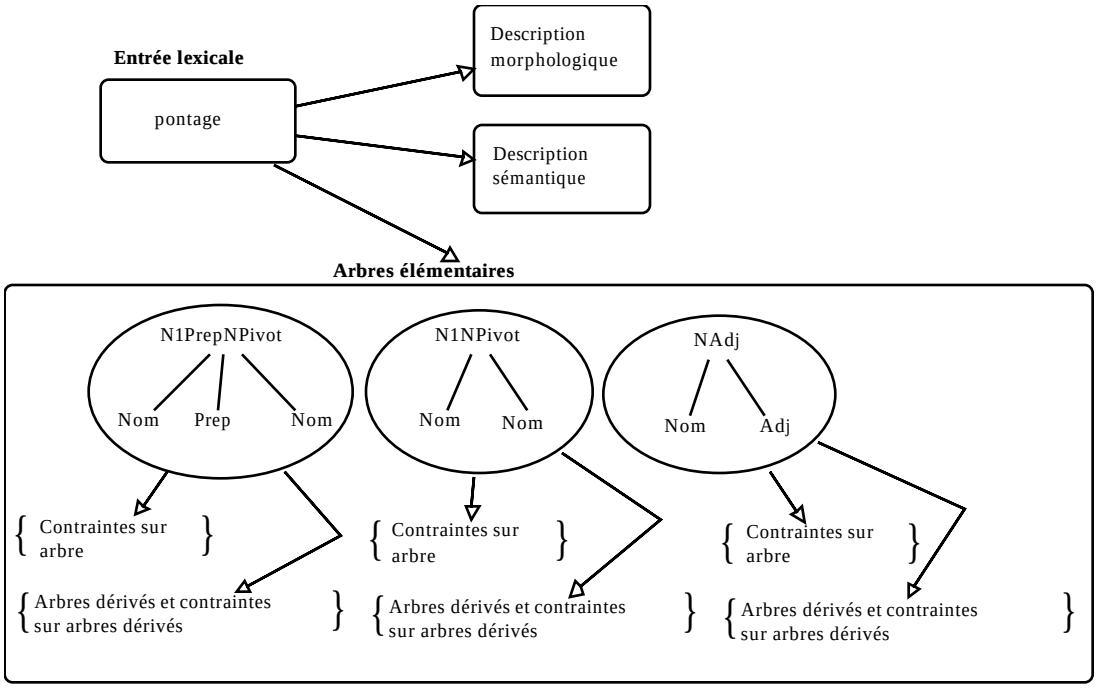


Figure 8.3 : Description des informations associées à `pontage`

A chaque entrée lexicale `#Dc-#Fc`, on associe les informations suivantes :

- une description morphologique de l'entrée lexicale `#Dm-#Fm`,
- une description sémantique de l'entrée lexicale `#Ds-#Fs`,

- une liste d'arbres élémentaires associés à l'entrée lexicale {#DAE-#FAE}. On peut aussi associer à chaque arbre élémentaire une liste de contraintes {#DCAE-#FCAE} et une liste d'arbres d'analyses {#DAL-#FAL}.

La phase de génération est conçue comme un processus évolutif. Si ces informations ne sont pas disponibles à une certaine phase de la représentation, rien n'empêchera d'affiner les représentations construites dès que ces informations seront établies ou disponibles. La détermination de celles-ci pourra se faire par projections des savoirs construits sur des savoirs définis par ailleurs ou en affinant la phase d'extraction de savoirs sur corpus avec là aussi projection sur des classes de savoirs qui précisent ce type d'informations. Il s'agit en fait de définir un format des informations que l'on peut recevoir pour la génération des prototypes associés aux mots : si les informations sont présentes, on pourra les représenter; si elles ne le sont pas, on attendra d'en disposer pour les représenter.

8.1.3. Mise en forme des connaissances

Phase 0 : Génération de prototypes pour "porter" les savoirs attachés à une entrée lexicale.

Un programme (écrit en C) est chargé de traiter le fichier précédent afin de le formater pour être lisible dans un environnement Self. Celui-ci crée une série de prototypes qui vont porter les informations lues. Pour chaque entrée lexicale, on construit un prototype qui a l'allure suivante :

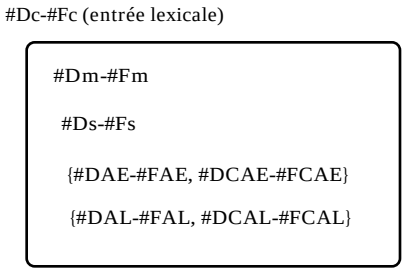


Figure 8.4 : Description des informations en entrée

Chacun des objets ainsi créé est enregistré dans un dictionnaire.

8.1.4. Une construction inductive des savoirs

8.1.4.1. Génération dynamique des prototypes lexicaux et des prototypes d'arbres

On reprend dans le schéma qui suit les différentes étapes du traitement mis en place pour la génération des mots et et arbres sous la forme de prototypes.

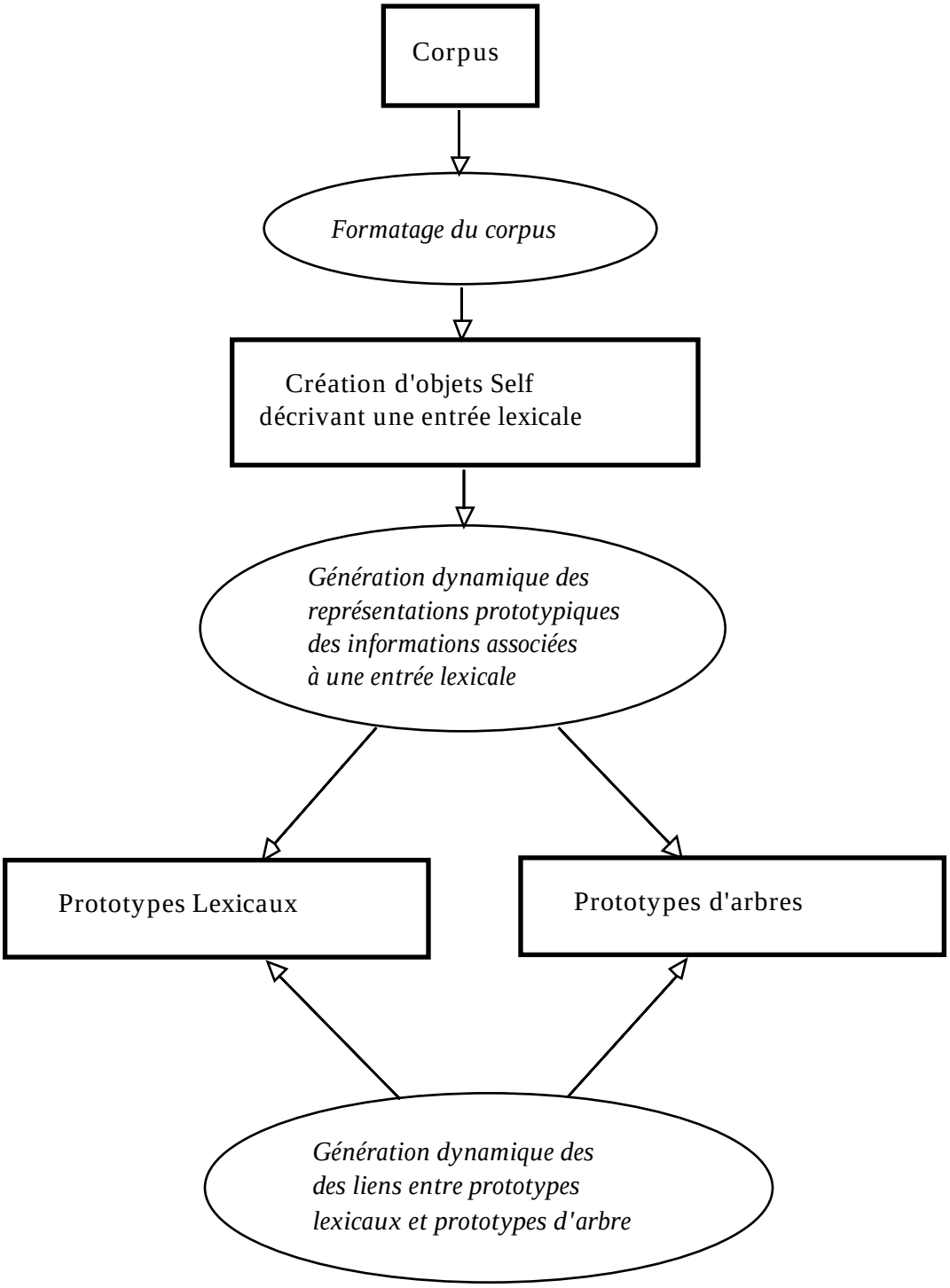


Figure 8.5 : Déroulement de la génération

A partir des objets précédents associés aux entrées lexicales, on construit les représentations prototypiques des informations contenues dans ces objets.

Phase 1 : Génération du prototype lexical en utilisant les informations décrivant la morphologie et la semantique du mot.

La génération d'un prototype lexical est réalisée de la manière suivante :

- Si le mot à représenter possède une représentation prototypique, on conserve l'objet trouvé.

- Si le mot à représenter ne possède pas de représentation prototypique et s'il existe déjà une représentation prototypique d'un élément de la même famille catégorielle, on utilise les opérations de clonage et d'ajustement pour représenter ce nouvel élément (en tenant compte des informations fournies pour décrire ce nouvel élément).

- Si le mot à représenter ne possède pas de représentation prototypique, et s'il n'existe aucune représentation prototypique de sa famille catégorielle, on commence par créer de toutes pièces une représentation prototypique de cette nouvelle famille catégorielle, puis on construit une représentation prototypique de ce nouveau représentant de cette famille (en tenant compte des informations fournies pour décrire ce nouvel élément).

Phase 2 : Génération des prototypes d'arbres élémentaires associés à chacun des mots.

Tous les arbres élémentaires créés sont enregistrés dans un dictionnaire. Et on utilise ce dictionnaire comme base de référence pour les arbres élémentaires disponibles. Si on a besoin d'un arbre non référencé dans ce dictionnaire, on le créé avec les outils adéquats.

- Si l'arbre élémentaire à représenter possède une représentation prototypique, on conserve l'objet trouvé.

- Si l'arbre élémentaire à représenter ne possède pas de représentation prototypique et s'il existe déjà une représentation prototypique d'un élément de la même famille catégorielle (avec le même type de composants), on utilise les opérations de clonage et d'ajustement pour représenter ce nouvel élément (par exemple, ajustements des contraintes sur les composants de l'arbre).

- Si l'arbre élémentaire à représenter ne possède pas de représentation prototypique, et s'il n'existe aucune représentation prototypique de sa famille catégorielle, on commence par créer de toutes pièces une représentation prototypique de cette nouvelle famille catégorielle, puis on construit une représentation prototypique de ce nouveau représentant de cette famille (en tenant compte des informations fournies pour décrire ce nouvel arbre).

Phase 3 : Génération des prototypes d'arbres d'analyses associés à chacun des arbres élémentaires.

On procède de la même manière que pour les arbres élémentaires.

Phase 4 : On construit ensuite les liens entre les prototypes lexicaux et les prototypes d'arbres élémentaires qui leurs sont associés.

Phase 5 : De même pour les liens entre les arbres d'analyses et les arbres élémentaires auxquels ils sont associés.

8.1.4.2. Génération du dictionnaire

Dans la phase 1, on utilise les outils déjà présentés (<- partie 2 chapitre 7) pour la génération de prototypes représentant des unités lexicales : on utilise les savoirs morphologiques et sémantiques fournis pour construire les représentations prototypiques des mots. Tous les prototypes créés, représentant des entrées lexicales, sont enregistrés dans un dictionnaire. Cette phase correspond donc à une construction dynamique du dictionnaire des entrées lexicales connues par le programme. On dispose en fait de plusieurs dictionnaires dont les contenus présentent les savoirs construits suivant des critères de recherche précis.

8.1.4.3. Génération d'un prototype de grammaire

Dans les phases 2 et 3, on utilise un générateur d'arbres similaire à celui déjà présenté (<- partie 2 chapitre 7) à la différence qu'il peut désormais construire des arbres auxiliaires (<- chapitre 3 : TAGs) utilisables dans les opérations d'adjonction (-> génération automatique d'arbres). Cette phase correspond à une construction dynamique d'une grammaire lexicalisée implicite qui associe à chaque entrée lexicale une suite d'arbres élémentaires et d'analyse. Tous les arbres créés sont enregistrés dans des dictionnaires correspondant à leur nature. La grammaire produite n'est donc pas construite explicitement mais elle se définit par les liens qui existent entre les prototypes lexicaux et les prototypes d'arbres créés.

8.1.4.4. Ajustement des bases de savoirs

Dans la mesure où il est possible à tout moment de modifier les savoirs initiaux attachés aux entrées lexicales, un ajustement des savoirs attachés à une entrée lexicale doit se propager sur les différentes représentations prototypiques associées à cette entrée lexicale. Il suffit donc :

- soit de modifier les objets issus de la lecture de corpus et de réitérer les Phases 0 à 5 (dans ce cas la phase 1 ne regénère pas le prototype lexical mais ajuste les représentations existantes). De même les phases 2 et 3 ne regénèrent que les entités non encore reconnues et ajustent les autres si nécessaire.

- soit de modifier dynamiquement les représentations prototypiques créées.

Si on doit par exemple modifier les contraintes associées à un arbre (élémentaire ou d'analyse), il est possible de modifier directement le prototype d'arbre contenu dans le dictionnaire correspondant.

On donne en annexe (-> annexe partie 2 chapitre 8) une trace de cette phase de génération des prototypes de mots et des contraintes associées. De plus une présentation complète des outils mis en place est faite elle aussi en annexe (-> incise 3).

8.1.5. Le réseau de prototypes dévoile une hiérarchie évolutive

8.1.5.1. Repérer les comportements partagés par les prototypes lexicaux

Si on reprend l'exemple précédent, l'approche retenue ici conduit à la mise en place des opérations schématisées ci-dessous. On dispose au départ d'une description textuelle des informations associées à un mot. A partir de celles-ci, on construit les prototypes pour représenter le mot et les arbres associés.

Le schéma ci-contre est une illustration partielle du micro-réseau de relations et de prototypes qui est défini sur une entrée lexicale. Si on poursuit plus avant ce travail de représentation sur de nouvelles unités lexicales, il est clair que l'on disposera de nouveaux points de vue sur les unités représentées. Notamment pour évaluer les liens qui unissent ces unités sur la base des comportements qu'elles partagent.

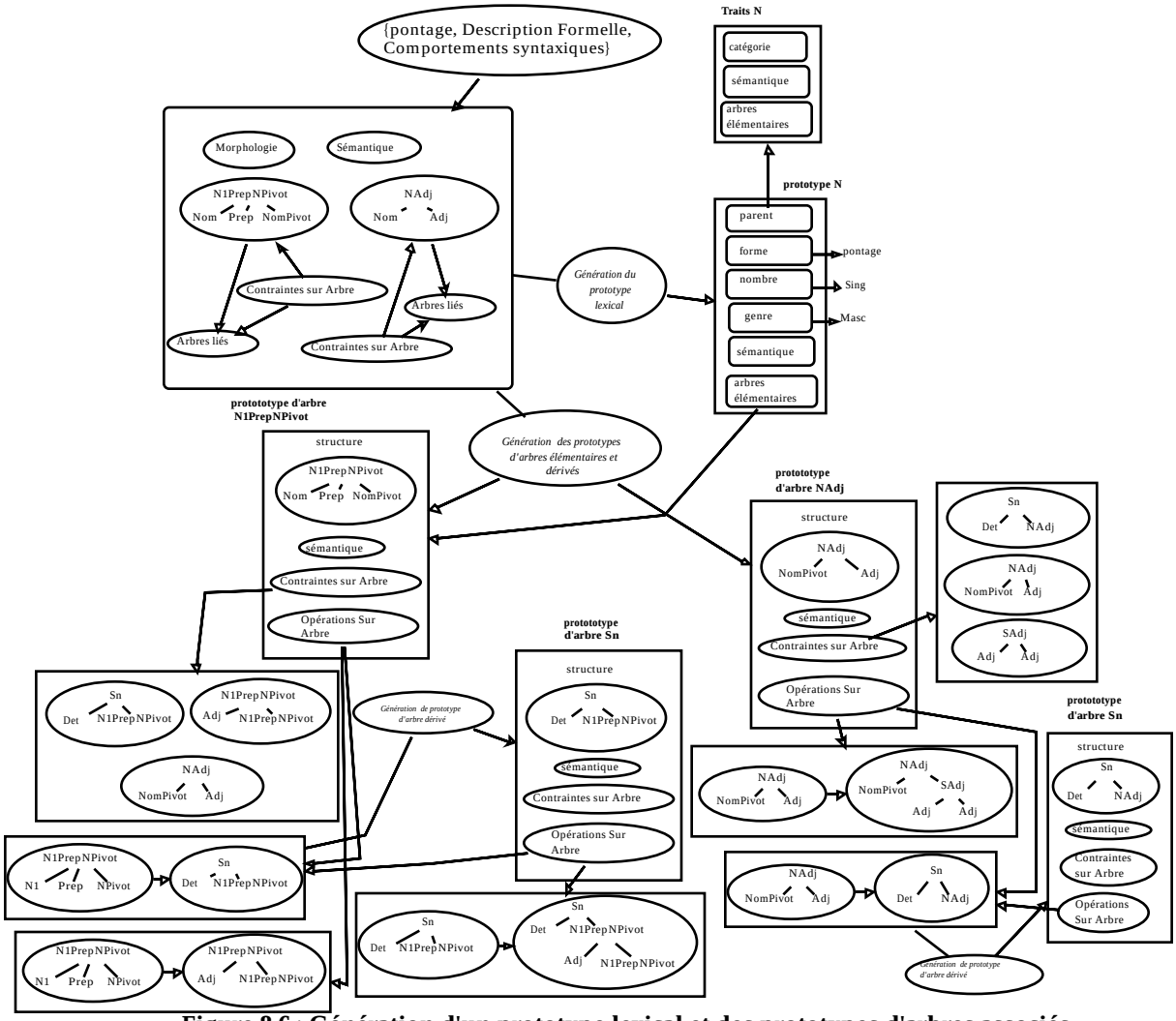


Figure 8.6 : Génération d'un prototype lexical et des prototypes d'arbres associés

A l'image de ce que nous faisions précédemment, nous pouvons désormais affiner notre travail de classement automatique des unités lexicales représentées sur la base de leurs comportements. Les savoirs attachés aux prototypes lexicaux se composent maintenant d'une liste d'arbres élémentaires (avec des contraintes possibles sur les noeuds de l'arbre) et pour chacun de ces arbres élémentaires d'une éventuelle liste d'arbres d'analyses (avec là aussi des contraintes possibles). Le classement construit se déroule de la manière suivante :

- Dans un premier temps, on cherche à repérer sur les prototypes lexicaux construits les comportements partagés (i.e les arbres élémentaires communs); on construit ensuite un pôle qui va porter ce savoir partagé.

Si on considère les noms `stenose` et `lesion`, ils partagent des savoirs et des comportements. Si on considère maintenant le nom `angioplastie`, celui-ci entre dans des constructions du type `"indication de angioplastie"`. Des arbres élémentaires sont communs à un sous-ensemble de prototypes représentant ces noms. On construit donc un pôle de comportements partagés qui va porter les arbres élémentaires communs. On établit ensuite un lien de délégation entre ce pôle et les prototypes concernés. Sur notre famille de mots comprenant les noms `stenose`, `lesion` et `angioplastie` on obtient le mini réseau suivant :

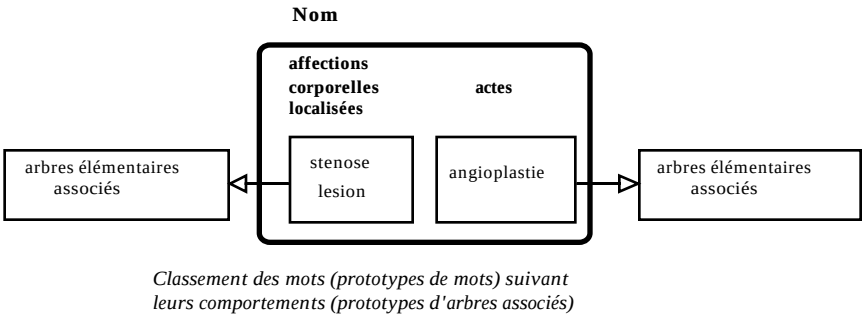


Figure 8.7 : Un mini-réseau de hiérarchies locales

- On évalue ensuite sur chaque pôle construit les extensions potentielles que les arbres associés à ce pôles permettent (-> classement des arbres élémentaires) quitte à construire de nouveaux pôles de comportements partagés par les arbres élémentaires si ces derniers mettent en avant des similarités comportementales (-> incise 4).

La figure 8.8 retrace les processus mis en place pour conduire des informations initiales au classement des unités lexicales.

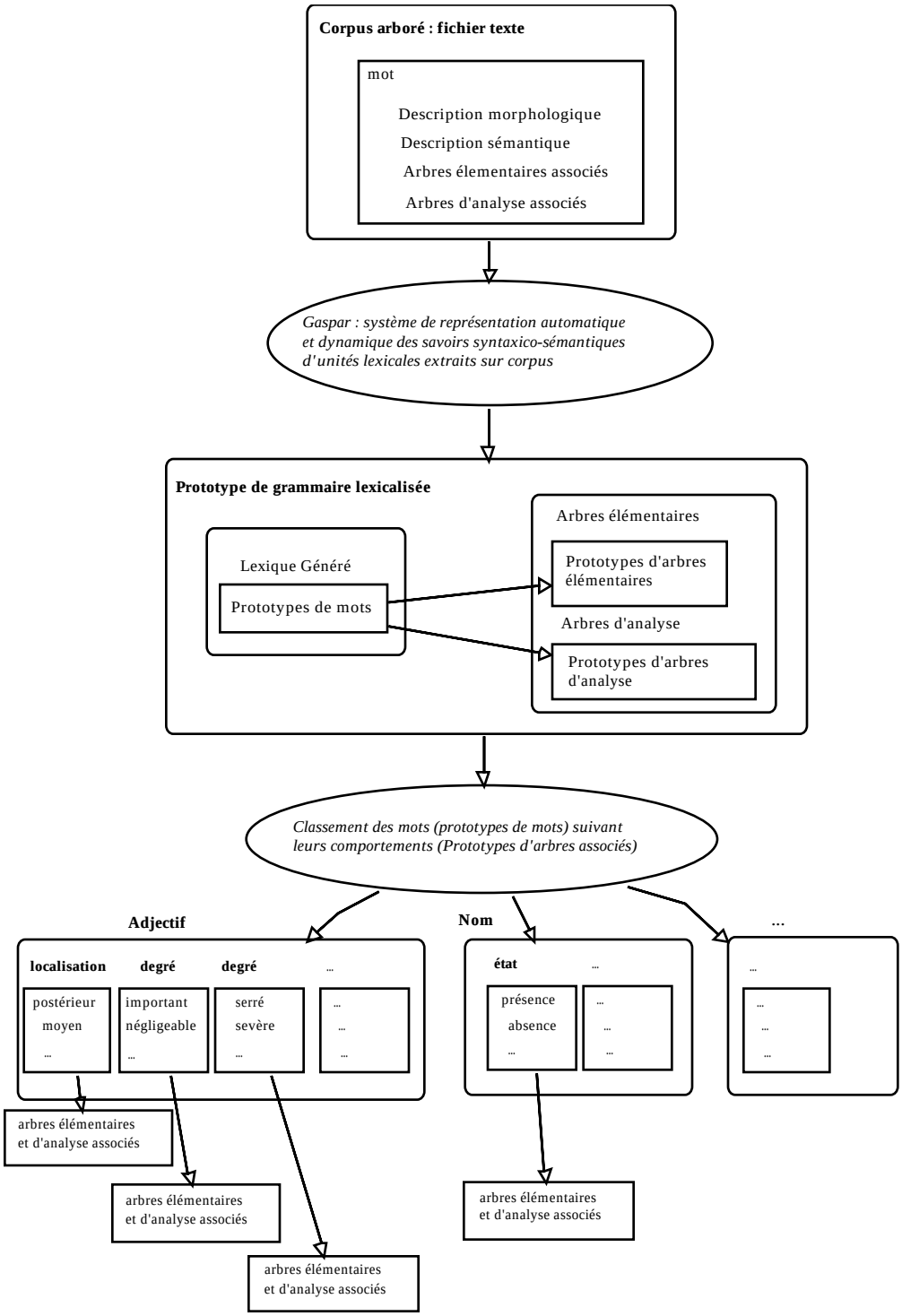


Figure 8.8 : Génération et classement des prototypes

Les étiquettes sémantiques présentes dans cette figure ne résultent pas du processus de classification mis en place. Il s'agit en fait d'un travail d'interprétation manuel fait à l'issue du classement. On a qualifié le classement obtenu en utilisant ces étiquettes.

On donne infra et en annexe (-> annexe partie 2 chapitre 8 : classement des mots et des arbres, -> expérimentations sur corpus) une trace du classement réalisé.

8.1.5.2. Rôle et utilisations des arbres élémentaires

Les arbres élémentaires : des amorces syntaxiques pour le classement des mots

Le rapport entre les arbres élémentaires (simplifiés) et les arbres d'analyses n'est pas exploité de manière définitive dans le classement des unités lexicales à partir des arbres élémentaires qui leur sont attachés. Il convient d'examiner plus précisément si les dérivations décrites par ces associations donnent des informations supplémentaires sur les unités lexicales auxquelles sont attachées ces structures syntaxiques.

arbre élémentaire <---> arbres d'analyses

Description des modifications des liens sémantiques entre les composants des arbres : glissements de sens, changement de type....

On peut par exemple examiner si le lien entre arbre élémentaire et arbres d'analyses produit du savoir. Le classement réalisé s'appuie principalement sur les arbres élémentaires associés aux mots. La phase de classement rapproche les unités lexicales de même catégorie qui partagent des arbres élémentaires. Elle indique aussi pour chaque arbre élémentaire d'un pôle construit de comportements partagés les arbres d'analyse associés à chacun de ces comportements. Mais elle ne dit rien de plus sur ces extensions potentielles que chaque arbre élémentaire peut induire : les différences comportementales des arbres élémentaires sont constatées mais ne sont pas interprétées (-> classement des arbres élémentaires). Les résultats présentés infra (-> expérimentations sur corpus) donnent une illustration du travail qu'il reste à faire pour interpréter plus avant les liens qui existent entre arbres élémentaires et arbres d'analyse associés.

Les arbres élémentaires ont une histoire syntaxique

Le rapport entre les arbres élémentaires et les arbres d'analyse associés n'est donc pas, à ce stade de notre travail, utilisé pour interpréter automatiquement les résultats construits dans le classement des mots. Il convient d'examiner manuellement si les dérivations décrites par ces associations apportent des informations supplémentaires sur les unités lexicales auxquelles sont attachées ces structures syntaxiques (-> classement des arbres élémentaires). Cet examen des liens entre arbre élémentaire et arbre d'analyse constitue une des étapes du travail manuel d'interpretation qu'il convient de faire pour évaluer les résultats construits.

arbre élémentaire <---> arbres d'analyses

Description formelle des mécanismes qui permettent de passer de l'un aux autres et réciproquement.

La phase de représentation des structures syntaxiques associées aux unités lexicales passe par un travail de formalisation des relations de dépendance qui sont mises en évidence dans la phase de simplification des arbres élémentaires. Nous disposons cependant d'une première série d'outils qui permettent de représenter dynamiquement les liens entre les arbres élémentaires associés aux unités lexicales et les arbres complexes dans lesquels ces arbres de base peuvent être insérés (-> génération automatique d'arbre).

Classement des arbres élémentaires

Le classement des mots présentés supra rapproche les unités lexicales qui partagent des arbres élémentaires mais n'utilise pas l'"histoire syntaxique" des arbres élémentaires pour qualifier les pôles de comportements construits. Le dispositif permet cependant d'evaluer automatiquement les différences

comportementales des arbres élémentaires. Il est en effet possible d'établir une recherche sur les arbres élémentaires de même catégorie des comportements partagés (arbres d'analyse) par ces arbres élémentaires. Ce classement utilise une démarche similaire à celle qui est utilisée pour classer les mots. Sur une famille d'arbre élémentaire de même catégorie, le dispositif va rechercher les prototypes d'arbres élémentaires qui possèdent des comportements similaires (i.e. les mêmes arbres d'analyse). S'il en trouve, il va construire automatiquement un pôle de comportements partagés pour les prototypes d'arbres élémentaires concernés et établir un lien de délégation entre ce pôle et les prototypes d'arbres.

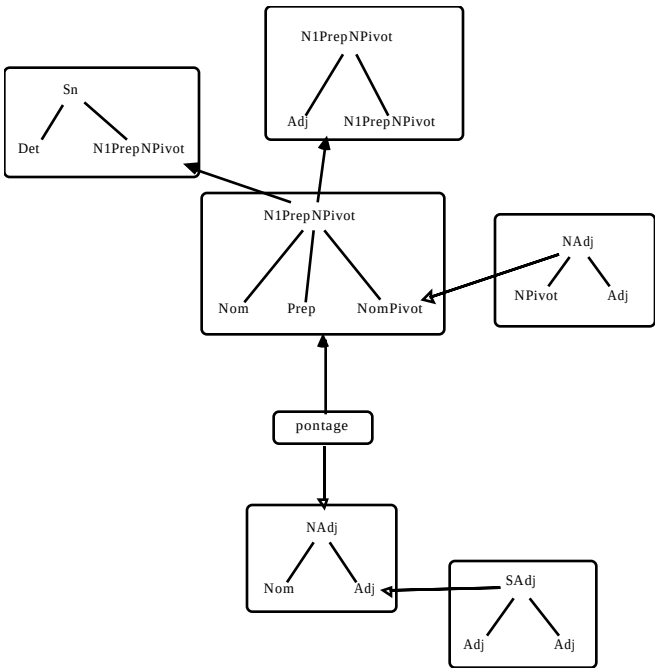
Les résultats construits sur le classement des arbres élémentaires peuvent donc être utilisé pour affiner le classement mis en place pour les mots : les arbres élémentaires associés à un pôle de comportements partagés par un ensemble de mots sont potentiellement porteurs de comportements partagés par un ensemble d'arbres élémentaires. C'est en examinant les résultats construits à la suite de ces deux phases de classement qu'il est possible de déterminer si ces délégations de comportements à double niveau produisent de nouveaux savoirs (-> incise 4).

Génération automatique d'arbres (au cours d'une analyse)

Le dispositif établit des liens entre arbres élémentaires et arbres d'analyse (ces liens sont enregistrés dans les dictionnaires construits lors de la phase de génération des arbres). On peut aussi vouloir décrire les opérations qui permettent de passer de l'arbre élémentaire aux arbres d'analyses. En effet, il est important, dans le cadre de travail développé ici, de pouvoir associer aux arbres élémentaires représentés (et donc aux prototypes associés) les processus qui permettent de construire les arbres d'analyses associés à ces arbres élémentaires. Si les arbres élémentaires à représenter sont susceptibles d'adjonction ou de substitution, on peut associer à la représentation prototypique de ces arbres élémentaires les processus qui permettent de réaliser et donc de construire les arbres obtenus par adjonction ou par substitution à partir des prototypes représentant les arbres élémentaires. On a ainsi une représentation dynamique de l'arbre élémentaire de base, celle-ci incluant les mécanismes de production dynamique des arbres d'analyses qui lui sont associés. Les représentations prototypiques des arbres élémentaires disposent ainsi d'un savoir sur les articulations potentielles que leur structure de base peut induire.

En particulier, on peut avoir besoin de produire l'arbre résultant d'une opération de substitution ou d'une opération d'adjonction sur un arbre construit (un prototype d'arbre construit est associé à une structure syntaxique dont les composants sont eux mêmes des prototypes d'arbre ou de mots construits) : c'est le cas par exemple au cours d'une analyse automatique.

Prenons un exemple : le mot `pontage` est associé aux arbres élémentaires et aux arbres d'analyse décrits dans le figure qui suit :



On doit donc pouvoir au cours d'une analyse reconstruire les arbres inscrits dans ces suites de dérivations potentielles. Pour réaliser cette génération de prototypes d'arbres, le dispositif construit utilise des opérateurs qui permettent de réaliser des opérations dynamiques sur les arbres, c'est-à-dire de construire dynamiquement les représentations prototypiques d'arbres obtenus par adjonction ou par substitution d'un prototype d'arbre sur une autre représentation prototypique d'arbre.

Construction des arbres de substitution

Pour réaliser cette génération de prototypes d'arbres construits, le dispositif construit utilise des opérateurs qui permettent de générer de nouveaux prototypes d'arbres à partir de ceux existants. Le générateur d'arbre prend en entrée les deux prototypes concernés et réalise l'opération de substitution : dans le cas d'une analyse avec le mot `pontage`, substitution de l'arbre construit `NPrepN` dans l'arbre (à construire) `Sn (Det NPrepN)`.

Construction des arbres auxiliaires pour l'adjonction

Pour l'adjonction de prototype, on dispose là aussi d'outils qui permettent de réaliser la génération de prototypes résultant de l'opération d'adjonction entre deux prototypes. L'opération se réalise de la manière suivante. On dispose d'une part d'un arbre élémentaire `Ae(i)` que l'on peut noter :

$$Ae(i)\ (X_i \rightarrow (X_{ij})\ (j=1...)$$

chacun des X_{ij} étant lui-même un arbre élémentaire. D'autre part, on dispose d'un arbre auxiliaire `Aa(k)` noté :

$$(Aa(k)\ (Y_k \rightarrow (Y_{kj})\ (j=1...))$$

et tel que Y_k soit dans $\{ X_{ij}\ (j=1...) \}$. L'adjonction peut donc se réaliser suivant le schéma ci-contre :

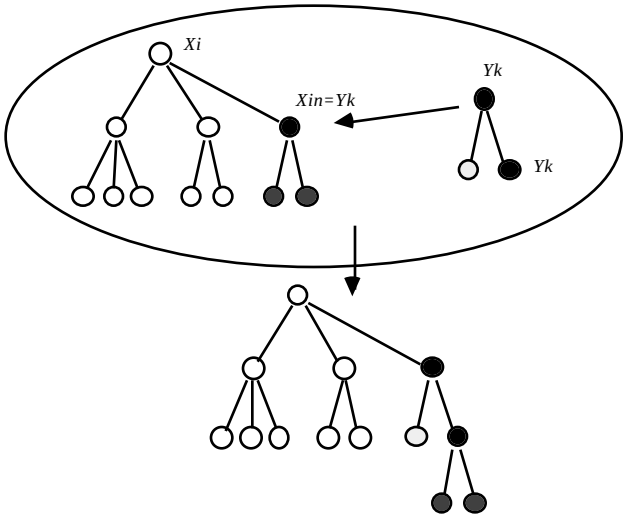


Figure 8.9 : Opération d'ajonction sur arbres

Le générateur d'arbre prend donc en entrée les deux prototypes pour mettre en oeuvre l'opération d'adjonction et réalise l'opération d'adjonction : dans le cas d'une analyse avec le mot `pontage` par exemple, ajout de l'arbre construit `SAdj` dans l'arbre construit `NAdj`.

On donne en annexe (-> annexe partie 3 chapitre 9 : analyse automatique avec `Gaspar`) une trace de la génération automatique d'arbres au cours d'une analyse.

8.2. Expérimentations sur corpus

8.2.1. Résultats construits sur un corpus de test

Le mini-corpus de test que nous utilisons contient 44 mots et une centaine d'arbres élémentaires et d'analyse. Sur ce corpus les processus de génération et de classement conduisent à la mise en place d'un réseau de prototypes dans lequel certains objets sont porteurs de comportements partagés.

Les figures qui suivent présentent des traces graphiques des prototypes générés lors du processus de génération et des pôles de comportements partagés construits. La trace dont le déroulement est donné en annexe (-> annexe partie 2 chapitre 8) ne présente que les deux premières contraintes traitées sur le corpus de test. Elle commence avec le mot pontage. Avant cette phase de travail, il n'existe aucune représentation prototypique de mot ou d'arbre. Le processus de génération doit tout d'abord créer les objets de toutes pièces puis en créer de nouveaux par clonage et ajustement. Comme il n'existe aucune représentation prototypique de la catégorie des noms, le processus construit un prototype pour représenter cette catégorie, puis un prototype pour représenter le mot. Pour traiter la deuxième contrainte, le processus construit une représentation du mot lesion par clonage et ajustement en utilisant ces prototypes initialement générés. Il en va de même pour la génération des arbres.

Sur le corpus suivant :

lesion, vraiment, pontage, de, a, avec, sur, gauche, proximal, moyen, distal, anterieur, inferieur, superieur, serre, severe, tres, peu, pas, coronaire, carotide, circonflexe, marginale, long, ulcere, complexe, inhomogene, absence, pas, risque, presence, apparition, constitution, persistance, residuel, significatif, important, negligeable, predominant, premier, double

on obtient les résultats ci-dessous.

- Dans la figure 8.10 on trouve tous les prototypes construits automatiquement par le processus de génération mis en place (tous ces objets , sur la gauche, ne sont pas expansés). On peut aussi y voir l'attribut de l'objet `Gaspar` qui porte tous les prototypes d'arbres construits (ces objets apparaissent sur la gauche). On peut ainsi accèder à ces objets via cet attribut. De même, on y voit (partiellement) un attribut de `Gaspar` portant tous les mots disponibles (`lexiqueGaspar`); les attributs nommés `proto*` correspondent aux prototypes des mots générés (ces objets apparaissent sur la gauche), et enfin on peut y voir une description détaillée du prototype construit associé au mot `pontage`.

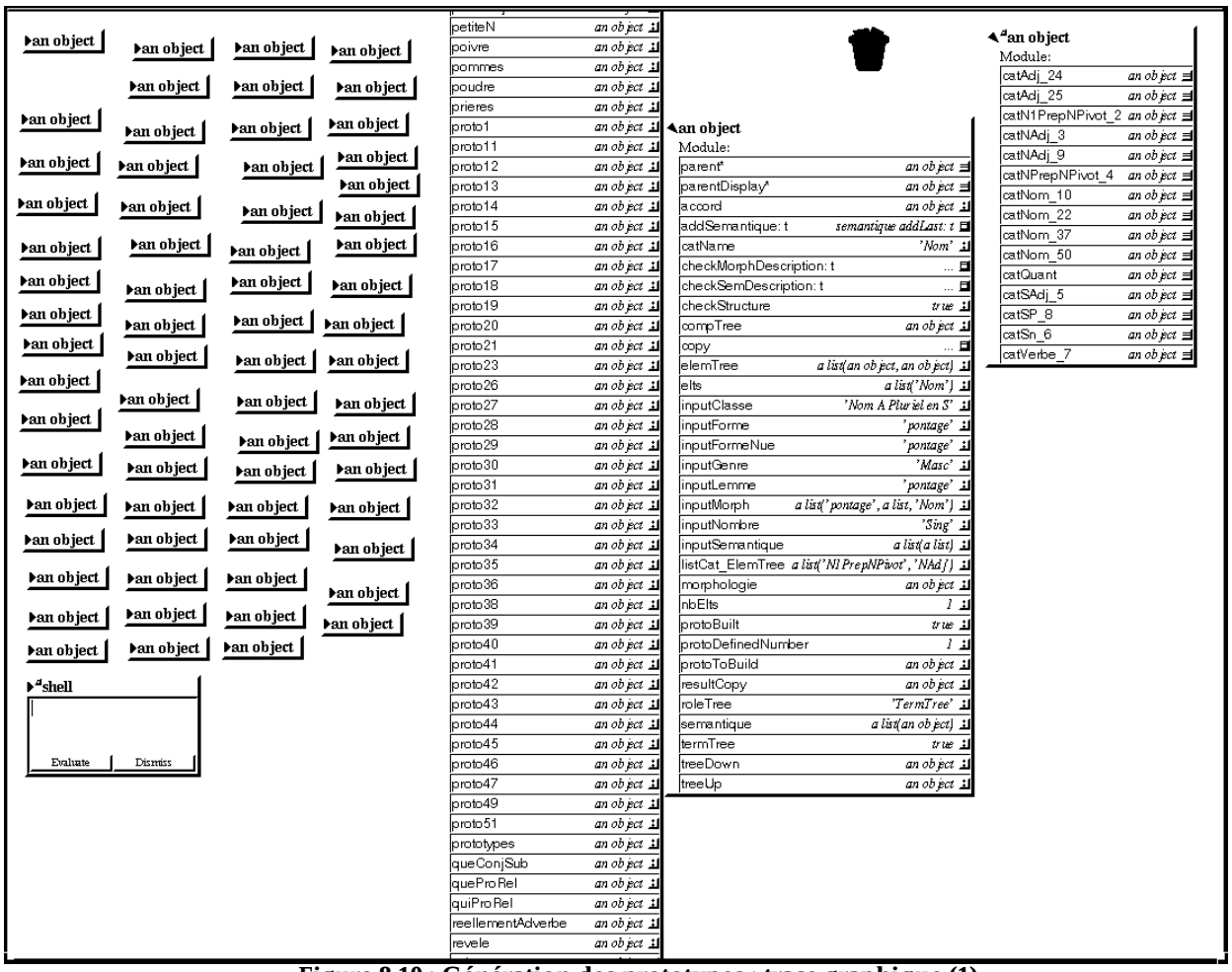


Figure 8.10 : Génération des prototypes : trace graphique (1)

- Dans la figure 8.11 on trouve les deux prototypes d'arbres élémentaires construits et associés au prototype lexical `pontage`.

Il s'agit des prototypes représentant l'arbre `NAdj` et l'arbre `N1 Prep NPivot`.

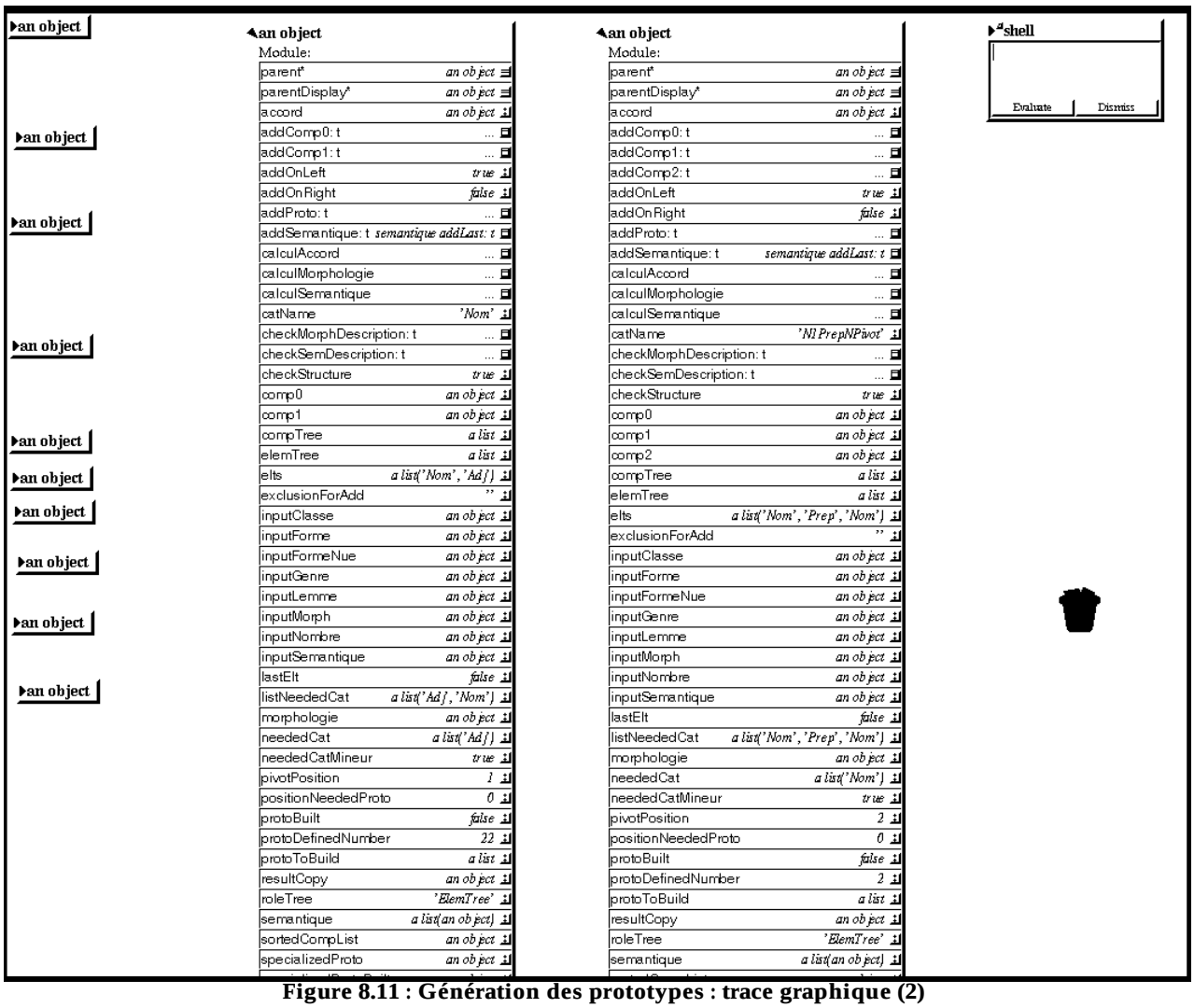


Figure 8.11 : Génération des prototypes : trace graphique (2)

- Cette troisième figure donne une trace graphique du pôle de comportement construit sur notre corpus (-> pôle n°5). Il s'agit du pôle de comportement regroupant les adjectifs `marginale`, `circonflexe`, `coronaire` et `carotide` (prototypes situés au bas de la figure). L'objet portant l'arbre élémentaire partagé est situé en haut de celle-ci.

Un certain nombre d'étiquettes sémantiques ont été associées manuellement aux prototypes lexicaux construits sur le corpus de test. Le petit nombre de mots représentés nous a permis d'ajuster les prototypes construits avec une liste limitée d'étiquettes. Ce travail reste à affiner.

```
Liste des prototypes partageant les memes Comportements :
     <Cat : Adj|Forme : carotide|Arbres elementaires : a list(an object)>
    Concept (0) : vaisseau
     <Cat : Adj|Forme : circonflexe|Arbres elementaires : a list(an object)>
    Concept (0) : vaisseau
     <Cat : Adj|Forme : coronaire|Arbres elementaires : a list(an object)>
    Concept (0) : vaisseau
     <Cat : Adj|Forme : marginale|Arbres elementaires : a list(an object)>
```

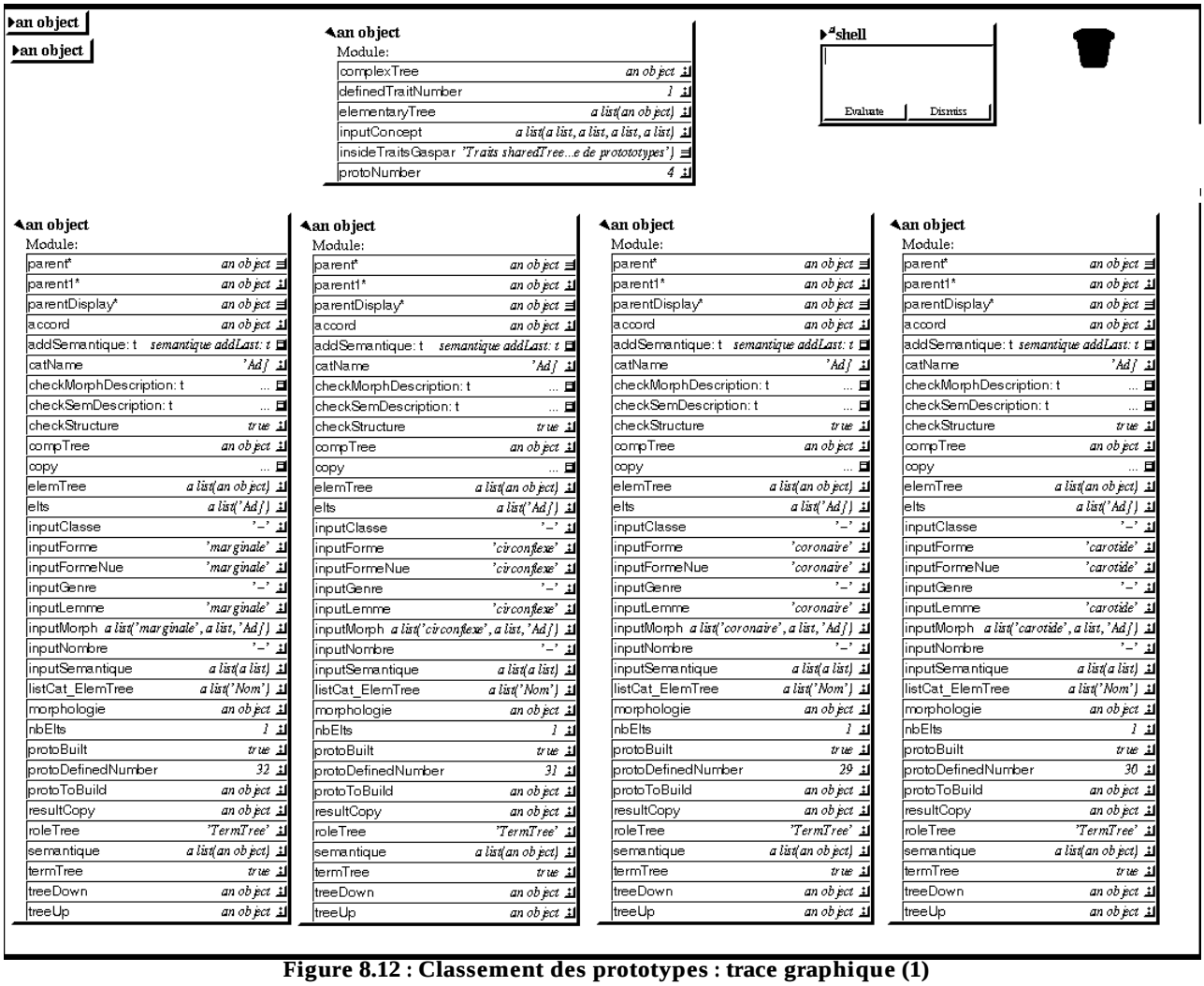


Figure 8.12 : Classement des prototypes : trace graphique (1)

- La figure 8.13 présente une trace graphique du pôle de comportement regroupant les noms `risque`, `apparition`, `persistance`, `pas`, `presence`, `absence` et `constitution` (les prototypes associés et situés au bas de la figure sont expansés). L'objet portant l'arbre élémentaire partagé est situé en haut de celle-ci.

```
Liste des prototypes partageant les memes Comportements :
     <Cat : Nom|Forme : risque|Arbres elementaires : a list(an object)>
    Concept (0) : etat
     <Cat : Nom|Forme : apparition|Arbres elementaires : a list(an object)>
    Concept (0) : proc
    Concept (1) : evt
     <Cat : Nom|Forme : persistance|Arbres elementaires : a list(an object)>
    Concept (0) : etat
     <Cat : Nom|Forme : pas|Arbres elementaires : a list(an object)>
    Concept (0) : etat
     <Cat : Nom|Forme : presence|Arbres elementaires : a list(an object)>
    Concept (0) : etat
     <Cat : Nom|Forme : absence|Arbres elementaires : a list(an object)>
    Concept (0) : etat
     <Cat : Nom|Forme : constitution|Arbres elementaires : a list(an object)>
    Concept (0) : proc
```

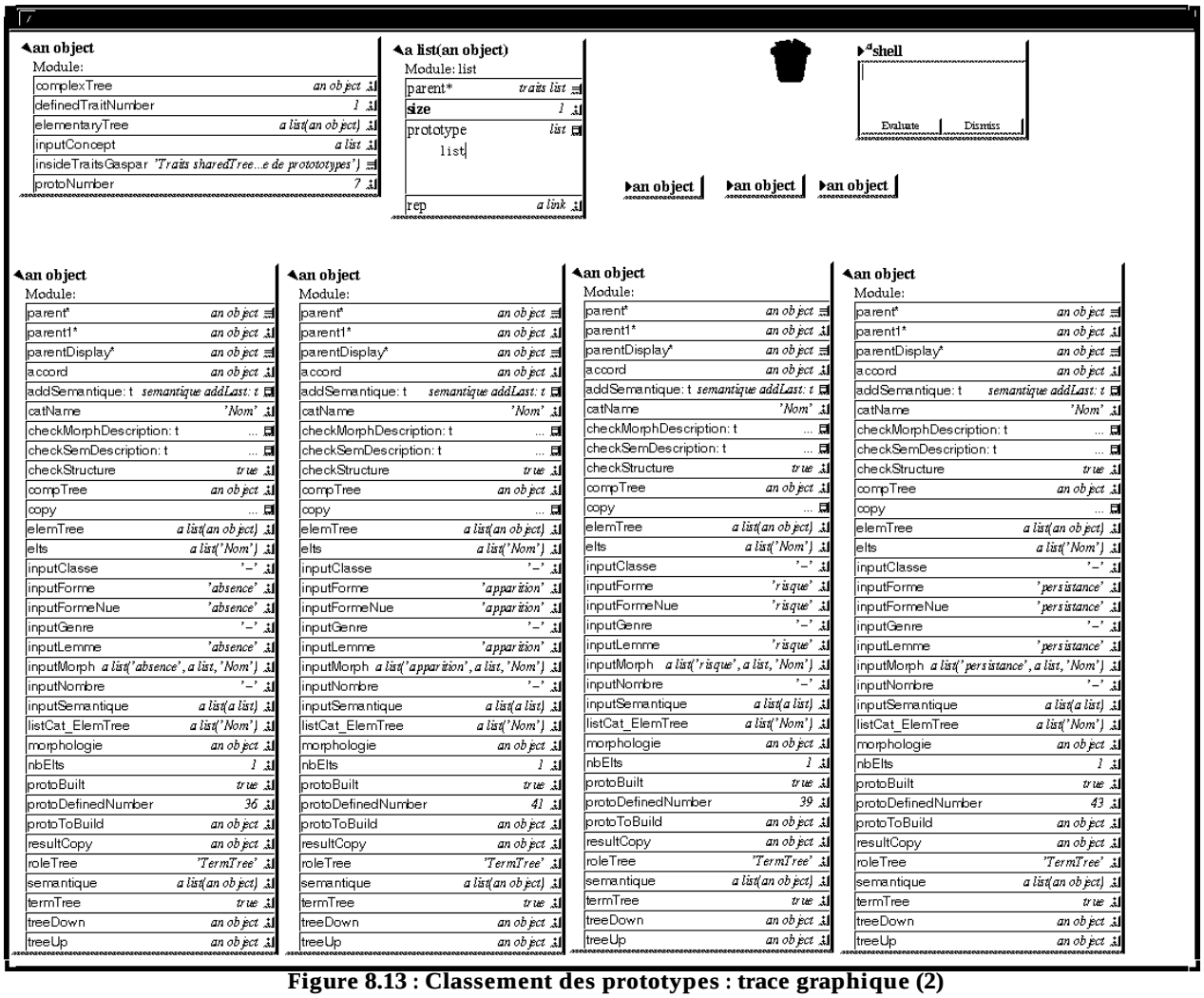


Figure 8.13 : Classement des prototypes : trace graphique (2)

Sur la catégorie grammaticale des adjectifs, on obtient les pôles de comportements partagés suivants :

Pour chaque pôle présenté infra, le processus de classement propose une trace dans la fenêtre shell qui donne la liste des prototypes de mots associés à ce pôle sous la forme suivante :

<catégorie du mot , forme du mot, arbres élémentaires associés à ce mot>

Sur un pôle donné, l'arbre ou les arbres élémentaires sont les arbres partagés par les mots visés.

- Pôle n°1 :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adj|Forme : posterieur|Arbres elementaires : a list(an object)>
     <Cat : Adj|Forme : moyen|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
     <Cat : Adj|Forme : inferieur|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
     <Cat : Adj|Forme : gauche|Arbres elementaires : a list(an object)>
     <Cat : Adj|Forme : distal|Arbres elementaires : a list(an object)>
     <Cat : Adj|Forme : superieur|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
     <Cat : Adj|Forme : anterieur|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
     <Cat : Adj|Forme : droit|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
     <Cat : Adj|Forme : proximal|Arbres elementaires : a list(an object)>
         Concept (0) : loc-lieu
```

- Pôle n°2 :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adj|Forme : inhomogene|Arbres elementaires : a list(an object)>
         Concept (0) : car
     <Cat : Adj|Forme : complexe|Arbres elementaires : a list(an object)>
         Concept (0) : car
     <Cat : Adj|Forme : long|Arbres elementaires : a list(an object)>
         Concept (0) : car
     <Cat : Adj|Forme : ulcere|Arbres elementaires : a list(an object)>
         Concept (0) : car
```

- Pôle n°3 :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adj|Forme : negligeable|Arbres elementaires : a list(an object)>
         Concept (0) : degre
     <Cat : Adj|Forme : predominant|Arbres elementaires : a list(an object)>
         Concept (0) : degre
     <Cat : Adj|Forme : significatif|Arbres elementaires : a list(an object)>
         Concept (0) : degre
     <Cat : Adj|Forme : important|Arbres elementaires : a list(an object)>
         Concept (0) : degre
```

- Pôle n°4 :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adj|Forme : serre|Arbres elementaires : a list(an object)>
         Concept (0) : degre
     <Cat : Adj|Forme : severe|Arbres elementaires : a list(an object)>
         Concept (0) : degre
```

- Pôle n°5 :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adj|Forme : carotide|Arbres elementaires : a list(an object)>
         Concept (0) : vaisseau
     <Cat : Adj|Forme : circonflexe|Arbres elementaires : a list(an object)>
         Concept (0) : vaisseau
     <Cat : Adj|Forme : coronaire|Arbres elementaires : a list(an object)>
         Concept (0) : vaisseau
     <Cat : Adj|Forme : marginale|Arbres elementaires : a list(an object)>
```

Sur la catégorie grammaticale des adverbes, on obtient le pôle de comportement partagé suivant :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Adverbe|Forme : pas|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Adverbe|Forme : peu|Arbres elementaires : a list(an object)>
     <Cat : Adverbe|Forme : tres|Arbres elementaires : a list(an object)>
         Concept (0) : degre
     <Cat : Adverbe|Forme : non|Arbres elementaires : a list(an object)>
```

Sur la catégorie grammaticale des noms, on obtient le pôle de comportement partagé suivant :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Nom|Forme : risque|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Nom|Forme : apparition|Arbres elementaires : a list(an object)>
         Concept (0) : proc
         Concept (1) : evt
     <Cat : Nom|Forme : persistance|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Nom|Forme : pas|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Nom|Forme : presence|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Nom|Forme : absence|Arbres elementaires : a list(an object)>
         Concept (0) : etat
     <Cat : Nom|Forme : constitution|Arbres elementaires : a list(an object)>
         Concept (0) : proc
```

Sur la catégorie grammaticale des quantitatifs, on obtient le pôle de comportement partagé suivant :

```
Liste des prototypes partageant les memes Comportements :

     <Cat : Quant|Forme : premier|Arbres elementaires : a list(an object)>
         Concept (0) : ord
     <Cat : Quant|Forme : double|Arbres elementaires : a list(an object)>
         Concept (0) : card
```

8.2.2. Résultats construits sur des gros corpus

8.2.2.1. Etude de la relation NAdj

Nous limitons ici notre étude à certaines configurations nominales et plus particulièrement aux séquences du type NAdj[91]. Il s'agit en particulier de rechercher au niveau des mots des régularités et des redondances d'utilisation puis d'évaluer les regroupements obtenus[92] (Justeson & Katz 1995). Deux corpus sont utilisés pour mettre en oeuvre les outils définis : le premier est obtenu via le travail d'extraction réalisé par Lexter, le second via l'extracteur AlethIP[93].

Remarques préliminaires sur les corpus étudiés

- Dans les deux corpus traités ci-dessous, nous n'avons retenu que la relation (mot, arbre élémentaire) : i.e. aucun arbre d'analyse n'est associé aux arbres élémentaires attachés aux mots.

- Les séquences retenues présentent, pour les deux corpus étudiés, un plus grand nombre de séquences dans lesquelles l'adjectif est en position de postposition[94] par rapport au nom.

Dans le premier corpus (via Lexter), on a la répartition suivante :

sur 1441 séquences NAdj sélectionnées, on a 1260 séquences dans lesquelles l'adjectif est postposé et 181 séquences où il est antéposé.

Dans le second corpus (via AlethIP), on a la répartition suivante :

sur 1679 séquences NAdj sélectionnées, on a 1444 séquences dans lesquelles l'adjectif est postposé et 235 séquences où il est antéposé.

Résultats Lexter et Cyclade : Caractérisation des relations pour les adjectifs (Habert & al. 1996)

Le réseau des adjectifs renseigne plutôt sur les relations et attributs à attacher aux notions. Il est dominé par les localisants, en particulier ceux qui renvoient à des artères : marginal coronaire circonflexe diagonal coronarien. L'opposition droit gauche constitue en fait une localisation auxiliaire, dans des contextes comme artere circonflexe gauche. Autre localisation, distincte, celle qui concerne la famille d'infarctus : anterieur apical inferieur. Si artere comme infarctus entrent dans une relation de localisation, celle-ci se réalise de deux facons différentes.

- Contextes artere

1. Lexter :

---

91 *"L'adjectif sert en premier lieu à déterminer un nom auquel il s'accorde en genre et en nombre. Dans cette détermination, la signification de l'adjectif constitue (au moins) un trait spécifique qui s'ajoute au nom"* (Weinrich 1989).

92 *"**The adjective-noun relation is directly pertinent to semantic attributes of both the adjective and the noun only when there is a deep relation between them.** In the case of the infinival and related constructions, no such relation holds; the noun modified by the adjective at the surface level is irrelevant. A principled approach to adjective disambiguation using nouns therefore requires a determination that the adjective modifies the noun at a deep syntactic level. It is therefore important to take into account the infinitival construction prior to disambiguating any adjective - even those for which it does not constitute an indicator"* (Justeson & Katz 1995).

93 AlethIP est un extracteur de groupes nominaux développé par GSI-ERLI dans le cadre du projet Eureka GRAAL.

94 *"Un adjectif qui détermine un nom lui est soit postposé soit antéposé. Lorsque l'adjectif est postposé, il insère dans la qualification tous les traits sémantiques qui sont les siens en vertu de sa signification lexicale. Lorsque l'adjectif est antéposé, il n'apporte à la qualification qu'un seul de ses traits lexicaux, un trait dont le pouvoir discriminant est de surcroît réduit...Les adjectifs issus de champs sémantiques spécialisés sont en règle générale postposés"* (Weinrich 1989).

```
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CORONAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Ppa NON_PONTE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj BRACHIAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj PULMONAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CIRCONFLEXE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj DIAGONAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj AURICULOVENTRICULAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj NON_STENOSE]]]
[/SN [/SN [/Nom ARTERE]][/SN [/XX COLLATERALISEE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj HUMERAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj INCOMPRESSIBLE]]]
[/SN [/SAdj [/Adj GROS]][/SN [/Nom ARTERE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj IRREGULIER]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj TRES_PATHOLOGIQUE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj RENAL]]]
[/SN [/SAdj [/Adj PETIT]][/SN [/Nom ARTERE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Ppa TRES_GRELE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CORONARIEN]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj DOMINANT]]]
[/SN [/SAdj [/Adj BEAU]][/SN [/Nom ARTERE]]]
[/SN [/SN [/Nom ARTERE]][/SN [/Nom LATERO-DIAGONALE]]]
```

2.AlethIP :

```
[/SN [/SAdj [/Adj AUTRE]][/SN [/Nom ARTERE]]]
[/SN [/SAdj [/Adj BEAU]][/SN [/Nom ARTERE]]]
[/SN [/SAdj [/Adj GROS]][/SN [/Nom ARTERE]]]
[/SN [/SAdj [/Adj PETIT]][/SN [/Nom ARTERE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj BRACHIAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CIRCONFLEXE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CORONAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CORONARIEN]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj DIAGONAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj DOMINANT]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj FEMORAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj HUMERAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj ILIAQUE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj INCOMPRESSIBLE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj LATERAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj MAMMAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj PULMONAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj RENAL]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj DOMINE]]]
[/SN [/SN [/Nom ARTERE]][/SN [/XX CIRCONTLEXE]]]
[/SN [/SN [/Nom ARTERE]][/SN [/XX COLLATERALISEE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj AURICULOVENTRICULAIRE]]]
[/SN [/SN [/Nom ARTERE]][/SAdj [/Adj CIRCONFLEXE]]]
```

- Contextes `infarctus`

1.Lexter

```
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj INFERIEUR]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj ANTERIEUR]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj BIOLOGIQUE]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj RECENT]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj INFERO-POSTERIEUR]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj INFERO-APICAL]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj MYOCARDIQUE]]]
```

2.AlethIP

```
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj ANTERIEUR]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj BIOLOGIQUE]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj INFERIEUR]]]
[/SN [/SN [/Nom INFARCTUS]][/SAdj [/Adj RECENT]]]
```

Le réseau permet également de préciser les conditions d'emploi de `coronaire` et `coronarien`. Malgré le nombre élevé de contextes partagés et leur appartenance commune à la clique `coronaire coronarien diagonal circonflexe`, les deux adjectifs ne sont pas synonymes.

- Cooccurrences `<coronarien+évaluatif>`

1. Lexter

```
[/SN [/SN [/SN [/Nom OBSTRUCTION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom OBSTRUCTION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom MALADIE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom ATHEROME]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/Nom LESION]][/SN [/SN [/Nom CORONARIEN]][/SAdj [/Adj TOUT-A-FAIT_SEVERE]]]]
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj TRITRONCULAIRES]]][/SAdj [/Adj SEVERE]]]

[/SN [/SN [/SN [/Nom ATHEROSCLEROSE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]
[/SN [/SN [/SN [/Nom ANOMALIE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]
```

2. AlethIP

```
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom MALADIE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]
[/SN [/SN [/SN [/Nom OBSTRUCTION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SEVERE]]]

[/SN [/SAdj [/Adj SEUL]][/SN [/SN [/SN [/Nom ANOMALIE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]]
[/SN [/SN [/Nom ABSENCE]][/SP [/Prep DE][/SN [/SN [/SN [/Nom ATHEROMATOSE]][/SN [/Nom CORONARIEN]]][/SAdj [/Adj
SIGNIFICATIF]]]]]
[/SN [/SN [/SN [/Nom ATHEROSCLEROSE]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONARIEN]]][/SAdj [/Adj SIGNIFICATIF]]]
```

- Cooccurrences `<coronaire+évaluatif>`

1. Lexter

```
[/SN [/SN [/Nom PAS]][/SP [/Prep DE][/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONAIRE]]][/SAdj [/Adj SIGNIFICATIF]]]]]
```

2. AlethIP

```
[/SN [/SN [/SN [/Nom LESION]][/SAdj [/Adj CORONAIRE]]][/SAdj [/Adj SIGNIFICATIF]]]
```

Ce réseau comprend enfin des antonymes : `residuel severe` et `important minime` qui peuvent exprimer la relation de `degré` à attacher à certaines affections.

8.2.2.2. Travail sur le corpus fourni par Lexter

Données traitées

Dans un premier temps nous avons travaillé sur des séquences `NAdj` extraites via Lexter. A partir de 8754 séquences comportant des groupes nominaux, nous avons extraits 609 mots (des noms) auxquels sont attachés 1441 arbres élémentaires de type :

```
Sn -> Nom Adj
Sn -> Adj Nom
Sn -> Adj XX
Sn -> XX Adj
```

Cette première sélection a donc consisté à ne garder que les arbres binaires portant les feuilles `Nom/XX` et `Adj`.

8.2.2.3. Travail sur le corpus fourni par AlethIP

Données traitées

A partir de 7766 séquences extraites via AlethIP comportant des groupes nominaux, nous avons extraits 718 mots (des noms) auxquels sont attachés 1680 arbres élémentaires de même type que précédemment.

8.2.2.4. Le travail sur les gros corpus et les limites de Self

"Prolog est né d'un pari : créer un langage de très haut niveau, même inefficace au sens des informaticiens de l'époque. L'efficacité consistait alors à faire exécuter très rapidement par une machine des programmes écrits laborieusement. Le pari

était donc de pouvoir écrire rapidement des programmes, quitte à ce que la machine les exécute laborieusement."
Colmerauer, préface à Giannesini & al. "Prolog", InterEditions, 1985.

Les processus de génération de prototypes présentés supra ont mis en avant les limites actuelles de l'implémentation de Self quand ces processus sont activés sur de gros corpus. Après la génération d'une cinquantaine de prototypes de mot de ces corpus et des arbres associés (environ une centaine d'objets), le système Self s'"écroule" par manque de ressources. Le problème est en fait lié à un manque de mémoire pour exécuter la construction dynamique de nouveaux objets. Self est de fait peu habitué à manipuler autant d'objets dans une même session de travail. Les créateurs[95] de Self ont essayé de remédier à ces problèmes en ajustant l'implémentation. Cette nouvelle version n'a pas encore pu être testée sur nos corpus de travail : des problèmes de compatibilité de structure de représentation ont été mises à jour entre cette version et le code écrit pour Gaspar. Cette incapacité actuelle du système Self à assimiler la génération d'un grand nombre d'objets met surtout en avant les problèmes complexes liés à la mise en oeuvre de ce type de modèle de représentations évolutives. Cette mise en oeuvre de la mouvance dans les outils de représentation coûte chère en ressources pour les systèmes sous-jacents. Ce contre-temps technique nous empêche donc pour le moment de produire des résultats pertinents sur les corpus Lexter et AlethIP.

Les processus sont prêts : **seule la "mémoire" leur manque pour exprimer et produire des pistes de sens.**

95 Je tiens une nouvelle fois à remercier David Ungar de l'attention qu'il a portée à mes différentes sollicitations et notamment pour mettre en oeuvre des solutions pour pallier ces problèmes.

8.3. Représentation et classification dynamiques : bilans et perspectives

8.3.1. Une démarche interprétative contrôlée et en spirale

Les énoncés de langue se réalisent sous la forme d'agencements entre des évènements, des idées, des concepts, des sujets... Ils se construisent comme une suite hétérogène d'éléments qu'il s'agit de faire tenir ensemble. Un tel agencement se comporte comme une structure qui organise les co-fonctionnements des unités qui la composent. Ce qui importe, ce n'est pas tant les éléments pris individuellement, mais les relations qu'ils tissent entre eux pour produire du sens. Ces relations se déploient entre les unités et ne sont pas des propriétés attachées aux unités qui composent un énoncé. Un agencement de langue établit donc des faisceaux de relations entre les unités linguitiques, faits de pleins et de vides, de blocs et de ruptures, de blancs et de reprises, de nuances et de brusqueries, d'additions dont la somme n'est jamais arithmétiquement construite... Le sens d'un énoncé construit de l'imperceptible à partir de micro-perceptions : comment formaliser ceci de manière prédéfinie? Les modèles doivent maintenir présente cette notion d'expérimentation permanente dans la production des faits de langue. Les programmes de TALN doivent construire des moyens de repérage pour conduire une expérimentation sur les faits de langue qui déborde nos capacités de prévoir. Les représentations d'unités de langue doivent être des outils qui permettent au linguiste d'opérer un travail d'expérimentation sur les faits représentés. La langage n'est pas une masse inerte. Les relations que les unités linguistiques tissent entre elles dans un énoncé peuvent évoluer sans que les unités elles-même soient modifiées. Self encourage l'expérimentation, et nous choisissons donc de ne pas nous imposer de modèles interprétatifs prédéterminées dans les représentations que nous construisons. Notre approche vise à définir une démarche en spirale : construire des représentations par affinements successifs avec projections de savoirs soit constitués soit établis par ailleurs; puis ajustements et affinements des représentations à chaque nouvelle étape (Mikheev & Moens 1994). Comme notre programme n'est pas encore capable de produire systématiquement de nouvelles connaissances à partir de celles qui existent déjà, l'acquisition de nouvelles connaissances reste tributaire d'une intervention extérieure au programme capable de transmettre ces informations. Soit on agit directement sur les savoirs construits (intervention manuelle), soit on affine les informations initiales et on réitère la génération ou l'affinement de la représentation des objets.

8.3.2. Des amorçages d'interprétation : le retour du linguiste

Notre approche tend à distribuer les savoirs sur des myriades d'éléments capables de travailler chacun de leur côté, tout en mettant en place des "micro-instances" centralisatrices qui structurent les savoirs représentés sur des ensembles réduits d'éléments. Les savoirs représentés se déploient sur les prototypes construits et les pôles de savoirs partagés permettent de tisser des liens entre les prototypes représentés. La démarche suivie s'inscrit donc dans une perspective expérimentale à différents niveaux :

- Travail sur des savoirs extraits de corpus.
- Représentation des mots : construire les représentations informatiques des informations linguistiques formelles associées aux mots.
- "Représentaction" sur les mots : construire les représentations des processus linguistiques ou des comportements linguistiques que l'on peut attacher à ces mots.
- Corpus arboré ⇒ (Représentations+Représentactions) ⇒ Un premier classement.

Le classement obtenu reste évidemment à affiner. Ce classement sur les entités représentées vise à la contruction de réseaux entre les prototypes définis à la manière d'un parcours entre ces prototypes

représentés pour y trouver le chemin interprétatif adéquat. Nous insistons sur le fait que le système construit s'inscrit dans une approche qui vise à établir un dialogue entre les dispositifs informatiques construits et les problèmes posés par les faits linguistiques. L'intervention manuelle du linguiste s'inscrit parfaitement dans une telle approche.

8.3.3. Un système qui module les flux d'informations

On peut disposer en amont de plusieurs couches d'informations. Si le travail d'extraction de savoirs est capable de mettre en évidence différents types de savoirs sur une entité lexicale, on utilisera ces informations dès la première phase de génération du prototype construit pour représenter cette unité lexicale. On peut aussi envisager de procéder à une acquisition de savoirs en réitérant la phase initiale de lecture de ces savoirs initiaux. On pourrait ainsi moduler et modulariser cette phase d'acquisition de savoirs en activant différents flux de savoirs. Si les savoirs attachés aux mots ne sont pas disponibles dès la première phase de génération des prototypes associés, il sera toujours possible d'ajuster les représentations construites en utilisant un nouveau flux de savoirs disponibles ultérieurement.

Il est aussi possible de projeter les résultats transitoires obtenus sur des savoirs établis par ailleurs et d'utiliser les résultats de ces projections pour ajuster les représentations.

- Phase1 : on utilise un corpus donné pour représenter puis classer les savoirs associés à ce corpus. On prend appui sur les savoirs généraux disponibles.

Cette première étape peut conduire à une mise à jour (manuelle ou automatique) sur les résultats construits : mise à jour sur les objets construits ou sur la base de savoirs généraux. Ces affinements résultent d'un travail d'interprétation fin des informations issues des activités de génération et de classement réalisées.

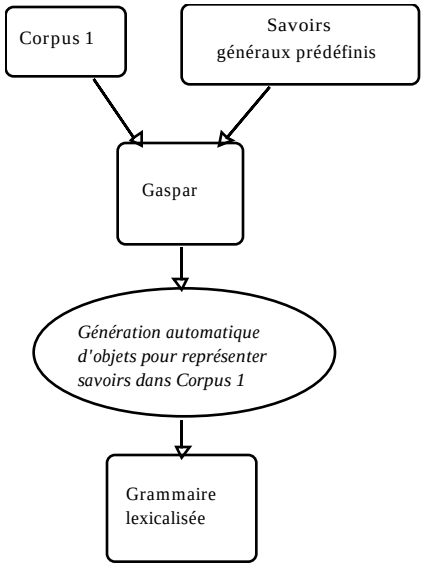


Figure 8.14 : Une démarche en spirale (1)

- Phase 2 : Si de nouvelles informations sont mises à jour (affinement des informations dans le corpus ou mises à jour dans les savoirs généraux), il est possible de réitérer les processus de représentation (affinement de l'existant ou recréation à la volée de nouveaux objets) ou de classement pour tenir compte de celles-ci.

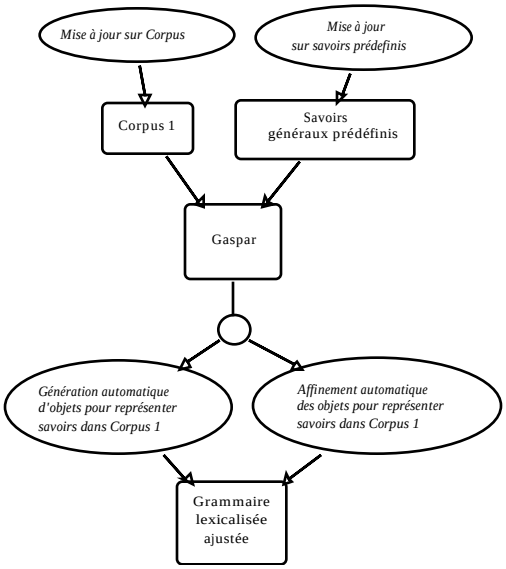


Figure 8.15 : Une démarche en spirale (2)

- Phase 3 : On peut aussi confronter les résultats établis sur différents états de corpus ou sur des corpus de différentes natures. Dans le schéma qui suit, on peut appliquer les processus sur deux corpus différents et produire des résultats relatifs à ces deux corpus. On peut aussi appliquer les processus au seul corpus 2 pour établir par exemple une confrontation des résultats produits sur les deux corpus traités.

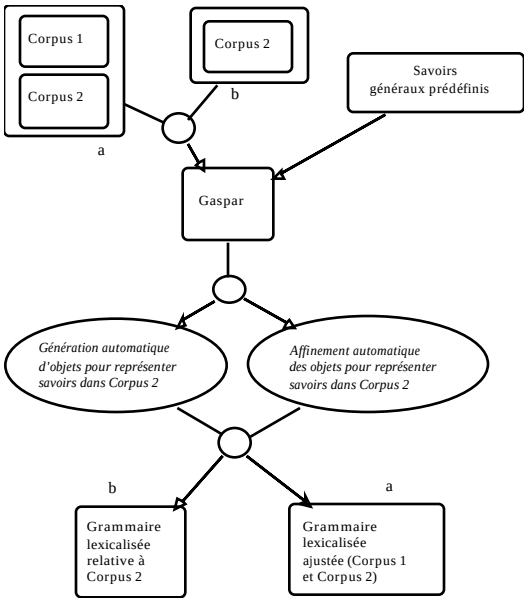


Figure 8.16 : Une démarche en spirale (3)

Le processus de représentation peut donc se développer en réitérant les phases suivantes :

représentations à partir de savoirs extraits d'un corpus projections des résultats ajustements des représentations ajustement du classement

Ces différentes étapes peuvent induire des phases intermédiaires de travail manuel pour corriger les états de représentation produits (Mikheev & Moens 1994). L'environnement de programmation choisi permet à tout moment d'affiner les représentations construites (-> incise 2). Le dispositif mis en place utilise ce potentiel évolutif pour aider à la mise en oeuvre de ces ajustements. On dispose ainsi d'une série d'interfaces et d'outils qui permettent d'interroger, de modifier... les objets construits dans une séance de travail (-> annexe partie 2 chapitre 8, -> incise 2).

La PàP s'accommode bien avec cette démarche de représentation. Elle permet en effet de construire la représentation du domaine visé puis de classer les savoirs représentés de manière progressive : les processus de représentation puis de classement permettent une approche incrémentale. La représentation du domaine étudié pouvant évoluer, sa représentation s'améliore en fonction des résultats obtenus.

8.3.4. Prototypie et représentation des savoirs pour le TALN

8.3.4.1. Processus de représentation et TALN

Les mots ont leur(s) histoire(s)

Dans des situations de communication traditionnelles, les mots véhiculent leurs propre(s) histoire(s); et même s'il y a des trous, personne ne s'en "plaint vraiment" dans la mesure où l'on peut, en général, les combler sans peine, quitte à "réécrire" une nouvelle "histoire" autour ou entre les mots.

Retrouver / rechercher les connaissances liées à chaque mot

Dans notre approche de travail, il s'agit de rechercher les savoirs associés aux mots puis de les représenter en tenant compte des interactions multiples entre toutes les connaissances qui sont à l'oeuvre entre les mots.

Processus de représentation évolutifs

Le modèle de représentation mis en oeuvre prend appui sur un corpus arboré pour opérer un apprentissage des savoirs à représenter; il s'agit ensuite de construire par affinements successifs les structures pour porter les savoirs mis à jour. On cherche à réaliser une adéquation entre les occurrences linguistiques réalisées et les prédictions de représentations construites. Il ne s'agit pas de produire d'emblée un résultat définif qui réalise cette adéquation de manière parfaite; mais plutôt de tendre vers cette adéquation, par touches successives, en affinant les prédictions construites.

La mise en oeuvre des représentations est donc conçue comme un mécanisme évolutif qui, d'une part, doit tenir compte d'un nombre important de sources de connaissances, et d'autre part, doit être capable d'intégrer de nouvelles informations à chaque étape. Les représentations à construire ne sont donc pas figées mais peuvent évoluer dynamiquement en fonction des nouveaux savoirs utilisés ou mis à jour. L'apprentissage de nouveaux savoirs ne prédétermine et ne limite pas les sources de connaissances capables d'induire de nouveaux savoirs à associer aux mots.

Importance du travail manuel sur le corpus initial, puis sur les représentations construites.

Deux tâches fondamentales sont nécessaires pour mettre en oeuvre l'apprentissage des savoirs à partir d'un corpus initial. Il s'agit évidemment dans un premier temps d'affiner le travail d'extraction de savoirs sur le corpus utilisé : le travail sur corpus est conçu lui aussi comme un travail évolutif, on n'épuise jamais un corpus. Puis, il s'agit d'affiner le travail de représentation sur les entités représentées, en utilisant soit des résultats construits par ailleurs, soit les nouveaux résultats obtenus par le travail continu effectué sur le corpus initial.

8.3.4.2. Affinements des représentations

On a vu précedemment que la génération des savoirs linguistiques mise en oeuvre avec Gaspar s'effectue par une construction dynamique de prototypes pour représenter aussi bien les mots que les contraintes syntaxiques (les arbres) associées à ces mots. Les différentes opérations qui réalisent la construction effective de ces prototypes utilisent les outils mis en place par Self pour définir des objets ou pour ajuster leurs représentations. On rappelle que par exemple l'opération de clonage (réalisée par la primitive `clone` appliquée à un objet) produit une copie du prototype auquel est adressé ce message. De même, les opérations d'ajout ou de retrait d'attributs (réalisées via les primitives `_AddSlots:`, `_AddSlotsIfAbsent:`, `_RemoveSlot:`...) permettent d'ajuster la représentation d'un objet donné en lui ajoutant un attribut (l'argument de cette primitive), ou en lui supprimant un attribut.

Clonage et ajustements

Dans les mécanismes mis en place par Gaspar pour la génération des prototypes, on n'utilise pas la primitive de clonage prédéfinie telle quelle. En effet, les prototypes construits sont définis par certains attributs qui peuvent par exemple pointer sur d'autres objets : le prototype `pontage` porte un attribut `morphologie` qui pointe vers un prototype `morphologie` décrivant la morphologie du mot. Si on appliquait l'opération de clonage au prototype `pontage` pour créer un nouveau prototype lexical, le prototype issu du clonage porterait lui aussi un attribut `morphologie` qui pointerait vers le même prototype `morphologie` associé au prototype `pontage`. On a donc défini des mécanismes de copie de prototypes qui enrichissent considérablement l'opération de clonage. La copie de prototype met en place des opérations qui initialisent tous les attributs de l'objet issu du clonage et qui peuvent aussi mettre à jour ces attributs si on dispose des informations adéquates.

Ajouts / Retraits d'attributs et effets de bord

Les opérations d'ajout et de retrait d'attributs génèrent des effets de bord dont il faut absolument tenir compte. En effet, si on ajoute un attribut à un prototype donné, on créé, en fait, un nouveau prototype (un clone du prototype initial) sur lequel est greffé le nouvel attribut. Il existe toujours une trace du prototype initial tel qu'il était défini avant l'opération d'ajout d'attribut. Dans notre démarche de génération dynamique de prototypes, on utilise évidemment ces mécanismes d'ajout et de retrait d'attribut pour ajuster la représentation des objets que l'on a à construire. Mais seul le prototype construit à l'ultime phase d'ajouts ou de retraits d'attribut est considéré comme pertinent dans l'environnement de Gaspar : les prototypes "transitoires" sont incohérents, ils correspondent à un état intermédiaire de la construction d'un certain type de prototype, ils ne seront donc pas pris en compte dans les opérations d'analyse définies par Gaspar. Pour contrôler cette génération de prototype, on a donc mis en place des mécanismes de vérification de la structure interne des objets construits : un prototype sera "cohérent" s'il est issu des différentes phases de construction des prototypes de mots ou des prototypes d'arbres. Ce qui indiquera qu'il possède tous les facettes nécessaires pour être considéré comme valide dans l'environnement de Gaspar. De plus, pour ne pas multiplier inconsidérement les prototypes incohérents, on dispose d'un mécanisme qui permet de "détruire" tous ces prototypes incomplètement construits ou tout au moins de les "poubelliser". Ces opérations sont définies en utilisant les outils de méta-représentation disponibles avec Self et qui seront présentées infra.

8.3.4.3. Retour sur le stochastique

Affecter des poids aux arbres associés aux mots

"Le fait que ces arbres élémentaires apparaissent en rafales peut être l'indication d'un fonctionement dénominatif (une entité suffisamment importante pour donner lieu à redondance locale). Par exemple, une ellision du type "angine stable" n'a pas le même statut (quant à la dénomination éventuelle que l'on peut postuler) selon qu'elle arrive immédiatement après une forme longue : "On peut distinguer deux types d'angines de poitrine. L'angine stable..." et selon qu'elle se produit sans forme longue dans un contexte proche : dans ce dernir cas, il est nettement plus probable qu'"angine de poitrine" est une dénomination, et "angine stable" un hyponyme " (Habert & al. 1995).

Si un mot donné est associé à un arbre élémentaire dont le nombre de réalisation dans le corpus donné est faible et si un autre mot est associé au même arbre élémentaire mais dont le nombre de réalisation est beaucoup plus important, il peut être intéressant de disposer de ces données quantitatives pour qualifier les résultats ou tout au moins pour mettre en avant ces différences de comportements des unités lexicales visées. On peut ainsi tenir compte dans le classement des unités lexicales du "poids" que peuvent avoir les arbres associés sur ces unités. De même, pour un mot inconnu, on peut utiliser les arbres les plus probables sur telle catégorie pour une analyse sur une séquence qui contient ce mot inconnu.

Apprentissage et mot nouveau

Les savoirs recensés et construits sur un certain état du corpus vont être utilisés pour le parsage. Si le parseur analyse une phrase qui contient des mots non recensés (dans la phase initiale d'extraction de savoirs sur corpus), il va évaluer les comportements de ces nouveaux mots en fonction des savoirs déjà construits. Le parseur gardera comme solution d'analyse, celle qui semble la plus probable. Plus précisément, si au cours de l'analyse le parseur rencontre un mot nouveau, il construit des prototypes lexicaux (pour chaque catégorie déjà recensée) pour représenter ce nouveau mot. De plus ne sachant pas si ce nouveau mot est forcément un représentant d'une des catégories grammaticales déjà reconnues, le parseur construit aussi un prototype lexical de catégorie inconnue pour représenter ce mot nouveau. Ce dernier type de prototype pourra entrer dans toutes les compositions syntaxiques sollicitées par d'autres prototypes d'arbres.

Utilisation des savoirs construits

En procédant ainsi, l'analyse ne bloque pas sur des formes non prédéfinies ou non représentées. Au contraire elle permet à l'analyse de construire de nouvelles connaissances (les nouveaux mots représentés) et toutes les connaissances qui peuvent y être associées. Par exemple, si on considère que le parseur doit analyser une phrase contenant le mot "pontage". Et s'il n'existe pas de prototypes lexicaux de même forme. Le parseur construit des prototypes lexicaux pour chacune des catégories grammaticales déjà construites avec cette forme. Les prototypes lexicaux ainsi définis vont hériter des comportements communs (s'il en existe) associés aux prototypes existants pour chacune de ces catégories. En particulier, si le classement des prototypes lexicaux a permis d'attribuer des arbres élémentaires aux objets qui portent les comportements communs des catégories construites, les prototypes lexicaux associés au mot "pontage" délègueront eux aussi à ces objets ces types de comportement. Ces savoirs hérités ne sont pas a priori des comportements associés aux nouveaux mots représentés, ils permettent simplement de relâcher l'analyse aux endroits où il peut y avoir blocage. Il reste ensuite à évaluer si cette association implicite de nouveaux savoirs à un mot initalement inconnu a permis de produire des savoirs pertinents sur ce mot et sur ces relations avec les autres mots.

De même, on utilise les arbres définis et contraints sémantiquement pour analyser des phrases qui pourraient contenir des mots nouveaux. A priori, si un mot est inconnu, les contraintes sur l'arbre vont bloquer la construction de cet arbre avec ce mot nouveau. On doit donc pouvoir relâcher ces contraintes si l'analyse rencontre un mot nouveau; il s'agira ensuite d'évaluer les constructions construites et les relâchements opérés pour produire un résultat significatif ou non. On opère ainsi ce type de relâchement dans le cas où l'analyse est confrontée à un prototype représentant un nouveau mot pour une éventuelle association avec tout prototype d'arbre rencontré.

On donne en annexe (-> annexe partie 3 chapitre 9 : analyse automatique avec Gaspar) une analyse qui illustre ce point. Dans l'analyse présentée, le parseur gaspar doit analyser la séquence "analyseur à prototypes". Cette séquence de mots contient une forme lexicale à laquelle le parseur ne va pas pouvoir, dans un premier temps, associer une ou plusieurs représentations prototypiques de mot. Il va associer, par défaut, à cette forme autant de représentations prototypiques de mot qu'il existe de catégorie pouvant potentiellement être associée à un mot. Il associe de plus à cette forme une représentation prototypique de mot de catégorie inconnue. L'analyse va donc se dérouler en utilisant ces savoirs construits automatiquement par le parseur en utilisant les savoirs déjà représentés.

8.4. Conclusion

1. Les processus de représentation des faits de langue sont contraints de faire évoluer les savoirs représentés (i.e. leurs structures de représentation construites) :

- Les savoirs linguistiques sont évolutifs.

- Leur représentation et leur classement sont donc en perpétuel ajustement.

La PàP permet de construire la représentation du domaine visé puis de classer les savoirs représentés de manière progressive : les processus de représentation puis de classement permettent une approche incrémentale. La représentation du domaine étudié pouvant évoluer, sa représentation s'améliore en fonction des résultats obtenus.

- Les savoirs qui interviennent dans les faits de langue sont nombreux et interagissent de manière complexe. Il est évident qu'une modification dynamique de ces savoirs ou de leurs liens réciproques implique une attention particulière. Ce contrôle ne peut être réalisé que si on peut exercer un méta-regard sur le domaine des connaissances représentées.

2. Le classement des unités lexicales mis en place précédemment est confronté aux problèmes suivants :

- Il est nécessaire là encore de disposer d'outils de contrôle pour réaliser le réseau de délégation induit par le classement défini.

- La classement ne qualifie pas les pôles de comportements partagés mis en place. Les méta-connaissances nécessaires pour nommer les choses construites ne sont pas disponibles ni même définies.

Incise Chapitre 8

Incise 1. Self 4.0, un environnement informatique qui dynamise les objets

Self (et notamment la version 4.0) permet une gestion dynamique de la mise à jour des objets représentés. Self est disponible à l'adresse suivante : http://self.sunlabs.com.

I.1.1. Self-4.0 (Smith, Maloney, Ungar 1995)

Environnement de programmation

• Définition et modification incrémentales des objets

• Techniques expérimentales d'interaction

• Manipulation directe des objets

• Animation

Démarrer Self

Self est activé en tapant `'Snapshot'` dans une session de travail. Self démarre avec :

• Un prompt interactif dans une fenêtre `shell`

La fenêtre `shell` permet d'interroger les objets disponibles dans le monde Self et en particulier de les inclure dans l'interface graphique.

• Une interface graphique interactive

• L'environnement graphique est appelé le monde Self

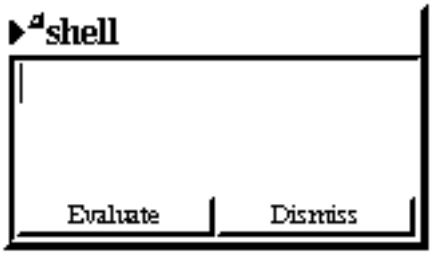


Figure 8.17 : Interface `shell`

La poubelle permet de nettoyer la fenêtre graphique en y déposant les objets que l'on ne souhaite plus y voir figurer.

Figure 8.18 : `"Trash-Can Self"`

I.1.2. Interfaces, outils, opérations et menus dans le monde Self

Menus des opérations sur le monde Self

En utilisant la souris dans la fenêtre graphique, on peut accéder à certains outils qui permettent d'agir sur le monde Self.

Figure 8.19 : Menu des opérations sur le monde Self

• La touche droite de la souris active un menu des opérations sur le monde Self :

Les options disponibles permettent en particulier de :

`Shell`
• créer un objet `shell`;

`Toggle Spy`
• activer un object qui contrôle les activités du système;

`Write Snapshot`
• sauvegarder l'état présent du monde Self;

`Quit`
• quitter la session de travail.

Menus des opérations spécifiques sur les objets Self

De même, tous les objets de la fenêtre graphique sont réactifs à la souris. On peut ainsi accéder à certains outils qui permettent d'agir sur les objets.

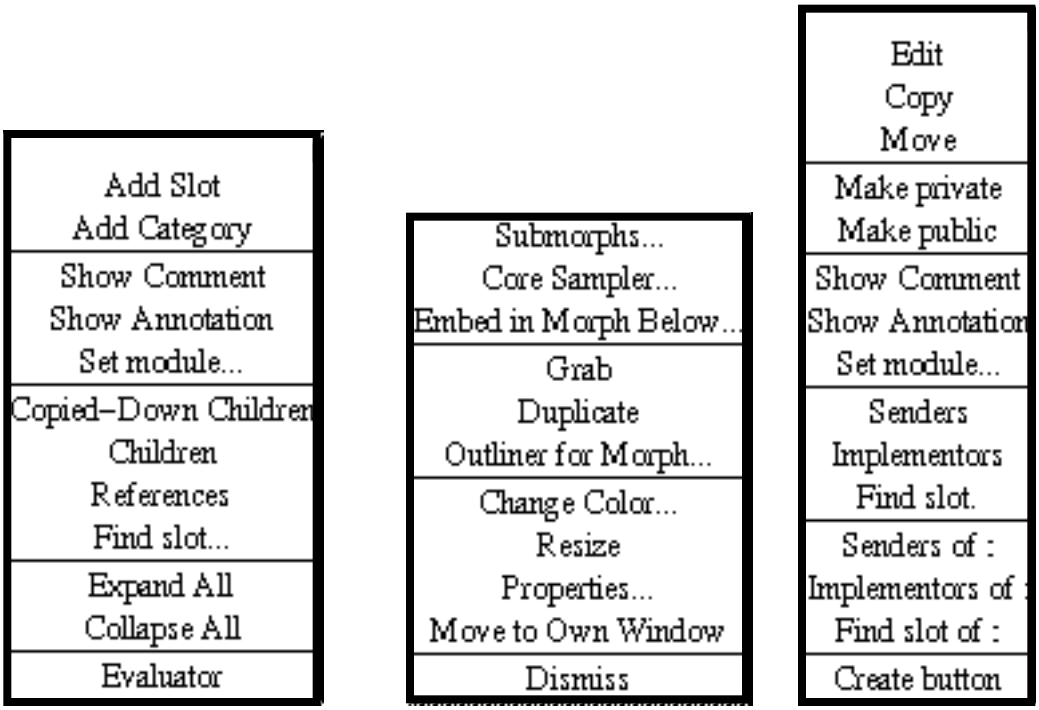


Figure 8.20 : Menus des opérations spécifiques sur les objets Self

• La touche de gauche de la souris permet d'"attraper" les objets et de les déplacer dans la fenêtre graphique

• La touche centrale de la souris active un menu des opérations spécifiques sur un objet

Menu des opérations spécifiques sur un objet

Les options disponibles permettent en particulier de :

`Add Slot`
• ajouter un attribut à l'objet visé;

`Comment`
• ajouter des commentaires à l'objet visé;

`Annotation ...`
• éditer un objet porteur d'annotations sur l'objet visé;

`Find Slot ...`

• créer un objet pour trouver un attribut dans l'objet visé;

`Structure Editor`
• créer un éditeur pour l'objet visé;

`Evaluator`
• créer un (ou plusieurs) évaluateur(s) pour l'objet visé;

• Le touche droite de la souris active un menu des opérations globales sur les objets :

Opérations globales

Les options disponibles permettent en particulier de :

`Duplicate`
• créer un copie de l'objet visé;

`Resize`
• retailler les dimensions de l'objet;

`Move to Own Window`
• créer un monde particulier (une nouvelle fenêtre) avec cet objet;

`Dismiss`
• faire disparaître l'objet de la fenêtre;

`Object expander`
• étendre ou de restreindre la présentation des attributs de l'objet.

On donne ci-dessous une présentation d'un objet tel qu'il apparaît dans la fenêtre graphique. Cette présentation précise les rôles des différents éléments constitutifs de cet objet.

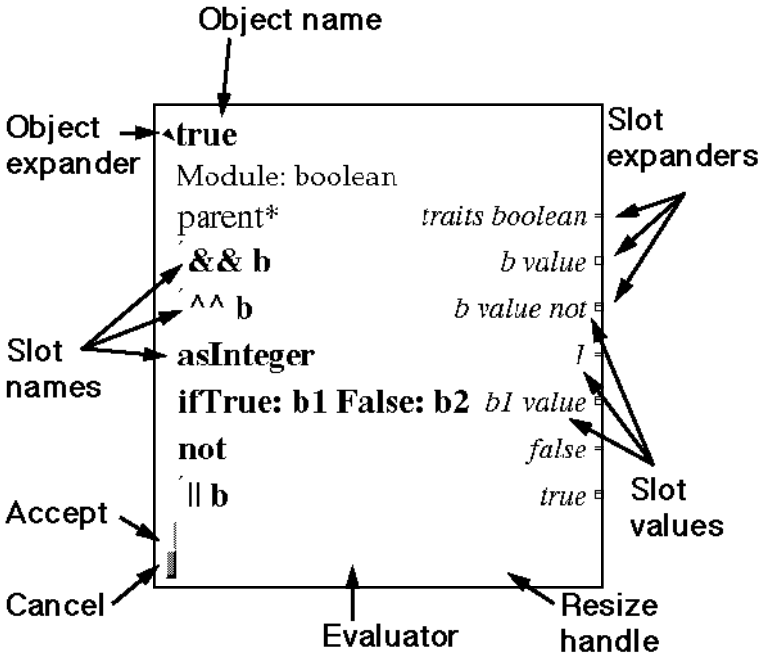


Figure 8.21 : Objet Self expansé (1) (Duvall 1995)

"Expanders"

Pour expanser ou restreindre la présentation des attributs de l'objet, on agit sur les points d'entrée définis :

Figure 8.22 : "Expander" pour les méthodes

Figure 8.23 : "Expander" pour les objets

"Evaluator"

Les évaluateurs constituent des zones d'édition qui permettent d'évaluer des expressions Self dans l'environnement particulier de l'objet.

• L'objet associé à un évaluateur est le récepteur direct des messages adressés dans cette zone.

Accept
• La validation de la zone Accept via le bouton de gauche de la souris permet d'activer l'exécution des messages définis dans l'évaluateur.

Cancel
• La validation de la zone Cancel via le bouton de gauche de la souris permet de désactiver l'évaluateur.

Il est donc possible de visualiser, manipuler ou modifier les attributs des objets de la fenêtre graphique. Pour cela, on peut tout d'abord activer les "slots expanders" pour avoir accès au contenu des attributs. On peut ensuite agir sur ces contenus.

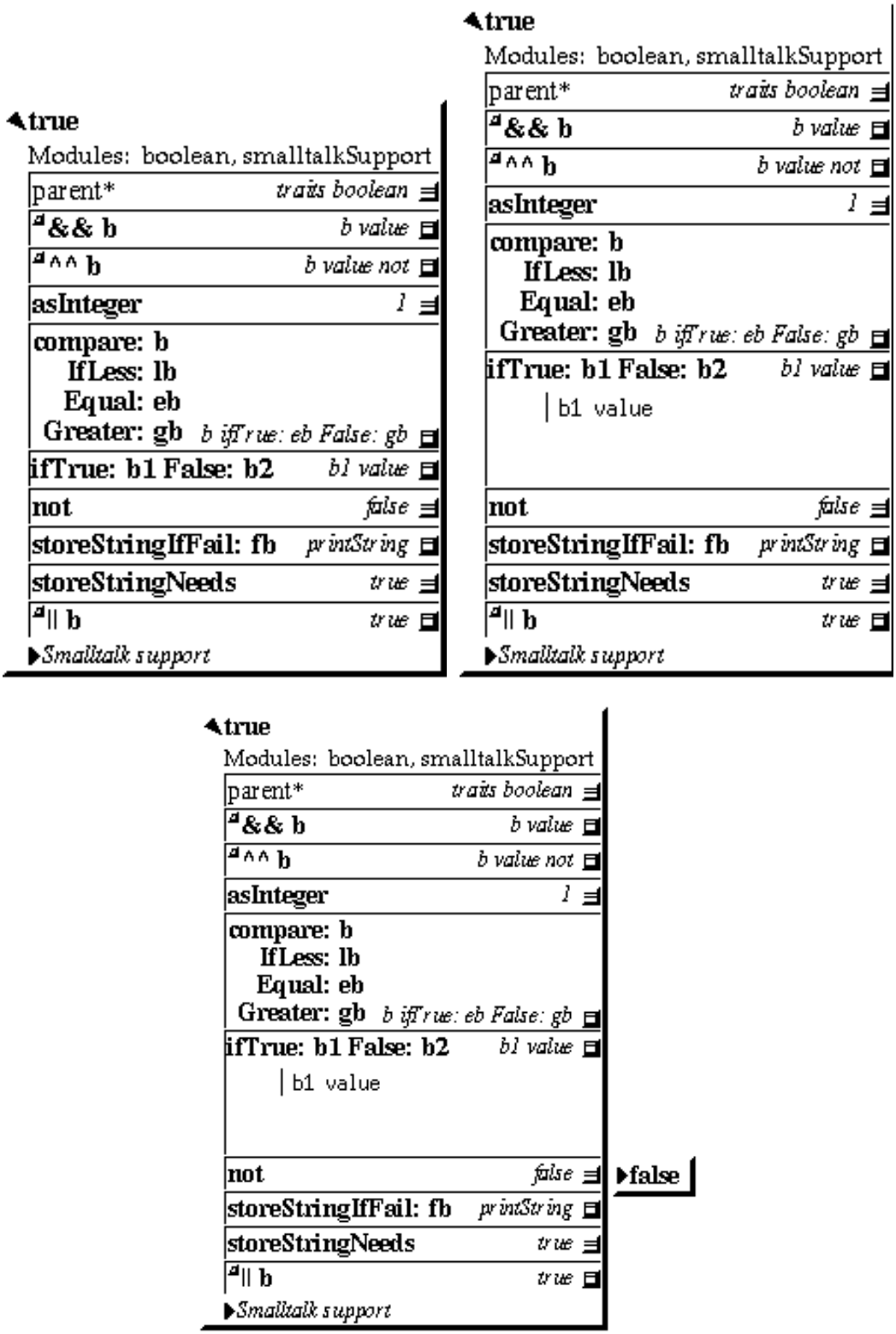


Figure 8.24 : Objet Self expansé (2)

La figure qui suit présente:

- l'objet `true` intégralement expansé,
- les objets expansés associés à certains attributs,
- une description du code définissant des méthodes attachées à d'autres attributs.

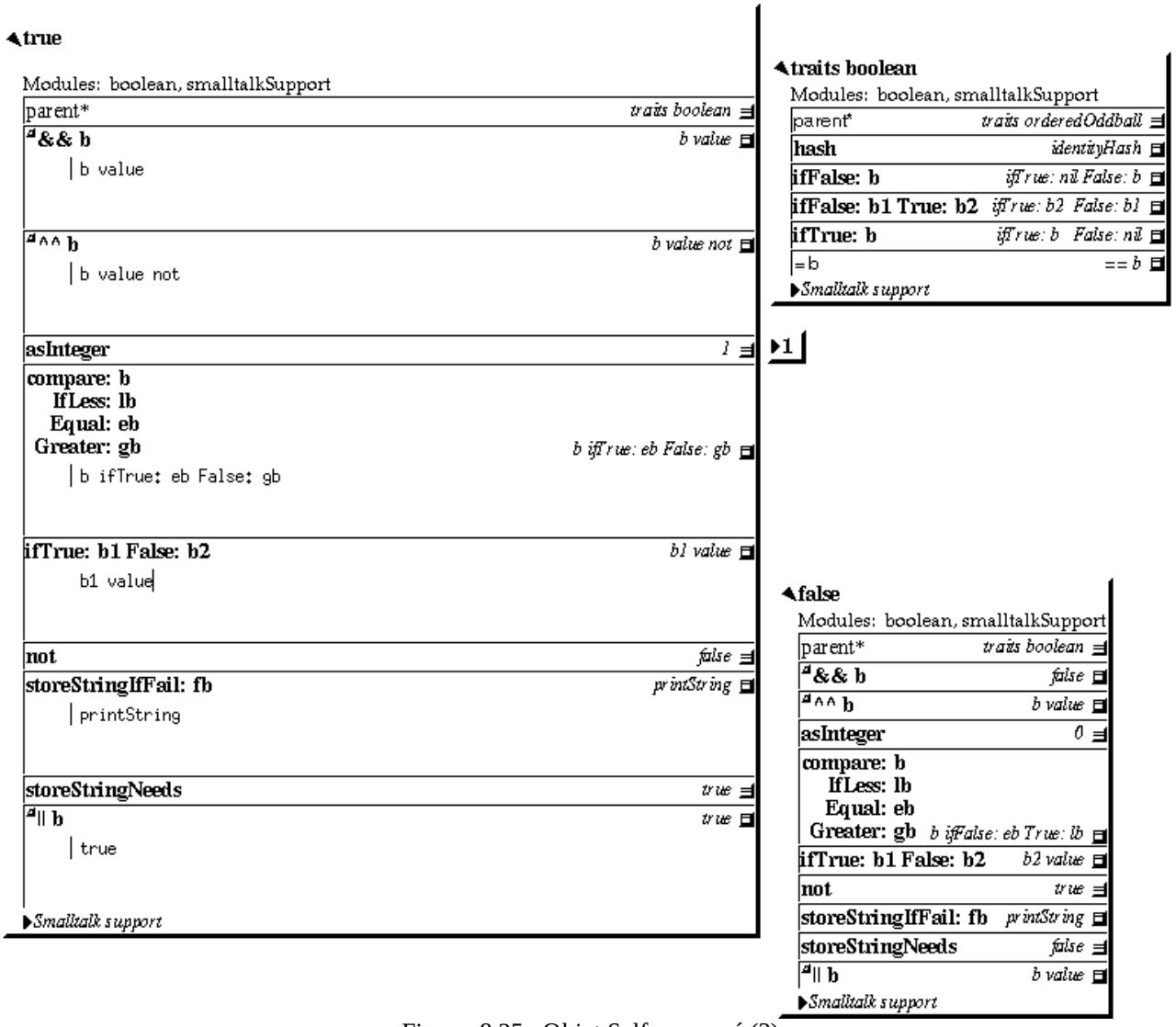


Figure 8.25 : Objet Self expansé (3)

Incise 2. Manipulation et intervention directes sur les objets avec Self

L'utilisateur constitue à lui tout seul une "entité" de méta-contrôle indispensable pour évaluer les résultats construits. Il peut lui même être à l'origine de modifications sur les savoirs représentés. Pour cela il peut intervenir directement sur les objets via la fenêtre graphique de Self. Dans les figures 8.26 et 8.27, on réalise une mise à jour dynamique d'un attribut de l'objet présenté dans la fenêtre graphique : il s'agit du prototype lexical représentant le nom `lesion`.

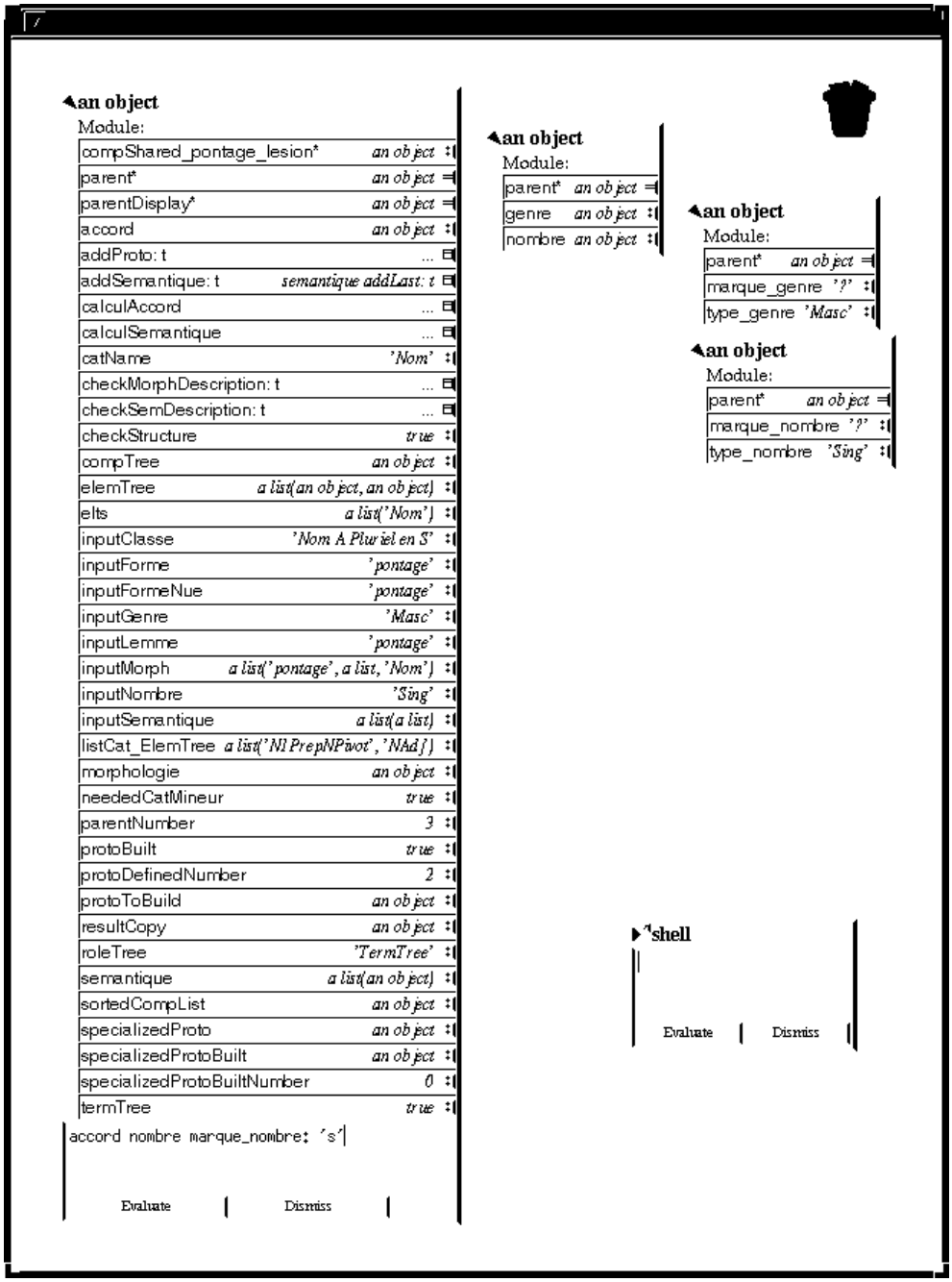


Figure 8.26 : Manipulation et modification directes d'objet (1)

On veut mettre à jour l'attribut `marque_nombre` associé à l'objet `nombre` porté par l'attribut `accord` associé au prototype lexical. Pour réaliser cette mise à jour, on commence par associer à l'objet un évaluateur (en activant la souris sur cet objet) : cette interface permet de lui adresser directement des messages. Dans la figure 8.26, on trouve les valeurs initiales associées aux différents objets présentés ainsi que le message de mise à jour `'accord nombre marque_nombre : s'` avant validation (`'evaluate'`). Dans la figure 8.27 on présente le résultat de la mise à jour précédente après la validation et la réussite de l'envoi du message.

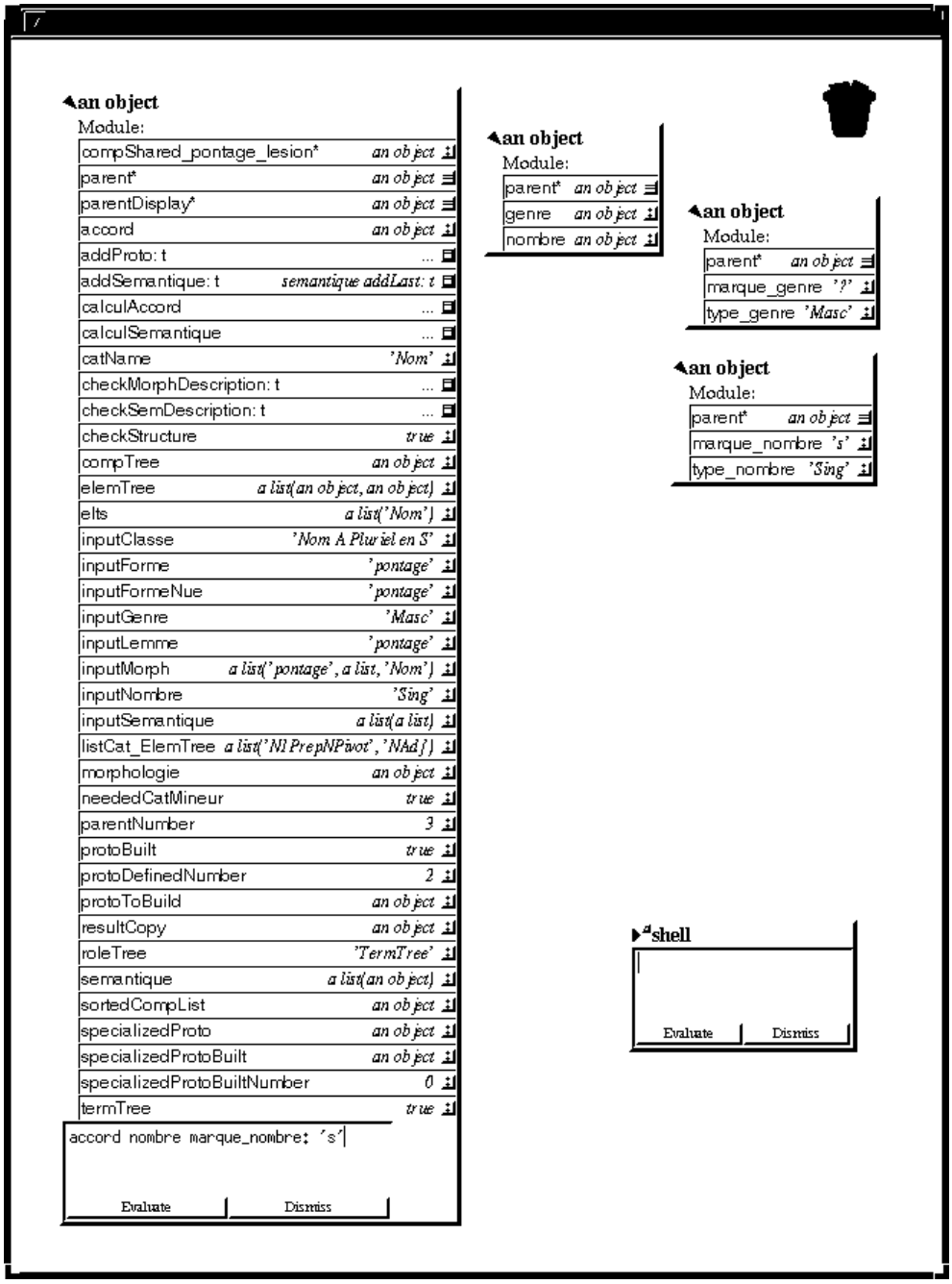


Figure 8.27 : Manipulation et modification directes d'objet (2)

Dans les figures 8.28, 8.29, 8.30, on effectue une mise à jour dynamique de la méthode associée à l'attribut `addSemantique` : on va donc modifier le code associé à l'écriture de cette méthode. Cette première figure présente la valeur initiale associée à cet attribut. En utilisant les menus accessibles sur l'attribut via la souris, on active la présentation de l'objet qui porte ce code initial, on active aussi l'affichage d'un éditeur qui permet d'accéder au code pour la mise à jour.

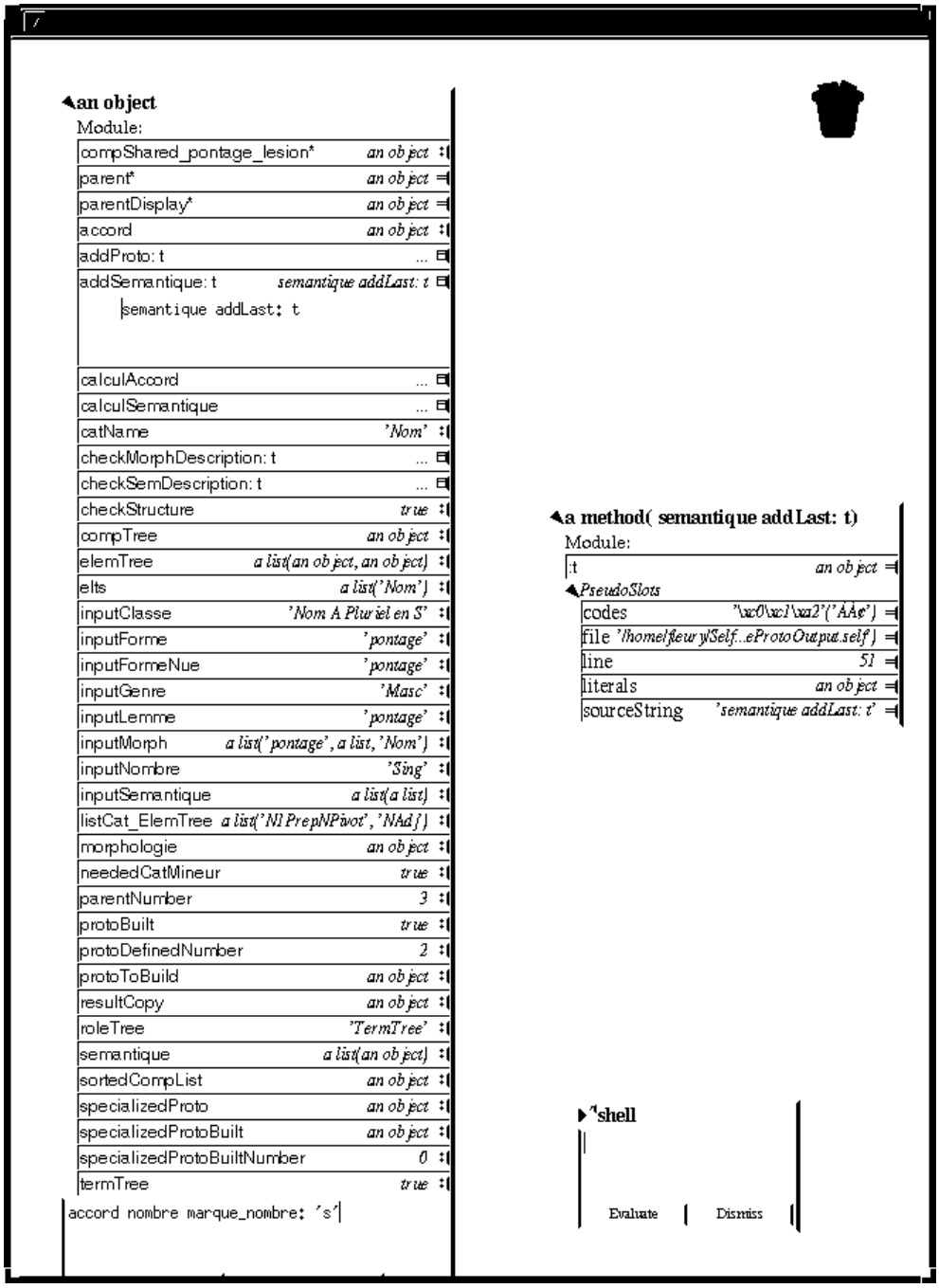


Figure 8.28 : Manipulation et modification directes d'objet (3)

Dans la figure 8.29, on effectue la mise à jour de l'attribut (la méthode) `addSemantique` : ajout d'un affichage après la mise à jour de l'attribut `semantique`. Une confirmation est demandée pour valider cette mise à jour (un voyant vert (celui du haut) pour la valider, un voyant rouge pour l'annuler (`<-"Accept, Cancel"`)). Si la mise à jour effectuée n'est pas syntaxiquement correcte, un objet portant un message d'erreur apparaît dans la fenêtre graphique et la mise à jour n'est pas enregistrée.

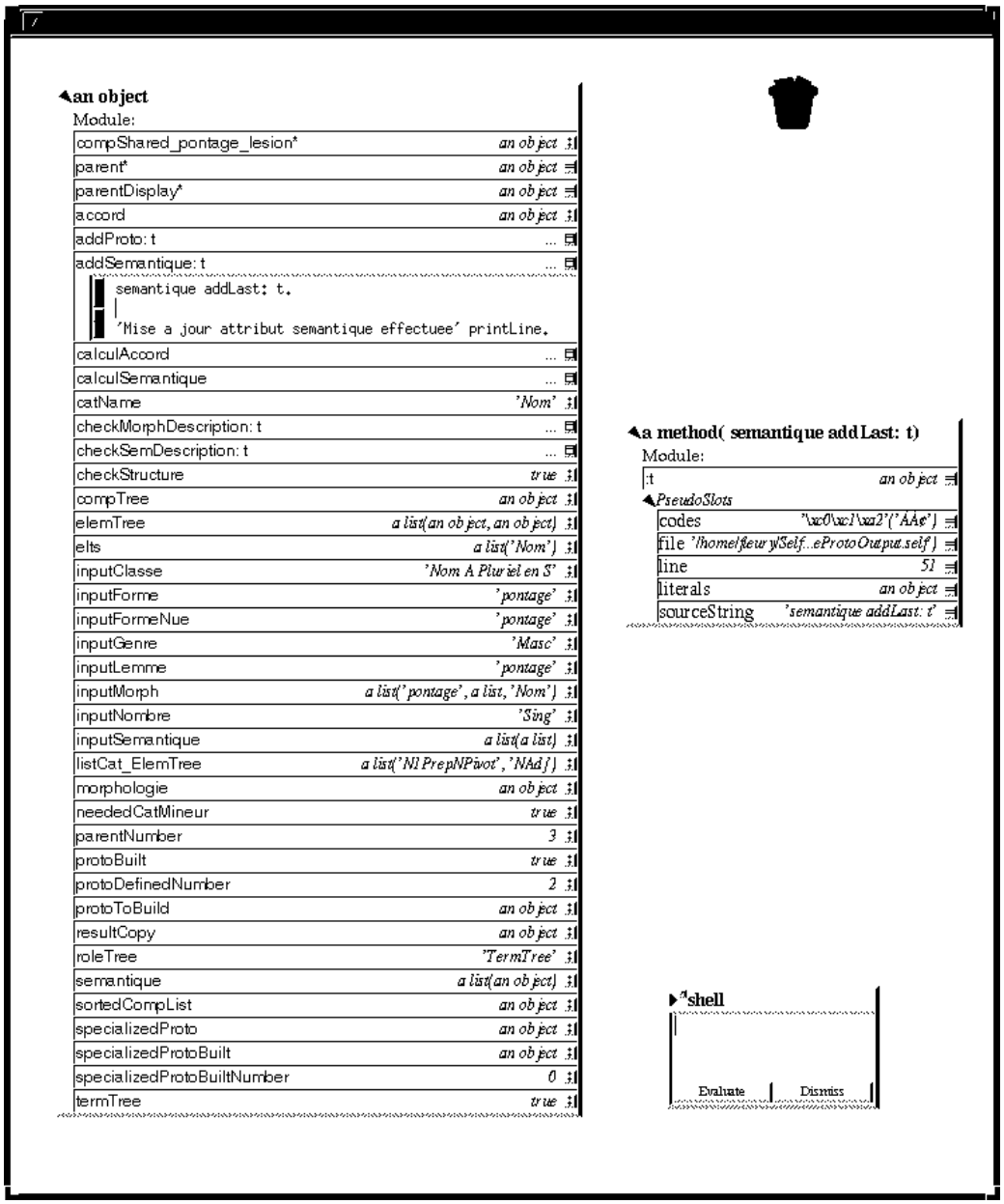


Figure 8.29 : Manipulation et modification directes d'objet (4)

Dans la figure 8.30, on trouve le résultat de la mise à jour réalisée et validée sur l'attribut (la méthode) `addSemantique`.

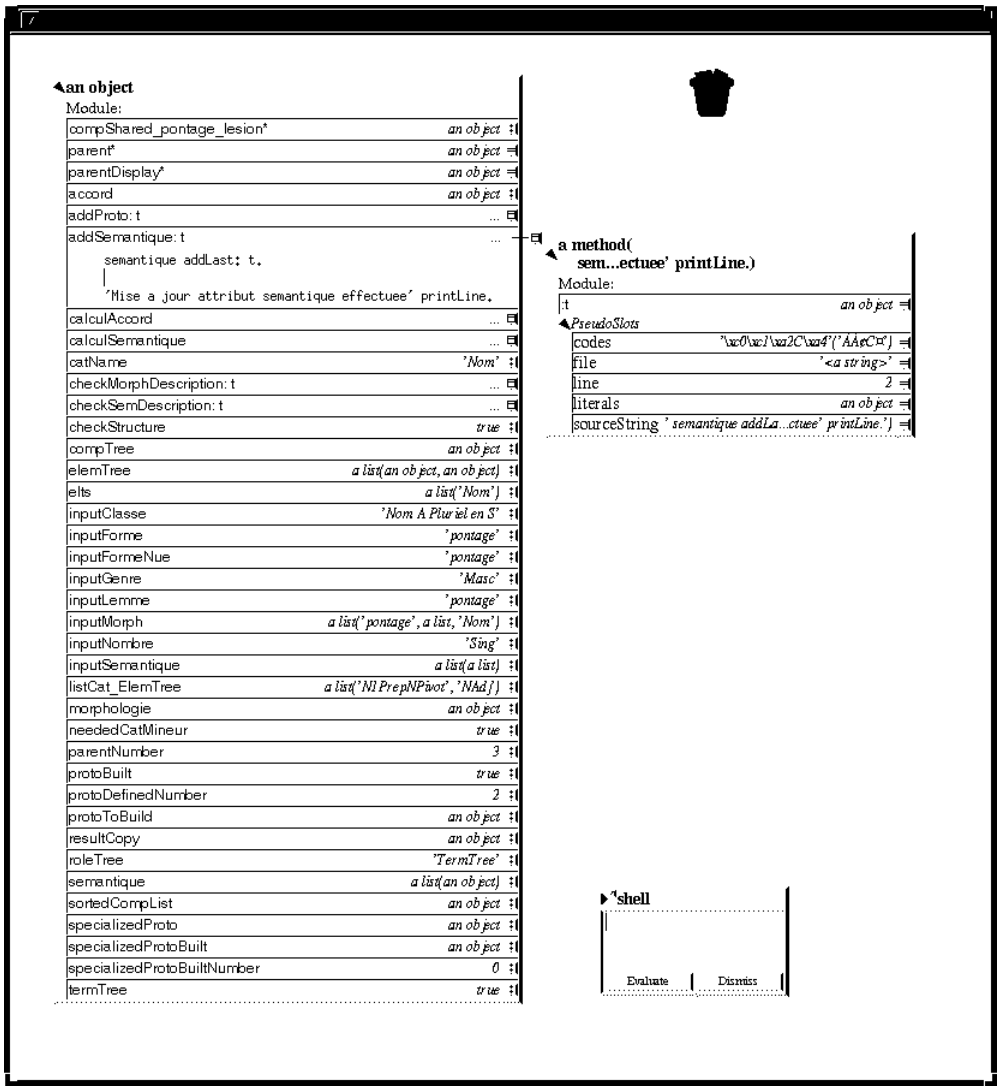


Figure 8.30 : Manipulation et modification directes d'objet (5)

Incise 3. Gaspar : un analyseur à mots implémenté avec Self-4.0

Nous présentons ici les outils mis en place pour l'implémentation de Gaspar, l'analyseur chargé de construire les représentations des mots puis leur classement. L'implémentation s'inscrit dans le cadre défini par la version 4.0 du langage à prototypes Self. Elle étend donc les outils d'analyse déjà présentés.

I.3.1. Catégorisation des outils pour l'analyse

L'analyseur Gaspar est organisé de la manière suivante:

Gaspar est un objet défini dans l'objet Self globals (un parent du lobby), il est donc accessible directement à partir du lobby (la "racine" du monde Self). Les différents outils définis ou à construire sont accessibles à partir de ce point d'entrée que constitue l'objet Gaspar. Tous les prototypes définis sous Gaspar sont donc des attributs définis dans cet objet ou dans ces sous-catégories. L'organisation des objets construits est présentée dans la figure qui suit.

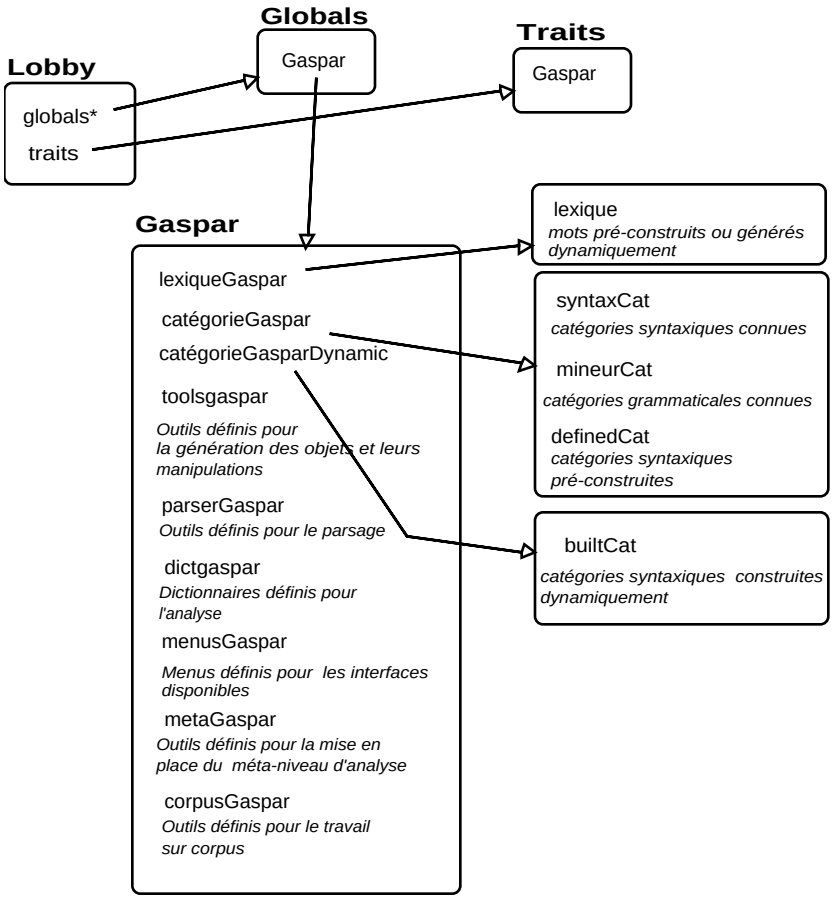


Figure 31 : Gaspar, un analyseur à prototypes

Chaque prototype dépendant de l'analyseur est catégorisé suivant son rôle dans l'"environnement" de cet analyseur. Les objets traits définis pour les objets associés à Gaspar sont définis dans l'objet Self traits qui porte tous les comportements partagés. En utilisant l'interface graphique de Self, on peut inspecter

l'objet Gaspar déclinant les différents niveaux de catégorisation présentés dans la figure précédente. Cette organisation initiale peut évidemment être modifiée: il est important de souligner qu'il est toujours possible avec Self de modifier dynamiquement et interactivement la structure des objets et donc aussi l'organisation des objets dans le monde Self.

I.3.2. Interfaces disponibles

Pour accèder à Gaspar et aux outils associés, des interfaces sont disponibles via les attributs menus définis par Gaspar. Ce dernier est le point d'entrée initial. On peut l'atteindre en chargeant le fichier initGaspar.self, ou en utilisant la syntaxe suivante:

```
gaspar menusGaspar menu
```

On peut ensuite atteindre les outils cherchés en sélectionnant une option parmi celles présentées.

- menu Load : chargement des données Gaspar

Il est bien sûr indispensable d'activer cette option pour disposer des outils Gaspar. On dispose pour le chargement des données d'une possibilité de chargement sélectif de celles-ci. De plus, on peut disposer ou non d'un lexique prédéfini ou de catégories déjà préconstruites.

- menu Parser : l'analyseur syntaxique Gaspar

L'analyseur syntaxique Gaspar peut travailler en utilisant des savoirs préconstruits (<- partie 2 chapitres 5, 6) pour l'analyse (-> partie 3 chapitre 9 : présentation du parseur). Il peut aussi travailler à partir des savoirs associés aux mots construits dynamiquement par Gaspar (<- partie 2 chapitres 7, 8 : génération automatique de prototypes à partir de savoirs extraits sur corpus).

- menu Lexique : outils pour inspecter les représentations prototypiques construites ou pour en créer.

- menu Define : outils Gaspar de génération et de classement

Ce menu regroupe les outils pour la génération dynamique des prototypes à partir de savoirs extraits sur corpus ainsi que les outils disponibles pour le classement des prototypes lexicaux et des prototypes d'arbre construits. D'autres outils sont également définis pour activer par exemple des opérations de substitution ou d'adjonction entre prototypes d'arbre.

- menu Dict : dictionnaires de Gaspar

Ce menu regroupe les outils qui permettent de lire les informations disponibles dans les différents dictionnaires définis par Gaspar.

- menu Meta : méta-niveau de Gaspar

Ce menu regroupe les outils qui permettent d'inspecter (à un niveau méta) tous les objets générés dans l'environnement défini par Gaspar et des outils qui permettent de modifier les liens entre ces objets (liens d'héritage en particulier) (-> chapitre 9). Ces outils utilisent la méta-représentation de structure définie par Self. L'objet metaGaspar est aussi l'objet initiateur des traçages potentiels de toutes les activités de Gaspar : on peut ainsi choisir de tracer les activités des différents processus définis (via l'interface graphique de Self et via la fenêtre shell), on peut en outre activer la présentation des objets construits ou sélectionnés dans la fenêtre graphique de Self. Il sera aussi capable de gérer les (méta)connaissances construites ou les évènements extérieurs qui interviennent dans les différentes phases de travail de Gaspar : ces options ne sont pas encore disponibles dans les résultats présentés ici (-> chapitre 9, -> annexe chapitre 9).

- `menu Corpus` :

Ce menu regroupe les outils qui permettent de sélectionner le corpus de travail et d'activer certaines actions sur les corpus définis (recherche de chaîne textuelle ou de cooccurrences dans les corpus disponibles).

Ces différents outils sont accessibles dans l'environnement Self, il reste évidemment possible d'utiliser l'interface graphique de Self pour manipuler ou visualiser les différents objets construits. Il est d'ailleurs possible d'activer dans la fenêtre graphique les différents objets associés à `Gaspar` via un menu `desktop` défini dans les menus construits. Ce dernier permet aussi l'ouverture et la fermeture de fenêtre(s) graphique(s).

Les outils d'interface construits constituent des raccourcis pour exécuter les différentes actions définies. Il est bien sûr plus facile de manipuler les objets individuellement via l'interface graphique. Pour visualiser par exemple les prototypes lexicaux ou syntaxiques construits dynamiquement par `Gaspar`, il suffit de les activer à partir de l'objet `Gaspar` dans la fenêtre graphique: pour faire apparaître `Gaspar` dans cette fenêtre, il suffit d'évaluer le prototype `Gaspar` dans la fenêtre `shell` disponible dans cette interface graphique. On peut ensuite parcourir les sous-catégories de cet objet pour y découvrir les objets cherchés.

On donne dans la figure qui suit une illustration de l'environnement disponible avec la fenêtre graphique de Self. On trouve dans cette figure :

- L'objet `Gaspar` (en haut et à gauche).
- La fenêtre `shell` qui permet d'activer des opérations dans cette fenêtre graphique.
- Les trois objets situés en bas et à droite sont des prototypes lexicaux ou d'arbres construits dynamiquement.
- L'objet en haut et à droite est un attribut de l'objet `Gaspar` qui contient tous les prototypes générés.

Et enfin, complètement à droite et en haut, la poubelle qui permet de nettoyer la fenêtre graphique.

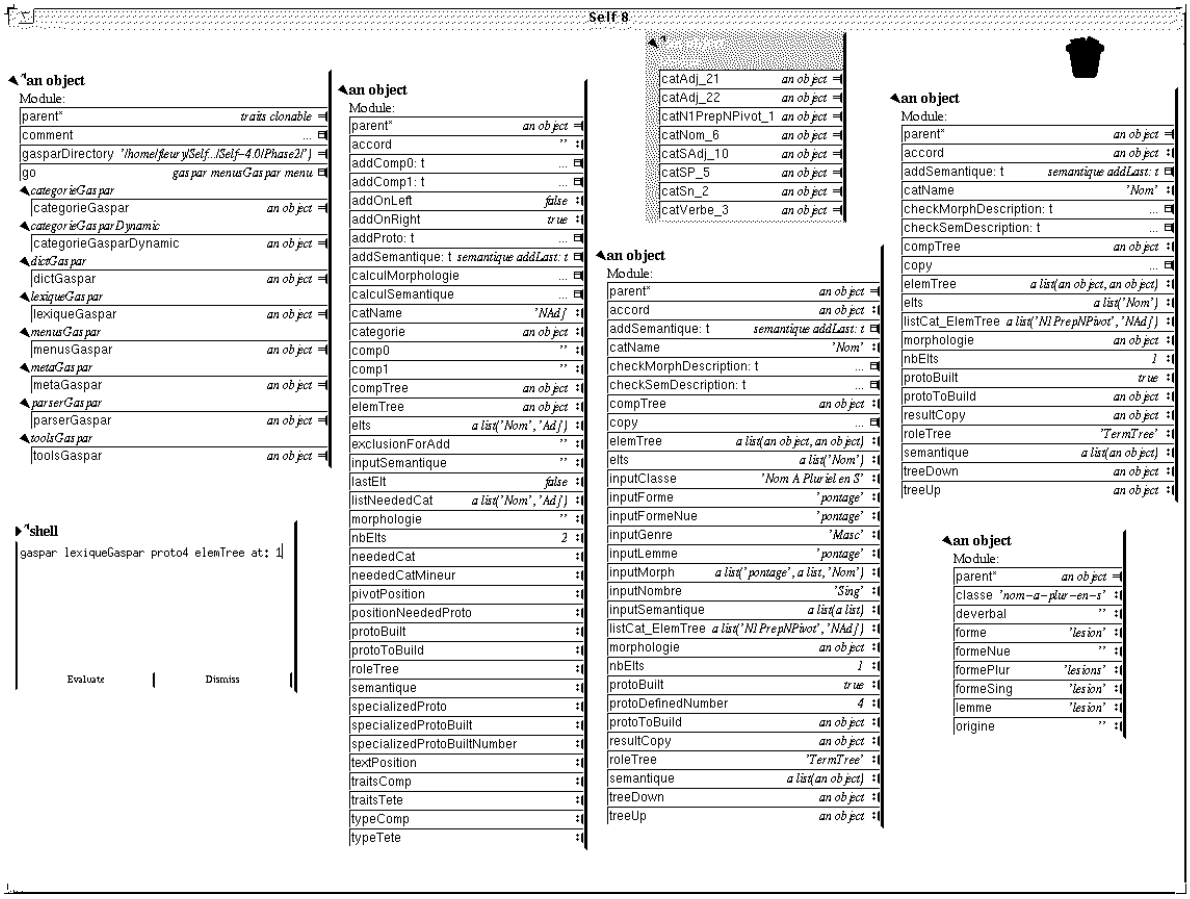


Figure 32 : `Gaspar` et l'interface graphique de Self

Incise 4. La classification évolutive est en marche

I.4. Des regards multiples sur les mots

La mise en oeuvre des processus de classement construits confirme les multiplicités de comportement possibles sur les mots. Comme on pouvait s'y attendre, on ne trouve jamais de comportement(s) partagé(s) par tous les membres d'une même famille catégorielle (<- expérimentations sur corpus). Il convient donc d'interroger les savoirs construits de manière plus fine si on veut y découvrir des similarités comportementales.

On donne ci-dessous une présentation détaillée des différentes opérations définies via Gaspar pour amorcer les classements des savoirs représentés.

I.4.1. Descriptif des opérations de classement réalisées

(1) Recherche sur tous les mots d'une même catégorie des comportements partagés (i.e. détermination des arbres propres à chacun d'eux)

Si tous les prototypes de mot d'une même catégorie partagent exactement les mêmes comportements, l'objet `traits` qui portent les comportements partagés de cette catégorie est mis à jour : ilportera ces comportements communs. Dans tous les cas, les prototypes de mots portent quant à eux leurs comportements propres .

(2) Recherche sur tous les mots d'une même catégorie des comportements partagés (i.e. détermination des arbres propres à chacun d'eux) et pour toutes les catégories

Même opération que dans (1) sur toutes les catégories.

(3) Recherche des arbres élémentaires communs à deux prototypes (i.e. détermination des arbres propres à chacun d'eux)

Si deux prototypes de mot d'une même catégorie partagent un ou plusieurs comportements, un objet `traits` est automatiquement construit pour porter ces comportements partagés. Dans ce cas on ajoute automatiquement aux prototypes concernés un attribut `parent` qui pointe sur ce nouvel objet porteur de comportements partagés. Les comportements propres à chacun des prototypes restent attachés à ces prototypes.

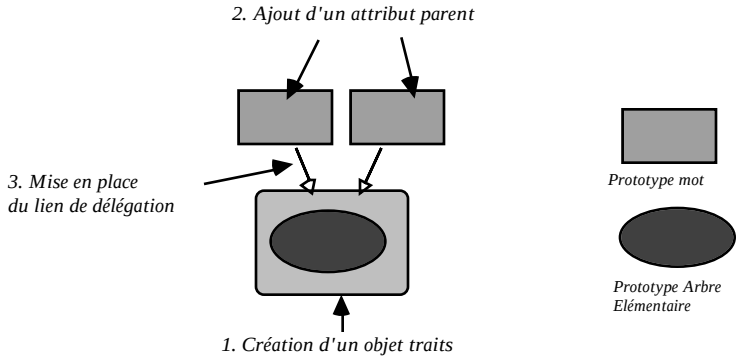


Figure 8.33 : Classement des mots deux à deux

Il est possible de réaliser ce type de recherche sur deux mots particuliers ou sur l'ensemble des mots (pris deux à deux et dans chaque catégorie). Dans la figure 8.18, on présente le résultat graphique obtenu sur la recherche de comportements partagés par deux prototypes de mot : il s'agit des deux prototypes associés aux mots `pontage` et `lésion` (-> annexe partie 2 chapitre 8 : trace de la recherche des comportements partagés par ces deux mots). Après évaluation de leur(s) comportement(s) partagé(s), un attribut parental est automatiquement ajouté à ces deux prototypes (`compShared_pontage_lesion*`) qui pointe sur l'objet (présenté à droite dans la fenêtre graphique) portant le comportement commun (une liste portant un arbre élémentaire).

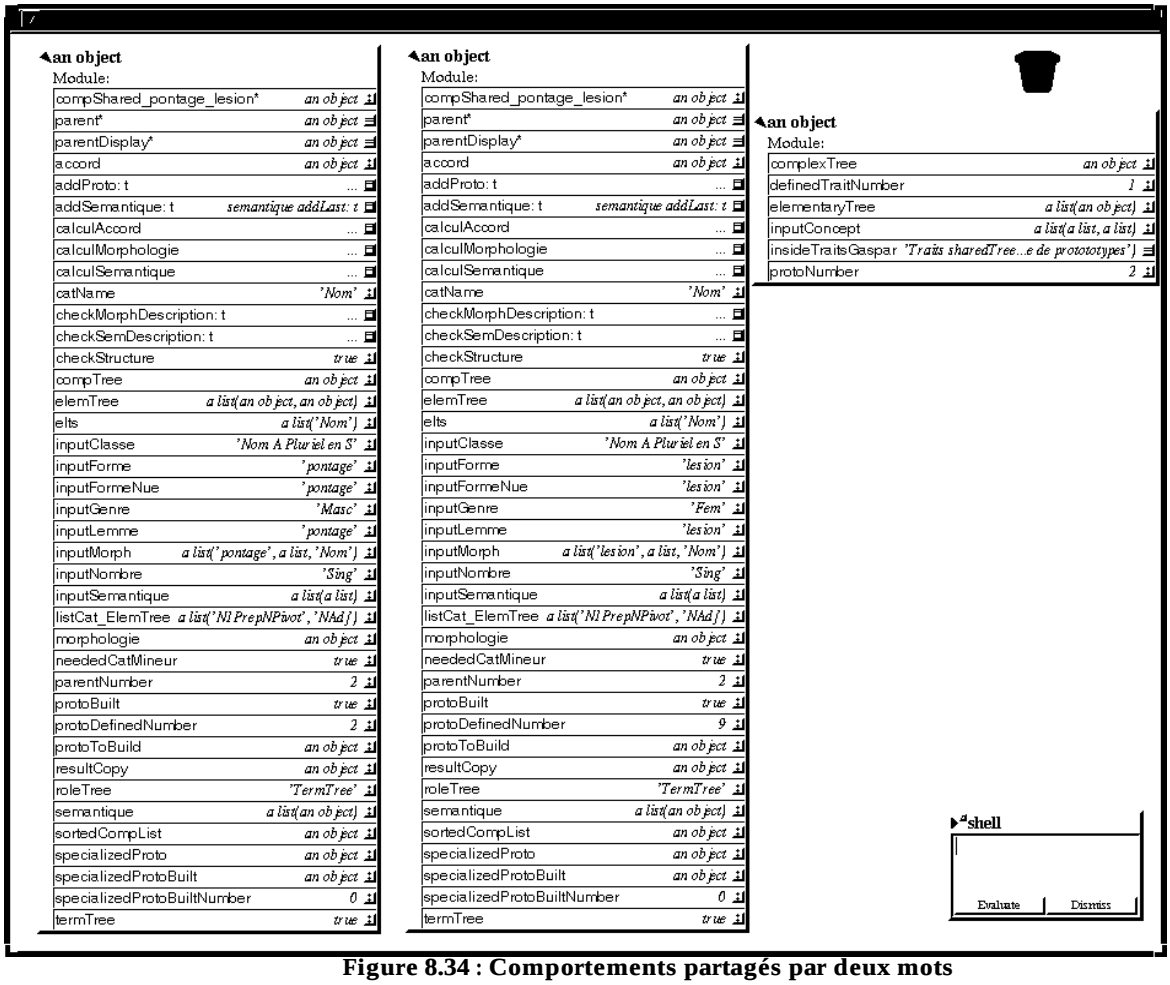


Figure 8.34 : Comportements partagés par deux mots

(4) Recherche sur des sous-familles de mots d'une même catégorie des comportements partagés et construction automatique d'un pôle de comportements partagés

Si plusieurs prototypes de mot d'une même catégorie partagent exactement les mêmes comportements, un objet `traits` est automatiquement construit pour porter ces comportements partagés. Dans ce cas on ajoute automatiquement aux prototypes concernés un attribut `parent` qui pointe sur ce nouvel objet porteur de comportements partagés. Les comportements propres à chacun des prototypes restent attachés à ces prototypes.

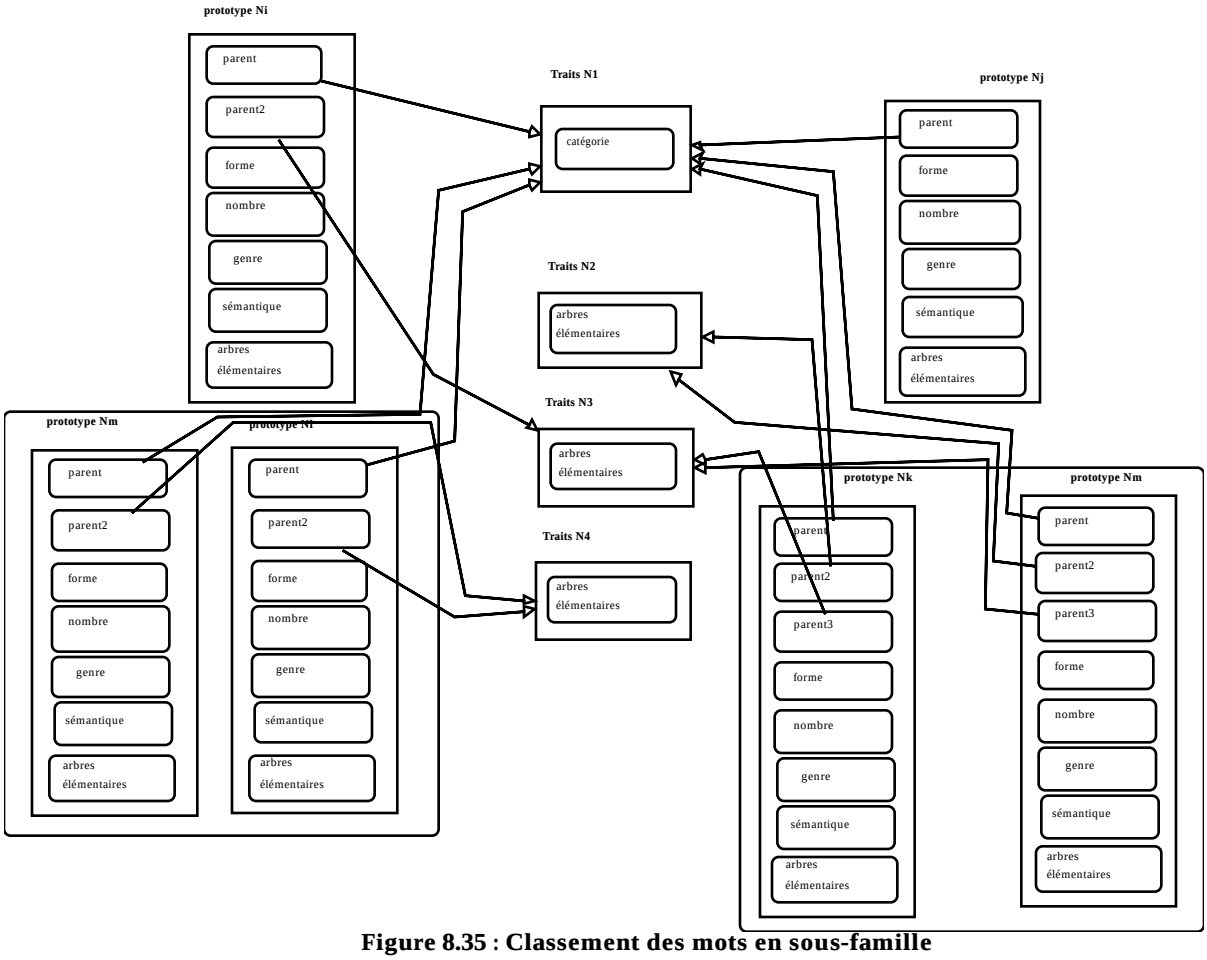


Figure 8.35 : Classement des mots en sous-famille

Dans la figure précédente, les prototypes construits (représentant des noms) sont regroupés en sous-famille en tenant compte des arbres élémentaires associés à ces prototypes de mots. Les prototypes d'une sous-famille délèguent les mêmes arbres élémentaires à un ou plusieurs pôles de comportements partagés.

(5) Recherche sur des sous-familles de mots d'une même catégorie (et pour toute les catégories) des comportements partagés et construction d'un pôle de comportements partagés

Même opération que dans (4) sur toutes les catégories.

(6) Recherche sur des sous-familles d'arbres élémentaires d'une même catégorie des comportements partagés (arbres d'analyse) et construction d'un pôle de comportements partagés

Si plusieurs prototypes d'arbre élémentaire d'une même catégorie partagent exactement les mêmes comportements, un objet `traits` est automatiquement construit pour porter ces comportements partagés. Dans ce cas on ajoute automatiquement aux prototypes concernés un attribut `parent` qui pointe sur ce nouvel objet porteur de comportements partagés. Les comportements propres à chacun des prototypes restent attachés à ces prototypes.

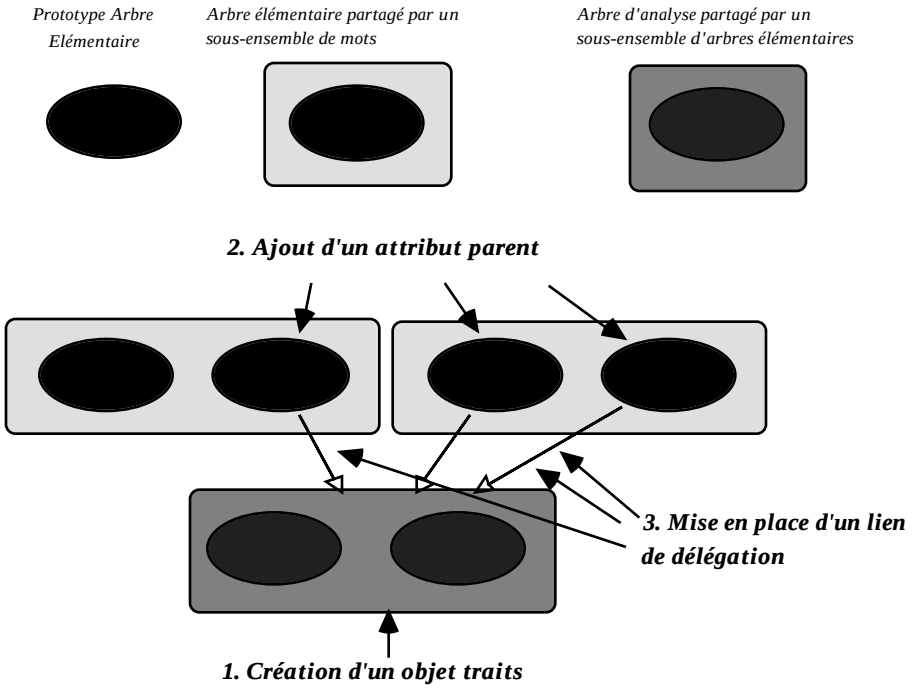


Figure 8.36 : Classement des arbres élémentaires

La détermination des comportements partagés par deux arbres élémentaires (à la manière de (3)) reste à faire.

On donne en annexe une trace de tous ces processus (-> annexe partie 2 chapitre 8).

I.4.2. Mise en place de réseaux de prototypes : autant de réseaux à interpréter

Le dispositif construit permet donc d'activer des processus de classement qui proposent des regards multiples et croisés sur les savoirs représentés. Ces processus construisent des réseaux de hiérarchies locales entre prototypes de mots et prototypes d'arbres ou entre prototypes d'arbres, ces liens multiples constituent autant de pistes de sens à interpréter.

- Examen des comportements partagés sur des sous-familles de mots d'une famille catégorielle

A partir des prototypes de mot d'une même catégorie (en haut de la figure) on construit une première couche de pôles de comportements partagés par des sous-familles de prototypes (opération (4) décrite supra).

- Examen des comportements partagés par les mots (deux à deux)

A partir de deux prototypes de mot d'une même catégorie, on peut aussi construire une seconde couche de pôles de comportements partagés par ces deux prototypes (opération (3) décrite supra).

- Examen des comportements partagés par les arbres élémentaires

Enfin, à partir des prototypes d'arbre élémentaire d'une même catégorie (les ellipses noires ou fortement grisées de la figure) on construit une nouvelle couche de pôles de comportements partagés par des sous-familles de prototypes (opération (5) décrite supra).

Le schéma qui suit est une illustration simplifiée des réseaux mis en place.

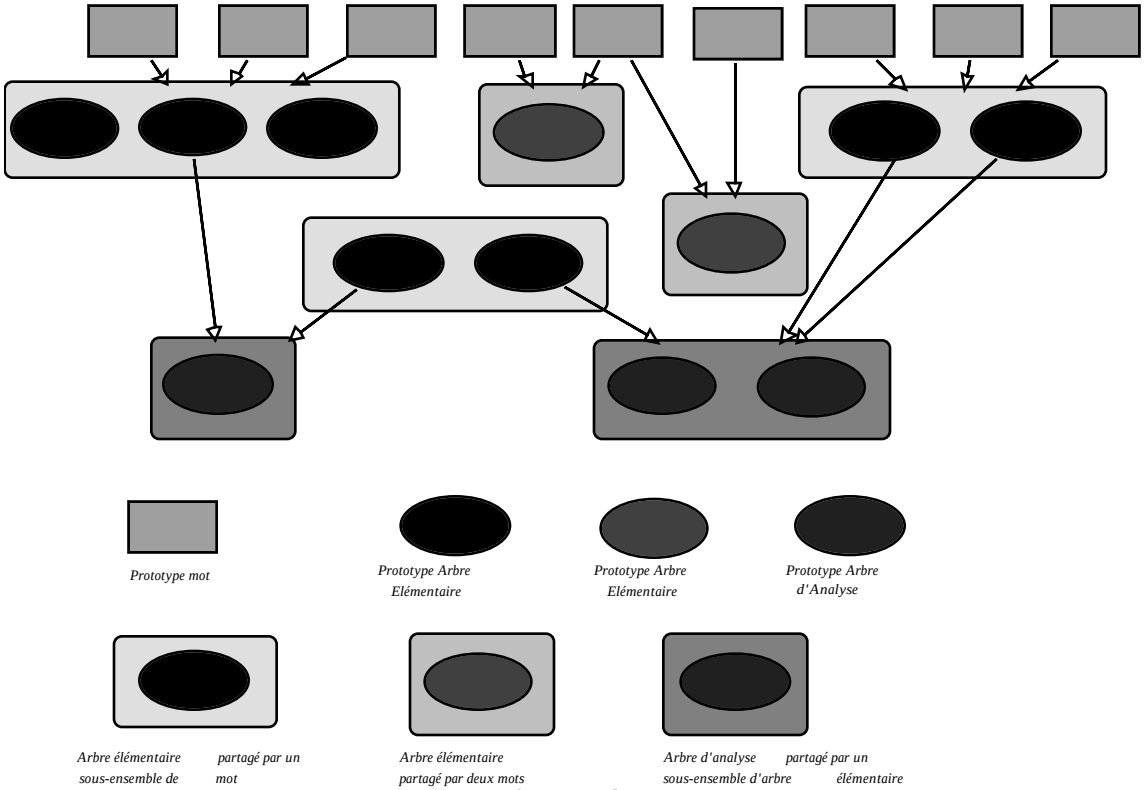


Figure 8.37 : La classification évolutive est en marche

Incise 5. Approche d'un méta-protocole

I.5. Une approche expérimentale guidée par le méta-regard du linguiste

Les outils construits produisent des amorces de résultat qu'il convient d'affiner (Habert & al. 1996). Ils permettent en outre de guider la mise au point de nouveaux résultats : on peut à chaque étape examiner les résultats déjà établis pour ensuite soit sélectionner parmi les outils disponibles ceux qui permettent d'améliorer la qualité des résultats, soit agir directement sur les savoirs représentés pour les modifier ou pour les remodeler.

Ainsi la détermination d'un classement global des mots d'une catégorie sur la base des comportements partagés par l'ensemble de ces mots (opération (1) supra) nécessite à coup sûr un examen plus fin de ces mots et de leurs comportements respectifs si on veut y déceler des regroupements de comportements partagés sur des ensembles restreints de mots.

La mise en place de regards croisés sur les savoirs représentés dépend donc directement du méta-regard de l'utilisateur sur les résultats qu'il génère en activant les outils définis : le méta-regard de l'utilisateur et ses interventions guident le travail d'affinement des résultats construits et leur interprétation (<- incise 4).

I.5.1. Du méta-regard à l'automodification

Le système Gaspar active plusieurs niveaux de représentation pour évaluer les savoirs manipulés :

1 Les programmes construits utilisent des méta-processus (-> partie 3 chapitre 9) capables d'évaluer la structure interne des objets construits. On peut vouloir sélectionner et rechercher parmi tous les objets disponibles un objet particulier avec un type donné d'attribut. Pour réaliser ce type d'opérations, on utilise des méta-processus capables de lire la structure des objets (-> partie 3 chapitre 9), et on récupère l'objet adéquat. Les programmes construits utilisent ce type de processus pour, par exemple, modifier dynamiquement des liens de délégation. On dispose aussi d'outils capables de supprimer les liens de délégation de même nature (i.e. les mêmes comportements étant portés par deux ou plusieurs pôles différents). Ces opérations, intégrées au programme construit restent évidemment transparentes pour l'utilisateur.

Le point important à souligner dans cette phase de développement de notre analyseur est que **les méta-processus disponibles avec Self ont été généralisés dans toutes les phases de travail qu'il est possible d'activer via Gaspar**. On a vu supra qu'il existait désormais une interface particulière dans Gaspar pour activer ce genre des outils. Ces derniers, activables par l'utilisateur, permettent un méta-regard sur les objets construits et des actions qui se réalisent en utilisant les méta-processus disponibles avec Self.

Ce méta-niveau reste principalement capable d'évaluer la définition structurelle des objets et d'agir sur elle : il n'est en aucun cas capable d'interpréter les méta-opérations réalisées. **Ce premier type d'automodification des représentations construites reste donc limité à la structure interne des objets portant les savoirs représentés.**

2 L'utilisateur constitue à lui tout seul une "entité" de méta-contrôle indispensable pour évaluer les résultats construits. C'est à lui d'interpréter et d'évaluer toutes les mises à jour réalisées : il peut d'ailleurs lui-même être à l'origine de modifications sur les savoirs représentés. Ce sont les outils construits à partir des fonctionnalités de Self (primitives du langage) qui permettent à l'utilisateur d'évaluer les objets construits.

Puisque l'utilisateur joue un rôle central dans notre approche expérimentale, il est important de souligner que si l'on veut qu'il soit capable d'intervenir pour évaluer ou modifier les objets ou les résultats construits,

il doit pouvoir suivre pas à pas les différentes opérations réalisées par les programmes construits et évaluer les changements d'états des objets manipulés (Habert & Fleury 1993a, 1993b). Il est donc impératif de disposer de processus de traçage de toutes ces opérations. Nous avons déjà indiqué et nous le reverrons infra (-> partie 3 chapitre 9) que les processus mis en place via Gaspar permettent d'activer à tout moment de tels processus de traçage qui présentent les différents états pertinents des traitements réalisés et des résultats construits à chaque étape.

I.5.2. Limites actuelles des méta-processus

Tous les méta-processus définis n'agissent en fait que de manière structurelle par rapport aux objets construits. Ils permettent soit d'ajouter un comportement, soit de retirer un comportement, soit enfin de modifier globalement un comportement. En effet, si on veut affiner un comportement dans un programme et de manière transparente pour l'utilisateur, il est nécessaire de le redéfinir complètement. Les processus disponibles ne sont pas encore capables de modifier partiellement la valeur d'un attribut, et notamment des attributs qui portent des procédures. Si l'on doit modifier ce type d'attribut de manière transparente pour l'utilisateur (et sans passer par l'interface graphique), il convient de modifier globalement la valeur de cet attribut i.e réécrire le code associé et le réaffecter à l'attribut visé. Si l'utilisateur peut intervenir dynamiquement et de manière interactive sur tous les objets construits en utilisant l'interface graphique de Self , il reste important de souligner qu'à moins de connaître toutes les finesses de la programmation avec Self, il reste difficile à un utilisateur non spécialiste de ce langage de gérer tout seul ce type de mise à jour.

La mise au point de tels outils de mise à jour dynamique et ponctuelle des objets construits est d'ailleurs loin d'être triviale (Cointe & al. 1993, Cointe & al. 1995) (-> partie 3 chapitre 9 : réflexivité comportementale).

Partie 3 : De la Science Fiction

Chapitre 9

La réflexivité pour une évolution des représentations dans le traitement automatique du langage naturel

9.1. Réflexivité, une introduction

L'énigme MU est présentée dans (Hofstadter 1986) pour illustrer les limites des systèmes formels. Au départ une question : "Pouvez-vous produire MU ?". Pour cela on dispose des éléments suivants :

L'axiome MI, les symboles M, U, I et les règles :

- Règle 1 : si vous possédez une chaîne se terminant par I, vous pouvez lui ajouter un U à la fin.
- Règle 2 : supposons que vous ayez Mx. Vous pouvez alors ajouter Mxx à votre collection.
- Règle 3 : si la chaîne des symboles III apparaît dans votre collection, vous pouvez former une nouvelle chaîne en remplaçant III par U.
- Règle 4 : si le groupe de symboles UU apparaît dans l'une des sous-chaînes, vous vous pouvez le supprimer.

La démarche raisonnable pour commencer à produire MU (en s'aMUsant) est de travailler à l'intérieur du système ainsi défini. C'est à dire à produire des dérivations successives à partir de MU :

règle 2 règle 2 règle 3
MI ----> MII ----> MIII ----> MIIIU ....

Jusqu'où peut-on aller dans cette dérivation sans perdre patience ? Il est préférable en fait d'examiner ce système et son fonctionnement à un niveau extérieur au processus de dérivation, pour évaluer pourquoi il est si difficile de produire MU. Si on regarde ce système de l'extérieur, on peut remarquer que :

- 1. Les nombres 2 et 3 jouent un rôle important, puisque les séries de trois I et de deux U sont éliminées.
- 2. Le système utilise des tendances opposées : des règles d'allongement (1 et 2), et des règles de raccourcissement (3 et 4).

Pour résoudre l'énigme il faut aller encore plus loin et utiliser un outil extérieur au système : la théorie des nombres. Si on compte le nombre de I des théorèmes du système, on s'aperçoit très vite qu'il n'est jamais nul : on ne peut pas éliminer tous les I. Les règles 1 et 4 n'ont aucun effet sur le nombre de I. Seules les règles 2 et 3 influent sur ce nombre. La règle 3 diminue de 3 le nombre de I. Dans une application de la règle 3, le nombre

de I pourrait très bien être un multiple de 3 si le nombre initial de I était lui aussi un multiple de 3 : cette règle ne peut créer un multiple de 3 qu'à condition de commencer avec un multiple de 3. La règle 2 double le nombre de I. De même, elle ne produira pas de multiple de 3 s'il n'y en avait pas un avant.

Pour résumer :

- 1. Le nombre de I initial est 1, et 1 n'est pas un multiple de 3.
- 2. Deux des quatre règles n'ont aucun effet sur le nombre de I.
- 3. Les règles 2 et 3 modifient le nombre de I de telle sorte qu'aucun multiple de 3 ne peut être produit.

La conclusion s'impose : le nombre de I ne peut jamais devenir un multiple de 3 et il est impossible que le nombre de I soit nul. C'est la méta-représentation du système MU dans la théorie des nombres qui a permis de résoudre cette énigme. Il faut se reculer pour observer l'ensemble de la situation. Puis choisir la méta-représentation qui servira de base de travail.

9.2. Concepts fondamentaux de la réflexivité : l'apport de Pattie MAES (MAES 1987)

9.2.1. (Méta) Système (Réflexif)

Réflexivité

"Il y a réflexivité chaque fois qu'un système s'applique à lui-même."

On parle de système pour décrire un mécanisme lié à une certaine partie du monde (le domaine). Ce système comprend une représentation des objets liés à ce domaine et les moyens de travailler avec/sur ces objets. Un méta-système est un système dont le domaine d'activité est lui-même un système lié à un domaine. On dira qu'un système est réflexif s'il raisonne et agit sur lui-même. Un système est réflexif si le méta-système associé est ce même système.

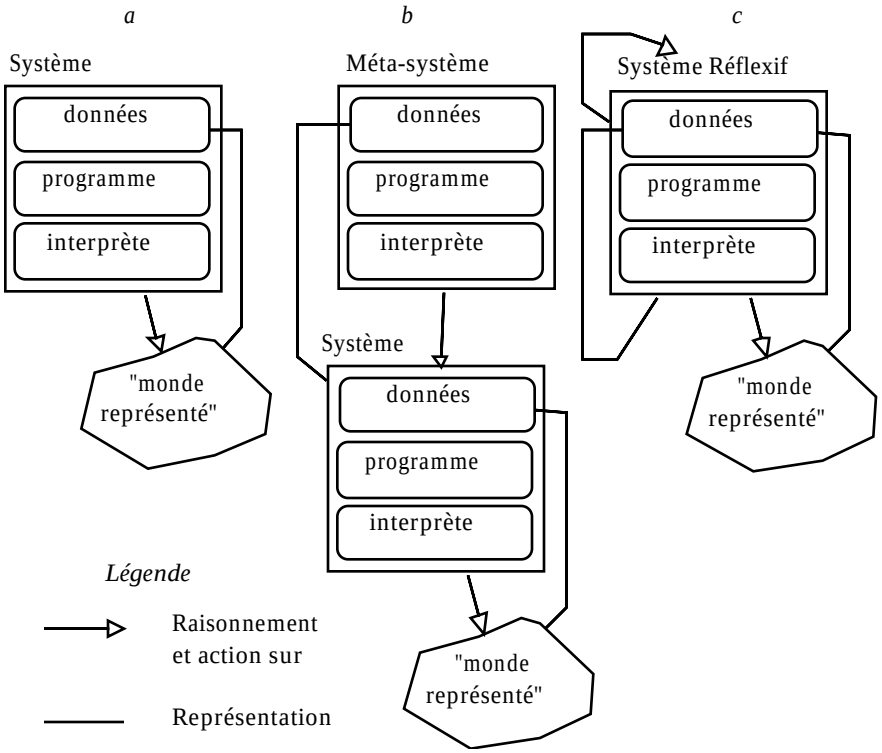


Figure 9.1 : (Méta) Système (Réflexif).

Ainsi on parle de réflexivité quand :

- Une partie d'un système fait référence au système dont elle fait partie, un énoncé déclaratif fait référence à une partie du système dont il fait partie :

  "Cette phrase contiendrait sept mots si elle en avait cinq de moins."

- Le système crée une copie de lui-même : dans les mécanismes cellulaires, les enzymes interviennent dans des créations de copie d'elles-mêmes (cf infra).

- Une partie A d'un système peut modifier une autre partie B de ce système, si A est capable d'apprendre à modifier B en vue d'obtenir de meilleures performances pour B.

- Le système s'observe sans se modifier pour évaluer son propre comportement (expertise de diagnostic, d'édition, d'interprétation...). On trouve très fréquemment dans les gros systèmes informatiques des outils de surveillance de l'activité de l'ensemble du système : CPU disponible, prise en compte des entrées sorties,...(que l'on trouve par exemple sous forme de “Monitoring", ou d'outils chargés de détecter des erreurs spécifiques (problèmes d'écriture sur disque...) et capables d'avertir le responsable (humain) du système pour qu'il intervienne.

La biologie moléculaire est d'ailleurs un très bon exemple de domaine où les phénomènes de différences de niveau de représentation agissent les uns sur les autres. On peut ainsi illustrer ce phénomène par l'exemple de la synthèse des protéines. Le gène provoquant la ou les protéines, fait partie d'un ensemble permettant de réguler la production des protéines en fonction des besoins de la cellule. Cet ensemble s'appelle l'opéron, il est composé d'une séquence d'ADN, appelé gène promoteur, d'une autre, le gène opérateur, et d'une troisième, le ou les gènes de structure. Un gène régulateur est associé à ce mécanisme. L'ADN se trouve dans le noyau des cellules. Il se compose de longues chaînes de molécules appelées nucléotides. Or si quelques protéines sont fabriquées dans le noyau, la plupart d'entre elles sont synthétisées à l'extérieur (dans le cytoplasme) pour y être utilisées. Il existe donc nécessairement des intermédiaires exécutant dans le cytoplasme le programme se trouvant dans le noyau. Une protéine est une suite d'acides aminés (aa). Avec les vingt aa existant, des milliers de protéines différentes peuvent se constituer : elles seront différentes de par leurs séquences d'aa et auront donc des rôles différents dans la cellule. Un programme inscrit sur le gène est très strict : il s'agit de déterminer avec précision l'ordre des aa composant la protéine produite. A chaque groupe de trois nucléotides correspond un aa particulier. En pratique, une copie du message du gène de structure est constituée sous la forme d'une molécule d'ARN-Messager (ARN-M). Cette molécule est capable de traverser la membrane nucléaire. D'autres intermédiaires (ARN de transfert) vont chercher dans le cytoplasme des aa qui s'assembleront selon la suite des nucléotides de l'ARN-M et donc de l'ADN. Ce qui est remarquable dans ce mécanisme, c'est que la cellule peut doser la production des substances dont elle a besoin. Lorsque la cellule n'a pas à produire une certaine protéine, la copie du gène de structure l'ARN-M ne peut pas s'effectuer. Lorsque la synthèse doit reprendre, les produits bloquant l'opéron du gène opérateur sont inactivés et l'ARN-M peut de nouveau produire.

9.2.2. Langages de programmation informatique et réflexivité

Un programme réflexif se définit au travers d'une reproduction intérieure des représentations construites et des modifications qu'il y apporte.

Plusieurs approches existent pour construire des systèmes réflexifs. Tout d'abord, certains systèmes incluent une part de réflexivité. Depuis peu les Systèmes d'Apprentissage sont implémentés en prenant en compte de manière adéquate les processus réflexifs. Il y a ensuite les systèmes qui font usage de processus réflexifs du fait qu'ils sont implémentés dans des langages de programmation qui fournissent des facilités pour l'utilisation de ces processus. LISP est un exemple d'un tel langage. Certaines fonctions lispiennes permettent un accès sur leur propre fonctionnement. Par exemple considérons les fonctions suivantes :

```
(defun integer-stream () '(0))
(defun integer-stream-element ()
   (let* (( list-of integers (integer-stream))
          (last-generated-integer (car list-of-integers)))
   (eval `(defun integer-stream ()
                  ',(cons (1+ last-generated-integer)
                     list-of-integers)))
   last-generated-integer))
```

Au premier appel de la fonction `integer-stream-element`, celle-ci retourne `0`, et elle redéfinit la fonction `integer-stream` qui devient :

```
(defun integer-stream () '(1 0))
```

et donc, au prochain appel de la fonction `integer-stream-element`, celle-ci retourne `1` ....

Il y a enfin les systèmes implémentés dans des langages dont la structure interne inclut une spécification explicite de la réflexivité (à l'image du langage 3-LISP, développé par B. C. Smith (1982)). Tout système implémenté avec un de ces langages peut raisonner et agir sur lui-même. On dit que de tels systèmes ont une architecture réflexive. Une caractéristique des langages construits avec une architecture réflexive est l'identité de représentation du niveau de l'activité liée à la réflexité et le niveau lié à l'exécution propre au système.

9.2.3. Architecture réflexive

On dit qu'un langage dispose d'une architecture pour X si le langage donne de l'importance au concept X et donc s'il fournit des primitives adéquates pour des activités réflexives. Ainsi un langage de programmation fournit une architecture réflexive si la réflexivité est un concept fondamental du système. Ceci implique que le langage produit du code spécifique au processus de réflexivité compréhensible par l'interpréteur de ce langage. Un langage possédant une architecture réflexive est tel que toute activité du système a une représentation d'elle-même au moment de son exécution. Une architecture réflexive permet par exemple d'utiliser des processus de réflexivité à un certain moment d'élaboration d'un programme pour par exemple mettre au point celui-ci. Une analyse automatique développée dans ce cadre pourrait ainsi construire des méta-représentations liées aux différents stades de l'analyse et ainsi permettre d'évaluer par exemple la circulation des traits manipulés par la grammaire. Les systèmes informatiques utilisent certains processus réflexifs pour par exemple : établir des statistiques sur les ressources utilisées ou disponible par ou pour le système, prévenir les utilisateurs si certaines erreurs sont détectées (problème d'écriture sur un disque...), permettre le traçage d'un programme, le "debuggage", etc. Mais à la différence des systèmes à architecture réflexive, les systèmes informatiques classiques n'ont pas été implémentés en prenant en compte la réflexivité comme un concept fondamental. Pour les systèmes à architecture réflexive, les processus de réflexivité sont développés de manière plus modulaire ce qui rend les efforts d'implémentation moins ardus et produit des systèmes plus élégants et plus ouverts. Le langage 3-LISP (Smith 1982), est conçu avec deux types de fonctions. Les fonctions simples construites avec `define-SIMPLE` ou `lambda-SIMPLE` et qui construisent des objets fonctionnels "classiques" pour le calcul informatique. Les fonctions réflexives construites avec `define-REFLECT` sont des fonctions dédiées au processus de réflexivité. Celles-ci ont accès à une représentation globale de l'environnement du système en activité. Ces fonctions travaillent sur des variables qui représentent l'environnement et peuvent les inspecter et les modifier.

9.2.4. Développement d'une architecture réflexive

9.2.4.1. Comment se fait l'auto-représentation du système ?

Etre capable de générer des informations sur l'état courant d'un processus en cours, cela signifie par exemple que l'interpréteur du langage qui dispose d'informations sur toutes les primitives du langage doit aussi être capable de construire des représentations de ces primitives lorsqu'elles sont instanciées (i.e une méta-représentation de ces primitives). De plus, il n'est pas suffisant de produire des informations sur l'état d'un système encore faut-il que cette représentation soit accessible. Comme on l'a dit précédemment le fait d'utiliser les mêmes schémas de représentation pour la partie réflexive et pour la partie exécution d'un système permet une interaction cohérente entre les deux niveaux de représentation. Il est cependant nécessaire d'établir des "frontières" entre ces deux niveaux de représentation. Ce que l'on peut illustrer de la manière suivante :

```
He said I'll do the job
He said : "I'll do the job"
```

Les guillemets ci-dessus introduisent cette différence de niveau entre les deux parties de l'énoncé. Dans le travail présenté dans DINOSOR (Fleury & al. 1992) on dispose de trois niveaux de représentation de la grammaire : une représentation associée à la grammaire initiale, une représentation associée à la

grammaire normalisée et une représentation associée à la grammaire compilée. Chacune de ces représentations est attachée à un objet spécifique et les trois objets ainsi créés sont nettement dissociés.

9.2.4.2. Comment programmer la partie réflexive ?

Un système doit être capable de modifier ses propres auto-représentations. Il doit aussi être possible d'effectuer un va-et-vient permanent entre la phase d'exécution d'un processus et la phase de réflexivité liée à ce processus.

9.2.4.3. Quel lien existe-t-il entre le système et la partie réflexive ?

Les données liées au domaine de la réflexivité doivent être liées au système de telle sorte qu'une fois achevées les manipulations dans une phase d'activité réflexive, le retour à l'exécution d'un processus du système implique que le système ait accès à ces données mises à jour (et réciproquement). Un travail de maintenance permanente est donc nécessaire pour la cohérence des deux niveaux de représentation. Il faut pouvoir disposer de programmes qui peuvent modifier le flot de contrôle quand ils sont en train de fonctionner. Il ne suffit pas de constater que quelque chose ne fonctionne pas dans le déroulement d'un programme, il faudrait pouvoir y remédier : on retrouve là le problème de la mise au point des programmes ou encore, dans le cas des analyseurs, le fait de savoir pourquoi ça ne marche pas comme on le voudrait... Avec 3-Lisp, il est possible d'indiquer comment changer le processus d'interprétation : si le programme détecte un certain type d'erreurs, il peut changer le comportement de l'interpréteur de façon à remédier à cette erreur sans passer par un programmeur pour modifier le programme.

9.2.5. Utilisation de la réflexivité

Il faut pouvoir se servir de ce que l'on a obtenu dès qu'on l'a obtenu. Pour passer d'une étape à une autre, on se sert des résultats que l'on vient d'obtenir. On verra infra comment la mise en place d'une représentation de la grammaire sous une forme normalisée s'est révélée primordiale pour la mise au point des modifications apportées au code initial. On doit donc pouvoir accéder aux représentations du système lui-même à tout moment de l'activité du système. Ce qui doit être accessible au processus de réflexivité :

- Les programmes du système.
- Les données du système.
- Le programme d'un processus en cours d'exécution.
- Les données d'un processus en cours d'exécution.
- La réflexivité rend possible le contrôle du flux des calculs de manière modulaire.

Un système réflexif peut comporter plusieurs modules et l'un d'entre eux modifie un autre. C'est le cas d'un système capable d'apprendre contenant une partie performance qui résout les problèmes et une partie apprentissage qui modifie la précédente au vue de ses résultats. Le système se modifie même si en réalité la partie qui modifie est distincte de la partie modifiée. Il est clair que pouvoir accéder aux représentations des programmes d'un système pendant leur déroulement est un moyen efficace de contrôle. La maintenance, l'acquisition et la communication de l'information contenue d'un système est une tâche qui suppose une connaissance importante sur toutes les activités d'un système donné et les interactions possibles entre certaines activités.

9.2.6. Dangers de la réflexivité

Si un système est capable de modifier son comportement, est-ce que cela entraîne un transfert de contrôle définitif de l'activité du système vers lui-même? Il faut ainsi remarquer que la réflexivité peut entraîner une perte d'informations. En effet si un système peut modifier son état courant quand il détecte une erreur, une incorrection ou quand les valeurs attachées à certaines variables provoque cette modification de l'état courant, on perd toute trace de l'environnement initial (à moins de mettre en place une nouvelle couche qui stocke les représentations des états précédents du système!). Or il arrive fréquemment que le programmeur (ou l'utilisateur) souhaite avoir accès à la représentation de l'environnement de l'état en cours du système pour évaluer par exemple la valeur attachée à une certaine variable. Ou, par exemple, lors d'une erreur du système qui nécessite sa réinitialisation, à la représentation du fichier généré par cet arrêt de la machine. Aussi l'auto-modification permanente d'un système peut entraîner une impossibilité d'accès aux représentations du système. On voit pourtant très bien l'intérêt que pourrait représenter l'existence de processus réflexifs notamment dans les périodes de mise au point de programme, c'est à dire à un moment où l'on souhaite avoir accès à tous les paramètres que le système en cours de mise au point manipule.

Un autre problème dans l'utilisation des processus réflexifs est celui de la gestion efficace des interactions entre les différents niveaux de représentation d'un système. Comment faire en sorte qu'il n' y ait pas de confusion entre les informations liées au système propre et les informations liées aux processus de réflexivité ? Dans le cas de l'analyse automatique, OLMES (Habert 1993) opère une transcription de la grammaire initiale. En plus de cette représentation de la grammaire, il faut pouvoir produire une représentation dédiée à la réflexivité et contenant les mêmes informations. Il convient donc d'établir une distinction nette entre ces deux niveaux de représentation. Même si le système en question est construit de telle sorte que les différents niveaux de représentation manipulent des primitives distinctes, il n'est pas évident que l'existence de ces deux niveaux de représentation rende la compréhension de tout ce qui est en jeu dans le système très claire pour le programmeur et/ou l'utilisateur.

9.2.7. Comment programmer la partie réflexive?

Les langages à architecture réflexive construisent une auto-représentation du système. Cette méta-représentation doit prendre en compte la formalisation des aspects sémantiques, syntaxiques, opérationnels, des phénomènes d'inférence propres au système. L'auto-référence peut-être implicite ou explicite. On parle de réflexivité implicite si le bon fonctionnement du système nécessite une activité réflexive permanente. Dans ce cas il est impératif d'assurer une cohérence constante et systématique entre le système et sa méta-représentation. (Le système s'applique à lui-même mais il ne le sait pas) On parle de réflexivité explicite si les processus réflexifs sont utilisés comme des fonctionnalités possibles mais non obligatoires. Ils interviennent que si l'on fait explicitement appel à eux. Il faut ensuite établir le type de lien entre le système et sa méta-représentation. Deux types de solution sont possibles :

- La réflexivité procédurale, le lien est implémenté.
- La réflexivité déclarative, le lien est spécifié.

9.2.7.1. Réflexivité procédurale

L'auto-représentation du système sert à implémenter le système. Le problème de la maintenance de cohérence des deux niveaux de représentations ne se pose donc pas. Il y a une seule représentation du système qui sert à l'implémenter et aussi à agir et réfléchir sur lui-même.

9.2.7.2. Réflexivité déclarative

Un système développé avec une architecture à réflexivité déclarative doit être capable de prendre en compte au moment de son exécution des contraintes et de poursuivre son exécution après avoir satisfait ces contraintes. Le système fournit les moyens de créer des méta-représentations et de récupérer le référent de

celles-ci. La cohérence des deux niveaux de représentation est assurée par un mécanisme qui maintient la relation entre ces deux niveaux. Le niveau méta-représentation est dédié à la description du système.

9.2.8. Vers une mise en place de processus réflexifs dans une analyse automatique

Le choix de la méta-représentation à adopter est extrêmement important car il implique la mise en place d'outils réflexifs spécifiques. Il faut rappeler que les langages à architecture réflexive permettent une plus grande modularité dans l'implémentation. Ces langages doivent être ouverts à toute extension qui permette de fournir de l'information sur l'activité d'un processus quelconque d'un système. Sans revenir sur les problèmes cruciaux qui existent pour la construction de langage à architecture réflexive, on peut quand même conclure provisoirement en disant que :

- La réflexivité est un outil utile pour toute activité de surveillance, de mise au point, de présentation raisonnée des résultats.d'un système.

- Les processus réflexifs doivent être limités et déterminés par des choix précis : les besoins de l'utilisateur ou du programmeur définiront ces processus. Leur implémentation doit d'ailleurs être introduite de manière modulaire et/ou ponctuelle.

- La réflexivité est envisageable dans les programmes orientés Objet (PàO et PàP) en raison de caractéristiques propres à ces langages : ce type de langage permet de développer la mise en place de méta-représentations, comme nous le verrons infra. Nous verrons plus loin comment mettre en place des processus réflexifs au sein d'un analyseur syntaxique développé avec le langage orienté objet CLOS puis avec le langage à prototypes SELF.

Dans le cas de l'analyse automatique, les processus réflexifs peuvent être développés pour mettre en place des activités de suivi ou de mise au point de l'analyse. Pour cela, il importe de construire différents niveaux de représentation des connaissances manipulées par l'analyseur. On peut par exemple construire plusieurs représentations de la grammaire utilisée par l'analyse à différents stade de celle-ci. On peut aussi envisager de construire des niveaux de représentations dédiés au pistage des transferts d'informations entre les différents acteurs de l'analyse. Les outils mis en place doivent présenter, de manière raisonnée, les différents états pertinents de l'analyse.

9.3. Qu’est-ce qu’on ne peut pas faire sans la réflexivité ?

"Le thème de la représentation des connaissances envahit aujourd'hui le champ des travaux en IA. Les ambiguïtés n'en sont pas pour autant levées : de quelles ***connaissances*** *s'agit-il? Et qu'appelle-t-on* ***représentation*** *? Savoir répondre à une liste de questions ne signifie pas forcément* ***connaître*** *un sujet. Toute vraie connaissance suppose l'usage à bon escient des informations qui sont les nôtres. Quant aux représentations qu'on pourra se faire de ces informations, elles dépendront de nombreux facteurs dont chacun, isolément, pourra les orienter différemment : la situation, le domaine concerné, la personne visée, les notions organisant la connaissance, le* ***langage*** *dont on dispose,.... Les classifications de savoir sur un domaine n'ont ainsi d'autre vertu que celle de nous aider à* ***ranger nos idées.*** *Comment ensuite les organiser, les orienter, tel est effectivement le problème des métaconnaissances nécessaires pour apprendre à savoir utiliser les connaissances et, par suite, à* ***délier*** *les moteurs d'inférence inhérents à de multiples systèmes experts"* (Vignaux 1991).

La tradition langagière considère souvent qu'il est nécessaire de "prendre du recul" pour évaluer le cours des choses. Il va de même pour la mise en place de dispositifs informatiques qui tentent de réaliser une représentation matérielle de l'analyse de certains faits de langue. L’utilisation qui est faite de la réflexivité dans les langages informatiques de haut niveau est de permettre de “parler des objets” manipulés (par les programmes mis en oeuvre dans ce langage) plutôt que de parler aux objets (via l’envoi de messages). Dans certaines étapes de développement d’un programme, il est souhaitable de pouvoir examiner la structure des objets et donc de disposer d’outils qui permettent l’activité réflexive sur le programme en cours de développement. Cette étape peut aussi se révéler souhaitable lors de l’exécution d’un programme en vue d’améliorer les processus mis en oeuvre par le programme[96].

[96] Il reste un choix pour le programmeur entre une écriture pragmatique, plutôt behavioriste et une écriture réflexive : ce choix peut être déterminé par des raisons nombreuses et variées (à expliciter), dans un but défini (Pourquoi faire?).

9.4. Connaissance et réflexivité en IA

9.4.1. De la méta-connaissance

Comme il est très difficile d'envisager d'inclure au sein d'un même module l'ensemble des connaissances que l'on peut avoir sur le langage naturel, il semble nécessaire d'envisager une expression déclarative des connaissances pour le TALN. On sépare ainsi les connaissances de leur mode d'emploi. Une telle approche implique donc que le système qui les utilise puisse avoir des connaissances pour manipuler ces connaissances (pour recréer le lien entre les connaissances et leur mode d'emploi, il s'agit ici de construire le mode d'emploi). Quand un système raisonne sur ce qu'il fait, il lui faut des connaissances sur les connaissances, appelées méta-connaissances. Un système doit pouvoir manipuler les connaissances dont il dispose afin d'expliquer ce qu'il a trouvé, d'apprendre en comprenant les raisons de ses succès et de ses échecs. Cela demande de travailler à deux niveaux : une niveau de base dans lequel on essaie de résoudre un problème et un niveau méta dans lequel on observe ce que l'on fait pendant que l'on résout le problème en question. Il est souvent bon de manipuler l'énoncé pour résoudre un problème. L'énoncé d'un problème est un sujet d'étude, étude qui se fait grâce à des métaconnaissances. Il en est de même pour notre propre utilisation du langage naturel. Nous ne naissons pas avec la connaissance d'une langue mais avec des métaconnaissances qui nous permettent d'apprendre une langue. Un nouveau problème fondamental surgit à ce stade de développement d'une approche déclarative de manipulation des connaissances. L'utilisation d'un très grande nombre de modules de connaissance implique un très grand nombre de méta-connaissances sur ces modules de connaissances pour pouvoir les manipuler. Il s'agit donc dans une telle approche de découvrir des ensembles de métaconnaissances pour traiter les connaissances déclaratives. Cette tâche n'est évidemment pas triviale. D'ailleurs (Pitrat 1993) souligne très bien la difficulté de travailler sur les métaconnaissances dans le cadre du langage naturel : dans ce domaine de savoirs, il est en effet difficile de manipuler et de maîtriser un grand nombre d'éléments de manière efficace. Le problème principal étant de trouver des métaconnaissances capables de produire de nouvelles connaissances.

9.4.2. Faciliter la production de méta-connaissances sur les entités linguistiques et l'analyse : Illustration avec OLMES (1993)

OLMES[97] (Habert 1991b) est un analyseur général pour le langage naturel. Il peut être utilisé pour des grammaires hors contexte comme pour des grammaires d'unification (Shieber 1986a). Il appartient à la famille des analyseurs à graphe actif. OLMES est écrit en CLOS - Common Lisp Object System (Keene 1989), (Steele 1990) - le système à objets de Common Lisp. Le recours à la programmation à objets, et particulièrement à l'héritage multiple, a permis la production de points de vue différents sur le parsage. A chaque occurrence de symbole de la grammaire correspond une classe. Une règle est représentée de la façon suivante : une classe est créée pour la partie gauche. Elle en mémorise l'étiquette. Dans une version de OLMES aujourd'hui modifiée, c'etait la super-classe de classes représentant les éléments de la partie droite, qui, elles, mémorisent l'étiquette de chacun de ces éléments. La figure qui suit donne le sous-graphe de classes traduisant `GN -> Det N Adj. GN7`[98] est la super-classe des classes `Det8`, `N9` et `Adj10`.

---

97 Il est utilisé au Laboratoire d'Etude des Textes Politiques (INaLF - CNRS) pour catégoriser et structurer des corpus et sert de base à un projet de comparaison d'outils d'analyse de texte, projet commun à l'ELI (ENS de Fontenay St Cloud) et au Centre d'Analyse de Textes par Ordinateur de l'UQAM (Université du Québec à Montréal).

98 Le recours à l'héritage peut surprendre : un élément droit de règle est dans une relation Partie-De avec la partie gauche de la règle, et non dans une relation Est-un qui est la relation que traduit habituellement l'héritage. Ce choix avait pour but de faciliter la représentation d'une règle comme un tout structuré, en utilisant des traits standards (réflexifs d'ailleurs) de CLOS. Chaque classe connait en effet ses super-classes et ses sous-classes directes. Concrètement cela signifie que chaque partie gauche de règle connait les éléments de partie droite et inversement. A partir de tout élément d'une règle, il est ainsi possible à

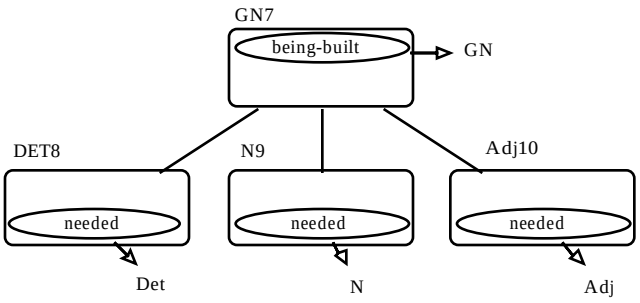


Figure 9.2 : Le sous-graphe de classes représentant `GN -> DET N ADJ`.

Une analyse montante de gauche à droite se déroule de la manière suivante. Chaque fois qu'est rencontré un mot ayant la même étiquette que l'une des classes correspondant à un terminal premier élément de partie droite de règle, une instance de cette classe est créée. Par exemple, si un Dét(erminant) est rencontré, une instance de `Det8` est créée, qui cherche le Dét(erminant) en cause (a) et le trouve. Cette instance crée alors une instance de la classe suivante dans le réseau, `N9`, qui cherche un N(om) et domine le Dét trouvé (b). Si l'instance de `N9` trouve un N(om), elle crée à son tour une instance de `Adj10`, cherchant un Adj(ectif) et dominant le Dét et le N trouvés aux pas précédents (c). Si cette dernière instance trouve un Adjectif, elle crée une instance de `GN7`, qui domine les mots ainsi reconnus (d), et qui déclenche de nouvelles règles, par le biais de la création de deux instances de `GN2` et `GN15`. Et la suite de l'analyse se poursuit de la même façon.

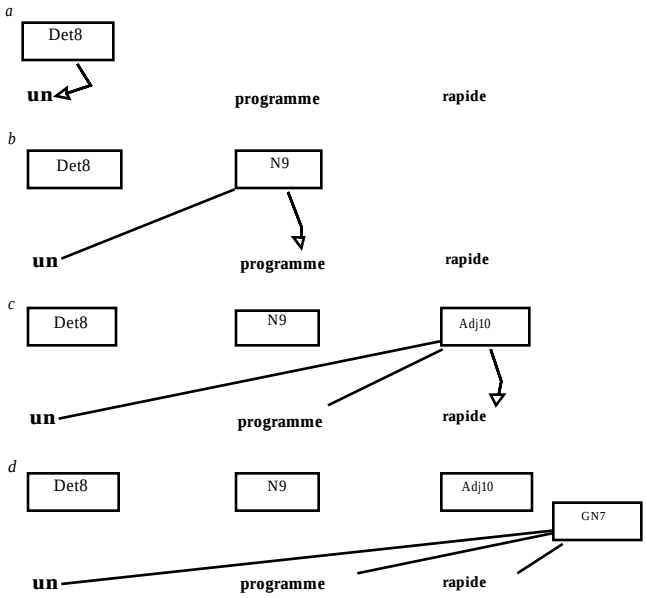


Figure 9.3 : Analyse par les instances d'un réseau de classes.

L'important est ici que l'entité "règle" est représentée en tant que telle, et peut donc être manipulée. La distinction est nette entre les choix que peut faire l'utilisateur en termes d'utilisation de la grammaire : direction d'exploration (gauche-droite, droite-gauche, bi-directionnelle) et stratégie (ascendante, descendante), choix qui se traduisent par des chaînages différents des classes représentant les éléments de la grammaire, et la structuration fondamentale de la grammaire par règles. Par ailleurs, une entité "méta-grammaire" est également créée, qui donne accès aux entités "règles" par le biais de leurs parties gauches.

---

l'exécution de reconstituer la règle entière. Une représentation qui n'introduise pas une telle ambiguïté (Habert & Fleury 1993) a été mise en place depuis et sera présentée infra.

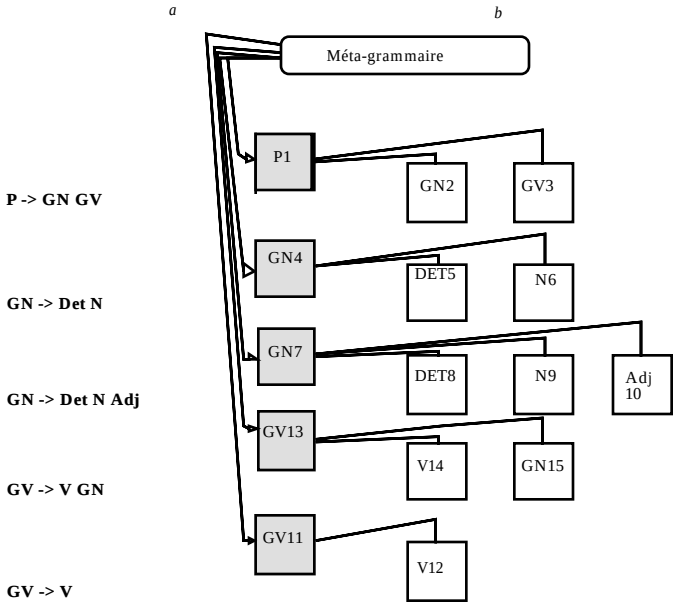


Figure 9.4 : Méta-grammaire et règles.

En OLMES, il y a donc une bijection entre les entités présentes à l'exécution et les éléments des règles définies par le linguiste, comme dans (Yonezawa & Ohsawa 1990). La transformation opérée dans (Yonezawa & Ohsawa 1990) est cependant à sens unique : elle manifeste le lien entre le comportement des unités de traitement et la grammaire de départ, elle ne permet pas pour autant une rétro-action sur la grammaire de départ à partir de l'observation de ce comportement. Le fait de constituer des méta-représentations des règles et de la grammaire en tant que telles rend possible au contraire en OLMES une telle action en retour. Toutefois, avant d'explorer les potentialités d'une telle architecture, il convient de réinterroger les notions de méta-connaissance et de réflexivité selon les cadres qui peuvent y faire appel.

9.4.3. Intelligence artificielle, formalismes d'unification, langages de programmation : des réflexivités distinctes ?

Le choix d'une architecture donnée pour un analyseur n'est pas purement technique. Il incarne également une certaine "idée" de la langue et de son traitement, tant naturel qu'automatique, et, partant, fixe le rôle que peuvent y avoir méta-connaissances et réflexivité. A l'inverse, il est nécessaire d'examiner le détail des architectures pour faire sortir les notions de réflexivité et de méta-connaissances de leur abstraction, et pour montrer les conditions techniques précises de leur emploi, au delà des espoirs ou des agacements que la perspective réflexive peut susciter. Ainsi se justifie la minutie du développement qui suit : limiter autant que faire se peut le risque de prendre ses désirs pour la réalité. Certains parseurs entendent fournir des représentations des structures de données présentes à l'exécution aussi "proches" que souhaitable des entités lexicales et syntaxiques définies par les utilisateurs linguistes. L'extension de telles architectures permet de produire des méta-connaissances sur les données linguistiques et éventuellement sur leur utilisation. Peut-être pourrait-on en effet contraster utilement les approches issues de l'intelligence artificielle, celles découlant des formalismes d'orientation linguistique et enfin celles autour des langages de programmation les plus évolués. En effet, si la recherche sur la réflexivité constitue une dimension de l'intelligence artificielle depuis une quinzaine d'années, si depuis près de dix ans, elle représente un courant dans la conception des langages de programmation, par contre, elle semble marginale dans l'évolution récente de la linguistique informatique. C'est ce constat que nous voulons développer et questionner, en restant bien conscients des passerelles et des interférences entre des problématiques que nous présentons comme distinctes, voire divergentes.

Domaines de la réflexivité.

Il n'est pas sûr que tous les types d'analyse aient à gagner à recourir à la réflexivité. Quel pourrait être son rôle dans un analyseur LR, par exemple ? Dans les analyseurs pour langages de programmation, une fois la grammaire définie, on n'a pas de raison, a priori, d'y revenir. On peut ne s'intéresser qu'au résultat : le compilateur produit pour le langage en cause. Pour le langage naturel, c'est rarement le cas. Les connaissances utilisées par l'analyseur sont sans cesse à affiner. La perspective réflexive paraît donc plus fructueuse pour les domaines où les connaissances résistent à la formalisation.

Constituer et articuler des points de vue.

Le système multi-agents CARAMEL (Sabah 1990, 1993) fournit un bon exemple. Il est destiné à réaliser diverses tâches liées au traitement automatique du français. Il part de l'hypothèse que le type de connaissances à employer pour traiter une phrase ne peut être déterminé a priori, ce qui exclut un enchaînement fixe de couches (morphologique, syntaxe, sémantique, pragmatique, par exemple), et également une intégration fixée à l'avance des divers types de connaissances. Dans le but de réaliser *"une architecture où chaque module [puisse] avoir accès aux connaissances et aux résultats d'autres modules selon ses besoins"* (Sabah 1993), CARAMEL comprend alors un ensemble d'agents, simples ou complexes, qui sont autant de systèmes-experts coopérant à la construction du sens de l'énoncé. Les agents complexes peuvent déclencher d'autres agents. Ils constituent autant de *"méta-systèmes contrôlant un ensemble d'agents pour l'accomplissement de leur but"* (ibid.). Un superviseur général contrôle l'analyse. La réflexité est ici fondamentale : aussi bien au niveau du superviseur général qu'à celui de n'importe quel agent complexe, il s'agit d'évaluer les représentations déjà produites, les capacités des agents déclenchables par l'agent complexe concerné et partant, les meilleurs moyens d'obtenir le but fixé, en déclenchant tel ou tel agent, en gérant le va-et-vient d'informations entre les agents contrôlés, et en adaptant la stratégie selon les informations reçues des agents déclenchés. Par ailleurs, *"dans sa mémoire de travail, le superviseur construit /./ une représentation globale du déroulement dynamique du raisonnement (une indication des agents qui ont été nécessaires pour mener à bien une tâche donnée), ce qui lui permet de donner par la suite des explications sur la stratégie qui a été employée"*. CARAMEL permet d'articuler différents savoirs linguistiques et non-linguistiques.

9.4.3.1. IA : la réflexivité pour la modularisation et l'articulation des savoirs

L'intelligence artificielle, plus directement confrontée aux relations entre les différents composants du savoir linguistique, et entre celui-ci et le savoir "sur le monde", a cherché à articuler ces différents types de savoirs, en utilisant par exemple l'activation de modules spécialisés, la mise en oeuvre ou le retrait de tel ou tel paquet de règles, et à essayer, depuis longtemps, de recourir aux méta-connaissances et à la réflexivité. Le développement des systèmes-experts (Laurière 1987) en particulier a très tôt poussé à représenter les connaissances que les experts ont sur leurs propres connaissances, en les formulant sous la forme de méta-règles guidant le moteur d'inférence. S'est avérée aussi fondamentale la capacité pour un système-expert donné de pouvoir expliquer comment il arrive à une conclusion donnée.

9.4.3.2. Les formalismes d'unification : peu de réflexivité

Au rebours de cette représentation modulaire et réflexive des connaissances et des traitements, les formalismes dits d'unification tendent à intégrer au long de l'analyse les différentes facettes de l'analyse linguistique. Ainsi dans la grammaire syntagmatique guidée par les têtes, HPSG (Pollard & Sag 1987), un "signe", entrée lexicale ou syntagme, comprend volets phonologique, syntaxique et sémantique. En tant que tels, ces formalismes ne favorisent guère, voire ne permettent pas, la séparation en différents modules explicites, destinés à interagir. Tout se passe comme si l'accord sur la manière de représenter les connaissances linguistiques et de s'en servir que constituent les formalismes d'unification s'était accompagné d'un certain enfermement sur syntaxe et sémantique pris dans un sens relativement restreint, et

parallèlement, d'une mise entre parenthèses de l'articulation des savoirs linguistiques entre eux et avec d'autres types de savoirs. La possibilité de fournir sous une forme relativement déclarative un certain nombre de contraintes sur la combinaison des constituants entre eux a relégué paradoxalement au second plan la réflexion sur les stratégies d'utilisation des connaissances linguistiques. Si bien qu'au total, l'intégration d'une approche réflexive aux analyseurs basés sur des formalismes d'unification paraît difficile, et en danger de rester "accessoire".

Les formalismes d'unification comprennent un certain nombre de principes, qui contrôlent la bonne formation des différents constituants. Au sein d'un formalisme donné, certains de ces principes peuvent être pensés comme universels, tandis que d'autres sont vus comme propres à une langue donnée. On trouve ainsi dans la grammaire lexicale-fonctionnelle, LFG (Bresnan 1982), les principes de complétude et de cohérence (toutes les "places" d'un prédicat verbal doivent être occupées, et il ne doit pas y avoir d'"argument" surnuméraire), dans la grammaire syntagmatique généralisée, GPSG (Gazdar & al. 1985), des principes d'instanciation de traits d'accord, de pied et de tête, ou les règles de précédence linéaire. Au delà de la règle générale que les principes doivent tous être simultanément satisfaits, la réalisation concrète de leur interaction, par exemple leur ordonnancement peut se révéler fort complexe, comme le montre S.Shieber (1986b). Pour tout dire, on voit réapparaître les problèmes de contrôle que la déclarativité des règles de réécriture annotées avait poussés dans l'ombre. Le Principle-Ordering Parser de S.Fong et R.Berwick (1991) a précisément pour objectif de permettre de tester des ordonnancements possibles des principes de la théorie du Gouvernement et du Liage en tenant compte de leurs dépendances logiques : "*Le PO-Parser a été délibérément construit d'une manière extrêmement modulaire pour permettre un maximum de flexibilité dans l'exploration des ordonnancements alternatifs de principes*". Les principes d'une théorie gagneraient sans doute à changer de statut et à constituer un méta-savoir explicite et utilisable en tant que tel au cours de l'analyse. Un tel méta-savoir pourrait veiller par exemple à l'application des différents principes d'unicité de la théorie du Gouvernement et du Liage (un seul noyau, une seule catégorie X par domaine, un seul support de fonction dans S etc.). Face aux deux réalisations possibles du GV avec objet direct en français, clitique préposé ou GN postposé au verbe, le méta-savoir incluant ce principe d'unicité éliminerait une configuration des hypothèses possibles si l'autre s'est réalisée.

9.4.3.3. Réflexions sur la réflexivité

Réflexivité approchée / implicite / explicite

Bascule d'un mode simple à un mode plus élaboré, envisageant davantage d'hypothèses, pondération des hypothèses, traitement des exceptions en tant que telles, adaptation aux énoncés traités, ces processus trouvent dans l'immédiat des approximations dans des analyseurs actuels : parsage auto-adaptatif (Lehman 1992), parsage probabiliste (Liberman 1991), dispositifs d'acquisition lexicale (Zernik 1991) ... Sans contester les apports de ce courant empiriste, numérique, non-symbolique, on peut faire l'hypothèse que l'intégration d'une approche réflexive dans la conception d'analyseurs peut permettre également de mettre en oeuvre ces dispositifs et ces fonctionnements, en leur donnant un statut plus clair. A ce titre, la comparaison entre CARAMEL et TALISMAN (Stefanini & al. 1992) est éclairante. TALISMAN repose aussi sur une architecture multi-agents, pour les mêmes raisons que CARAMEL : *"compose[r] différentes phases d'analyse"*, *"ordonner convenablement les divers sous-ensembles de règles et /./ les activer au bon moment"*. Les deux systèmes s'inscrivent donc dans le cadre de l'Intelligence Artificielle Distribuée (Bond & Gasser 1988), où l'"*intelligence est répartie entre des agents. Ceux-ci sont des entités autonomes capables d'agir rationnellement sur elles-mêmes et sur leur environnement en fonction de leurs observations, de l'état de leurs connaissances et de leurs interactions"* (Stefanini & al. 1992). Alors que CARAMEL repose sur des tableaux noirs pour le partage d'informations, TALISMAN s'appuie sur la communication par messages entre agents. Pour éviter les problèmes de synchronisation, de concurrence, de partage de ressources que fait naître une trop grande autonomie des agents, TALISMAN se voit contraint d'ajouter des "lois". *"Les lois régulent et modifient les échanges de messages entre les agents. Elles sont explicites et modifiables "* (ibid.). Ces lois jouent en fait le rôle des superviseurs de CARAMEL : elles *"activent un agent spécialisé, en fonction d'indices prélevés dans le texte et de caractéristiques d'un problème d'analyse particulier"* (ibid.), elles évitent certains conflits entre agents et en gèrent d'autres. Il s'agit bien de méta-savoirs. Leur reconnaître ce statut pose de manière plus nette la question de la constitution de la méta-connaissance au

cours des traitements, c'est-à-dire en définitive le problème d'architectures réflexives qui aident à dégager des modes de coopération optimaux entre les agents, en notant par exemple les conflits rencontrés, les duplications d'informations etc. Thématiser la réflexivité en tant que telle paraît en définitive plus fructueux que d'y recourir implicitement ou d'utiliser des approximations.

Réflexivité naturelle et réflexivité artificielle

Nous sommes restés précédemment (relativement) agnostiques sur le rapport entre la réflexivité "naturelle" dont semble faire preuve le locuteur lors de l'analyse d'un énoncé, et que marque la constante activité métalinguistique, et la réflexivité qui nous semble devoir intervenir dans les analyseurs. A cela, au moins trois raisons. Les arguments psycho-linguistiques sont tout d'abord fréquemment suspects : la diversité tant des hypothèses que de leur utilisation rend prudent. En outre, mimer le comportement humain n'est pas forcément la meilleure solution. Enfin, ce sont plutôt les raisons internes à la conception d'analyseurs de recourir à la réflexivité qui nous paraissent pertinentes. C'est a priori dans le cadre d'une vision modulaire de l'analyse automatique que les apports de la réflexivité sont les plus évidents. Pourtant, le courant actuel des grammaires d'unification peut également en bénéficier. Pour reprendre les termes de P. Maes (1987a), *"une implémentation réflexive ne contribue pas directement à la résolution des problèmes du domaine représenté par le système. Elle contribue plutôt à l'organisation interne du système ou à son interface avec le monde extérieur. Son but est de garantir un fonctionnement efficace et harmonieux des calculs liés au problème traité"*. Les divers cadres de pensée des travaux sur la réflexivité offrent donc des perspectives stimulantes. Pour reprendre P. Maes (1987a) : *"le calcul réflexif est tellement constitutif des systèmes en vraie grandeur qu'il doit être développé comme un outil fondamental dans les langages de programmation"*. Pourquoi n'en irait-il pas de même pour l'analyse automatique du langage naturel ? En gardant en mémoire l'avertissement de J. Pitrat (1992) *"Travailler sur l'utilisation des métaconnaissances est un détour qui aura des conséquences pour les systèmes manipulant les langues naturelles, mais nous ne devons pas en attendre des résultats spectaculaires à court terme"*.

9.5. Tradition de la programmation réflexive

9.5.1. Réflexivité et Méta-niveau dans les langages orientés Objet

On étudie ici certains langages dotés d'une capacité d'"introspection" permettant à un système conçu dans de tels langages de s'adapter et de s'optimiser. Est visée une adaptation non seulement statique mais aussi dynamique (au cours de l'évolution des calculs). On entend en outre contrôler le comportement d'un système non pas à un très bas niveau d'expression mais à un niveau au moins équivalent à celui du système lui-même. Un méta-niveau représente et décrit alors l'exécution du niveau de base, c'est-à-dire le déroulement du programme. Il est possible que le méta-niveau ait lui-même un méta-niveau et ainsi de suite, ce qui construit éventuellement ce qu'on appelle une tour réflexive (d'autant qu'il y a identité de langage entre niveau de base et méta-niveau).

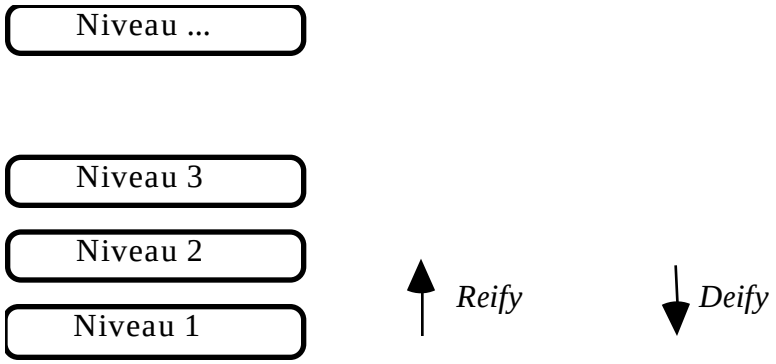


Figure 9.5 : Une tour réflexive.

Le fait qu'un langage soit explicitement doté d'une méta-représentation de lui-même permet par exemple d'obtenir un langage qui puisse évoluer de manière contrôlée, et servir ainsi de terrain d'expérimentation tout en gardant les avantages de la standardisation. C'est le cas par exemple en CLOS où les classes sont elles-mêmes des instances de métaclasses qui déterminent leur comportement et leur structure. Créer de nouvelles métaclasses permet alors d'introduire de nouveaux comportements (autres schémas de calcul de l'héritage multiple, classes connaissant leurs instances ...), sans modifier pour autant le fonctionnement normal du système (Cointe & Graubé 1988, Graubé 1989, Kickzales & al. 1991). La plupart des langages orientés Objets mettent en avant les notions conjointes de Classes et Métaclasses. Cette approche s'est révélée une solution adéquate pour mettre en place des mécanismes de réflexivité (cf infra avec CLOS), que l'on peut qualifier de réflexivité structurale. Cela signifie que toutes les données présentes dns le système sont réifiées (données et programmes). Le développement des langages à prototypes tels Self a mis en avant de manière plus précise la notion de reflexivité comportementale qui introduit la notion de réification de l'exécution d'un objet : cet aspect n'a pas été résolu dans le premier type de langage et est en cours de développement avec le second (Self (the Self Group 1987-1996), Moostrap (Cointe & Mulet 1993)). Il importe de préciser tout de suite que le développement de réflexivité et d'architectures à méta-niveaux ne signifie qu'il soit nécessaire de développer ou utiliser des langages qui soient eux-mêmes complètement réflexifs. Il s'agit plus précisément que ces langages permettent de spécifier ou d'engendrer des activités réflexives pour répondre à des sollicitations précises et à des besoins particuliers prédéfinis.

9.5.2. Clos et la Réflexivité

CLOS encourage un style de programmation qui fasse de chaque classe une sorte de boite noire à laquelle les utilisateurs n'ont accès que par l'interface fourni par les fonctions génériques sans qu'ils puissent connaître le détail de l'implémentation. CLOS ne force cependant pas à séparer nettement les comportements "privés"

de la classe, accessibles par ses instances et les instances de ses sous-classes, et les savoir-faire "publics", utilisables par le tout-venant, sauf à recourir au Meta-Object-Protocol (MOP par la suite).

9.5.2.1. Méta-Object

9.5.2.2. Le MOP de CLOS

CLOS est défini en termes d'objets : aux classes, aux définitions de champs, aux méthodes, aux fonctions génériques correspondent des méta-objets qui réalisent le méta-protocole[99]. Ce méta-protocole permet à l'utilisateur de se définir ses propres méta-classes (par exemple des classes qui connaissent toutes leurs instances) et d'écrire des extensions portables de CLOS. L'apport CLOS est que ces sur-langages gardent comme sous-bassement les potentialités de CLOS, tout en les redéfinissant. L'utilisation du MOP pour construire une couche supplémentaire de la représentation des informations dans l'analyse syntaxique faite par Olmes a permis d'enrichir considérablement la connaissance de l'utilisateur sur ce qui est en jeu dans l'analyse syntaxique (-> annexe chapitre 9 : trace d'une analyse). Le Méta-protocole de CLOS est construit de telle sorte que l'utilisateur de processus réflexifs peut spécifier les éléments sur lesquels il a besoin de définir une utilsation de tels mécanismes. *"For this class, this slot, perform slot access in this different way"* plutôt que *"To all accesses, this different way"* (Kiczales 1990). La structure méta définie en CLOS s'inscrit dans le même modèle que celui dans lequel CLOS est construit. Les éléments de base de CLOS (classes, méthodes, fonctions génériques) sont définis comme des objets : i.e, chaque classe, chaque fonction générique, chaque méthode est représenté sous la forme d'un métaobjet. Le terme "Classe métaobjet" réfère aux classes dont les instances sont des métaobjets. Chaque classe est représentée par une Classe métaobjet qui enregistre ses superclasses, les slots définis dans la classe et les slots hérités. Et tous les mécanismes qui définissent les comportements des objets (héritage de slot, structure des instances, accès au slot, initialisation d'instance...) sont codés au niveau du métaprotocole par le biais de fonctions génériques qui agissent sur les Classes métaobjet.

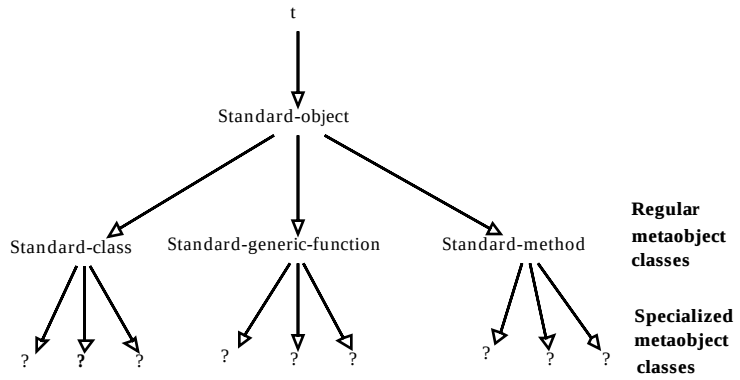


Figure 9.6 : Classes MetaObject en CLOS.

Il est possible de d'envisager de mettre en place des outils de "surveillance" de toutes les informations présentes au cours de l'activité d'un processus en cours. Par exemple si on envisage de ne manipuler au cours de l'analyse que des objets, il est possible d'établir un historique des accès aux champs des classes manipulées. Si les traits utilisés dans la grammaire sont associés à des objets, on peut ainsi tracer la circulation de ces traits au cours de l'analyse. Le MOP fournit cette possibilité en utilisant ce que (Bobrow & al. 1990) nomment le `Monitoring-Slot-Access`. Pour ce faire il faut définir un métaobjet. Les classes

---

99 *"Les protocoles méta-objets sont des interfaces qui donnent aux utilisateurs la possibilité de modifier de façon incrémentale le comportement du langage et son implémentation, aussi bien que la possibilité d'écrire des programmes dans ce même langage... L'implémentation de ce langage est elle-même structurée comme une programme orienté objet... Les éléments de base du langage de programmation, classes, méthodes et fonctions génériques sont accessibles comme le sont les objets"* (Bobrow & al 90).

dont on souhaite faire un historique de l'accès à leurs champs héritent de cette classe. L'utilisation habituelle du protocole d'accès permet ainsi de mettre en place ce travail d'enregistrement des accès aux champs. La généricité étant ce qu'il convient d'utiliser sous CLOS, on peut aussi envisager de construire un compteur qui enregistre le nombre de fois où une fonction générique et les méthodes associées sont appelées. Pour ce faire on peut en effet définir au niveau du MOP des fonctions génériques et des méthodes qui réalisent cette incrémentation. On peut ainsi établir complètement l'historique des interactions entre les objets. Ces informations ajoutées aux informations fournies par le `Monitoring-Slot-Access` pourraient donc produire une importante trace de l'activité d'un analyseur syntaxique. Il serait alors possible d'établir un relevé de toutes les opérations exécutées et d'identifier par exemple les performances de ces opérations.

9.5.2.3. Les limites de `change-class`

Si on change la définition d'une classe donnée, cette opération peut conduire à perdre des informations mémorisées par des instances déjà créées (Habert 1996). A moins de définir des prologues ou des épilogues qui accompagnent les redéfinitions de classe (Ibid.). Ces accompagnements posent malgré tout des problèmes d'une extrême finesse, et impliquent une attention particulière pour mettre à jour les classes existantes. Surtout si on envisage de modifier dynamiquement les définitions des structures définies dans un processus de classification automatique ou de gérération dynamique des savoirs associées. De plus changer la définition des classes existantes nécessite un méta-regard sur le processus de représentation en cours pour évaluer les interactions ou concurrences à l'oeuvre dans les savoirs à représenter et dans les objets déjà exitants. Une maîtrise totale des savoirs en jeu implique un méta-contrôle sur les objets construits ou à construire (et donc les méta-outils associés) qu'il n'est évidemment pas trivial de définir. S'il est possible de changer la définition d'une classe et de propager ce changement aux instances d'icelle (en respectant les mises en garde précédentes), il est aussi possible de changer la classe d'une instance donnée (Ibid.). La fonction générique `change-class` est l'outil qui réalise cette opération. Cette fonction générique a malgré tout des limites. Elle opère un transfert d'information entre les attributs communs aux deux classes en jeu (les valeurs des attributs de la classe d'origine passent dans les attributs de la nouvelle classe), et initialise les attributs spécifiques à la nouvelle classe avec les valeurs définies par défaut dans celle-ci. Par contre les attributs spécifiques à la classe d'origine sont oubliés. Cette opération peut donc conduire à une perte d'information. Et il n'est pas sûr que cette perte d'informations soit justifiée / voulue dans tous les cas. On peut ainsi vouloir, pour une instance donnée, changer sa classe pour y préciser certaines caractéristiques que l'on ne possède pas encore (du type : `moulin` devient `+humain` dans `moulin à paroles`), tout en conservant certaines informations déjà présentes.

9.5.3. Self et la réflexivité

Comme on l'a vu dans la présentation de Self faite précedemment, une des caractéristiques de Self est que les objets manipulés sont complètement déterminés par les comportements qui leur sont associés. Cette priorité de représentation permet une extension et une flexibilité dans le développement de nouvelles applications. Self dispose aussi d'une facette fondamentale : on peut souhaiter pouvoir examiner ou manipuler les objets au cours d'un processus de calcul. On utilise les objets comme des données et non plus seulement comme des entités associées à la représentation d'un concept. Ce type de réflexivité permet de "parler" à propos d'un objet plutôt que de lui parler (via l'envoi de messages en Self). Cette action est faite en Self avec des objets appelés miroirs.

9.5.3.1. Réflexion de structure en Self

La notion de reflect

Un objet Self est essentiellement un dictionnaire (association attributs et valeurs). Les attributs peuvent être constants ou modifiables, parents ou non, privés ou publics. Ces informations sont conservées dans des entités appelées `maps`. Une réflexion de structure est introduite en Self par des entités appelées `miroirs`.

Ces entités rendent visibles au niveau du langage les informations contenues dans la représentation interne des objets. Un `miroir` sur un objet `x` permet essentiellement de faire des requêtes sur la représentation interne de cet objet. Dans la réalité, les `miroirs` sont des objets qui ne contiennent aucune information en eux-mêmes. Un `miroir` est simplement un receveur capable de répondre à un ensemble de primitives du système qui donnent accès en lecture aux informations sur l'objet.

9.5.3.2. Réflexion de comportement

Dans les langages à taxonomie de classes, les classes et les méta-classes définissent la réflexion de structure. Dans les langages à prototypes de type Self, la notion de classe n'existe pas pour définir un modèle d'auto-représentation des objets. (Cointe & al. 91) propose une implémentation de la réflexion de comportement en Self. Ce protocole est basé sur l'utilisation de méta-objets pour décrire le comportement des objets lors de la réception des messages. A chaque objet Self est associé un méta-objet dont la fonction est d'expliciter la manière dont l'objet se comportera à la réception d'un message. Le problème de la régression infinie par le lien "méta" est réglé par l'introduction un méta-objet de base, `BasicMetaObject`, qui s'admet lui-même comme son propre méta-objet et qui définit les comportements de base pour la recherche des attributs et leur application. Le lien "méta" peut être explicite (via un attribut `metaobject`) ou implicite (dans ce cas l'objet a pour méta-objet `BasicMetaObject`).

A l'envoi d'un message, l'interprète effectue les opération suivantes :

- 1. Il envoie au méta-objet du receveur le message

`` `lookup: From: SendingMethodHolder:` ``

effectuant la recherche de l' attribut et retournant un descripteur de l'attribut représentant l'attribut trouvé.

- 2. Il envoie ensuite à ce descripteur d'attribut le message

`` `slotApplyTo: WithArgs:` ``

qui déclenche l'exécution du contenu de l'attribut trouvé.

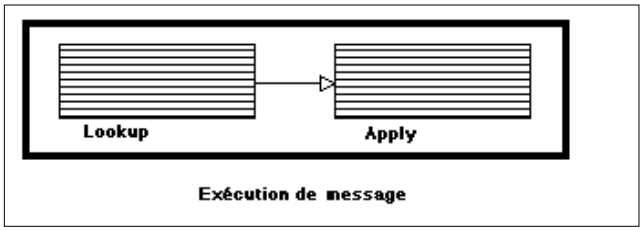


Figure 9.7 : Exécution de message en Self.

Ce protocole réifie l'exécution des messages en paramétrant la recherche des attributs et leur application. Les messages

`` `lookup: From:SendingMethodHolder:` ``
et
`` `slotApplyTo: WithArgs:` ``

fournissent les points d'entrée nécessaires pour introduire les modifications voulues par l'utilisateur. Le protocole de réflexion de comportement est donc basé sur l'utilisation de méta-objets pour décrire le comportement des objets lors de la réception de message. L'exécution d'un message est décomposé en deux phases : une phase de recherche de l'attribut correspondant au sélecteur (`lookup`) et une phase

d'application de l'attribut trouvé dans le contexte de l'objet receveur. Cette extension pose des problèmes fondamentaux sur le développement de protocole de réflexion de comportement : la notion de lien méta est à étudier, une réification totale des attributs semble difficile à mettre en place en Self. La réflexivité comportementale pose deux problèmes fondamentaux. D'une part, ce type de mécanisme doit décrire le comportement d'un objet agissant sur d'autres objets. D'autre part, on doit pouvoir disposer de moyens pour intervenir sur l'exécution d'un processus affectant les objets du système pour éventuellement modifier le cours des évènements.

9.5.3.3. Moostrap : un noyau réflexif pour un langage à prototypes

(Cointe & Mulet 1993) propose, avec le langage à prototypes réflexif MOOSTRAP, une nouvelle approche pour une étude de la réflexion de comportement dans un langage à prototype. Il ne s'agit plus de construire une extension d'un langage existant mais de construire directement un langage dans lequel le protocole de réflexion de comportement s'inscrit comme un choix fondamental dans le développement d'un langage à prototypes.

9.6. Outils disponibles pour le TALN

9.6.1. Un lien Méta

Méta-classe : homogénéité du paradigme, toute entité est un objet, ie une instance d'une classe.

Définir un lien méta est assurément le plus sûr moyen pour pouvoir disposer de mécanismes qui coopèrent dans un système de représentation des connaisssances réflexif. La question fondamentale que pose ce type de système est le type de distinction qui doit exister entre le niveau méta et le niveau de base. On peut définir un système à architecture méta :

- 1°) On met en place un niveau méta de référence. Il s'agit de définir les mécanismes d'inteaction de référence entre le niveau méta et le niveau de base.

- 2°) On dispose de plusieurs méta-niveaux indépendants (par rapport au lien méta qu'ils entretiennent avec le niveau de base) et il s'agit dans ce cas de définir les interactions entre ces différents méta-niveaux.

Dans le premier cas de figure, on définit un certain méta-niveau et on s'y tient, et parallèlement on met en place les processus réflexifs sur le niveau de base. Dans le second, chaque type de problème étudié dans le niveau de base définit son propre méta-niveau et les processus réflexifs évaluent les interactions entre ces méta-niveaux.

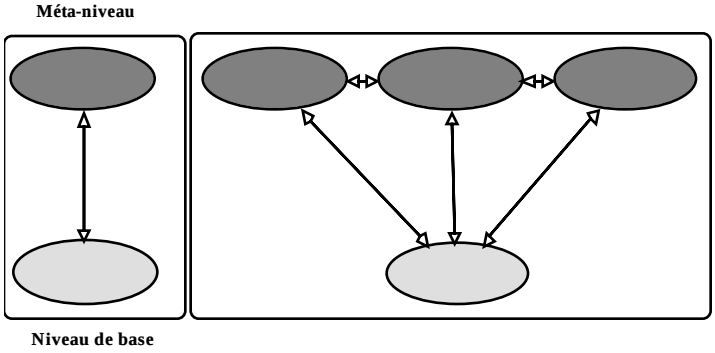


Figure 9.8 : Un lien Méta.

9.6.2. Analyse Automatique et Méta-niveau

OLMES offre des points d'entrée distincts pour le linguiste qui affine la grammaire et pour le programmeur qui met au point l'analyseur. Les entités actives lors de l'analyse combinent en effet de nombreuses facettes structurelles ou comportementales : structure de base d'une unité de traitement (type de constituant cherché...), structure et comportement nécessaires aux relations de dominance, mémorisation d'hypothèses syntaxiques et capacité à les déclencher si nécessaire, direction d'exploration (gauche-droite ou inversement), étape dans la vérification d'une règle ou construction du constituant correspondant à la vérification complète de la règle, chaînage avec les étapes précédente et suivante... L'héritage multiple sert alors à organiser le comportement et les structures utilisés par les éléments de la grammaire. Chaque classe créée pour une occurrence de symbole dans la grammaire hérite en effet de différentes classes abstraites qui représentent les différentes facettes du comportement et de la structure de cet élément de la grammaire pendant l'analyse. L'héritage a pour rôle ici de structurer les méta-connaissances. La tâche du compilateur est alors, au vu de la grammaire et des choix de l'utilisateur, de donner les ancêtres adéquats à chaque classe correspondant à un élément de la grammaire, de manière à lui fournir les structures et les

comportements pertinents. Ce recours intensif à l'héritage multiple fait coexister des points de vue différents et indépendants sur une règle de grammaire représentée par un graphe de classes et sur son utilisation possible. La multiplicité, organisée, des points de vue possibles sur le réseau de classes permet de créer différents protocoles d'accès nettement distincts. L'utilisateur en particulier doit seulement connaître l'existence du réseau de classes et d'une bijection entre les occurrences de symboles dans la grammaire et les noeuds de ce réseau. Il lui faut connaître en outre un sous-ensemble des facettes que peut comporter un noeud de ce réseau, les facettes les plus proches de sa propre représentation de ce qu'est une grammaire. Enfin, il lui faut connaître la partie correspondante du protocole qu'emploient au moment de l'analyse les entités produites par la compilation, pour pouvoir y greffer traçage et pas à pas, par le biais de prologues et d'épilogues. En CLOS, il est possible en effet d'ajouter des prologues et/ou des épilogues aux fonctions disponibles (cf supra) : ces ajouts sont utilisés pour produire des effets de bord, et, sauf exception, ne changent pas pour le reste le fonctionnement ou la valeur retournée par les fonctions en cause (Muller & al. 1989). Grâce à ce savoir à dessein restreint, et sans devoir pour autant connaître l'ensemble des choix faits pour l'implémentation, l'utilisateur dispose de "vues" orthogonales pour pister le comportement de la grammaire au cours de l'analyse. Dans ce contexte, "orthogonal" signifie qu'il est possible d'observer une facette particulière d'un élément du réseau sans pour autant ouvrir la "boîte noire" et sans être confronté à des détails implémentatoires non pertinents. L'accès donné à l'implémentation se situe à un niveau d'abstraction suffisamment élevé (Kickzalez & al 1991). Il s'oppose aux deux approches qui sont disponibles habituellement lorsque, contrairement à toute attente, l'analyseur ne produit pas de dérivations pour la phrase proposée ou produit des dérivations erronées. La première revient à ouvrir la "boite noire" que constitue l'analyseur : on a recours alors aux outils de traçage et de mise au point du langage d'implémentation. La seconde approche consiste à enrichir la boite à outils de fonctionnalités de traçage spécifique (Boguraev & al. 1988), (Bouchard & al. 1992a). Ces deux approches présentent pourtant de sérieux inconvénients. La première force le linguiste utilisateur à manipuler des structures de données sans rapport évident avec les entités définies dans la grammaire et le lexique, et qui de toute façon sortent de son domaine de compétence propre. La seconde l'oblige à maîtriser une nouvelle couche logicielle qui s'ajoute à l'analyseur proprement dit.

Donnons quelques exemples d'utilisation de ce type de pistage :

- Lorsque l'utilisateur veut connaître l'utilisation effective des règles de la grammaire sur un corpus donné, le compilateur ajoute un prologue déclenché à la création des instances de la classe correspondant au premier élément d'une règle, et un autre à la création de celle correspondant à la construction du constituant (dans l'exemple choisi et présenté infra, à chaque création d'une instance de DET5 et de GN7). Ces prologues incrémentent des compteurs, ce qui permet de savoir le nombre de fois où chaque règle a été déclenchée et le nombre de fois où elle a réussi. Les résultats fournissent des aperçus instructifs sur l'organisation de la grammaire et permettent de l'améliorer, en ordonnançant les règles par ordre de probabilité. Tracer une règle donnée se traduit par l'ajout de prologues et d'épilogues aux fonctions de combinaison des entités représentant les constituants déjà trouvés avec celles qui représentent les constituants dont la présence est en cours de vérification.

- L'insertion de points d'arrêt dans de tels ajouts permet d'examiner en mode pas à pas le fonctionnement de la règle, et donne l'occasion d'explorer à loisir l'environnement d'exécution précisément quand et où c'est pertinent. En outre, il est souvent important dans des formalismes qui associent des contraintes arbitraires aux éléments des règles, comme c'est le cas pour les formalismes d'unification, de pouvoir pister non une règle ni même un ensemble de règles, mais des éléments au sein de différentes règles qui ont recours à un même attribut, de manière à observer la percolation de cet attribut au cours de la dérivation. Un pistage lié aux règles n'est pas approprié : il produit trop d'information. Dans notre modèle, il est aisé d'attacher des instructions de pistage aux seules entités pertinentes, et ceci à travers toute la grammaire.

Cette approche de la mise au point est efficace du point de vue de l'utilisateur, dans la mesure où elle repose sur une relation causale et immédiate entre la grammaire de départ et les entités présentes à l'exécution. (Carroll & al. 1991) indiquent d'ailleurs qu'un travail efficace n'est possible que si l'on peut revenir des règles fautives pendant l'exécution à l'endroit de la grammaire où elles sont définies. Par contraste, le traçage proposé par exemple par (Bouchard & al. 1992) permet de voir le graphe évoluer, pas de suivre une règle dans son fonctionnement, tandis que (Boguraev & al. 1988) font le lien entre la

grammaire et ses résultats par le biais d'étiquettes attachées aux règles et l'ajout de ces étiquettes aux structures produites... En outre, la distinction est claire entre les traitements normaux et les comportements ajoutés en phase de mise au point.

Au total, par la constitution de points de vue, on peut voir dans une approche réflexive une façon de donner une meilleure prise sur des systèmes touffus. Pour reprendre les termes de (Sabah 1993) : "*[La méta-représentation] est /./ sélective (ce qui permet une réduction de la complexité du système. En outre, le contenu de la méta-représentation dépend du type de raisonnement qui va l'utiliser : elle est spécialisée (un système 'conscient' a donc la possibilité de disposer de plusieurs méta-modèles du même objet). En conséquence, cette représentation est partielle (ce n'est pas une simple image complète du système, mais une représentation des seuls éléments pertinents pour la tâche considérée)*".

9.7. Réflexivité : modification des programmes

9.7.1. Hétéro-modifications

9.7.1.1. Savoir ce qui se passe, vers des traces pour linguistes

Etape de mise au point de traceurs.

Cette phase de mise au point du code de Olmes a permis l'affinement de la présentation des résultats dans le but prédéfini de fournir des informations pertinentes pour les premiers utilisateurs de OLMES à savoir des linguistes. Il fallait mettre en place des mécanismes qui soient capables de tracer les points cruciaux de l'analyse et de fournir des résultats intermédiaires et terminaux pertinents pour les utilisateurs. De plus, la mise au point progressive d'un méta-niveau de l'analyse a permis de garder au delà de l'analyse des informations présentes au moment de l'analyse et donc de pouvoir procéder à une rétro-lecture de ces informations à l'issue de l'analyse.

Les traceurs.

Les traceurs pour l'analyse avec OLMES utilisent principalement la potentialité offerte par CLOS d'ajouter ou de supprimer des méthodes.

- 1. Ajout de méthodes :

CLOS permet de réaliser l'ajout d'une méthode à une fonction générique en aval de la création de ces deux entités. Cet ajout se fait grâce à la fonction générique `add-method`. Cette fonction générique réalise les vérifications de congruence de lambda-list, d'existence d'une méthode utilisant la même forme de discrimination et de l'unité d'utilisation de cette méthode.

- 2. Retrait de méthode :

Les traceurs peuvent à certains moments produire des informations non-pertinentes : on doit pouvoir les activer et les désactiver en fonction de certains besoins particuliers. CLOS offre la possiblité d'exécuter une extraction de méthode en utilisant la fonction générique `remove-method`. Cette action est rendue possible car CLOS dispose de plus d'une fonction générique (`find-method`) capable de retrouver une méthode utilisant une certaine forme de discrimination sur ses arguments.

L'ajout et le retrait de méthodes `before` et `after` s'est révélé être un outil fondamental pour fournir des résultats au cours de l'analyse. On rappelle que la combinaison des méthodes par défaut en CLOS (dite standart) accepte quatre catégories de méthodes, les méthodes primaires, les post-méthodes (`:after`), les pré-méthodes (`:before`) et les méthodes englobantes (`:around`). Les outils de traçage manipulent les post et pré méthodes qui permettent des effets de bord. Les méthodes définies pour l'analyse automatique peuvent être étendues en y adjoignant des pré-méthodes ou des post-méthodes dédiées à fournir des résultats ou des informations sur les objets manipulés par ces méthodes tout au long de leur activité. Les traceurs mis en place ont été développés pour fournir aussi bien des résultats terminaux que des résultats intermédiaires d'une analyse. Un exemple du code défini pour les traceurs est donné en annexe, accompagné des différents résultats produits correspondants aux différentes étapes de développement de ces outils de traçage (-> annexe partie 3 chapitre 9 : une analyse complète est donnée précédée du code complet de l'interface des outils associés à OLMES).

Un méta-niveau dans OLMES.

La première phase de développement d'un méta-niveau pour OLMES (DINOSOR (Fleury & al. 1992), (GASPAR (Fleury 1992)) a permis d'introduire la nécessité de disposer d'informations sur la grammaire compilée à l'issue de l'analyse. Ce travail a donc consisté à introduire une méta-grammaire pour l'analyse. Il s'agissait de "garder" un point d'entrée sur la grammaire à l'issue de l'analyse. Cette phase de développement d'un méta-niveau a permis de mettre en lumière l'importance de travailler à un méta-niveau pour la mise au point des modifications opérées sur le code initial : c'est grâce au développement d'un méta-niveau intermédiaire (la grammaire normalisée) que la mise au point de la méta-grammaire a été facilitée. Le schéma ci-contre illustre cette première phase de développement d'un méta-niveau pour l'analyse :

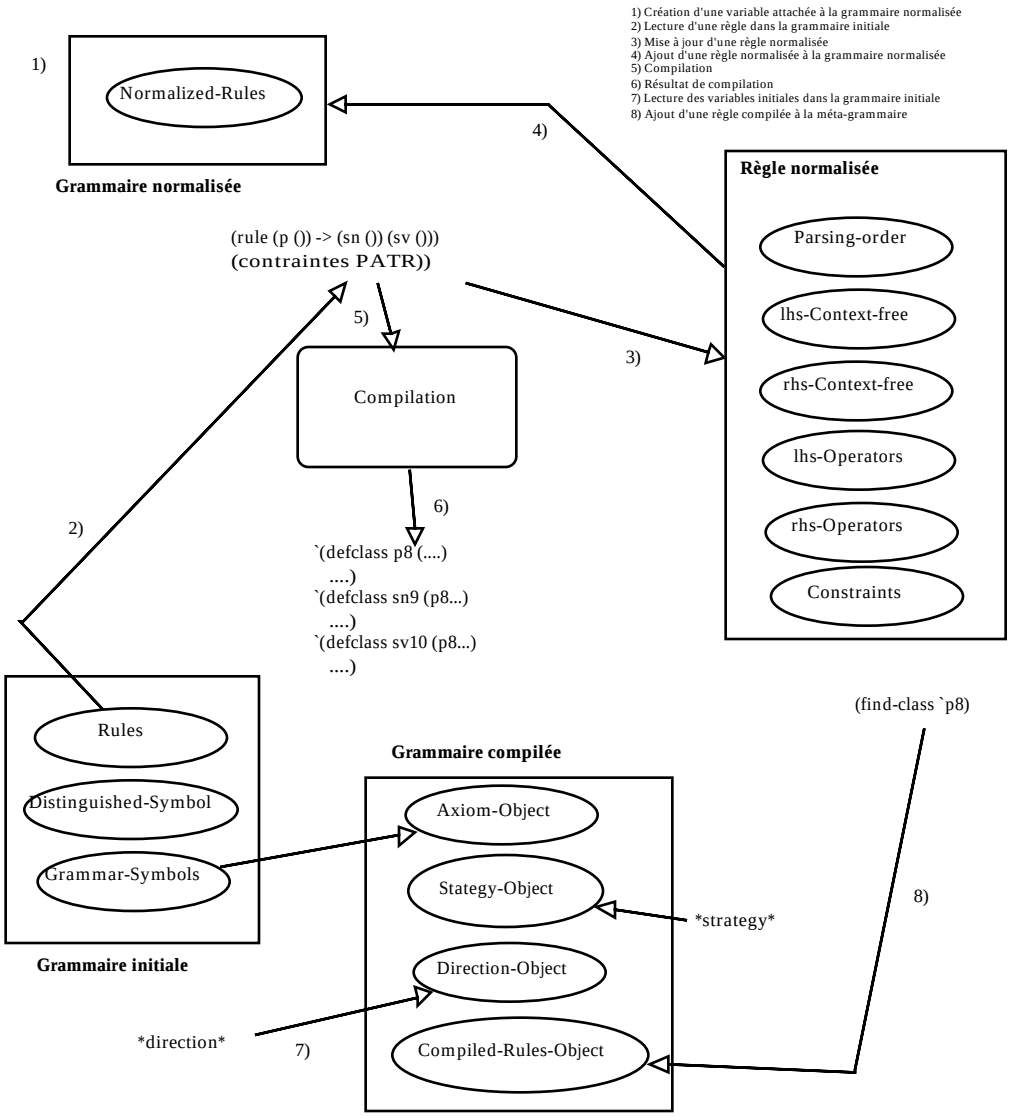


Figure 9.9 : DINOSOR 92 : un premier méta-niveau pour OLMES.

Cette phase de développement a permis de mettre en évidence les défauts de cette présentation (<- le problème mentionné de dépendance hiérarchique entre partie gauche d'une règle et partie droite). La solution retenue dans (Habert & Fleury 1993) maintient la définition d'un méta-niveau pour la grammaire comme suit :

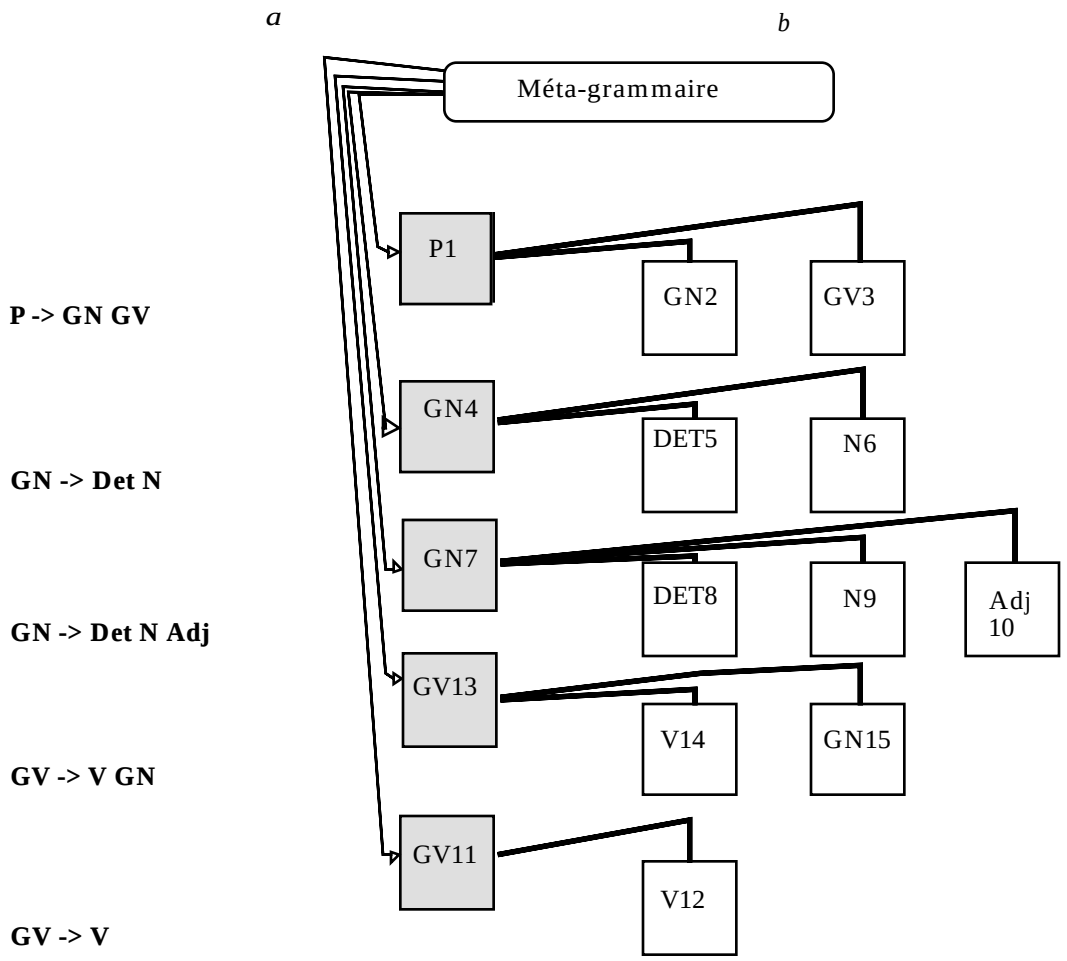
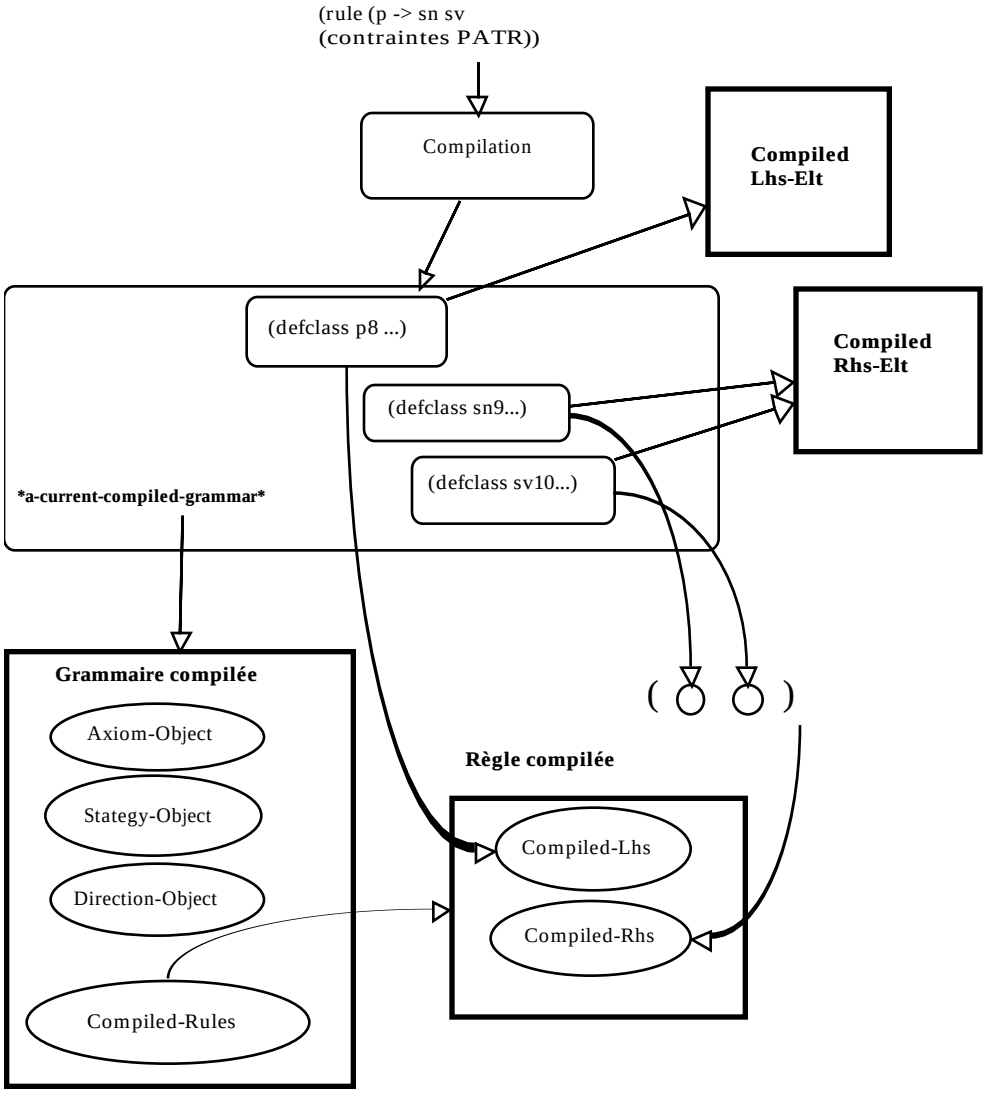


Figure 9.10 : OLMES 93 : une méta-grammaire pour OLMES.

Cette version a permis de prendre en compte de manière fondamentale la nécessité de mettre en place un méta-niveau pour l'analyse (Habert & Fleury 1993). Les défauts déjà mentionnés sur cette version ont été repris et une version ultérieure a amplifié cette notion de méta-niveau. Le point principal a été de ne plus considérer la partie gauche d'un règle comme la super-classe des éléments de la partie droite d'une règle.



Compilation avec OLMES

Figure 9.11 : OLMES 94 : un méta-niveau pour l'analyse avec OLMES.

Le fait d'éclater la règle en autant d'objets qu'il y a d'éléments dans cette règle a permis notamment dans le travail de traçage de pouvoir isoler la création de chacun de ces éléments pour par exemple pouvoir compter le nombre d'instanciation de ces éléments. Cette phase de mise au point d'outils pour une lecture et une compréhension active de l'analyse a aussi pris en compte la nécessité de disposer d'une interface graphique des résultats fournis par les outils développés.

9.7.1.2. De la méta-connaissance à la réflexivité

Montrons avec OLMES comment passer de la production de méta-connaissances à une réelle réflexivité, c'est-à-dire à la modification dynamique du comportement de l'analyseur en fonction de ces méta-connaissances. L'objectif d'une approche réflexive est en effet que chaque entité active non seulement possède les moyens d'observer son comportement à l'exécution, mais qu'elle utilise ces "notes" pour modifier, si nécessaire, ce comportement. CLOS donne des outils pour une telle approche réflexive. Il dispose en effet d'un Méta-Protocole (Kickzales & al. 1991) : chaque entité de la couche objet-classe, méthode ou champ - est une instance d'une méta-classe, donc, en d'autres termes, un objet. Parce que chaque

classe est également un objet, les classes représentant la grammaire de départ possèdent une mémoire propre, et peuvent par conséquent noter leurs observations sur l'analyse. Le nombre de règles déclenchées et de succès pour une règle peut être mémorisé "au sein de chaque règle". La figure qui suit montre les changements qui en résultent pour les classes `GN7` et `Det8` (<- exemple présenté supra) et la mémorisation des essais et des succès dans la version de OLMES présentée dans (Habert & Fleury 1993) :

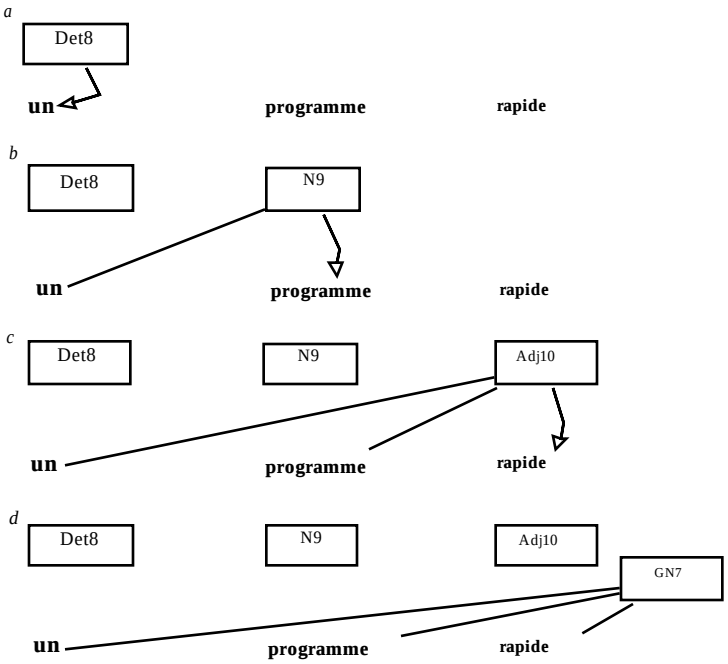


Figure 9.12 : Analyse par les instances dans un réseau de classe.

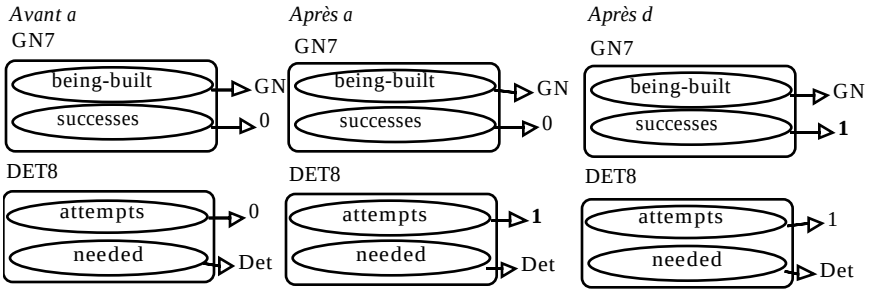


Figure 9.13 : Mémorisation des essais et des succès dans les classes d'une règle.

De nouvelles méta-classes peuvent alors servir à affiner le comportement de ces unités. C'est cette piste prometteuse qui peut permettre la réorganisation du réseau par lui-même en fonction des probabilités attachées à chaque règle après le passage des règles sur un corpus d'essai et l'observation des déclenchements et réussites des règles. Les transformations apportées au modèle de (Yonezawa & Ohsawa 1990), le lien entre la grammaire de départ et les entités qui la représentent lors de l'analyse, permettent précisément cette action en retour de l'analyse sur la grammaire.

9.7.2. Auto-modification

9.7.2.1. `Gaspar` : Un analyseur à mots non entièrement définis

Comment définir des "démons" capables de redéfinir les comportements généraux à factoriser

Pour reprendre ce que nous disions dans les chapitres précédents (<- partie 2 chapitres 4, 6) sur la (re)construction d'une représentation sémantique que l'on peut attacher à `moulin`, il s'agit ici d'évaluer comment il serait possible d'étendre cette représentation sémantique aux noms d'instruments : comment mettre en oeuvre une catégorisation plus générale tout en préservant une évolution potentielle de cette représentation initiale ? La représentation sémantique proposée supra tend à identifier (à isoler) les comportements sémantiques que l'on peut attacher à `moulin` et qui permettent de construire des (débuts d') interprétations possibles qu'il reste possible d'affiner. Si ces comportements traduisent des opérations sémantiques plus générales qu'il serait possible de factoriser pour une classe de noms particuliers (en particulier pour les noms d'instrument), il conviendrait de restructurer l'organisation des représentations sémantiques attachées aux entités représentées pour disposer d'une représentation ouverte et cohérente du domaine à représenter. La figure ci-dessous rappelle l'opération de recatégorisation présentée.

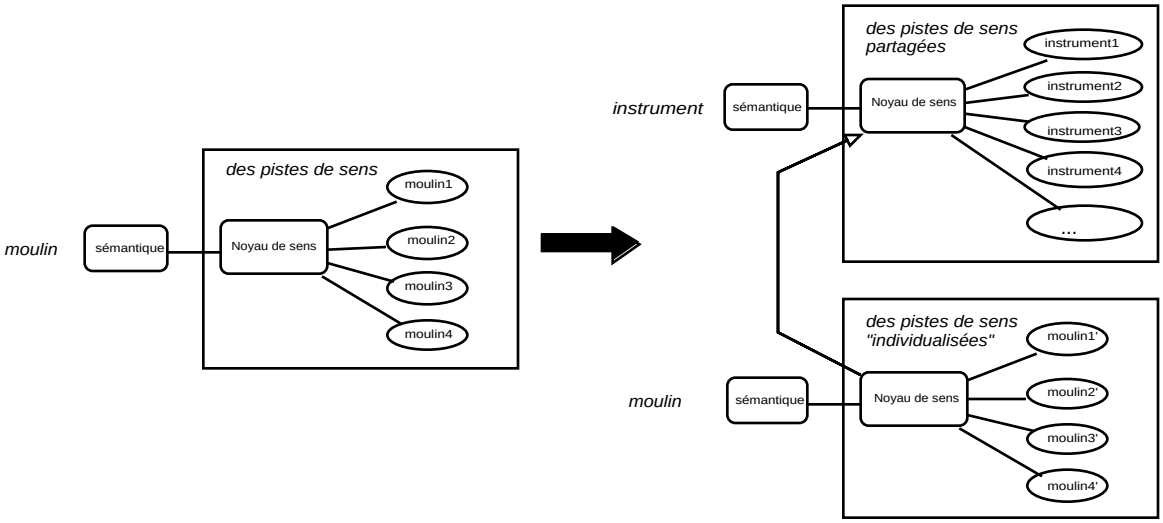


Figure 9.14 : `Moulin` et noms d'instrument, des comportements sémantiques partagés.

Si une approche prototypique se révèle adéquate pour rendre compte d'un micro-phénomène particulier, il convient de ne pas perdre de vue la place de ce micro-phénomène dans un système plus complexe, ou tout au moins de dégager de l'étude particulière de celui-ci, les éléments qui le dépassent.

Du méta-regard sur l'analyse à l'automodification des représentations

Le problème à résoudre peut se poser de la manière suivante :

Tel type de comportement sémantique attaché à `moulin` se révèle être un comportement sémantique plus général pour une classe de noms plus large. Est-il possible de réorganiser cette part de savoir plus général ? Comment effectuer cette réorganisation du savoir ? Quel est l'impact d'une telle réorganisation sur les savoirs représentés ? Il convient pour cela de disposer de savoirs capables de détecter que tel comportement attaché à `moulin` se retrouve sous une forme assez proche dans la représentation sémantique attachée à un autre nom d'instrument. Puis d'étendre ce rapprochement de comportements partagés à un ensemble d'entités représentées. La difficulté ne s'arrête évidemment pas là, il s'agit ensuite de définir les modalités d'une réorganisation des savoirs partagés. La question de savoir si un tel aménagement est souhaitable n'est d'ailleurs pas à écarter : ne risque-t-on pas de définir des regroupements aléatoires dont la pertinence n'est pas justifiée? Il peut en effet y avoir des recoupements entre les comportements sémantiques de plusieurs classes de noms, il faut pouvoir y retrouver une cohérence à moins d'y perdre toute volonté de classer les choses "raisonnablement".

Un méta-niveau pour l'analyse avec `Gaspar`

<u>Utilisation de la réflexion de structure en Self</u>

L'approche présentée supra (<- partie 2 chapitres 7, 8) réalise une génération dynamique des représentations prototypiques des savoirs linguistiques à partir de données recueillies sur un corpus. Celle-ci met en avant la nécessité de disposer d'un méta-niveau de représentation pour la gestion des objets créés ou à modifier. On doit pouvoir ajuster :

- Les représentations des objets associés à des unités de langue (ajout ou retrait d'informations).
- Les processus qui agissent sur ces unités.
- Les liens qui existent entre ces unités (modifier, ajouter des liens d'héritage ou autre sur ces unités).

Etant donné la variation potentielle des structures représentées, on doit pouvoir connaître la structure d'un objet donné à un moment particulier de son activité. Si l'on veut modifier cette structure, on doit disposer d'un méta-regard sur la composition de cet objet : les attributs qui le composent, les liens d'héritage qui lui sont associés... L'interface graphique de Self-4.0 permet à tout moment d'agir sur la représentation d'un objet donné : ajouts ou suppressions d'information, modifications de valeurs ou des comportements... On peut donc agir directement et interactivement sur les objets représentés. Pour construire un programme de TALN capable d'agir dynamiquement sur ces mêmes objets, on doit disposer d'outils qui produisent les mêmes résultats mais qui agissent de manière transparente pour permettre une modification dynamique des représentations construites. On présente ci-dessous une suite d'opérations qui permettent d'agir sur les objets représentés sans passer par l'interface graphique de Self (-> annexe partie 3 chapitre 9 : trace des (méta-)opérations via `Gaspar`). On choisit volontairement cette solution pour donner à penser ce que le programme construit réalise lui de manière dynamique.

Utilisations de méta-savoirs avec Gaspar

1. Un méta-regard sur les objets

Dans les différentes phases de travail présentées (<- partie 2 chapitres 6, 7 et 8), les structures de représentation construites pour représenter les unités de langue peuvent évoluer à tout moment de leur existence : on peut ajuster, remodeler la représentation des objets en y ajoutant, en y supprimant ou en y modifiant les informations associées. Il est donc impossible de prévoir toutes les facettes d'un objet donné : on ne prévoir toutes les évolutions que peuvent subir les représentations des unités de langue. Par contre, il reste possible à tout moment de connaître la structure de cet objet. Le dispositif permet à tout moment, via les (méta-)outils construits, d'interroger les objets pour connaître leur structure. On donne en annexe (-> annexe partie 3 chapitre 9) une présentation du prototype lexical associé au mot `lesion` en utilisant les outils disponibles avec `Gaspar`. En utilisant l'interface graphique de Self, on dispose aussi des outils définis par cette interface pour obtenir une présentation interactive de la structure interne d'un objet. Il reste cependant important de pouvoir travailler avec ces savoirs structurels dans un programme en cours d'exécution (de manière transparente pour l'utilisateur) et donc de pouvoir récupérer ces informations de manière automatique si on doit agir sur celles-ci. On a vu aussi que l'on pouvait pour un ensemble de prototypes donnés construire un pôle destiné à porter les comportements communs à ces prototypes. La construction de ce pôle s'accompagne de la mise en place de nouveaux liens entre les prototypes concernés et le pôle construit. Si l'on s'intéresse à l'un des prototypes de la famille, et si l'on veut connaître les attributs que celui-ci délègue à d'autres objets, il faut pouvoir détecter parmi ses attributs ceux qui révèlent un lien de délégation vers un autre objet pour examiner les attributs délégués. Pour réaliser ce genre de requête sur un prototype Self, on utilise des outils disponibles avec Self qui agissent à un méta-niveau et qui permettent de décrire et de modifier la structure d'un objet donné.

2. Sélection des savoirs à manipuler

L'environnement mis en place avec Self facilite les manipulations concrètes et directes sur les entités représentées. Un utilisateur peut "communiquer" avec les agents du programme directement pour les modifier éventuellement sans avoir à connaitre précisément les mécanismes qui permettent ces modifications. Un programme Self est constitué d'entités "vivantes" et non d'entités inertes et immuables. Pour changer un objet, il est inutile de recompiler, il suffit de modifier l'objet suivant les besoins. Notre approche expérimentale s'accorde bien avec la possibilité dans l'environnement programmatique choisi de

modifier interactivement les représentations informatiques créées. L'interface de Self permet une visualisation des objets, et l'utilisateur peut "ausculter" les contenus des différents attributs ou des méthodes associés aux objets. Il peut aussi agir sur ces différentes facettes d'un objet. Nous avons déjà indiqué que certaines phases de travail manuel peuvent être réalisées pour affiner les résultats construits quand une action directe sur les représentations produites est nécessaire. Il n'en reste pas moins qu'une part importante de notre programme gère automatiquement les savoirs rencontrés. Il doit donc pouvoir sélectionner les savoirs à traiter s'il doit agir sur eux. Pour réaliser ce type de sélection, le dispositif utilise les méta-outils mis en place par Self pour interroger la structure d'un objet donné. Prenons un exemple :

Sélectionner le destinataire d'un message

Si un attribut de même nom est disponible dans un prototype et dans un ou plusieurs de ses parents, on peut sélectionner le récepteur auquel on veut adresser une requête sur cet attribut. Dans notre travail de classement on a montré précédemment que les processus définis mettent en place un réseau complexe de liens de délégation entre les prototypes de mots. Il est donc nécessaire de pouvoir retrouver et utiliser les liens de délégation définis pour adresser les messages pertinents aux objets adéquats. De plus, ces liens de délégation ne sont pas figés de manière définitive. Il est donc important de pouvoir sélectionner le destinataire d'un message si un objet donné délègue des comportements à différents parents : un mot peut déléguer des comportements (des arbres) à plusieurs parents, il convient donc de pouvoir retrouver les différents chemins possibles pour récupérer le comportement adéquat (<- partie 2 chapitre 7). Il convient pour cela de pouvoir :

- 1. Connaître pour un prototype les liens d'héritage qui lui sont associés.
- 2. Récupérer tous ses parents.
- 3. Retenir le destinataire visé.

Self permet d'interroger la structure de tous les objets disponibles dans le système : on peut ainsi déterminer si un attribut donné dans un objet est un attribut parental, (i.e. cet attribut induit un lien de délégation entre l'objet visé et un autre objet). En outre, on dispose avec Self de primitives qui permettent de "forcer" l'envoi d'un message à un parent donné. On peut donc adresser le message souhaité au parent visé en utilisant par exemple la syntaxe suivante : `'message' sendTo: Objet DelegatingTo: parentChoisi.`

3. Ajouter de la délégation

Self ne dispose pas de classes mais permet de construire des liens d'héritage dynamiquement entre les objets visés. Le classement construit sur les prototypes lexicaux présenté supra (<- partie 2 chapitre 8) construit des hiérarchies locales entre un sous-ensemble de prototypes et un pôle de comportements partagés par ces prototypes. Cette opération réalise en fait une construction dynamique de liens hiérarchiques entre objets. Ajouter un lien de délégation à un prototype est une opération qui consiste à établir un lien entre deux objets, on indique au prototype fils qu'il délègue des informations (attributs ou méthodes) à un prototype père. Ceci n'est qu'une application naturelle de Self qui permet à tout moment d'ajuster la représentation des objets disponibles. Le dispositif réalise automatiquement cette mise en place de délégation entre prototypes (-> annexe partie 2 chapitres 8, 9); les processus mis en place agissent à un méta-niveau pour évaluer les modifications effectuées par ces opérations dynamiques de changement d'état d'un objet.

4. Supprimer la délégation

Si on peut ajouter de la délégation, l'opération inverse est elle-aussi possible. Il convient simplement de connaître le lien à supprimer (i.e. le(s) prototypes fils et le(s) prototype(s) père concerné(s)). On dispose avec Self de mécanismes qui permettent d'interroger les attributs d'un objet, on peut en particulier savoir si un attribut donné est un attribut qui délègue des comportements. Le dispositif utilise donc ces mécanismes pour réaliser des opérations de suppression de lien de délégation entre objets. Dans le système `Gaspar`, on dispose par exemple de processus capables de supprimer automatiquement les liens de délégation de même nature (i.e. les mêmes comportements étant portés par deux ou plusieurs pôles différents).

5. Un Méta-traçage

Le travail sur des processus réflexifs pour un traitement automatique passe impérativement par une phase de développement de processus de traçage des activités d'un programme de traitement automatique. Dans les différentes phases de travail présentées, les structures de représentations construites peuvent subir des manipulations diverses : elles peuvent être structurellement modifiées ou affinées. On conçoit donc qu'il soit utile, voire indispensable, de pouvoir suivre ces évolutions potentielles. A cet effet, les processus réflexifs disponibles avec le cadre de représentation choisi ont permis de développer des outils de suivi ou de mise au point de l'analyse. Ces outils présentent, de manière raisonnée, les différents états pertinents des traitements réalisés et des résultats construits à chaque étape. Les différents outils créés au cours de ce travail ont d'abord été conçus comme des modules indépendants capables de gérer eux-mêmes le traçage de leurs activités. Le développement d'un méta-niveau de travail a conduit à modifier cet isolement initial. En fait, chaque module reste indépendant en ce qui concerne son éventuel traçage mais utilise un module de traçage qui se situe à un méta niveau.

```
GasparMetaLook queryForActiveTrace ActiveOutputTrace <- true/false
```

6. Un Méta-regard pour maintenir la cohérence des objets construits

On a vu précedemment que pour contrôler la génération des prototypes, des mécanismes de vérification de la structure interne des objets construits ont été définis : un prototype sera "cohérent" s'il est issu des différentes phases de construction des prototypes de mots ou des prototypes d'arbres. Ce qui indiquera qu'il possède tous les facettes nécessaires pour être considéré comme valide dans l'environnement de Gaspar. Pour ne pas multiplier inconsidérement les prototypes incohérents, on dispose d'un mécanisme qui permet de "détruire" tous ces prototypes incomplètement construits ou tout au moins de les "poubelliser". Ces opérations sont définies là aussi en utilisant les outils de méta-représentation disponibles avec Self qui permettent de supprimer tous les attributs à un objet donné.

Classification automatique et processus réflexifs

Dans l'état actuel du classement effectué, une intervention manuelle est indispensable pour interpréter les résultats construits. Les limites de l'automatisation marquent le champ de travail qu'il reste à effectuer manuellement pour mener à bien le classement des unités lexicales. Le classement à mettre en place est fortement dépendant des informations qui ont permis de définir ce classement. Il reste donc indispensable de s'interroger sur la nature des informations linguistiques qui sont manipulées pour ce classement. Les informations recueillies actuellement en amont doivent être renforcées et affinées. Notre travail doit d'une part préciser quelle est la nature des informations à traiter et suivant les informations manipulées, quel type de classement on obtient, et quel type de classement veut-on ou peut-on obtenir sur la base de ces informations. Il doit aussi mettre en place des processus très fins de classement capables de mettre de l'ordre dans les structures construites. Plus précisément, puisque le classement présenté supra met en place des liens de délégation entre des prototypes et des pôles de comportements partagés, une automatisation plus fine de ce classement doit passer par la mise en place d'un vaste et complexe réseau des liens de délégation. Il reste donc à évaluer les interactions et concurrences qu'un tel réseau induit. Il ne suffit pas d'ajouter des liens de délégation dans tous les sens pour produire le classement escompté. Il faut pouvoir évaluer les implications de la mise en place effective de plusieurs liens de délégation sur un même prototype donné : il faut par exemple s'assurer que l'on ne produit pas arbitrairement de l'ambiguïté dans les liens mis en place. La question de l'attestation ou de la non-attestation, par le système lui-même, de ses propres représentations est un point crucial pour l'ajustement conceptuel. On conçoit donc qu'il soit nécessaire de disposer de processus qui agissent à un méta-niveau et qui soient chargés de ce contrôle (Bachimont 1992). Les outils présentés précédemment ne constituent qu'une première phase de développement de tels outils de contrôle.

Mettre en place la génération de nouveaux savoirs

Notre travail vise à permettre aux objets créés d'apprendre par ajustements successifs de nouvelles connaissances. Dans les chapitres précédents, les prototypes lexicaux apprennent dynamiquement les

savoirs syntactico-sémantiques qu'il est possible de leur associer. Cet apprentissage est réalisé sous la forme d'une relation d'apprentissage d'un maître à un élève. Le maître en l'occurrence se matérialise tout d'abord sous la forme des informations initiales recueillies sur un corpus donné. Le classement produit peut aussi conduire l'utilisateur à intervenir directement sur les résultats construits pour affiner ces résultats. Cet apprentissage via l'acquisition de connaissances proposées par un maître ne suffit pas (Pitrat 1995). Un programme de TALN doit pouvoir construire de nouvelles connaissances à partir des connaissances connues. Nous avons déjà indiqué les difficultés fondamentales que posent cette dernière assertion. Nous nous limiterons ici à présenter quelques exemples simples qui illustrent cette génération de nouveaux savoirs à partir de savoirs existants.

Affiner les processus autodescriptifs des objets

Les objets construits doivent évaluer leurs fonctionnements et leurs comportements par rapport aux autres objets. Il convient donc de mettre en place des processus fins pour permettre aux objets construits de donner une bonne description d'eux-mêmes afin qu'ils puissent comparer cette description à celles des autres objets. Si deux objets (représentant des mots ou des arbres) doivent "dialoguer" pour évaluer leurs descriptions respectives et celles de leurs comportements, il convient que ces objets puissent activer des processus capables de produire ces descriptions.De plus, dans la mesure où les objets construits sont susceptibles d'évoluer, il convient que les processus d'autodescription des objets construits puissent être affinés dynamiquement pour tenir compte de ces évolutions potentielles. Self permet la mise en place de mécanismes qui réalisent cette description des attributs d'un objet donné à tout moment de son existence dans le système. La mise en oeuvre de telles opérations sur tous les objets du système construit reste à faire. De même il convient de poursuivre la mise en place de processus capables d'interroger plus précisément les attributs (de comportements) attachés aux mots et à ceux des arbres associés si l'on veut donner du sens aux savoirs ainsi représentés.

Création de pôles de comportements

La phase de travail qui consiste à repérer les savoirs communs puis à définir et à créer les pôles de comportements partagés peut être assimilée à une première étape dans la construction de nouveaux savoirs. Celle-ci reste malgré tout à affiner et à développer. Ces regroupements de savoirs partagés par un ensemble de prototypes lexicaux ou par un ensemble de prototypes d'arbre ne sont en effet pas identifiés ou nommés. L'opération qui consiste à évaluer la ressemblance comportementale entre les mots ne qualifie pas les ressemblances constatées.

Création automatique de prototypes de mot au cours de l'analyse

On a déjà indiqué que si au cours de l'analyse le parser rencontre un mot nouveau, il construit automatiquement des prototypes lexicaux (pour chaque catégorie déjà recensée) pour représenter ce nouveau mot. De plus ne sachant pas si ce nouveau mot est forcément un représentant d'une des catégories grammaticales déjà reconnues, le parser construit aussi un prototype lexical de catégorie inconnue pour représenter ce mot nouveau. Ce dernier type de prototype pourra entrer dans toutes les compositions syntaxiques sollicitées par d'autres prototypes d'arbres. Cette phase de génération automatique de nouveaux prototypes de mot au cours de l'analyse utilise les représentations déjà construites pour en produire dynamiquement de nouvelles : le parser prédit différentes représentations prototypiques pour ce mot nouveau en tenant compte des représentations prototypiques de mot qu'il connaît déjà. la mise en place de ce type de processus reste encore à affiner. Il faut en particulier pouvoir enregistrer si les représentations construites par défaut ont permis de donner une solution satisfaisante ou non et dans tous les cas garder trace de ces résultats.

Des règles pour la génération des nouveaux objets

La phase de génération de prototypes mise en place supra (<- partie 2 chapitre 8) reconstruit systématiquement et de toutes pièces un nouveau prototype s'il n'existe aucun représentant de sa famille catégorielle sans vérifier plus avant si les prototypes existants sont capables de fournir des informations pour la génération de ce nouvel élément. Il serait en fait possible de modifier cette génération de prototype en utilisant de manière plus fine les savoirs déjà construits. Ce qui permettrait de plus d'améliorer les performances du système construit en limitant par exemple le nombre d'écritures sur disque d'informations générées pour réaliser la génération d'un nouvel objet. En effet, en examinant plus précisément la structure des objets déjà construits, il ne serait plus nécessaire de reconstruire de toutes pièces un nouvel objet; en s'inspirant de la structure des objets construits, il suffirait d'ajuster ces structures pour produire de nouveaux objets en utilisant par exemple les règles de génération suivantes :

Méta-Règle n°1 :

Si un prototype d'arbre a x composants
S'il n'existe pas de moule d'arbre à x composants
Alors créer ce moule en s'appuyant sur ce prototype

Méta-Règle n°2 :

Si un Objet Ae (Arbre élémentaire) a x composants.
S'il n'existe pas de moule d'arbre à x composants.
Alors créer ce moule en s'appuyant sur Ae.

Ce travail reste à définir et à mettre en place.

Créer un dictionnaire d'erreurs

Nous avons déjà indiqué qu'une part de notre travail d'interprétation des structures construites passe par une phase de travail manuel qui va consister à agir directement sur les objets construits pour affiner leurs représentations. Afin de garder trace de ces interventions manuelles (faite au méta-niveau que constitue le méta-regard de l'utilisateur), toute intervention manuelle ou provenant du programme sur les représentations sera enregistrée dans un dictionnaire. Le méta-niveau permet dans ce cas d'évaluer les représentations disponibles et de les corriger si nécessaire, tout en gardant trace de ces interventions sur celles-ci. **Cette trace des erreurs assimile l'apprentissage mis en place à un processus qui enregistre le positif et le négatif**.

9.9. Bilan des activités gaspariennes et méta-perspectives

9.9.1. Activités gaspariennes

On présente ci-dessous un schéma global des différentes activités définies via `Gaspar`.

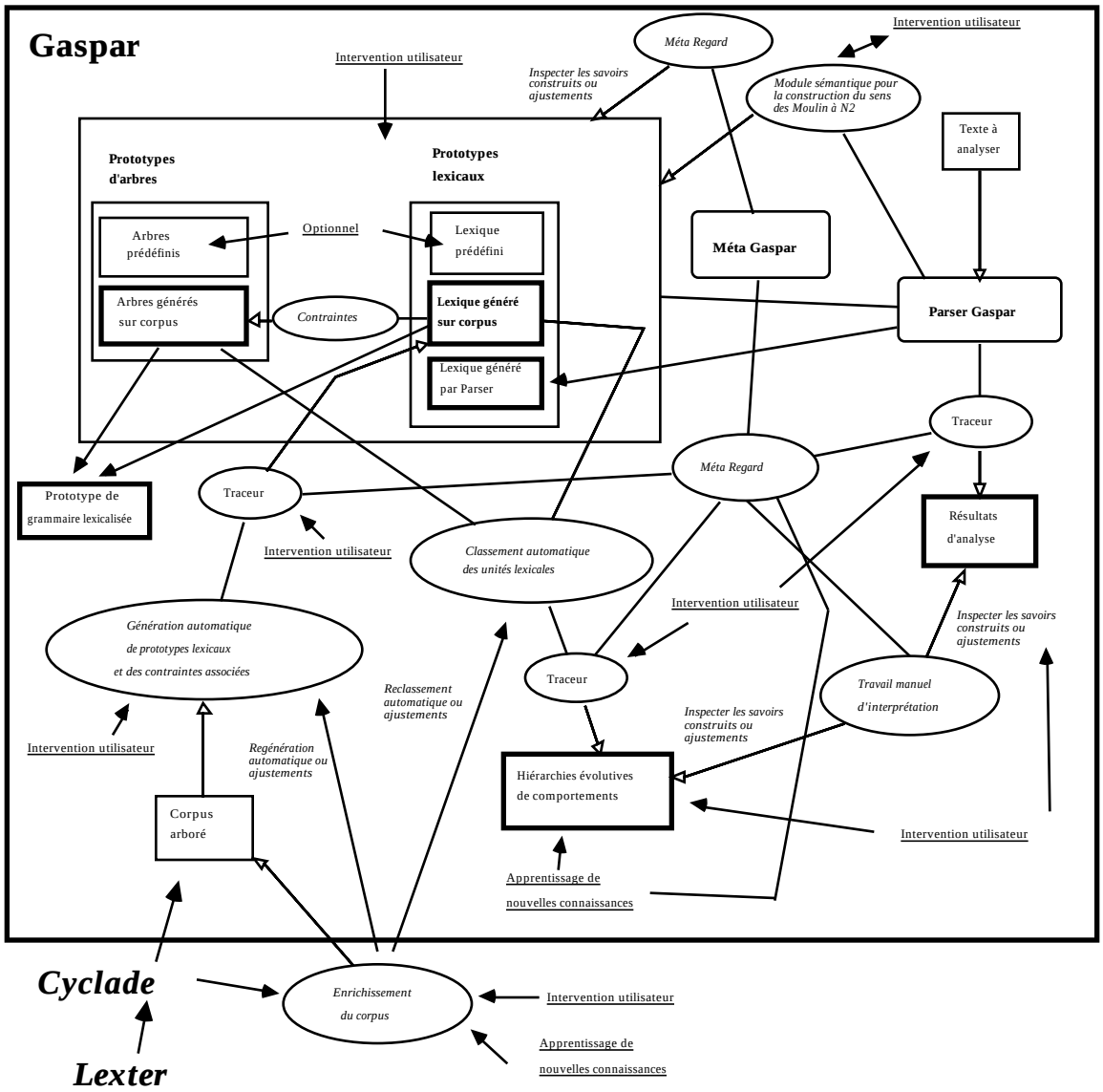


Figure 9.15 : Schéma global des activités gaspariennes.

Ce schéma ne rend évidemment pas bien compte du fait que de cette hétérarchie enchevêtrée naît de la représentation. Il met simplement en avant la complexité des mécanismes mis en place et de leurs connexions. La myriade de prototypes qui résultent du travail de génération automatique de prototypes de mots et d'arbres est en effet difficilement réductible à une présentation schématique.

Le dispositif `Gaspar` (sous Self-4.0) est composé de plusieurs modules (<- partie 2 chapitre 8 : catégorisation des attributs Gaspar) :

- Un module de calcul sémantique des séquences du type `moulin à N2` avec intervention éventuelle de l'utilisateur pour affiner les résultats (<- partie 1 chapitre 1, 4 <- partie 2 chapitres 5, 6, 7).

- Un module pour la génération automatique de prototypes de mot et de prototypes d'arbre à partir de savoirs extraits sur corpus (<- partie 2 chapitre 8).

- Un module de classement des prototypes construits ou disponibles (<- partie 2 chapitre 8).

- Un module méta (<- paragraphes précédents).

Ce dernier module regroupe différents processus qui sont utilisés par les différents modules mis en place. Il permet d'activer des méta-regards sur les savoirs représentés : il intègre notamment les processus de traçage que l'utilisateur peut activer sur toutes les opérations disponibles dans chaque module. Ces processus de traçage décrivent les différentes étapes des opérations déclenchées. L'utilisateur a aussi la possibilité d'intégrer automatiquement une présentation des résultats construits dans l'interface graphique de Self. Il peut ainsi avoir une présentation complète des représentations manipulées ou construites. **Ces objets sont alors manipulables et modifiables à volonté et de manière dynamique**. Ce méta-module peut aussi activer des actions de modification dynamique des représentations construites : ces processus peuvent agir de manière transparente pour l'utilisateur.

- Un module d'analyse syntaxique (<- partie 2 chapitres 5, 6).

La parseur permet plusieurs alternatives lors d'une analyse :

- Il peut travailler (à la demande) avec des contraintes prédéfinies (<- partie 2 chapitre 5, 6) ou avec les contraintes générées automatiquement (<- partie 2 chapitre 8).

- Il peut utiliser les différents opérateurs mis en place pour la construction automatique d'arbres par substitution ou adjonction (<- partie 2 chapitre 8).

- Il peut activer (à la demande) le module particulier de calcul sémantique sur les séquences du type `moulin à N2` avec intervention éventuelle de l'utilisateur pour affiner les résultats.

- Le parseur peut enfin utiliser (à la demande) les représentations déjà construites pour en produire dynamiquement de nouvelles (<- partie 2 chapitre 8 et paragraphes précédents).

Un unifieur est en cours d'implémentation. Il utilisera les contraintes associées aux mots et aux arbres; ces contraintes étant soit issues du travail d'extraction de savoirs à partir de corpus soit mises à jour dynamiquement sur les représentations construites (par l'utilisateur par exemple). Cette implémentation doit permettre, en particulier, de restreindre le nombre de prototypes construits mais non pertinents à l'issue d'une analyse (notamment quand la phrase à analyser contient des mots inconnus (-> annexe partie 3 chapitre 9 : analyse avec `Gaspar`)). La figure qui suit présente plus précisément les activités gaspariennes mises en place pour l'analyse :

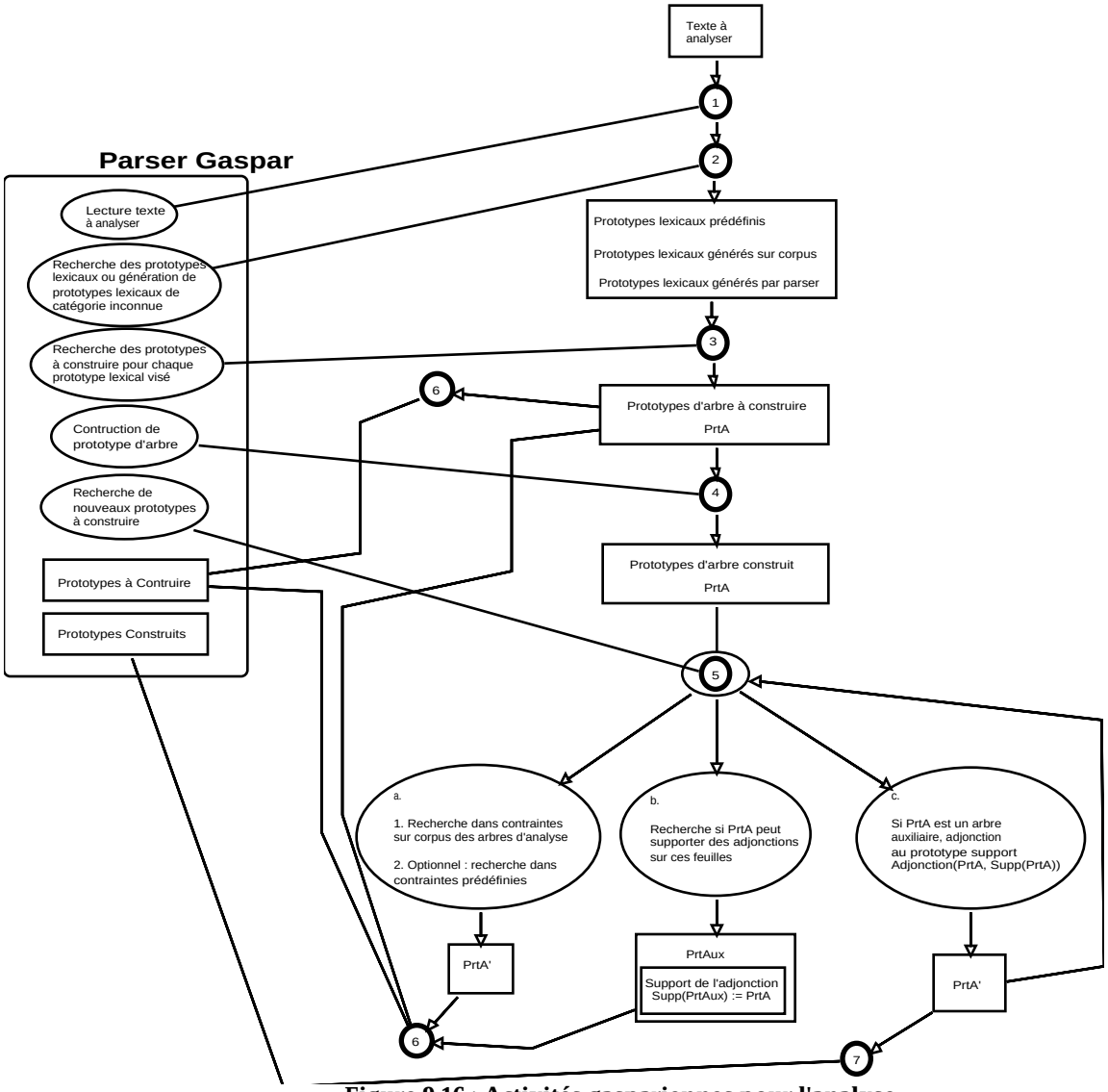


Figure 9.16 : Activités gaspariennes pour l'analyse.

9.9.2. Perspectives

Les différentes phases de travail présentées ont mis en avant des points cruciaux pour la mise en place de processus de représentation des faits de langue.

La matière à représenter est "vivante" : elle évolue en permanence, dans le temps et dans ces mises en action. Les outils de représentation doivent donc ne pas figer les savoirs représentés : les structures de représentation construites doivent restées ouvertes et évolutives. La mise en place de telles structures de représentation évolutives pour un traitement automatique du langage naturel pose le problème fondamental du contrôle permanent à définir pour maintenir une cohérence dans le domaine représenté. Et on perçoit aisément les difficultés auxquelles on est confronté pour réaliser un dispositif de traitement automatique : la langue articule un grand nombre de connaissances et "se joue" de leurs représentations.

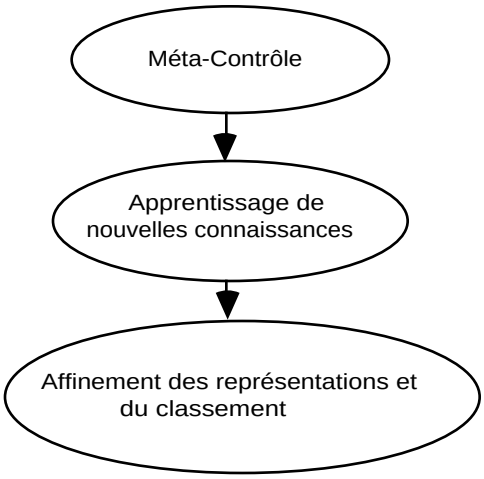


Figure 9.17 : Méta-perspective.

Notre travail n'est qu'une illustration de ces problèmes. En restreignant la couverture linguistique, on a pu mettre en avant l'immensité de la tâche à accomplir. Ce travail doit être poursuivi.

En particulier, le développement d'outils pour la représentation et pour le classement dynamiques des savoirs linguistiques doit être étendu. Ce travail est contraint par une double nécessité. D'une part, il est nécessaire de mettre en place des outils de contrôle capables à tout moment d'évaluer les différents types de savoirs manipulés et leurs interrelations existantes ou à venir. D'autre part, la réalisation d'un classement automatique général des unités lexicales semble hors d'atteinte tant qu'on ne pourra pas construire de nouvelles connaissances capables d'enrichir les savoirs déjà établis : en sachant de plus que ce classement à atteindre ne sera jamais un résultat définitif. Il reste malgré tout possible d'affiner les processus de classement mis en place et de fait d'affiner le classement visé (en connaissant ses limites) tout en continuant à travailler sur les métaconnaissances à l'oeuvre dans les faits de langue utilisés.

Glossaire

Glossaire

catégorie

En linguistique, le mot catégorie s'applique aux notions telles que le genre, le nombre, le temps, l'aspect...

La réponse classique, aristotélicienne, *"est que la catégorisation se fait sur la base de propriétés communes. Le rassemblement dans une même catégorie d'objets différents ne fait en effet plus de difficultés si l'on admet que les éléments réunis présentent un ceratins nombre d'attributs en commun. Pour décider de l'appartenance d'un x à la catégorie des chiens, il suffit de vérifier si le x en question possède les attributs qui constituent le dénominateur commun de la catégorie, autrement dit, s'il est un animal, un mammifère, etc... S'il vérifie ces propriétés, ce sera un chien... La catégorisation ainsi conçue répond à un modèle de conditions nécessaires et suffisantes (CNS)"* (Kleiber 1991). Ce modèle aristotélicien de la catégorie repose sur les propriétés suivantes:

- **1. Les concepts ou catégories sont des entités aux frontières clairement délimitées.**
- **2. L'appartenance d'une entité particulière à une catégorie répond au système du vrai ou faux : un x quelconque est un chien ou n'est pas un chien, selon qu'il satisfait ou non aux conditions critériales de la catégorie "chien".**
- **3. Les membres d'une même catégorie ont un statut catégoriel égal, puisque chaque membre possède les propriétés requises par la définition de la catégorie. Eu égard à la catégorie à laquelle ils appartiennent, chaque membre est un aussi "bon" membre que les autres (Kleiber 1991).**

En fait, ce qu' Aristote pensait établir comme la liste des catégories de pensées[100] n'est en réalité qu'une liste de catégories de langue[101] : *"Aristote a défini dix catégories comme genres suprêmes de l'Etre. Ces catégories, à valeur ontologique par définition, sont des catégories logiques. Elles représentent toutes les façons de prédiquer d'un sujet : selon la substance, quantité, qualité, relation, lieu, temps, position, possession, action, passion. La proposition "S est P" est prise en autant de façon qu'il y a de catégories : ce sont les "diverses classes auxquelles ce Philosophe a voulu réduire tous les objets de notre pensée, en comprenant toutes les substances sous la première, et tous les accidents sous*

---

100 *"Chacune des expressions n'entrant pas dans une combinaison signifie: la substance; ou combien; ou quel; ou relativement à quoi; ou où; ou quand; ou être en posture; ou être en état; ou faire; ou subir. "Substance" par exemple, en général, "homme", "cheval"; - "combien", par exemple "de deux coudées; de trois coudées"; - "quel", par exemple "blanc; instruit"; - "relativement à quoi", par exemple "au lycée; au marché"; - "quand", par exemple "hier; l'an passé"; - "être en posture", par exemple "il est couché: il est assis"; - "être en état", par exemple "il est chaussé; il est armé"; - "faire", par exemple "il coupe; il brûle"; - "subir", par exemple "il est coupé; il est brûlé". Aristote pose ainsi la totalités des prédicats que l'on peut affirmer de l'être, et il vise à définir le statut logique de chacun d'eux"* (Benveniste 1966).

101 *"Pour autant que les catégories d'Aristote sont reconnues valables pour la pensée, elles se révèlent comme la transposition des catégories de langue. C'est ce qu'on peut dire qui délimite et organise ce qu'on peut penser. La langue fournit la configuration fondamentale des propriétés reconnues par l'esprit aux choses. Cette table de prédicat nous renseigne donc avant tout sur la structure des classes d'une langue particulière"* (Benveniste 1966).

les neuf autres." (...) Ces catégories sont issues du langage. Notre façon de parler, encadrée par les catégories, détermine ce qui les choses peuvent être. (...) Benveniste a montré le caractère proprement linguistique des catégories qui au lieu de possèder l'universalité ontologique qu'y voyait Aristote, sont plutôt des attributs de la langue grecque. La logique aristotélicienne est d'origine linguistique" **(Bachimont 1992).**

Toute catégorisation langagière s'organise comme un compromis entre un système de marques expressives et celui des régulations cognitives c'est-à-dire des visées de connaissances, qu'à chaque fois, de tels emplois permettent ou favorisent. Il s'agit en permanence d'une négociation entre deux systèmes, celui du penser pour faire dire et celui de l'exprimer pour amener à ainsi penser. Les opérations au coeur de ces mécanismes ne relèvent ni du mental isolé, ni du linguistique : ***"il s'agit plutôt de jeux sur l'extension des formes et sur l'intension que ces formes peuvent véhiculer"*** **(Vignaux 1986). Même si "*****les catégories de la grammaire ne sont en aucune façon la traduction de quelconques "lois de la pensée"; néanmoins, elles donnent à penser sur ce que peut être "l'ordre du penser" lorsqu'il se doit confronter à emprunter ce système essentiel d'expression qu'est le langage"*** **(Vignaux 1986).**

catégorisation

Toute description de la langue nécessite que l'on nomme les unités que l'on étudie (i.e les mots constituant les unités de la langue). L'utilisation de symboles catégoriels a toujours été un caractère définitoire de la description linguistique qui n'a pu se passer d'une analyse en "parties du discours". On classe les mots selon leur nature, leur forme, leur fonction, leur sens. En français, on peut considérer que les parties du discours recouvrent en général les familles suivantes : le nom, l'adjectif, l'article, le pronom, le verbe, l'adverbe, la préposition, la conjonction et l'interjection. Si l'on évalue l'évolution de la définition d'une catégorie comme celle de l'adjectif depuis la grammaire de Port Royal jusqu'à celle donnée par la Nouvelle Grammaire Française (Grévisse 1989) on mesure la difficulté qui existe pour la classification des formes adjectivales. En effet dans la Grammaire de Port Royal, la distinction entre le substantif et l'adjectif s'exprime ainsi:

"Les noms qui servent à exprimer les choses s'appellent substantifs (...) c'est aussi ceux qui signifient premièrement et directement les modes."

"Les noms qui signifient les choses comme modifiées sont appelés adjectifs."

Ce qui distingue les noms des adjectifs repose donc ici sur la nature de l'opération par laquelle l'esprit saisit le mode ou la chose. L'adjectif n'est considéré que dans un rapport avec un nom et n'est pas envisagé comme une catégorie grammaticale indépendante. Cette définition est avant tout notionnelle. Dans la Nouvelle Grammaire Française, on trouve la définition suivante associée à la classe des adjectifs :

"L'adjectif est un mot qui varie en genre et en nombre, genre et nombre qu'il reçoit par le phénomène de l'accord, du nom auquel il se rapporte. Il est apte à servir d'épithète et d'attribut... Du point de vue sémantique, l'adjectif exprime une manière d'être, une qualité de l'être ou de la chose désignée par le nom auquel il se rapporte".

Cette définition dégage des éléments définitoires syntaxiques, sémantiques, morphologiques et fonctionnels. Ce changement d'analyse notoire, fondée sur l'étude d'un convergence de différents critères définitoires illustre un modèle d'appréhension nouveau pour cette catégorie. Dans tous les cas, on sent bien que la difficulté, pour traiter du problème de la classification des adjectifs, vient de ce que chaque critère, pris isolément, ne suffit pas à donner une définition claire de l'adjectif.

Parce qu'elle constitue le coeur de la structure théorique de la grammaire occidentale, la théorie des classes de mots a acquis très tôt une certaine autonomie et donné lieu à des problèmes qui proviennent explicitement de cette autonomisation :

- **a) lister de façon exhaustive les classes correspondant aux parties du discours,**

- **b) justifier l'ordre d'exposition des classes (opération qui correspond à des contraintes spécifiques, par exemple, celles qui tiennent à la structure logique de la définition),**

- **c) catégoriser toutes les formes d'une langue,**

- **d) trouver un classement qui permette de catégoriser toutes les formes de toutes les langues connues réelles ou possibles.**

Les parties du discours (ou classes de mots) ***"ont rempli des fonctions précises. Des fonctions qui peuvent nous apparaître maintenant extralinguistiques : quand les philosophes grecs posent la question de l'adéquation de la proposition, au sens logique du terme, avec la description du monde, ils utilisent le langage, mais sans le considérer comme la fin de leur étude. Des fonctions pratiques : par exemple, servir à l'apprentissage de l'écriture d'une langue naturelle; aider à la traduction d'une langue dans une autre... Des fonctions théoriques : permettre d'édicter des règles grammaticales générales portant sur des classes générales /./"*** **(Colombat 1988).**

clonage

Dans les langages à prototypes, l'opération de clonage permet de dupliquer les objets existants.

Clos

CLOS est un langage à objets (Keene 1989, Steele 1990).

cognition

F.J. Varela (Varela 1996) décrit le programme de recherche cognitiviste dans une approche symbolique de la manière suivante:

"Question 1 : Qu'est-ce que la cognition ?
Réponse : Le traitement de l'information : la manipulation de symboles à partir de règles.

Question 2 : Comment cela fonctionne-t-il ?
Réponse : Par n'importe quel dispositif pouvant représenter et manipuler des éléments physiques discontinus : des symboles. Le système n'interagit qu'avec la forme des symboles (leurs attributs physiques), et non leur sens.

Question 3 : Comment savoir qu'un système cognitif fonctionne de manière appropriée ?
Réponse : Quand les symboles représentent adéquatement quelque aspect du monde réel, et que le traitement de l'information aboutit à une solution efficace du problème soumis au système."

Puis dans une approche non-symbolique:

"Question 1 : Qu'est-ce que la cognition ?
Réponse : L'émergence d'états globaux dans un réseau de composants simples.

Question 2 : Comment cela fonctionne-t-il ?
Réponse : Des règles locales gèrent les opérations individuelles et des règles de changement gèrent les liens entre les éléments.

Question 3 : Comment savoir qu'un système cognitif fonctionne de manière appropriée ?
Réponse : Quand les propriétés émergentes (et la structure résultante) sont identifiables à une faculté cognitive - une solution adéquate pour une tâche donnée."

Enfin dans une approche énactive[102]:

"Question 1 : Qu'est-ce que la cognition ?

[102] Enacter est synonyme de faire émerger.

Réponse : L'action productive : l'historique du couplage structurel qui énacte (fait-émerger) un monde.

Question 2 : Comment cela fonctionne-t-il ?
Réponse : Par l'entremise d'un réseau d'éléments inter-connectés, capables de subir des changements structuraux au cours d'un historique non-interrompu.

Question 3 : Comment savoir qu'un système cognitif fonctionne de manière appropriée ?
Réponse : Quand il s'adjoint à un monde de signification préexistant, en continuel développement (comme c'est le cas des petits de toutes les espèces), ou qu'il en forme un nouveau (comme cela arrive dans l'histoire de l'évolution)."

cognitif

Se rapporte à la connaissance en général et plus précisément se dit des facultés supérieures de l'intelligence comme le langage, le raisonnement, la mémoire, l'attention...

computation

Ce terme est traditionnellement employé à partir du sens anglo-saxon de traitement par ordinateur.
"Le traitement computationnel est une opération qui est effectuée sur des symboles, c'est-à-dire sur des éléments qui représentent ce à quoi ils correspondent. La notion en jeu ici est la représentation, ou l'intentionnalité, terme du philosophe pour la qualité de ce qui est 'à propos de quelque chose" **(Varela 1996).**
Pas de computation sans représentation : ***"L'hypothèse cognitiviste prétend que la seule façon de rendre compte de l'intelligence et de l'intentionnalité est de postuler que la cognition consiste à agir sur la base de représentations qui ont une réalité physique sous forme de code symbolique dans un cerveau ou une machine"*** **(Varela 1996).**

concept

"Depuis G.Frege (Frege 1893), un concept peut être vu comme une fonction définie sur un certain domaine et à valeur dans l'ensemble de valeurs de vérité {vrai,faux}. Cette fonction a pour but de discriminer les objets auxquels s'appliquent le concept..." **(Haton & al 1991). Le concept est le fruit de l'activité de l'intelligence (conception). Selon Kayser, un concept est une** ***"entité symbolique recevant des propriétés inférentielles".*** **Selon Lyons, il** ***"exprime que toute idée, toute pensée ou toute construction mentale au moyen de laquelle l'esprit appréhende les choses ou parvient à les reconnaitre."***

connaissance

"Ce que nous appelons connaissances peut être ainsi défini comme prenant la forme de structures stabilisées dans la mémoire à long terme. Ce sont en vérité tous nos savoirs de base servant à la reconnaissance, à la compréhension des situations et de l'action. Ces savoirs vont se manifester comme représentations, c'est-à-dire des états provisoires de connaissance, résultant de nos activités de construction et d'interprétation des situations ou des évènements. Certaines de ces représentations vont être stockées en mémoire puisque pertinentes ou correspondant à des savoirs réutilisables; elles prendront, de ce fait, le statut de connaissances, et on parlera à ce propos d'acquisition ou de construction des connaissances" **(Vignaux 1991).**

contrôle

Le contrôle en IA traite de la conduite et de la supervision du raisonnement utilisé pour résoudre un problème donné et peut être résumer ainsi : à chaque instant du processus de résolution d'un problème donné, quelle action le système doit-il exécuter?

déclarativité

Une connaissance est déclarative quand elle est donnée sans son mode d'emploi.

délégation

La délégation est un mécanisme mis en place dans les langages à prototypes pour factoriser des comportements communs à un ensemble de prototypes au sein d'objets qui seront ainsi porteurs de ces

comportements partagés. Les prototypes qui partagent des comportements délèguent ceux-ci aux objets définis pour regrouper de tels comportements.

flexibilité

Une propriété fondamentale de l'intelligence est la notion de flexibilité. L'être humain est capable de conduire un raisonnement par constructions successives en intervenant pas à pas dans l'élaboration de ce raisonnement et si nécessaire en éliminant certaines étapes défaillantes de son raisonnement. Si les dispositifs informatiques doivent agir intelligemment ("comme un être humain le ferait"), ceux-ci doivent nécessairement être flexibles.

fonction générique

Une fonction générique en CLOS est une interface pour l'utilisation des méthodes. Elle indique le patron syntaxique commun aux diverses méthodes associées à cette fonction générique.

formalisme

(-> langage formel)

frame

Les frames (Minsky 1975) sont des entités regroupant de manière structurée l'ensemble des connaissances relatives à un objet, un concept... Un frame est composé d'un ensemble d'attributs correspondant aux diverses notions relatives à la chose ainsi représentée. Un ensemble de frames peut être organisé sous la forme d'une hiérarchie d'objets traduisant les relations entre les objets representés dans un domaine donné.

grammaire

On appelle grammaire le quadruplet G=(X,V,S,P), où X est un alphabet avec lequel sont écrites les phrases du langage engendré, V est un alphabet auxiliaire ou non terminal, S est un élément de V, appelé axiome, P est un ensemble de règles de production (-> langage formel).

grammaire stochastique

Une grammaire stochastique est définie de la manière suivante:

G = (V,X,P,S) où P est un ensemble de règles de la forme : (p : α **->** β**, p** $\in$ [0,1]),

$$\text{avec } \alpha \in (V \cup X)^* V(V \cup X)^* \text{ et } \beta \in (V \cup X)^*.$$

héritage

"Un système à héritage permet de représenter des connaissances sous la forme d'une hiérarchie d'objets. Un objet est défini par une collection de propriétés structurelles et procédurales qui décrivent son état et les opérations qu'il est capable d'exécuter. Les objets sont organisés par niveau d'abstraction et ils partagent des propriétés par l'intermédiaire du mécanisme d'héritage : un objet hérite des propriétés des objets qui sont plus généraux que lui tandis que ses propriétés sont héritées par les objets qui sont plus spécifiques que lui" **(Haton & al 1991).**

héritage et non-monotonie

Ce type de raisonnement est communément rencontré quand on utilise des hiérarchies basées sur l'héritage. Si on définit la classe des oiseaux comme une classe d'éléments qui se construit par héritage successifs, ce sont les états à chaque étape de la hiérarchie qui détermineront les propriétés des éléments constitutifs du graphe d'héritage définit pour cette classe. Comme le fait de voler est reconnu comme une situation typique de la classe des oiseaux, ce typage se répandra tout au long du graphe d'héritage (par défaut) tant que l'on ne donnera pas une information supplémentaire/contradictoire à un certain niveau de ce graphe pour modifier ce défaut. La notion d'héritage manipulée dans les langages à taxonomie de classes est en quelque sorte comparable à un mécanisme de copie virtuelle "par défaut" des informations de la sur-classe vers la sous-classe qui, elle, spécifie ses propres caractéristiques. La mise en place d'un graphe d'héritage est un affinement progressif des informations déjà représentées vers une représentation dont le "grain" est de plus en plus précis.

heuristique

Une heuristique est une méthode d'aide à la découverte.

hiérarchie

Organisation dans laquelle les éléments sont répartis dans une relation de spécialisation de sorte que chacun soit supérieure (plus général) au suivant (plus particulier).

interface

Une interface correspond au contexte informatique qui permet à l'utilisateur d'un ordinateur d'échanger des informations avec celui-ci.

langage

On appelle langage sur un alphabet X toute partie de X^* (-> langage formel).

langage de classes

Les langages de classes permettent de modéliser un domaine à l'aide d'une hiérachie de classes (<- héritage).

langage formel

Un langage formel est défini par la donnée d'un alphabet (constitué de signes) et de règles de formation des expressions (les règles de dérivation). Le tout circonscrit un ensemble de mots, i.e des séquences de signes construites suivant les règles de formation.

langage à prototypes

Un concept donné, décrivant une famille d'objets, peut être représenté par un exemplaire particulier de cette famille qui est porteur des propriétés les plus fréquemment associées aux membres de la famille d'objets considérés : cet objet particulier est un prototype. Le prototype est une manière de représenter les connaissances par défaut. Tout objet est une spécialisation d'un prototype donné et l'on créé les nouveaux objets par copie différentielle d'un prototype.

logique et monotonie

L'inférence déductive conventionnelle est une illustration de la monotonie. On peut l'illustrer par le syllogisme qui conclue que "Socrate est mortel" à partir des faits suivants : "Socrate est un homme" et "Tous les hommes sont mortels". Si on considère maintenant le fait "Les oiseaux volent" (A) , s'il n'est pas vrai que tous les oiseaux volent, le fait de voler est reconnu comme une situation typique pour un oiseau; il est donc raisonnable de penser que si l'on considère l'énoncé "Gaspar est un oiseau" (B) à la suite du précédent, on peut dire que "Gaspar peut voler" (C). Et cela est raisonnable tant qu'aucun élément contradictoire ne conduira à dire que Gaspar est un oiseau atypique de l'ensemble des oiseaux qui ne vole donc pas. La conclusion (C) est donc nuancée par l'absence d'informations sur l'atypicalité de Gaspar. Si cette information est fournie, il sera nécessaire de réviser (C). L'inférence conduite ici n'est plus monotone. C'est l'absence d'information contradictoire qui porte la marque d'un raisonnement non-monotone.

méta-connaissance

Une méta-connaissance est une connaissance portant sur des connaissances (<- connaissance).

méta-langage

Un méta-langage est un langage permettant de décrire les propriétés du domaine étudié et les traitements sur ce domaine.

méthode

Une méthode dans un langage à Objets est un protocole d'accès aux objets.

Moostrap

Langage à prototypes réflexif (Cointe & Mulet 1993).

ontologie

L'ontologie est une science qui produit des ensembles : les ontologies. Le terme 'ontologie' vient de la philosophie où il désigne "la partie de la métaphysique qui s'intéresse à l'Etre en tant qu'Etre. Mais l'ontologie est habituellement davantage comprise comme une science des étants que comme une science de l'être en tant qu'être, c'est-à-dire qu'elle s'interesse davantage à ce qui existe (les étants ou existants) qu'au principe de ce qui existe (l'Etre) (Bachimont & al. 1996).

Une ontologie est l'ensemble des objets reconnus comme existants dans le domaine visé. Une ontologie comprend tout ce à quoi il faut penser dans un domaine, tous les objets de pensée d'un domaine. La façon d'y penser ou d'en parler indique le type ontologique de l'objet (Bachimont & al. 1996).

Construire une ontologie d'un domaine, c'est donc décider quels sont les objets que l'on retient comme existants i.e. décider quels objets possèdent une consistance ontologique et lesquels n'en ont aucune (Bachimont & al. 1996).

paradigme

"Le paradigme définit la norme de ce qu'est une activité légitime à l'intérieur du domaine scientifique qu'il régit" **(Chalmers 1976).**

programme (informatique)

Succession d'instructions élémentaires destinées à être exécutées par une machine, les programmes sont décrits par des langages formels.

procédurale

Une connaissance procédurale est liée à un mode d'emploi (vs déclaratif).

prototype

"We call an object which is (in some way) typical of a group of objects a prototype" **(Bussman 1991) (-> typicalité).**

raisonnement

Un raisonnement peut être vu comme une suite d'énoncés ou de représentations symboliques conduit en fonction d'un but à atteindre. Dans un système à base de connaissances, un raisonnement peut être considéré comme une découverte des connaissances s'appuyant sur des données connues et menant à un but déterminé.

raisonnement non-monotone

En logique, si q est une conclusion valide d'un ensemble de prémisses $(p_i)_{0<i<n}$ elle le restera si l'on dispose d'un ensemble $(p_i)_{0<i<n+1<m}$. Cette propriété de raisonnement logique témoigne de ce que la logique n'exprime que des modèles trop idéalisés de situations ordinaires. En effet les $(p_i)_{n+1<i<m}$ peuvent signifier que l'on se trouve dans une situation exceptionnelle dans laquelle la conclusion n'est plus assurée. Cette situation est fréquente dans les activités habituelles dans lesquelles on utilise plus des normes que des lois infaillibles. La compréhension de textes n'est pas par exemple un processus qui partant de données (les perceptions) aboutirait à un résultat (le sens) : il s'agit beaucoup plus d'une recherche d'équilibre entre différentes contraintes dont le résultat est toujours provisoire puisque remis en question par la prise en compte de nouvelles informations. Il est clair que tous les raisonnements humains ne peuvent être traduits en utilisant la déduction "monotone" manipulée par la logique traditionnelle. Certains raisonnements prennent en compte dans leur déroulement l'absence d'informations non formulées ou manquantes par défaut et parallèlement les moyens de recours potentiel pour activer, recupérer les informations si cela s'avère nécessaire. Au contraire des logiques monotones, les logiques non-monotones prennent en compte le fait que la donnée de nouveaux phénomènes peut amener à réviser les premières conclusions construites.

réflexivité

Il y a réflexivité chaque fois qu'un système s'applique à lui-même.

Le film de John Cassavetes, intitulé Opening Night **peut être vu comme la présentation d'un moment réflexif. Outre le fait que ce film nous parle de quelques américains et qu'il fonctionne donc comme une (re)présentation de certaines connaissances sur la vie de ces quelques américains, ce film est aussi une présentation du travail des acteurs de théâtre, et leur capacité de jouer plusieurs rôles successifs et aussi de créer des masques, des artifices. Afin de décrypter le problème de la fabrication d'un spectacle, ce film utilise le théâtre tout en parlant du cinéma. La démarche suivie par ce film, faire d'un théâtre, d'une pièce, d'une actrice le sujet et le centre d'un film, constitue un moment réflexif particulier, une focalisation sur les processus de création d'univers de représentation qui se superpose à la représentation de l'espace filmique. Tout le film est un va-et-vient permanent entre les "différents niveaux de représentation mis en place dans le système" que constitue cet espace filmique. La principale qualité de ce film réside dans le fait que le film peut être "lu" (et vu) comme un unique texte d'une ambivalence assumée, volontaire, maîtrisée, valant pour les différents domaines, scènes de théâtre et hors-scène : il n'y a qu'une seule matière spatiale pour ce film structurée sur plusieurs niveaux de représentation. Le spectateur a d'ailleurs la possibilité de "voir" dans une même image ces différents niveaux : la caméra filme les scènes de tous les côtés, le résultat est que les liens complexes entre les différents univers de représentation développés dans le film (entre l'actrice et ses publics) sont accessibles dans la même image. Le film montre la capacité d'une actrice à créer de multiples figures. Myrthe Gordon (l'actrice) ne confond pas la réalité (celle qui est liée au personnage du film) avec le rôle qu'elle interprète : même si tout ce qu'elle dit hors-scène, et de plus en plus au fur et à mesure que le film avance, peut être rapporté à la pièce qu'elle interprète et réciproquement. Le film raconte comment une actrice parvient à traverser un couloir et à passer la porte : la porte qui sépare la scène des coulisses.**

réification

La réification permet de définir informatiquement une chose quelconque sous forme d'un objet : passage d'une chose, d'un évènement vers une entité qui représente cette chose ou cet évènement.

sens conceptuel

Un concept prescrit les sens des termes en déterminant le contenu sémantique qu'ils doivent véhiculer et les relations conceptuelles qu'ils entretiennent avec les autres termes du discours. Un concept est de nature prescriptive dans la mesure où il renvoie à un contenu réel i.e. objectif : les objets du domaine visé (Bachimont & al. 1996), (Bachimont 1995).

sens linguistique

Un signifié est une entité descriptive. Il rend compte de la signification d'un morphème, décrit ses rapports sémantiques aux autres morphèmes du discours, mais il ne détermine pas selon des règles quels rapports sont obligatoires et lesquels sont interdits. L'énoncé "boire comme un trou" possède un sens linguistique mais viole les prescriptions conceptuelles des termes (seul un animé peut boire et seul un objet liquide peut être bu). Un énoncé possèdant un contenu conceptuel (i.e exprimant une connaissance) peut commander une action. L'énoncé "la sténose est serrée" prescrit qu'il faut dilater l'artère pour restaurer la circulation. Le contenu linguistique ne prescrit aucune action à entreprendre, il décrit le sens et ne prétend pas renvoyer au monde objectif (Bachimont & al. 1996), (Bachimont 1995).

Self

Self est un langages à prototypes (Self Group 1987, 1995).

script

Un script (Schank & Abelson 1977) est une suite standard d'actions, supposée familière, et à laquelle on peut se référer sans avoir besoin de l'expliciter.

système symbolique

Définition mathématique pour donner sens à des expressions sans se référer à des conventions arbitraires. les systèmes symboliques partent d'axiomes pour construire des théorèmes à l'aide de règle de dérivation. Les axiomes et le règles de dérivation sont exprimés dans un langage formel parfaitement défini. On peut ainsi envisager une programmation informatique des systèmes

symboliques.

typicalité

"Il ressort de nombreuses expériences que les catégories semblent codées dans l'esprit non pas sous forme de listes énumérant les individus membres de la catégorie, pas davantage sous forme de listes énumérant les critères nécessaires et suffisants qui détermineraient l'appartenance à cette catégorie, mais plutôt sous la forme d'un prototype représentant le membre type de la catégorie considérée. Le codage le plus économique, sur le plan cognitif, pour représenter une catégorie, est bel et bien une image concrète du membre moyen de cette catégorie" **(Rosch 1977).**

L'approche prototypique ainsi présentée permet de prendre en compte les propriéts typiques des domaines représentés : l'énoncé "Les oiseaux volent" contient l'idée communément partagée que le fait de voler est une aptitude associée aux oiseaux, i.e. on peut dire que typiquement, un oiseau vole, bien que certains oiseaux ne volent pas. L'approche prototypique est donc une représentation par défaut qui prend en compte des propriétés partagées par l'ensemble des individus d'un groupe donné à moins de déclarer explicitement qu'un individu de ce groupe ne partage pas cette propriété. Une autruche ne vole pas, elle est atypique dans la famille des oiseaux pour cette propriété.

Références bibliographiques

Références bibliographiques

Abréviations employées:
AAAI : American Association for Artificial Intelligence
ACL : Association for Computational Linguistics
COLING : Computational Linguistics
CSLI : Center for the Study of Language and Information, Chicago Press
DFKI : Deutsches Forschungszentrum für Künstliche Intelligenz Gmbh
DRLAV : Revue de Linguistique, CNRS et Université Paris VIII
EACL : European Conference of the Association for Computational Linguistics
ECAI : European Conference on Artificial Intelligence
ECOOP : European Conference on Object-Oriented Programming
IJCAI : International Joint Conference on Artificial Intelligence
JOOP : Journal of Object-Oriented Programming
LINX : Revue de Linguistique, Université de Paris X Nanterre
MOTS : Revue publiée par les Presses de la Fondation Nationale des Sciences Politiques
OOPSLA : Object-Oriented Programming : Systems, Languages and Applications
TOOLS : Technology of Object-Oriented Programming and Systems
TSI : Technique et Science Informatiques, AFCET-Bordas
TAL : Traitement Automatique des Langues, revue de l'ATALA, Association pour le Traitement Automatique des LAngues.

Abeillé Anne
1989
"L'unification dans une grammaire d'arbres adjoints : quelques exemples en syntaxe française", in T.A informations, revue internationale du traitement automatique du langage, volume 30, numéro 1-2, Librairie Klincksieck.

Agesen Ole
1996
"Concrete Type Inference : Delivering Object-Oriented Applications", Thèse de Doctorat, Stanford University.

Aho Alfred
1968
"Indexed grammars : an extension of Context-Free grammars", Journal of the ACM, vol.ume 15, numéro 4.

Aho Alfred V., Ravi Sethi, Jeffrey D.Ullman
1986
Compilers : principles, techniques and tools, Addison-Wesley.

Andler D.
1992
Introduction aux sciences cognitives, Gallimard, Point Seuil.

Assadi H., Bourigault D.
1995
"Classifications d'adjectifs extraits d'un corpus pour l'aide à la modélisation des connaissances", Actes Journées de l'Analyse des Données Textuelles.

Auroux S.
1988
"La grammaire générale et les fondements philosophiques des classements des mots", in Langages numéro 92, Larousse.

Authiez-Revuz
1992
"Les non-coïncidences du dire et leur représentation méta-énonciative. Etude linguistique et discursive de la modalisation autonymique". Thèse de doctorat ès Lettres.

Authiez-Revuz
1994
"L'énonciateur glosateur de ses mots : explicitation et interprétation" in Langage n° 103 *Le lexique : construire l'interprétation*, Larousse.

Bachimont Bruno
1992
Le contrôle dans les systèmes à base de connaissances, Hermès.
1995
"Ontologie régionale et terminologie : quelques remarques méthodologiques et critiques" in (Otman 1995)

Bachimont Bruno, Jacques Bouaud, Jean Charlet, Pierre Zweigenbaum
1996
"Ontologie et réutilisabilité : expérience et discussion", Acquisition et Ingénierie des connaissances, Toulouse 1996, Céjaduès éditions.

Bentch Peter A., Walter J. Savitch
1995
"An occurrence-based model of word categorization" in Annals of Mathematics and Artificial Intelligence 14(1995) 1-16, J.C Baltzer AG, Science Publishers.

Berlin B.
1978
"Ethnobiological Classification", in Cognition and Categorization, E.Rosch et B.Lloyd (editors), Hillsdale (N-J) : L.Erlbaum.

Berlin B., Kay P.
1969
"Basic Colors Terms:Their Universality and Evolution", University of California Press, Berkeley.

Biber Douglas
1993

"Using register-diversified corpora for general language studies", Computational Linguistics, volume 19, numéro 2, p. 219-241.

Biber Douglas, Edward Finegan
1994
"Intra-textual variation within medical research articles" in Corpus-based Research into Language, W.Oostdisk & P. de Haan editors, Rodopi, Amsterdam.

Bobrow Daniel G.
1990
"Object Invoked Computational Reflection in CLOS", (Briot & al. 90).

Bobrow Daniel G., Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales and David A. Moon
1988
Common Lisp Object System Specification : programmer interface concepts and functions in the programmers interface, X3J13 Document 88-002R, June [également Lisp and Symbolic Computation, V. 1, #3/4, January 1989 et ACM SIGPLAN Notices, V23 Special Issue, September 1988].

Bobrow Daniel G., Kenneth Kahn, Gregor Kickzales, Larry Masinter, Mark Stefik, and Frank Zdybel
1986
"CommonLoops : merging Lisp and Object-Oriented Programming", OOPSLA'86, p.17-29.

Bobrow Daniel G., Gregor Kiczales
1988a
"The Common Lisp Object System Metaobject Kernel : a status report", First International Workshop on Lisp Evolution and standardization, 22-23 février 1988, AFNOR, INRIA, LITP, AFCET.
1988b
"The importance of being Meta", First CLOS users and implementors workshop, Palo Alto.
1989
"Programming and metaprogramming in the Common Lisp Object System", Tutorial, TOOLS'89.

Bod Rens
1993
"The Problem of Computing the Most Probable Tree in Data-Oriented Parsing and Stochatisc Tree Grammars", ACL'93.
1995
"Enriching Linguistics with Statistics : Performance Models of Natural Language", ILLC Dissertation Series 1995-14, Department of Computational Linguistics, Universiteit van Amsterdam.

Boguraev Branamir
1988
"A natural language toolkit reconciling theory with practice", in (Reyle & Rohrer 1988, p.95-130).

Boguraev Bran, John Carroll, Ted Briscoe, Claire Grover
1988
"Software Support for Practical Grammar Development", COLING'88, p.54-58.

Bolc Leonard (Editor)
1983
The design of interpreters, compilers, and editors for Augmented Transition Networks, Springer-Verlag.
1987
Natural language parsing systems, Springer-Verlag.

Bond Alan H., Les Gasser (Editors)
1988
Readings in Distributed Artificial Intelligence, Morgan Kaufmann.

Bouchard Lorne, Emirkanian Louisette, Estival Dominique ,Fay-Varnier Christine, Fouquere Christophe, Prigent Gilles, Zweigenbaum Pierre
1992
"EGL : a french linguistic development environment", Actes 12èmes *Journées Internationales Intelligence Artificielle, Systèmes Experts, Langue Naturelle,* p. 229-238, Avignon.

Bourigault D.
1993
"An endogeneous Corpus-Based Method for Structural Noun Phrase Disambiguation", in Proceedings, European Chapter of The ACL, EACL'93.

M.Bras, D. Coulon, J.P Desclés, C. Fuchs, F. Gayral, J. Jayez, D. Kayser, F. Nef, D. Reppert, P. Saint Dizier, C. Tollu, Y. Toussaint, B. Victorri
1990
"La sémantique des langues naturelles. Eléments d'une approche comparative" in Actes PRC GDR IA, Bernadette Bouchon-Meunier éditeur, Hermès.

Bresnan Joan (Editor)
1982
The Mental Representation of Grammatical Relations, MIT Press.

Briot Jean-Pierre, Foote Brian, Kickzales Gregor, Ibrahim Mamdouh, Matsuoka Satoshi, Wanatabe Takuo (Editors)
1990
Actes Reflection and metalevel architectures in Object-Oriented Programming, ECOOP (European Conference on Object Oriented Programming) / OOPSLA (Object-Oriented Programming : Systems, Languages and Applications) Workshop, Ottawa.

Bronckart Jean-Paul (avec la collaboration de D. Bain, B. Schneuwly, C. Davaud et A. Pasquier)
1985
Le fonctionnement des discours : un modèle psychologique et une méthode d'analyse, Delachaux et Niestlé.

Bussman Stefan
1991
Prototypical Concept Formation : An alternative Approach to knowledge Representation, Technical Memo (TM 91-15), DFKI.

Cadiot Pierre
1992
"à entre deux noms : vers la composition nominale", Lexique 11, P.U.L.

Cadiot Pierre, François Nemo
1997a
"Propriétés extrinsèques en sémantique lexicale", in Journal of French Language Studies 7, Cambridge University Press.
1997b
"Pour une sémiogénèse du nom", in Langue Française n°113, Mars 1997.

Carroll John, Ted Briscoe, Claire Grover
1991
"A Development Environment for Large Natural Language Grammars", University of Cambridge Computer Laboratory, Technical Report 233.

Chalmers Alan F.
1976
"What is this thing called Science? An assessment of the nature and status of Science and its methods", University Of Queensland, St Lucia.

Chambers Greg
1992
"The Design and Implementation of The Self Compiler, an Optimizing Compiler for Object-Oriented Programing Language", Ph.D. Thesis, Computer Science Department, Stanford University, April 1992.

Church K.W., R.L. Mercer
1993
"Introduction to the special issue on computational linguistics using large corpora." Computational Linguistics, vol. 19, n°1, p. 1-24.

Cointe Pierre, Philippe Mulet
1993
"Définition d'un noyau réflexif pour un langage à prototypes", in RPO *"Représentations par objets"*, (Le point sur sa recherche et ses applications), La Grande Motte 17-18 Juin 1993.

Cointe Pierre, Philippe Mulet, Jacques Malenfant
1995
"Towards a methodology for Explicit Composition of Meta-Objects" in OOPSLA'95.

Cointe Pierre, J. Malenfant, C. Dony, Philippe Mulet
1991
"Etude de la réflexion en Self, un langage à prototypes", LITP

Cointe Pierre, Graube Nicolas
1988
"Programming with metaclasses in CLOS", Actes "*First CLOS users and implementors workshop*", Palo Alto.

Coleman L.
1975
"The Case of the vanishing presupposition", Proceedings of the first annual meetings of the Berkeley linguistic society.

Coleman L., Kay P.
1981
"Prototype semantics : the english word Lie", Language 57, p. 26-44.

Colombat B.
1988
"Elements de réflexion pour une histoire des parties du discours", in Langages numéro 92, Larousse.

Conein B.
1997
"Descriptions multiples et catégorisations : le journal dans tous ses états", in Langue Française n°113, Mars 1997.

Copestake Ann, Ted Briscoe
1993
"Semi-productive polysemy and sense extension".

Cori Marcel, Marandin Jean-Marie
1993
"Grammaires d'arbres polychromes", TAL volume 34, numéro 1.

Corbin Danielle
1990
"Homonymie structurelle et définition des mots construits (Vers un dictionnaire dérivationnel)", Centre d'étude du lexique *"La definition"* , Larousse 1990.

Corbin Danielle, Georgette Dal, Agnès Mélis-Puchulu, Martine Temple
1993
"D'où viennent les sens a priori figurés de mots construits? Variations sur lunette(s), ébéniste et les adjectifs en -esque. " Verbum (Rhétorique et science du langage) n°1-2-3

Dachelet Roland
1994
"Sur la notion de sous-langage", Thèse de Doctorat de l'Université Paris VIII.

Daladier A.
1990
"Aspects constitutifs des grammaires de Harris", Langage 99, p. 57-84

Danlos Laurence
1985
Génération automatique de textes en langues naturelles, Masson.

Dejong G.
1979
"Prediction and substantiation : a new approach to Natural Language Processing", in Cognitive Science, volume 15.

Desclés, Jean-Pierre
1973
"Linguistique et formalisation", in Le langage, RETZ, p.304-323.
1982
"Quelques réflexions sur les rapports entre linguistique et mathématiques", in Penser les mathématiques, F. Guenard et G. Lelièvre responsables, Seuil, p.88-107.

Des Rivières Jim, Gregor Kickzales, G. Bobrow
1990
The Art Of Meta-Object Protocol, Xerox Corporation.

Dubois Danièle
1991
Sémantique et Cognition (Ouvrage collectif), Editions du CNRS.

Ducournau Roland, Michel Habib, Marianne Huchard, Marie-Laure Mugnier, Amadéo Napoli
1995
"Le point sur l'héritage multiple", Technique et science informatiques, volume 14, numéro 3, p. 309 - 345.

Duvall Robert
1995
"The Self Environment : a Tutorial", CS196b : Object-Oriented Programming Languages for Computer Science Education, Brown University.

Earley Jay
1986
"An efficient context-free parsing algorithm", in *Readings in Natural Language Processing*, B. Grosz, K. S. Jones, B. Weber editors, Morgan Kauman Publishers.

Fabre Cécile
1996
"Interpretation of Nominal Compounds : Combining Domain-Independent and Domain-Specific Information", Actes de COLING'96, Copenhague, Danemark.

Fauconnier G.
1992
"Galilée, les étoiles et la grande syntaxe" in L. Tasmowski & A.Zribi-Hertz editeurs, *De la musique à la linguistique. Hommages à Nicolas Ruwet*, Communication & Cognition, Ghent.

Fiala P.
1994
"L'interprétation en lexicométrie. Une approche quantitative des données lexicales", in Langage n°103 Le lexique : construire l'interprétation, Larousse.

Fillmore C.J.
1971
"Types of lexical information", in Semantics; an interdisciplinary reader in philosophy, linguistic and psychology, D.D Steinberg et L.A Jakobovits (editors), Cambridge University Press.
1977
"Topics in lexical semantics", in Current issues in linguistics theory, R. Cole (editor), Indiana University Press.
1978
"The case for Case", in Universal in linguistic theory, H. Bach et R.T. Harms (editors), Holt Rinehart and Winston.
1982a
"Towards a descriptive framework for spatial dexis", in Speech, place and Action, R.J. Jarvella, W. Klein (editors), Chichester.
1982b
"Frame semantics", in Linguistics in the morning calm, Linguistic Society of Korea.

Fleury Serge
1992
"GASPAR : General Analysis about Syntactic Parsing And Reflexion", Mémoire de DEA, Université de Paris VII.

Fleury Serge, Molia André, J.J Peyra,
1992
DINOSOR : Development IN Olmes Sweet Olmes for Reflection. Projet de séminaire consacré à la construction d'analyseurs syntaxiques, E.N.S Fontenay.

Fodor J.A
1986
La modularité de l'esprit, Minuit.

Fong Sandiway, Robert C. Berwick
1991
"The Computational Implementation of Principle-Based Parsers", in Current Issues in Parsing Technology, Masaru Tomita (editor), Kluwer Academic Publishers, p.9-25.

Fradin Bernard
1993
RS des léxèmes complexes

Franckel Jean-Jacques, Daniel Lebaud
1992
"Lexique et opérations, le lit de l'arbitraire" in *La théorie d'Antoine Culioli, ouvertures et incidences*, Ophrys.

Frege G.
1893
Ecrits Logiques et philosophiques, Seuil 1971.

Fuchs Catherine (éditeur)
1988
"L'ambiguïté et la paraphrase. Opérations linguistiques, processus cognitifs, traitements automatisés", centre de publications de l'université de Caen.

Fuchs Catherine
1994
"The challenge of continuity for a linguistic approach to semantics" in (Fuchs & Victorri 1994).

Fuchs C., Le Goffic
1985
"Ambiguïté, Paraphrase et Interprétation", *Modèles Linguistiques,* VII, 2, Lille, Presses Universitaires.

Fuchs Catherine, Victorri Bernard, Gosselin Laurent
1991
"Polysémie, glissements de sens et calcul des types de procès" in Les typologie des procès,Klinckieck.

Fuchs & Victorri (editeurs)
1994
Continuity in Linguistics Semantics, Linguisticae Investigations (Supplementa), Volume 19, John Benjamins Publishing Company, Amsterdam/Philadelphia.

Fuchs Catherine, Victorri Bernard
1990
"Construire un espace sémantique pour représenter la polysémie d'un marqueur grammatical : l'exemple de encore", Linguistica Investigationes 16.

Garde P.
1983
"Présupposés linguistiques de la théorie des parties du discours" Travaux I, 1-8 Aix.

Garfield J.L.

"Modularity in Knowledge Representation and Natural Language Understanding", MIT Press.

Gazdar, Gerald
1985
"NLs, CFLs and CF-PSGs", in (Jones, Wilks 1985, p.81-93).
1987
"Linguistic Application of Default Inherence Mechanisms" in Linguistic Theory And Computer Applications, Academic Press.
1988
"Applicability of indexed grammars to natural languages", in (Reyle & Rohrer 1988, p.69-94).
1990
"An introduction to DATR", in (Evans & Gazdar 1990, p.1-14).

Gazdar Gerald, Ewan Klein, Geoffrey Pullum, Ivan Sag
1985
Generalized Phrase Structure Grammar, Harvard University Press.

Gazdar, Gerald and Chris Mellish
1989
Natural language processing in Lisp, Addison-Wesley.

Gazdar Gerald, Geoffrey K. Pullum
1987
"Computationally relevant properties of natural languages and their grammars", (Savitch & al. 1987, p.387-438).

Gazdar, Gerald, Geoffrey K. Pullum, Robert Carpenter, Ewan Klein, Thomas E. Hukari, Robert D. Levine
1987

"Category structures", Report CSLI-87-102, (Repris dans Computational Linguistics, volume 14, number 1, Winter 1988, p. 1-19, traduction française, "Les structures de catégories", (Miller & Torris 1990, p. 245-301))

Godard Danièle, Jacques Jayez
1994
"Types Nominaux et Anaphores : le cas des objets et des événements", Anvers.
1995
"Principles as lexicals methods" in *Proceedings of the AAAI workshop on representation and acquisition of lexical knowledge*, March 27-29, Stanford University.

Graube Nicolas
1989
"Architectures réflexives et implémentations des langages à taxonomie de classes en Lisp : applications à ObjVlisp, Common Lisp Object System et TELOS", thèse de doctorat, Université Paris 6.

Grefenstette Gregory
1993
"Automatic Thesaurus Generation from Raw Text using Knowledge-Poor Techniques", 9th Annual Conference of the university of Waterloo, Centre for the New Oxford English Dictionary and Text Research, Septembre 1993, Oxford.
1994
"Corpus-Derived First, Second and Third-Order Word Affinities", EURALEX'94, Amsterdam 1994.

Grishman Ralph, Richard Kittredge (Editors)
1986
Analyzing language in restricted domains : sublanguage description and processing, Lawrence Erlbaum Associates.

Gross Gaston
1994
"Classes d'objets et description des verbes" in Langage n°115 *"Sélection et sémantique"* numéro dirigé par Jacqueline Giry-Schneider, Septembre 1994, Larousse.

Habert Benoît
1991a
"Olmes, a versatile and extensible parser in CLOS", in TOOLS'91.
1991b
"Knowledge representations for Parallel Parsing Strategies", in Proceedings Computer Architecture for Machine Perception 91
1991c
"Spécialiser les règles syntaxiques", RFIA 91 Lyon France.
1995
"Introduction", in TAL, volume 36, numéro1-2,
1996
Objectif : CLOS, Guide d'autoformation en 10 séquences, MASSON Collection Objectif.

Habert Benoit , Didier Bourigault
1997
"A Frontier to Root Transducer for the Evaluation of Two Noun Phrase Extractors", Journées Scientifiques et Techniques 1997.

Habert Benoit , Philippe Barbaud, Fernande Dupuis, Christian Jacquemin
1995
***"Simplifier des arbres d'analyse pour dégager les comportements syntactico-sémantiques des formes d'un corpus"* in Cahiers de Grammaire, Université de Toulouse**

Habert Benoît, Christian Jacquemin
1993
"Noms composés, termes, dénominations complexes : problématiques linguistiques et traitements automatiques" , TAL, volume 34, numéro 2.

Habert Benoit, Pierre Cadiot
1997
"Au sources de la polysémie nominale", in Langue Française n°113, Mars 1997.

Habert Benoit, Cécile Fabre
1993
"Approche hiérarchique de la construction du sens de dénominations complexes de la forme N1 à N2", Actes du colloque "Informatique et Langage Naturel", Nantes.
(à paraître)
"Symplifying Nominal Parse Trees to Find Semantic Types in Corpus" in Proceedings ALLC-ACH.

Habert Benoît, Serge Fleury
1993a
"Vers des analyseurs réflexifs", TAL, volume 34, numéro 1.
1993b
"Suivi fin de l'analyse automatique du langage naturel basé sur l'héritage et la discrimination multiples", in RPO *"Représentations par objets"*, (Le point sur sa recherche et ses applications), La Grande Motte 17-18 Juin 1993.

Habert Benoît, Helka Folch
1996
"Quasi-Trees : a logical formalism to relate texts and certified terms in structured indexing", ELI, ENS de Fontenay-St Cloud.

Habert Benoît, Pierre Fiala
1989
"La langue de bois en éclat : les défigements dans les titres de la presse quotidienne", MOTS 21, Presses de la Fondation Nationale des Sciences Politiques.

Habert Benoît, Adeline Nazarenko
1996
"La syntaxe comme marche-pied de l'acquisition de connaissances : bilan critique d'une expérience", Journées d'Acquisition de Connaissances 1996.

Habert Benoît, Eric Gaussier
1997
"Langue spécialisée : des séquences observées aux noms possibles" in D.Corbin et al. Actes *Mots possibles et mots existants*, Lille 1997.

Habert Benoît, Elie Naulleau, Adeline Nazarenko
1996
"Symbolic word classification for medium size corpora", Coling'96.

Harris Zellig
1968
"Analyse du discours", Languages n°13, Larousse.
1970
"La structure distributionnelle", Langages n°20, Larousse.
1988
"Language and Information", Columbia University Press, New York, 1988.

Haton Jean Paul, Nadjet Bouzid, François Charpillet, Marie Christine Haton, Brigitte Lâasri, Hassan Lâasri, Pierre Maquis, Thierry Mondot, Amadeo Napoli.
1991

Le Raisonnement en Intelligence Artificielle : Modèles, Techniques et Architectures pour les sytèmes à base de connaissances, InterEditions.

Hofstadter Douglas
1985
Gödel, Escher, Bach : les brins d'une guirlande éternelle, InterEditions.

Hopcroft John E., Jeffrey D. Ullmann
1979
Introduction to Automata Theory, Language and Computation, Addison-Wesley.

Johnson Mark
1991
"Features and Formulae", Computational Linguistics.

Joshi K.
1987
"Word Order Variation in Natural Language Generation", AAAI'87.

Justeson John S. , Slava M. Katz
1995
"Principled Disambiguation : Discriminating Adjective Senses with Modified Nouns" in ACL'95,Volume 21, numero 1.

Kaplan Ronald, Joan Bresnan
1982
"Lexical-Functional-Grammar : a Formal System for Grammatical representation", in The mental Representation of Grammatical Relations (Chap. 4), Joan Bresnan Editor, The MIT Press.

Kameyama Megumi
1988
"Atomization in Grammar Sharing", ACL'88, p.194-203.

Karttunen Lauri
1989
"Radical Lexicalism", in Alternative Conceptions of Phrase Structure, Mark R. Baltin & Anthony S. Kroch editors, Chicago Press, p.41-65.

Karttunen Lauri, Martin Kay
1985
"Parsing in a free word order language", Natural language parsing : psychological, computational and theoretical perspectives, David R Dowty, Lauri Karttunen, Arnold M Zwicky (editors), Cambridge University Press, p.279-306.

Kay Martin
1980
"Algorithm schemata and data structures in syntactic processing", in Readings in Natural language processing (Barbara J. Grosz, Karen Sparck Jones and Bonnie Lynn Webber, eds.), p.35-70, Morgan Kaufman : Los Altos, 1986.
1985a
"When meta-rules are not meta-rules", in (Jones & Wilks 1985 p.94-116).
1985b
"Unification in grammar", in Natural language understanding and logic programming (Veronica Dahl and Patrick Saint-Dizier, eds) p.233-240, North-Holland : Amsterdam.
1985c
"Parsing in functional unification grammar", In Natural language parsing (David R. Dowty, Lauri Karttunen and Arnold M. Zwicky, eds), p.251-278, Cambridge University Press, Cambridge.

Kayser Daniel
1987

"Une sémantique qui n'a pas de sens", (Rastier 87a, p.33-46).
1988a
"What kind of thing is a concept", Computational Intelligence, 4, p. 158-165.
1988b
"Ambiguïté, paraphrase et profondeur variable", (Fuchs 1988, P.237-249).
1989a
"Réponse à Kleiber et Riegel" in Linguisticae Investigationes XIII:2, p. 419-422, John Benjamins B.V. Publishing Company, Amsterdam.
1989b
"La sémantique des langues naturelles et les logiques", Annales des Télécommunications, 44, n° 1-2, p. 34-52.
1990
"Adéquation et inadéquation de la logique au traitement sémantique des langues", Modèles Linguistiques, tome XII, fasc. 1, p.119-136.
1992
"Profondeur variable et sciences cognitives", in Andler Daniel (resp.), Introduction aux sciences cognitives, Gallimard, Folio Essais, p. 195-218.
1994
"What kind of models do we need for the simulation of understanding" in (Fuchs & Victorri 1994)
1995
"Texte et dénotation" in (Otman 1995)
1997
"La sémantique lexicale est d'abord inférentielle", in Langue Française n°113, Mars 1997.

Keene Sonya E.
1989
Object-Oriented programming in Common Lisp, a programmer's guide to CLOS, Addison-Wesley.

Kiczales Gregor
1990
"Making Reflection Safe for Real-World Users"
In OOPSLA/ECOOP'90 Workshop, *"Reflection and Metalevel Architectures in Object-Oriented Programming"*.

Kiczales Gregor, Daniel G. Bobrow
1990
"CLOS specification : 3 Metaobject Protocol", Xerox.

Kiczales Gregor, Jim des Rivières, Daniel G. Bobrow
1991
The Art of the MetaObject Protocol, The MIT Press.

Kiczales Gregor, Luis Rodriguez
1990
"Efficient method dispatch in PCL", Proceedings of the 1990 ACM conference on Lisp and Functional Programming, p.99-105.

Kiczales Gregor, John Lamping
1992
"Issues in the Design and Specifications of Class Libraries", OOPSLA'92, p.435-451.

Kleiber Georges
1990
"La sémantique du prototype", PUF.
1994
"Contexte, interprétation et mémoire : approche standard vs approche cognitive" in Langage n° 103 *Le lexique : construire l'interprétation*, Larousse.

Kleiber Martin, Georges Riegel
1989
"Une sémantique qui n'a pas de sens n'a vraiment pas de sens" in Linguisticae Investigationes XIII:2, p. 405-417, John Benjamins B.V. Publishing Company, Amsterdam.
1991
"Sens lexical et interprétations référentielles. Un écho à la réponse de D. Kayser" in Linguisticae Investigationes XV:1, p. 181-201, John Benjamins B.V. Publishing Company, Amsterdam.

Knuth Donald D.
1968
Semantics of the context free languages, Math Systems Theory 2.

Lakoff G.
1986
"Classifiers as a reflexion of Mind", in Noun Classes and Categorization, C. Craig (editor), Amsterdam : Benjamins.
1987
Women, Fire and Dangerous Things, Chicago University Press.

Laurière Jean-Louis
1987
Intelligence artificielle : résolution de problèmes par l'Homme et la machine, 3ème édition, Eyrolles.

Lehman Jill Fain
1992
Adaptative parsing : self-extending language interfaces, Kluwer Academic Publishers.

Lemaire Florence, Francois Rechenmann
1995
"Intégration de connaissances terminologiques dans les grandes bases d'objets, exemple en biologie moléculaire" in (Otman 1995).

Le Pesant Denis
1994
"Les compléments nominaux du verbe lire, une illustration de la notion de classe d'objets" in Langage n°115 *"Sélection et sémantique"* numéro dirigé par Jacqueline Giry-Schneider, Septembre 1994, Larousse.

Liberman Mark Y.
1991
"The Trend towards statistical models in Natural Language Processing", Natural Language and Speech, E. Klein & F. Veltman (editors), Springer-Verlag, p.1-7.

Liebermann Henri
1986
"Using Prototypical Objects to implement Shared Behavior in Object Oriented System", Artificial Intelligence Laboratory, MIT.
1987
"Concurrent Object Oriented Programming in Act1." In Concurrent Object Oriented Programming, Akira Yonesawa & Tokoro Editors, The MIT Press.

Maes Pattie
1987a
Computational Reflection, Technical Report 87-2. Artificial Intelligence Laboratory.
1987b
"Concepts and experiments in computational reflection", Actes OOPSLA'87, p. 147-155, Orlando.

Maes Pattie, Nardi D. (responsables)
1987

Meta-Level Architectures and Reflection, North Holland.

Maingueneau Dominique
1991
L'analyse automatique du discours - Introduction aux lectures de l'archive, Hachette supérieur.

Malenfant Jacques
1995
"On the Semantic Diversity of Delegation-Based Programming Languages" in OOPSLA'95.

Marandin Jean Marie
1992
"La perception syntaxique" in Le Gré des Langues n°4.
1993
"Analyseurs syntaxiques", TAL, volume 34, numéro 1.

Marcus Mitchell
1980
A theory of syntactic recognition for natural language, MIT Press.

Marr D.
1986
Vision, W.H Freeman and Company.

McNaught J.
1993
"User needs for textual corpora in Natural Language Processing", Literary and Linguistic Computing, volume 8, numéro 4, p. 227-234

Mel'cuk Igor
1988
Dependency Syntax, Suny, New York.

Miclet Laurent
1984
Méthodes structurelles pour la reconnaissance des formes, Eyrolles.

Milner Jean Claude
1989
Introduction à une science du langage, Le Seuil.

Mikheev Andrei, Marc Moens
1994
"Acquiring and Representing Background Knowledge for a natural Language Processing System" in AAAI'94.

Mikheev Andrei, Steven Finch
1995
"A Workbench for Acquisition of Ontological Knowledge from Natural Language" in ACL'95.

Minsky M.
1975
"A framework for representing Knowledge", in The Psychology of Computer Vision, P.H. Winstoon (editor), McGrawHill.

Molia André
1991
"Analyse des phrases interrogatives dans le formalisme LFG", rapport de maîtrise, Université de Paris VII.

Muller Hans, John Rose, James Kempf, Tayloe Stansbury
1989
"The use of multimethods and method combination in a CLOS based window interface", in OOPSLA'89, p. 239-253.

Myers Brad A., Dario A. Guise, Brad Vander Zanden
1992
"Declarative Programming in a Prototype-Instance System : Object-Oriented Programming Without Writing Methods", OOPSLA'92, p.184-200.

Nijholt Anton
1991
"Overview of Parallel Strategies", in Current Issues in Parsing Technology, Masaru Tomita Editor, Kluwer Academic Publishers.

Norvig Peter
1992
Paradigms of Artificial Intelligence Programming : cases studies in Common Lisp, Morgan Kaufman.

Numaoka Chisato, Mario Tokoro
1990
"Decentralized Parsing Method using Communication multiple Concurrent Object", in TOOLS'90.

Gabriel Otman (éditeur)
1995
La banque des mots numéro spécial 7-1995, Terminologie et Intelligence Artificielle, Texte réunis et présentés par G.Otman

Paroubek P., Y. Shabes, A. Joshi
1992
"XTAG : a grammatical Workbench for developing Tree Adjoin Grammars", 4th Conference on Applied Natural Language Processing, Trento.

Pêcheux M.
1969
Analyse Automatique du Discours, Dunod

Péry-Woodley Marie Paule
1995
"Quels corpus pour quels traitements automatiques ?", in TAL, volume 36, numéro1-2, p. 213-232.

Pitrat Jacques
1991
Méta-Connaissances, Hermès.
1992
"Intelligence Artificielle et langues naturelles", TAL, volume 33, numéro 1-2, p. 191-208.
1993
Penser autrement l'informatique, Hermès.
1995
De la machine à l'intelligence, Hermès.

Pollard Carl, Ivan A. Sag
1987
Information-based syntax and semantics, Chicago Press.

Poncet-Montange Anne
1991
"Les groupes nominaux de structure NAN et NAV", Thèse de doctorat en Linguistique, Université Paris 13.

Prince Violaine, Ruxandra Bally-Ispas
1993
"Un algorithme pour le transfert de règles pragamatiques par inférence d'analogie dans le procesus complexe GLACE, in *"Modélisation des connaissances pour le traitement automatique du langage naturel"*, Thèse d'habilitation à diriger les recherches en Sciences sous la direction de Gérad Sabah, Notes et Documents du LIMSI:93-34, LIMSI-CNRS.

Pullum Geoffrey K., Gerald Gazdar
1987
"Natural languages and context-free languages", in (Savitch & al. 1987, p. 138-182).

Pustejovsky J.
1991
"The generative lexicon" in Computational Linguistics.

Pustejovsky J., B. Boguraev
1993
"Lexical knowledge representation and natural language processing" in Artificial Intelligence 63, p193-223.

Rady M.
1983
"L'ambiguïté du langage naturel est-elle source du non-déterminisme des procédures de traitement?", Thèse de doctorat, Université de Paris VII.

Rajman Martin
1995
"Approche probabiliste de l'analyse syntaxique", in TAL, volume 36, numéro 1-2.

Rastier François
1987a
"Sémantique et intelligence artificielle", in Langages 87.
1987b
Présentation, (Rastier 87a, p.5-20).
1987c
"Représentation du contenu lexical", (Rastier 1987a, p.103-122).
1987d
Sémantique interprétative, PUF.
1991
Sémantique et recherches cognitives, PUF.
1995
"Le terme : entre ontologie et linguistique" in (Otman 1995).

Rastier F., Cavazza M., Abeille A.
1994
Sémantique pour l'analyse, Masson.

Recanati F.
1997
"La polysémie contre le fixisme", in Langue Française n°113, Mars 1997.

Reyle U., C. Rohrer (Editors)
1988
Natural language parsing and linguistic theories, D. Reidel Publishing Company.

Resnik Philip Stuart
1993
"Selection and Information : a Class-Based Approach to Lexical relationships", (Ph.D Dissertation), University of Pennsylvalia IRCS Report 93-42.

Rosch E.
1975
"Cognitive representation of semantic categories", in journal of Experimental Psychology , volume 104.
1976
"Classification d'objets du monde réel : origine et représentations dans la cognition", in Bulletin de Psychologie, S.Ehrlich, E.Tulving (Editors), numéro spécial : La mémoire sémantique.
1977
"Human Categorization", in Studies in Cross Cultural Psychology, volume 1, N.Warren (Editor), Academic Press, New York.

Rosch E., Lloyd B (Editors)
1978
Cognition and Categorization, Hillsdale.

Russell Bertrand
1969
Signification et Vérité, Flammarion.

Sabah Gérard
1988
L'intelligence artificielle et le langage : 1. Représentation des connaissances, Hermès.
1989
L'intelligence artificielle et le langage : 2. Processus de compréhension, Hermès.
1990
"CARAMEL : un système multi-experts pour le traitement automatique des langues", *Modèles Linguistiques*, tome XII, fasc. 1, p. 95-118.
1993
"Vers une conscience artificielle", in Modèles et concepts pour la science cognitive, *Hommage à J.-F. Le Ny*, Michel Denis & Gérard Sabah (eds), PUG, p. 207-222.

Sabah Gérard, M. Rady
1983
"A deterministic syntactic Parser", in Proceedings of the 8th IJCAI'83 Karlsruhe.

Sager N
1981
Natural Language Information Processing : A Computer Grammar of English and its Applicatons, Addison Wesley.
1986
"Sublanguage : linguistic Phenomenon, Computationel Tool", in *Analyzing Language in Restricted Domains : Sublanguage Description and Processing*, Grishman R., Kittredge R. editors, Lawrence Erbaum Associates, Hillsdale, New Jersey.

Savitch Walter J. et alii
1987
The formal complexity of natural language , D. Reidel Publishing Company.

Schank Roger C.
1987
Entretien avec François Rastier (Rastier 87a, p.123-128).

Schieber, Stuart M.
1986a
An introduction to unification-based approaches to grammar, Center for the Study of language and information, Chicago Press.
1986b
"A Simple Reconstruction of GPSG", Actes 12ème *COLING*, p. 211-215, Bonn.

Self Group 93 : Agesen O., Bak L., Chambers C., Chang B.W., Maloney J., B.Smith R., Ungar D., Wolczko M
1993
"How to use Self 3.0", Sun Microsystems, Inc. and Stanford University.

Self Group 95 : Agesen O., Bak L., Chambers C., Chang B.W., Hölzle U., Madsen O.L, Maloney J., B.Smith R., Ungar D., Wolczko M
1995
"Self 4.0 Read This First", Sun Microsystems, Inc. and Stanford University.

Shriver Bruce, Wegner Peter (Editors)
1987
Research directions in object-oriented programming, The MIT Press.

Shinoda Yoichi, Takuya Katayama
1990
"Object-Oriented Extension of Attribute Grammars and its implementation Using Distributed Attribute Evaluation Algorithm", in Attribute Grammars and their Applications, Pierre Deransart & Martin Jourdon Editors, Springer-Verlag.

Small Steve L.
1987
"A distributed word-based approach to parsing.", in Natural Language Parsing Systems, Leonard Bolc Editor, Springer-Verlag.

Smith Walter R.
1995
"Using a Prototype-based Language for User Interface : the Newton Project's Experience" in OOPSLA'95.

Steele Guy
1990
Common Lisp : The Language, Digital Press.

Steele Guy, Gabriel Richard
1993
"The evolution of Lisp", *ACM Sigplan Notices*, volume 28, #3, March 93, p. 231-270.

Stefanini Marie-Hélène, Alain Berrendonner, Geneviève Lallich, Flavio Oquendo
1992
"TALISMAN : un système multi-agents gouverné par des lois linguistiques pour le traitement de la langue naturelle", COLING'92, p.490-497.

Sueur Jean-Pierre
1978
"Sur la syntaxe du récit oral", Université d'Orléans, multigraphié.
1982
"Pour une grammaire du discours : élaboration d'une méthode; exemples d'application", Mots 5, p.143-184.

Taylor John, Trevor Hopkins
1995
Introduction to Object Oriented Concepts through Self, University of Manchester and Sun Micosystems Inc.

Tesnières Lucien
1959
Eléments de syntaxe structurale, Klincksieck, Paris.

Testard-Vaillant François-Xavier
1989

"CLAS : a Parallel Extension of CLOS", Proceedings of the Second CLOS users and implementors workshop.

Thompson Henry
1981
"Chart parsing and rule schemata in PSG", ACL'81 p.167-172.
1983
"MCHART : a flexible, modular chart parsing system", AAAI'83, p.408-410.

Vigay-Shanker K., A. Joshi
1992
"A feature-based Tree Adjoining Grammars", in Unification grammars, The MIT Press.

Torris Thérèse, Philip Miller
1990
Les Formalismes grammaticaux pour le TAL, Hermès.

Ungar David
1995
"How to Program Self-4.0", Sun Microsystems, Inc. and Stanford University.

Ungar David, Randall B. Smith
1987
"SELF : The power of Simplicity", in OOPSLA'87, Orlando, Florida, October 1987.
1995
"Programing as an experience : The Inspiration for SELF", Stanford University.

Ungar David, Randall B. Smith, Bay-Wei Chang
1995
"Getting Closer to Objects : Object-Focused Programing Environments", in Visual Object Oriented Programing Language, M.Burnett, A.Goldberg & T.Lewis Editors, Prentice-Hall, 1995, p. 185-198.

Ungar David, Randall B. Smith, John Maloney
1995
"The Self-4.0 User Interface : Manifesting the System-wide Vision of Concretness, Uniformity and Flexibility," in OOPSLA'95.

Ungar David,Lynn Andrea Stein, Henri Liebermann
1988
"A shared view of Sharing : The Treaty Of Orlando" , in Object Oriented Concepts, Applications and databases , Won Kim and Fred Lochovsky Editors, ACM Press.

Van der Linden E.A.
1992
"Incremental Processing and the Hierarchical Lexicon", in Computational Linguistics.

Varela Francisco J.
1996
Invitation aux sciences cognitives, Editions du Seuil.

Victorri Bernard
1994
"The use of continuity in modelling semantic phenomena" in (Fuchs & Victorri 1994).

Vignaux G.
1991
Sémantique et Cognition (Ouvrage collectif), Editions du CNRS.

Vijay-Shanker K.
1987
"A study of Tree Adjoining Grammar", PhD Thesis, University of Pennsylvania..

1992
"Using Descriptions of Trees in a Tree Adjoining Grammar", Computational Linguistics, Décembre 1992, volume 18, numéro 4.

Wegner Peter
1987
"The object-oriented classification paradigm", (Shriver & Wegner 1987, p.479-560).

Weinrich Harald
1989
Grammaire textuelle du français, Didier/Hatier .

Weizenbaum Joseph
1966
Eliza, CACM 6.

Wolczko Mario, Randall B. Smith
1995
Prototype-Based Application Construction Using Self 4.0, Sun Microsystems, Inc.

Yonesawa Akinori, Ichiro Ohsawa
1990
"Object-Oriented Parallel Parsing for Context-Free Grammars", n ABCL : an Object-Oriented Concurrent System, Akinori Yonesawa Editor, The MIT Press.

Zernik Uri (Editor)
1991
Lexical Acquisition : Exploiting On-Line Resources to build a Lexicon, Lawrence Erlbaum Associates.

Zweigenbaum Pierre
1992
LN2-3, Un Analyseur Syntaxique pour Grammaires Lexicales Fonctionnelles, in T.A informations.
1994
"MENELAS : an Access System for Medical Records using Natural Language", Computer Methods and Programs in Biomedicine, volume 45, p. 117-120.

Zweigenbaum Pierre, Bruno Bachimont, Jacques Bouaud, Jean François. Boisvieux
1995
"A Multi-Lingual Architecture for Building a Normalised Conceptual Representation from Medical Language", Reed M. Gardner (Editor), Proc. 17th Annual Symposium on Computer Applications in Medical Care.

Zweigenbaum Pierre, Jacques Bouaud, Benoît Habert, Adeline Nazarenko
1997
"Coopération apprentissage en corpus et connaissances du domaine pour la construction d'ontologies", Journées Scientifiques et Techniques, FRANCIL, Avignon, 1997.

Annexes

Annexes Partie 2 Chapitre 5

A-2-5.1. Phase0 : Filtrage Catégoriel

- **Définition des `traits` (comportements)**[103] **:**

```
traits applications _AddSlotsIfAbsent: ( | catN = () | )
traits applications _AddSlotsIfAbsent: ( | genre = () | )
traits applications _AddSlotsIfAbsent: ( | nombre = () | )
traits applications _AddSlotsIfAbsent: ( | semantique = () | )
traits applications _AddSlotsIfAbsent: ( | category = () | )
traits genre _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ printString = (
      '(',  type_genre printString, ', ',
            marque_genre printString, ')'   ).
| )
traits nombre _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ printString = (
      '(',  type_nombre printString, ', ',
            marque_nombre printString, ')'   ).
| )
traits semantique _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ printString = (
      '(',  traits printString, ', ',
            sense printString, ', ',
            classe printString, ')'  ).
| )
traits category _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ printString = (
      '(',  type printString,  ')'   ).
| )
traits catN _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
| )
```

- **Définition des prototypes :**

```
prototypes applications _AddSlotsIfAbsent : ( | catN = () | )
prototypes applications _AddSlotsIfAbsent: ( | genre = () | )
prototypes applications _AddSlotsIfAbsent: ( | nombre = () | )
prototypes applications _AddSlotsIfAbsent: ( | semantique = () | )
prototypes applications _AddSlotsIfAbsent: ( | category = () | )
genre _Define: ( |
  _ parent* = traits genre.
  ^ type_genre <- ''.
  ^ marque_genre <- ''.
| )
nombre _Define: ( |
  _ parent* = traits nombre.
  ^ type_nombre <- ''.
  ^ marque_nombre <- ''.
| )
semantique _Define: ( |
  _ parent* = traits semantique.
  ^ traits <- ''.
  ^ sense <- ''.
  ^ classe <- ''.
```

[103] Dans ce qui suit, l'écriture du code fait référence à la version 2.0 de Self.

```
| )
category _Define: ( |
  _ parent* = traits category.
  ^ type <- ''.
| )
catN _Define: ( |
  _ parent* = traits catN.
  ^ forme <- ''.
  ^ genre <- ''.
  ^ categorie <- ''.
  ^ nombre <- ''.
  ^ semantique <- ''.
| )
```

Pour créer de nouvelles entités prototypiques représentant des unités lexicales donné, on dispose des prototypes dont les attributs sont associées à des chaînes vides. Ces attributs seront valués à la création d'un prototype associé à l'élément lexical donné. Ci-dessous, le créateur des prototypes `catN` :

```
_AddSlots: (| newN = () |)
newN _Define: (|
_ parent* = traits catN.
^ genre <- ''.
^ nombre <- ''.
^ forme <- ''.
^ categorie <- ''.
^ semantique <- ''.
^ addforme: t = ( forme: t ).
^ addgenre: t = ( genre: t ).
^ addsemantique: t = ( semantique: t ).
^ addcategorie: t = ( categorie: t ).
^ addnombre: t = ( nombre: t )
|)
```

Le lexique de test comprend des éléments construits de la manière suivante :

```
_AddSlots: (| lumiere |)
lumiere: newN copy
lumiere addforme: 'lumiere'
lumiere addgenre: (genre copy type_genre:'Fem')
lumiere addnombre: (nombre copy type_nombre: 'Sing')
lumiere addcategorie: (category copy type: 'Nom')
lumiere addsemantique: (((semantique copy traits: '')classe: '') sense: 'cf Petit Robert')
_AddSlots: (| moulin |)
moulin: newN copy
moulin addforme: 'moulin'
moulin addgenre: (genre copy type_genre:'Masc')
moulin addnombre: (nombre copy type_nombre: 'Sing')
moulin addcategorie: (category copy type: 'Nom')
moulin addsemantique: (((semantique copy traits: '')classe: 'Producteur de NaN')sense: 'cf Petit Robert')
_AddSlots: (| a |)
a: newPrep copy addforme: 'a'
a addcategorie: (category copy type: 'Prep')
```

On peut aussi adopter le dispositif standard de création des prototypes, à savoir le clonage, pour multiplier les prototypes lexicaux: on se sert d'un prototype lexical existant pour en créer un autre représentant une entité proche du précédent.

```
_AddSlots: (| paroles |)
paroles: lumiere copy
paroles addforme: 'paroles'
paroles addnombre: (nombre copy type_nombre: 'Plur')
paroles addsemantique: (((semantique copy traits: '')classe:'') sense: 'cf Petit Robert')
```

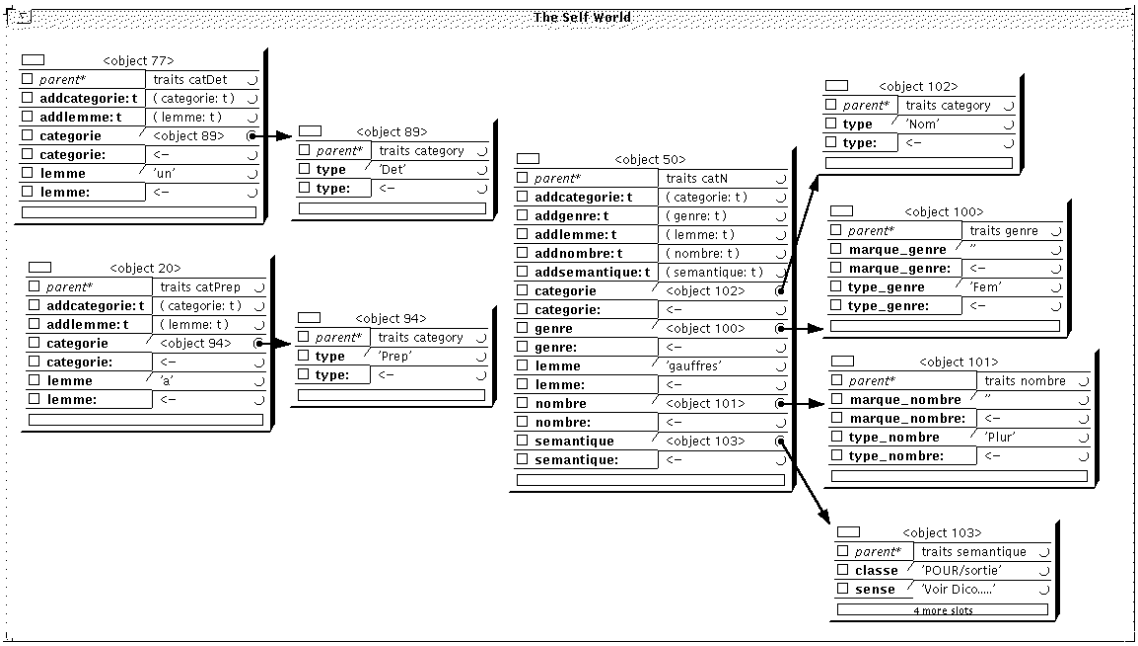


Figure a-2-5.1 : Un réseau de prototypes pour représenter le lexique.

Le code correspondant au prototype `regle` est le suivant :

```
traits applications _AddSlotsIfAbsent: ( | regle = () | )
traits regle _Define: ( |
    parent* = traits clonable.
    |)
prototypes applications _AddSlotsIfAbsent: ( | regle = () | )
regle _Define: ( |
    parent* = traits regle.
    ^ number <- ''.
    ^ position <- 0.
    ^ lhs <- vector copy.
    ^ lhsContent <- vector copy.
    ^ rhs <- vector copy.
    ^ rhsContent <- vector copy.
    ^ rhsContentForme <- vector copy.
    ^ addlhs: t = ( lhs: lhs copyAddLast: t).
    ^ addrhs: t = ( rhs: rhs copyAddLast: t).
    ^ newPosition: t = (position: (position + 1)).
    ^ addLhsContent <- ''.
    ^ addRhsContent: t = (
      rhsContent: ( rhsContent copyAddLast: t).
      ).
    ^ addRhsContentForme: t = (
      rhsContentForme: (rhsContentForme copyAddLast: t).
      addLhsContent:
      (((rhs size) _Eq: (rhsContent size))
          ifTrue: [
           (lhsContent: ((((aNaN copy
                          addN1: (rhsContent at: 0))
                          addprep: (rhsContent at: 1))
                          addN2: (rhsContent at: 2)))]).).|)
```

Le démarrage de chaque analyse initialise part les attributs du prototype associé à la règle à construire et enregistre la suite de mots à analyser.

```
regle lhsContent: (vector copy)
regle rhsContent: (vector copy)
regle rhsContentForme: (vector copy)
regle lhs: (vector copy)
regle rhs: (vector copy)
regle addrhs: 'Nom'
regle addrhs: 'Prep'
regle addrhs: 'Nom'
regle addlhs: 'NaN'
regle position: 0
textExample setInputTextExample: 'moulin'
textExample setInputTextExample: 'a'
textExample setInputTextExample: 'paroles'
inspect: textExample
```

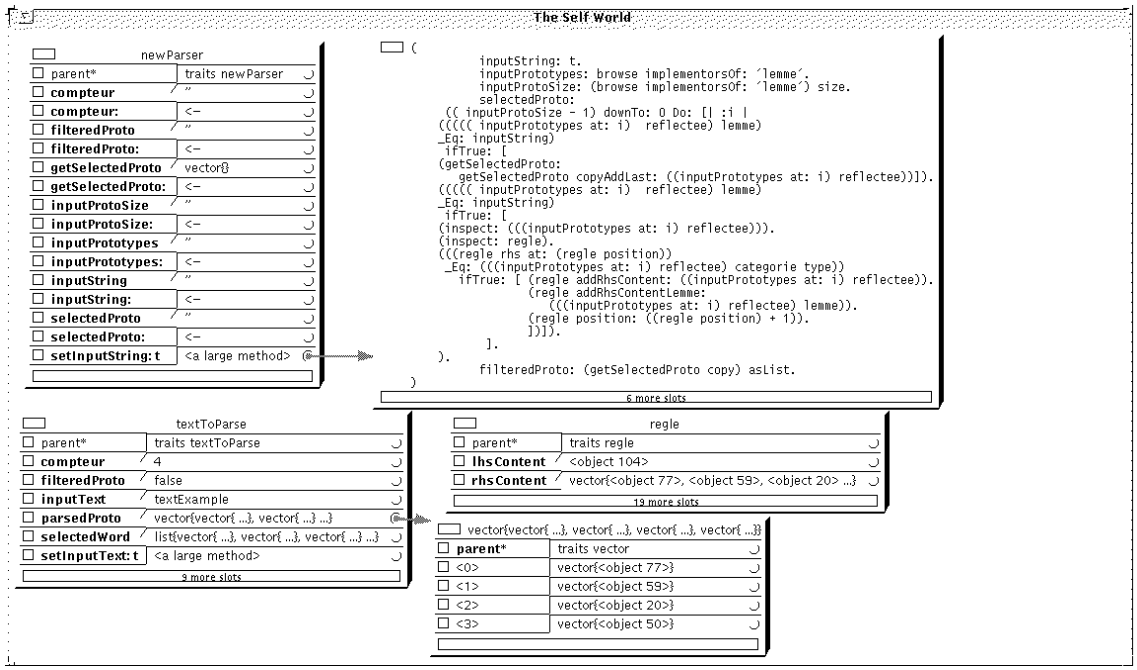


Figure a-2-5.2 : Un prototype de parseur.

L'analyse commence et se déroule de la manière suivante :

1. Lecture de la suite de mots à analyser.

```
textToParse setInputText: textExample
```

2. On prend les entrées lexicales enregistrées une par une et on recherche les prototypes lexicaux associés qui permettent de mettre à jour la règle à construire :

Description du prototype contenant la règle à construire

```
( |   parent* = <an object>.
    ^ number <- ''.
    ^ position <- 0.
    ^ lhs <- vector{'NaN'}.
    ^ lhsContent <- vector{}.
    ^ rhs <- vector{ 'Nom', 'Prep', 'Nom'}.
    ^ rhsContent <- vector{}.
    ^ rhsContentForme <- vector{}.
    ^ addlhs: t = ( lhs: lhs copyAddLast: t).
    ^ addrhs: t = ( rhs: rhs copyAddLast: t).
    ^ newPosition: t = ( position: (position + 1)).
    ^ addLhsContent <- ''.
    ^ addRhsContent: t = (
        rhsContent: ( rhsContent copyAddLast: t). ).
    ^ addRhsContentForme: t = (
        rhsContentForme: (rhsContentForme copyAddLast: t).
        addLhsContent:
        (((rhs size) _Eq: (rhsContent size))
 ifTrue:   [ (lhsContent: ((((aNaN copy addN1: (rhsContent at: 0)) addprep: (rhsContent at: 1))addN2: (rhsContent at:
2)))]).). | )
```

Modalités d'utilisation des attributs décrits ci-dessus

- L'attribut `position` donne la position dans la règle en construction;
- L'attribut `lhs` contient la catégorie du Lhs à construire;
- L'attribut `lhsContent` contiend le prototype Lhs construit (s'il est construit...);
- L'attribut `rhs` contient la description de la règle à construire;
- L'attribut `rhsContent` contiend tous les prototypes `rhs` construits au cours de l'analyse et correspondants au valeurs catégorielles contenues dans l'attribut `rhs`;
- L'attribut `rhsContentForme` donne la valeur du forme des prototypes de l'attribut précédent.

Les autres attributs permettent de construire les attributs pré-cités.

Sur l'exemple donné la règle construite a l'allure suivante :

- **Description de la règle construite**

```
( |   parent* = <an object>.
    ^ number <- ''.
    ^ position <- 3.
    ^ lhs <- vector{'NaN'}.
    ^ lhsContent <- <an object>.
    ^ rhs <- vector{ 'Nom', 'Prep', 'Nom'}.
   ^ rhsContent <- vector{ <an object>, <an object>, <an object>}.
    ^ rhsContentForme <- vector{ 'moulin', 'a', 'paroles'}.
    ^ addlhs: t = ( lhs: lhs copyAddLast: t).
    ^ addrhs: t = ( rhs: rhs copyAddLast: t).
    ^ newPosition: t = ( position: (position + 1)).
    ^ addLhsContent <- <an object>.
    ^ addRhsContent: t = (
        rhsContent: ( rhsContent copyAddLast: t).
    ).
    ^ addRhsContentForme: t = (
        rhsContentForme: (rhsContentForme copyAddLast: t).
        addLhsContent:
        (((rhs size) _Eq: (rhsContent size))
            ifTrue: [
             (lhsContent: ((((aNaN copy
                            addN1: (rhsContent at: 0))
                            addprep: (rhsContent at: 1))
                            addN2: (rhsContent at: 2)))]).). | )
```

- **Description du `Lhs` Construit**

```
( | _ parent* = <an object>.
      ^ n1 <- <an object>.
    ^ prep <- <an object>.
    ^ n2 <- <an object>.
    ^ categorie <- 'NaN'.
    ^ addN1: t = ( n1: t ).
    ^ addprep: t = ( prep: t ).
    ^ addN2: t = ( n2: t ).| )
```

- **Description de chacun des éléments du `Lhs` Construit**

1. Le prototype Nom associé à `moulin`:

```
( | _ parent* = <an object>.
    ^ genre <- <an object>.
    ^ nombre <- <an object>.
    ^ forme <- 'moulin'.
    ^ categorie <- <an object>.
    ^ semantique <- <an object>.
    ^ addforme: t = ( forme: t ).
    ^ addgenre: t = ( genre: t ).
    ^ addsemantique: t = ( semantique: t ).
    ^ addcategorie: t = ( categorie: t ).
```

2.Le prototype Préposition associé à `a`

```
( | _ parent* = <an object>.
    ^ forme <- 'a'.
    ^ categorie <- <an object>.
    ^ addcategorie: t = ( categorie: t ).
    ^ addforme: t = ( forme: t ). | )
```

3.Le prototype Nom associé à `paroles`

```
( | _ parent* = <an object>.
    ^ genre <- <an object>.
    ^ nombre <- <an object>.
    ^ forme <- 'paroles'.
    ^ categorie <- <an object>.
    ^ semantique <- <an object>.
    ^ addforme: t = ( forme: t ).
    ^ addgenre: t = ( genre: t ).
    ^ addsemantique: t = ( semantique: t ).
    ^ addcategorie: t = ( categorie: t ).
    ^ addnombre: t = ( nombre: t ). |)
```

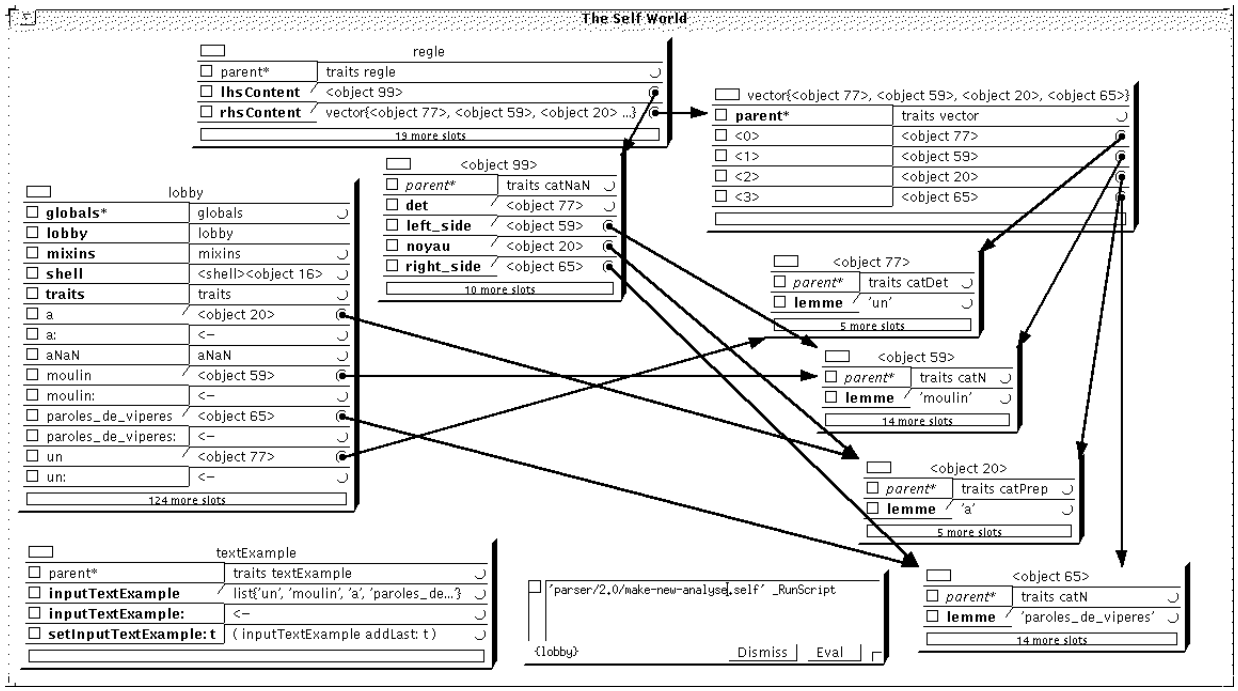


Figure a-2-5.3 : Résultat d'analyse sur "un moulin à paroles de vipères".

A-2-5.2. Phase1 : Affinement du filtrage

On donne ici le code[104] associé aux classes sémantiques.

```
traits applications _AddSlotsIfAbsent: ( | pourEntrée = () | )
traits pourEntree
 _Define: ( |
    _parent* = traits clonable.
    ^copy = (clone).
    ^printString  = ('(', classe printString, ')' ).
    |)
traits applications _AddSlotsIfAbsent: ( | pourSortie = () | )
traits pourSortie
 _Define: ( |
    _parent* = traits clonable.
    ^copy = (clone).
    ^printString  = ('(', classe printString, ')' ).
    |)
traits applications _AddSlotsIfAbsent: ( | avecEnergie = () | )
traits avecEnergie
 _Define: ( |
    _parent* = traits clonable.
    ^copy = (clone).
    ^printString  = ('(', classe printString, ')' ).
    |)
traits applications _AddSlotsIfAbsent: ( | avecPartieDe = () | )
traits avecPartieDe
 _Define: ( |
    _parent* = traits clonable.
    ^copy = (clone).
    ^printString  = ('(', classe printString, ')' ).
    |)

globals applications _AddSlotsIfAbsent: ( | pourEntree  = () | )
 pourEntree: _Define: ( |
   _parent* = traits pourEntree.
   ^classe = 'POUR/Entrée'.
    |)
globals applications _AddSlotsIfAbsent: ( | pourSortie  = () | )
 pourSortie: _Define: ( |
   _parent* = traits pourSortie
   ^classe = 'POUR/Sortie'.
    |)
globals applications _AddSlotsIfAbsent: ( | avecEnergie  = () | )
 avecEnergie: _Define: ( |
   _parent* = traits avecEnergie.
   ^classe = 'AVEC/Energie'.
    |)
```

[104] Le code présenté ici fait référence à la version 3.0 de Self.

```
globals applications _AddSlotsIfAbsent: ( | avecPartieDe  = () | )
avecPartieDe: _Define: ( |
   _parent* = trait avecPartieDe.
   ^classe = 'AVEC/Partie-de'.
    |)
```

On a défini les objets `traits` et les prototypes associés aux catégories définissant les formes complexes `N1` à `N2` de la manière suivante :

```
traits applications _AddSlotsIfAbsent: ( | catN1aN2 = () | )
traits catN1aN2
 _Define: ( |
    _parent* = traits clonable.
    ^copy = (clone).
   ^printString  = (
          '(Classe: ',  (semantiqueClasse classe ) printString,  ', Categorie: ',
                            categorie printString, ',  LhsN1àN2: ',
                            (n1 forme) printString, ', PrepN1àN2: ',
                            (prep forme) printString, ', RhsN1àN2: ',
                            (n2 forme) printString, ', formeN1àN2: ',
                            (forme_N1àN2) printString, ')' ).
    |)
globals applications _AddSlotsIfAbsent: ( | catN1aN2  = () | )
cat1aN2: _Define: ( |
   _parent* = traits catN1aN2.
   ^n1    <- ''.
   ^prep     <- ''.
   ^n2 <- ''.
   ^semantiqueClasse <- ''.
   ^forme_N1aN2  <- ''.
   ^addN1: t = (n1: t).
   ^addPrep: t = (prep: t).
   ^addN2: t = (n2: t).
   ^addSemantiqueClasse: t = (semantiqueClasse: t).
   ^addFormeN1aN2: t = (forme_N1aN2: t).
    |)
```

On créé les prototypes `N1` à `N2` de la manière suivante :

```
_AddSlots: ( | moulin_a_paroles  = () | )
moulin_a_paroles: catN1àN2 copy
moulin_a_paroles addN1: moulin   (ce dernier représente le prototype lexical associé à "moulin")
moulin_a_paroles addN2: paroles
moulin_a_paroles addPrep: a
moulin_a_paroles addSemantiqueClasse: (pourSortie copy)
moulin_a_paroles addForme_N1aN2: 'moulin a paroles'
```

Pour créer un nouveau prototype `N1` à `N2`, soit on répète cette opération en effectuant les assignations adéquates, soit on effectue une copie du prototype `moulin_a_paroles` et on modifie les attributs du prototype cloné :

```
_AddSlots: ( | moulin_a_gauffres  = () | )
moulin_a_gauffres: moulin_a_paroles copy
moulin_a_gauffres addN2: gauffres
moulin_a_gauffres addFormeN1aN2: 'moulin a gauffres'
```

- Prototypes lexicaux créateurs de prototypes :

```
un protoToBuild: (vector copy) asList
_AddSlots: (| sn2 |)
sn2: catSnMoulinaN2 copy
un protoToBuild addLast:sn2
leD protoToBuild: (vector copy) asList
_AddSlots: (| sn1 |)
sn1: catSn copy
_AddSlots: (| p1 |)
p1: catP copy
sn1 protoToBuild: (((vector copy) asList) addLast: p1)
leD protoToBuild addLast:sn1
moulin protoToBuild: (vector copy) asList
moulin protoToBuild addLast:  (catN1aN2 copy)
 analyseV protoToBuild: (vector copy) asList
_AddSlots: (| sv2 |)
sv2: catSVt copy
 analyseV protoToBuild addLast: sv2
```

A-2-5.3. Phase3 : Analyse par un réseau de prototypes

- Début de l'analyse:

```
Self 27> 'Phase4/4.1/parser_prototypes.new' _RunScript
reading ./Phase4/4.1/parser_prototypes.new...
```

```
manager
'Self 28>'Phase4/4.1/ph1-proto-declencheurs.new' _RunScript
 reading ./Phase4/4.1/ph1-proto-declencheurs.new...
```

Initialisation de la phrase à analyser :

```
Self 29>textExample inputTextExample removeAll
textExample setInputTextExample : 'le'
textExample setInputTextExample : 'manager'
textExample setInputTextExample : 'analyse'
textExample setInputTextExample : 'un'
textExample setInputTextExample : 'moulin'
textExample setInputTextExample : 'a'
textExample setInputTextExample : 'paroles'
list{<object 41>}
list{}
Self 30>  list{'le'}
Self 31> list{'le', 'manager'}
Self 32> list{'le', 'manager', 'analyse'}
Self 33> list{'le', 'manager', 'analyse', 'un'}
Self 34> list{'le', 'manager', 'analyse', 'un', 'moulin'}
Self 35> list{'le', 'manager', 'analyse', 'un', 'moulin', 'a'}
Self 36> list{'le', 'manager', 'analyse', 'un', 'moulin', 'a', 'paroles'}


Self 37> manager parser : (textExample inputTextExample)

Noeuds initiaux :

(0-('Clit'/'le')-1)

(0-('Det'/'le')-1)

(1-('Nom'/'manager')-2)

(2-('Vt'/'analyse')-3)

(3-('Det'/'un')-4)

(4-('Nom'/'moulin')-5)

(5-('Prep'/'a')-6)

(6-('Nom'/'paroles')-7)

(Prototype initial :
-------------------
'le'/'Clit')

(Prototype initial :
-------------------
'le'/'Det')

(Prototype(s) nouveau(x) a construire :
--------------------------------------
0-(Cat :'Sn'-NeededCat :'Nom'-Forme :'(le *)')-1)

(Prototype initial :
-------------------
'manager'/'Nom')

(Prototype initial :
-------------------
'analyse'/'Vt')

(Prototype(s) nouveau(x) a construire :
--------------------------------------
2-(Cat :'Sv'-NeededCat :'Sn'-Forme :'(analyse *)')-3)

(Prototype initial :
-------------------
'un'/'Det')

(Prototype(s) nouveau(x) a construire :
--------------------------------------
3-(Cat :'Sn'-NeededCat :'MoulinaN2'-Forme :'(un *)')-4)

(Prototype initial :
-------------------
'moulin'/'Nom')
(Prototype initial :
-------------------
'moulin'/'Nom')

(Prototype(s) nouveau(x) a construire :
--------------------------------------
4-(Cat :'N1aN2'-NeededCat :'Prep'-Forme :'')-5)

(Prototype initial :
-------------------
'a'/'Prep')

(Prototype initial :
-------------------
'paroles'/'Nom')

( Prototype extrait a construire :
```

```
 -------------------------------
4-(Cat a construire :'N1aN2'-Forme construite :''-NeededCat :'Prep')-5)

( Prototype disponible retenu :
-----------------------------
5-(Cat :'Prep'-Forme :'a')-6)

( Prototype resultant :
----------------------
4-(Cat :'N1aN2'-Forme construite :''-NeededCat :'Nom')-6)

( Prototype extrait a construire :
 -------------------------------
3-(Cat a construire :'Sn'-Forme construite :'(un *)'-NeededCat :'MoulinaN2')-4)

( Prototype extrait a construire :
 -------------------------------
2-(Cat a construire :'Sv'-Forme construite :'(analyse *)'-NeededCat :'Sn')-3)

( Prototype extrait a construire :
 -------------------------------
0-(Cat a construire :'Sn'-Forme construite :'(le *)'-NeededCat :'Nom')-1)

( Prototype disponible retenu :
-----------------------------
1-(Cat :'Nom'-Forme :'manager')-2)

( Prototype resultant :
----------------------
0-(Cat :'Sn'-Forme construite :'(le manager)'-NeededCat :'')-2)

(Prototype construit :
---------------------
0-(Cat :'Sn'-Forme construite :'(le manager)')-2)

(Nouveau prototype a construire :
--------------------------------
0-(Cat :'P'-Forme :'((le manager) *)')-2)

( Prototype extrait a construire :
 -------------------------------
0-(Cat a construire :'P'-Forme construite :'((le manager) *)'-NeededCat :'Sv')-2)

( Prototype extrait a construire :
 -------------------------------
4-(Cat a construire :'N1aN2'-Forme construite :''-NeededCat :'Nom')-6)

( Prototype disponible retenu :
-----------------------------
6-(Cat :'Nom'-Forme :'paroles')-7)

( Prototype resultant :
----------------------
4-(Cat :'N1aN2'-Forme construite :'(moulin a paroles)'-NeededCat :'')-7)

( Prototype extrait a construire :
 -------------------------------
3-(Cat a construire :'Sn'-Forme construite :'(un *)'-NeededCat :'MoulinaN2')-4)

( Prototype disponible retenu :
-----------------------------
4-(Cat :'MoulinaN2'-Forme :'(moulin a paroles)')-7)

( Prototype resultant :
----------------------
3-(Cat :'Sn'-Forme construite :'(un (moulin a paroles))'-NeededCat :'')-7)

(Prototype construit :
---------------------
3-(Cat :'Sn'-Forme construite :'(un (moulin a paroles))')-7)

( Prototype extrait a construire :
 -------------------------------
2-(Cat a construire :'Sv'-Forme construite :'(analyse *)'-NeededCat :'Sn')-3)
( Prototype disponible retenu :
-----------------------------
3-(Cat :'Sn'-Forme :'(un (moulin a paroles))')-7)

( Prototype resultant :
----------------------
2-(Cat :'Sv'-Forme construite :'(analyse (un (moulin a paroles)))'-NeededCat :'')-7)

(Prototype construit :
---------------------
2-(Cat :'Sv'-Forme construite :'(analyse (un (moulin a paroles)))')-7)

( Prototype extrait a construire :
 -------------------------------
0-(Cat a construire :'P'-Forme construite :'((le manager) *)'-NeededCat :'Sv')-2)

( Prototype disponible retenu :
-----------------------------
2-(Cat :'Sv'-Forme :'(analyse (un (moulin a paroles)))')-7)

( Prototype resultant :
----------------------
0-(Cat :'P'-Forme construite :'((le manager) (analyse (un (moulin a paroles))))'-NeededCat :'')-7)
```

```
(Prototype construit :
---------------------
0-(Cat :'P'-Forme construite :'((le manager) (analyse (un (moulin a paroles))))')-7)
nil
Self 38> manager displayResult
(0-( Cat : 'Clit'/'le')-1)
(0-( Cat : 'Det'/'le')-1)
(1-( Cat : 'Nom'/'manager')-2)
(2-( Cat : 'Vt'/'analyse')-3)
(3-( Cat : 'Det'/'un')-4)
(4-( Cat : 'Nom'/'moulin')-5)
(4-( Cat : 'Nom'/'moulin')-5)
(5-( Cat : 'Prep'/'a')-6)
(6-( Cat : 'Nom'/'paroles')-7)
(0-( Cat : 'Sn'/'(le manager)')-2)
(4-( Cat : 'MoulinaN2'/'(moulin a paroles)')-7)
(3-( Cat : 'Sn'/'(un (moulin a paroles))')-7)
(2-( Cat : 'Sv'/'(analyse (un (moulin a paroles)))')-7)
(0-( Cat : 'P'/'((le manager) (analyse (un (moulin a paroles))))')-7)
nil
```

- Prototypes lexicaux créateurs de prototypes :

```
une protoToBuild: (vector copy) asList.
_AddSlots: (| sn1 |)
sn1: catSn copy
_AddSlots: (| p1 |)
p1: catP copy
sn1 protoToBuild: (((vector copy) asList) addLast: p1)
une protoToBuild addLast:sn1
_AddSlots: (| sn3 |)
sn3: catSnAdjN copy
_AddSlots: (| p2 |)
p2: catP copy
sn3 protoToBuild: (((vector copy) asList) addLast: p2)
une protoToBuild addLast:sn3
laD protoToBuild: (vector copy) asList.
_AddSlots: (| sn2 |)
sn2: catSn copy
laD protoToBuild: (((vector copy) asList) addLast:sn2)
laClit protoToBuild: (vector copy) asList.
_AddSlots: (| sv1 |)
sv1: catSVtClit copy
laClit protoToBuild addLast: sv1
briseV protoToBuild: (vector copy) asList.
_AddSlots: (| sv2 |)
sv2: catSVt copy
briseV protoToBuild addLast: sv2
petiteAdj protoToBuild: (vector copy) asList.
_AddSlots: (| adjn1 |)
adjn1: catAdjn copy
petiteAdj protoToBuild addLast: adjn1
```

- Initialisation de la phrase à analyser :

```
Self 4> textExample inputTextExample removeAll
textExample setInputTextExample: 'une'
textExample setInputTextExample: 'petite'
textExample setInputTextExample: 'brise'
textExample setInputTextExample: 'la'
textExample setInputTextExample: 'glace'
list{}
Self 5> list{'une'}
Self 6> list{'une', 'petite'}
Self 7> list{'une', 'petite', 'brise'}
Self 8> list{'une', 'petite', 'brise', 'la'}
Self 9> list{'une', 'petite', 'brise', 'la', 'glace'}
```

- Début de l'analyse :

```
Self 10> manager parser: (textExample inputTextExample)

Noeuds initiaux:

(0-('Det'/'une')-1)

(1-('Nom'/'petite')-2)

(1-('Adj'/'petite')-2)

(2-('Nom'/'brise')-3)

(2-('Vt'/'brise')-3)

(3-('Det'/'la')-4)

(3-('Clit'/'la')-4)

(3-('Clit'/'la')-4)

(4-('Nom'/'glace')-5)

(4-('Vt'/'glace')-5)
```

```
(Prototype initial:
-------------------
'une'/'Det')

(Prototype(s) nouveau(x) a construire:
--------------------------------------
0-(Cat:'Sn'-NeededCat:'Nom'-Forme:'(une *)')-1)

(Prototype(s) nouveau(x) a construire:
--------------------------------------
0-(Cat:'Sn'-NeededCat:'AdjN'-Forme:'(une *)')-1)

(Prototype initial:
-------------------
'petite'/'Nom')

(Prototype initial:
-------------------
'petite'/'Adj')

(Prototype(s) nouveau(x) a construire:
--------------------------------------
1-(Cat:'AdjN'-NeededCat:'Nom'-Forme:'(petite *)')-2)

(Prototype initial:
-------------------
'brise'/'Nom')

(Prototype initial:
-------------------
'brise'/'Vt')

(Prototype(s) nouveau(x) a construire:
--------------------------------------
2-(Cat:'Sv'-NeededCat:'Sn'-Forme:'(brise *)')-3)

(Prototype initial:
-------------------
'la'/'Det')
(Prototype initial:
-------------------
'la'/'Det')

(Prototype(s) nouveau(x) a construire:
--------------------------------------
3-(Cat:'Sn'-NeededCat:'Nom'-Forme:'(la *)')-4)

(Prototype initial:
-------------------
'la'/'Clit')

(Prototype initial:
-------------------
'la'/'Clit')

(Prototype(s) nouveau(x) a construire:
--------------------------------------
3-(Cat:'Sv'-NeededCat:'Vt'-Forme:'(la *)')-4)

(Prototype initial:
-------------------
'glace'/'Nom')

(Prototype initial:
-------------------
'glace'/'Vt')

( Prototype extrait a construire:
 --------------------------------
3-(Cat a construire:'Sv'-Forme construite:'(la *)'-NeededCat:'Vt')-4)

( Prototype disponible retenu:
------------------------------
4-(Cat:'Vt'-Forme:'glace')-5)

( Prototype resultant:
----------------------
3-(Cat:'Sv'-Forme construite:'(la glace)'-NeededCat:'')-5)

(Prototype construit:
---------------------
3-(Cat:'Sv'-Forme construite:'(la glace)')-5)

( Prototype extrait a construire:
 --------------------------------
3-(Cat a construire:'Sn'-Forme construite:'(la *)'-NeededCat:'Nom')-4)

( Prototype disponible retenu:
------------------------------
4-(Cat:'Nom'-Forme:'glace')-5)

( Prototype resultant:
----------------------
3-(Cat:'Sn'-Forme construite:'(la glace)'-NeededCat:'')-5)

(Prototype construit:
```

```
---------------------
3-(Cat:'Sn'-Forme construite:'(la glace)')-5)

( Prototype extrait a construire:
 ------------------------------
2-(Cat a construire:'Sv'-Forme construite:'(brise *)'-NeededCat:'Sn')-

( Prototype disponible retenu:
----------------------------
3-(Cat:'Sn'-Forme:'(la glace)')-5)

( Prototype resultant:
----------------------
2-(Cat:'Sv'-Forme construite:'(brise (la glace))'-NeededCat:'')-5)

(Prototype construit:
---------------------
2-(Cat:'Sv'-Forme construite:'(brise (la glace))')-5)

( Prototype extrait a construire:
 ------------------------------
1-(Cat a construire:'AdjN'-Forme construite:'(petite *)'-NeededCat:'Nom')-2)

( Prototype disponible retenu:
----------------------------
2-(Cat:'Nom'-Forme:'brise')-3)

( Prototype resultant:
----------------------
1-(Cat:'AdjN'-Forme construite:'(petite brise)'-NeededCat:'')-3)

(Prototype construit:
---------------------
1-(Cat:'AdjN'-Forme construite:'(petite brise)')-3)

( Prototype extrait a construire:
 ------------------------------
0-(Cat a construire:'Sn'-Forme construite:'(une *)'-NeededCat:'AdjN')-

( Prototype disponible retenu:
----------------------------
1-(Cat:'AdjN'-Forme:'(petite brise)')-3)

( Prototype resultant:
----------------------
0-(Cat:'Sn'-Forme construite:'(une (petite brise))'-NeededCat:'')-3)

(Prototype construit:
---------------------
0-(Cat:'Sn'-Forme construite:'(une (petite brise))')-3)

(Nouveau prototype a construire:
--------------------------------
0-(Cat:'P'-Forme:'((une (petite brise)) *)')-3)

( Prototype extrait a construire:
 ------------------------------
0-(Cat a construire:'Sn'-Forme construite:'(une *)'-NeededCat:'Nom')-1

( Prototype disponible retenu:
----------------------------
1-(Cat:'Nom'-Forme:'petite')-2)

( Prototype resultant:
----------------------
0-(Cat:'Sn'-Forme construite:'(une petite)'-NeededCat:'')-2)

(Prototype construit:
---------------------
0-(Cat:'Sn'-Forme construite:'(une petite)')-2)

(Nouveau prototype a construire:
--------------------------------
0-(Cat:'P'-Forme:'((une petite) *)')-2)

( Prototype extrait a construire:
 ------------------------------
0-(Cat a construire:'P'-Forme construite:'((une petite) *)'-NeededCat:'Sv')-2)

( Prototype disponible retenu:
----------------------------
2-(Cat:'Sv'-Forme:'(brise (la glace))')-5)

( Prototype resultant:
----------------------
0-(Cat:'P'-Forme construite:'((une petite) (brise (la glace)))'-NeededCat:'')-5)

(Prototype construit:
---------------------
0-(Cat:'P'-Forme construite:'((une petite) (brise (la glace)))')-5)

( Prototype extrait a construire:
 ------------------------------
0-(Cat a construire:'P'-Forme construite:'((une (petite brise)) *)'-NeededCat:'Sv')-3)

( Prototype disponible retenu:
----------------------------
```

```
3-(Cat:'Sv'-Forme:'(la glace)')-5)

( Prototype resultant:
----------------------
0-(Cat:'P'-Forme construite:'((une (petite brise)) (la glace))'-NeededCat:'')-5)

(Prototype construit:
---------------------
0-(Cat:'P'-Forme construite:'((une (petite brise)) (la glace))')-5)

nil
```

- Fin de l'analyse. Le `manager` contient les éléments construits au cours de l'analyse :

```
Self 11> manager displayResult
(0-( Cat: 'Det'/'une')-1)
(1-( Cat: 'Nom'/'petite')-2)
(1-( Cat: 'Adj'/'petite')-2)
(2-( Cat: 'Nom'/'brise')-3)
(2-( Cat: 'Vt'/'brise')-3)
(3-( Cat: 'Det'/'la')-4)
(3-( Cat: 'Det'/'la')-4)
(3-( Cat: 'Clit'/'la')-4)
(3-( Cat: 'Clit'/'la')-4)
(4-( Cat: 'Nom'/'glace')-5)
(4-( Cat: 'Vt'/'glace')-5)
(3-( Cat: 'Sv'/'(la glace)')-5)
(3-( Cat: 'Sn'/'(la glace)')-5)
(2-( Cat: 'Sv'/'(brise (la glace))')-5)
(1-( Cat: 'AdjN'/'(petite brise)')-3)
(0-( Cat: 'Sn'/'(une (petite brise))')-3)
(0-( Cat: 'Sn'/'(une petite)')-2)
(0-( Cat: 'P'/'((une petite) (brise (la glace)))')-5)
(0-( Cat: 'P'/'((une (petite brise)) (la glace))')-5)
nil
Self 12>
```

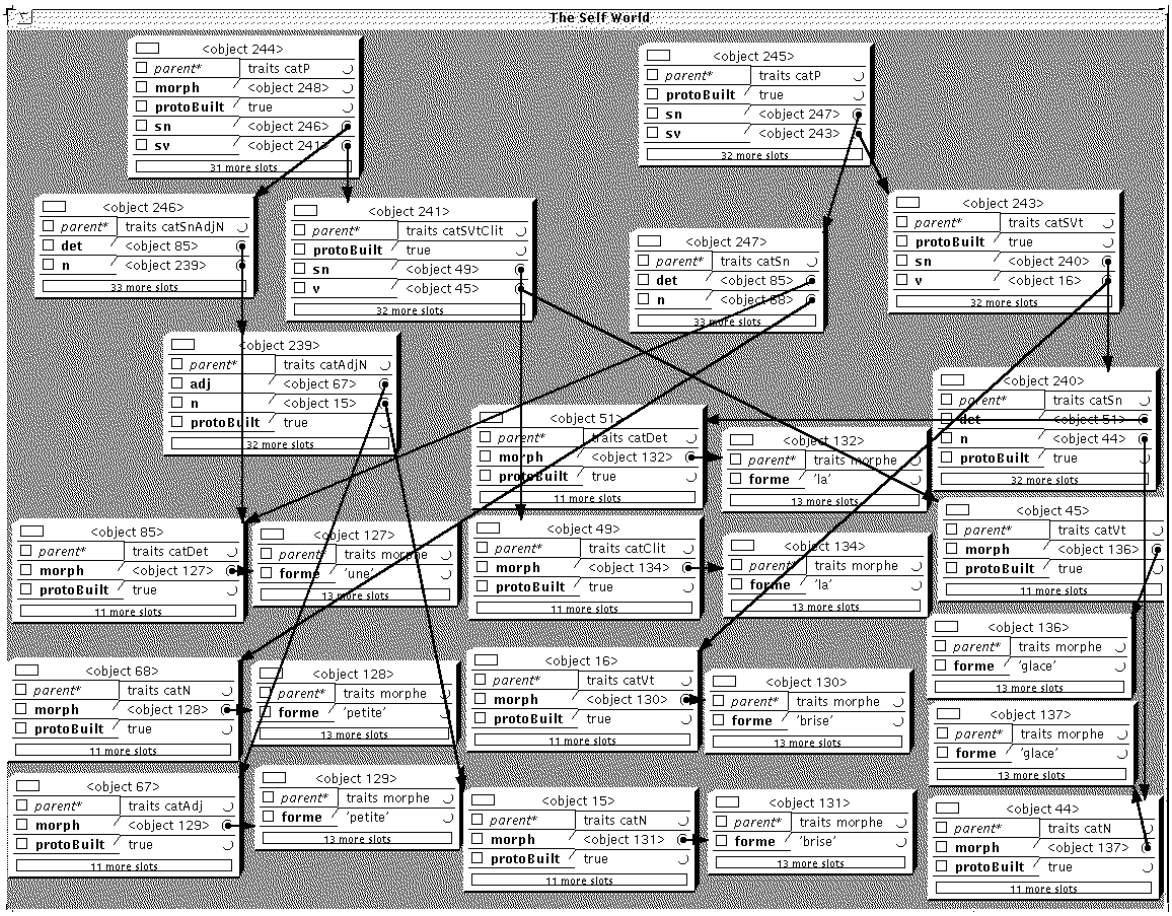


Figure a-2-5.4 : Un réseau de prototypes pour "`une petite brise la glace`".

A-2-5.4. Mise en place d'un affinage progressif pour la construction du sens des séquences moulin à N2

```
Self 79> Debut Analyse....

Noeuds initiaux :

(0-('Nom'/'moulin')-1)

(1-('Prep'/'a')-2)

(2-('Nom'/'paroles')-3)

(Prototype initial :
-------------------
'moulin'/'Nom')
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'AVEC/Energie'>
Ajout du N1 au prototype specialise MoulinaN2 <classe :'AVEC/Partie-de'>
(Prototype(s) nouveau(x) a construire :
-------------------------------------
0-(Cat:'N1aN2'-NeededCat:'Prep'-Forme:'moulin' )')-1)

(Prototype initial :
-------------------
'a'/'Prep')

(Prototype initial :
-------------------
'paroles'/'Nom')

( Prototype extrait a construire :
 ------------------------------
0-(Cat a construire :'N1aN2'-Forme construite : 'moulin')'-NeededCat :'Prep')-1)

( Prototype disponible retenu :
-----------------------------
1-(Cat :'Prep'-Forme :'a')-2)

Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'AVEC/Energie'>
Ajout de la Prep au prototype specialise MoulinaN2 <classe :'AVEC/Partie-de'>
( Prototype resultant :
----------------------
0-(Cat :'N1aN2'-Forme construite : 'moulin a ')-NeededCat :'Nom')-2)

( Prototype extrait a construire :
 ------------------------------
0-(Cat a construire :'N1aN2'-Forme construite : 'moulin a ')-NeededCat :'Nom')-2)

( Prototype disponible retenu :
-----------------------------
2-(Cat :'Nom'-Forme :'paroles')-3)
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'AVEC/Energie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'AVEC/Partie-de'>
```

La sémantique construite n'entre pas dans le modèle sémantique construit.
Le menu qui suit vous permet de modifier les valeurs sémantiques attachées
aux entrées lexicales représentées.

```
-------------------------------------------------------
<1 : Display Semantique>
<2 : Modify Semantique>
<3 : New Semantique>


Select a number or q to quit Gaspar-Semantique :1
-------------------------------------------------------
<1 : Display Semantique N1>
<2 : Display Semantique Prep
<3 : Display Semantique N2

Select a number or q to quit Gaspar-Display-Semantique :3
-------------------------------------------------------
Categorie  :
```

```
(Type :'Nom')
Morph  :
 <Lemme : 'parole'><Forme : 'paroles'><Forme Plur : 'paroles'><Forme Sing : 'parole'>
 <Classe : 'nom-a-plur-en-s'>
Semantique  :
Trait :
 (comptable : '-')
 (humain : '-')
 (artefact : '-')
 (vivant : '-')
 (anime : '-')
 (abstrait : '-')
 (concret : '+')
 (Nparole : '+')
 (ali : '-')
 (synth : '-')
 (energie : '+')
 (deverbal : '-')
Classe :
''
Predicat :
Relation :
Accord  :
 <Genre : 'Fem'><Nombre : 'Plur'>
-------------------------------------------------------
<1 : Display Semantique N1>
<2 : Display Semantique Prep
<3 : Display Semantique N2

Select a number or q to quit Gaspar-Display-Semantique :q
-------------------------------------------------------
Gaspar-Display-Semantique Shut Down
-------------------------------------------------------
<1 : Display Semantique>
<2 : Modify Semantique>
<3 : New Semantique>

Select a number or q to quit Gaspar-Semantique :2
-------------------------------------------------------
-------------------------------------------------------
<1 : Mod Semantique N1>
<2 : Mod Semantique Prep>
<3 : Mod semnatique N2>

Select a number or q to quit Gaspar-Mod-Semantique :3
-------------------------------------------------------
Nouvelles valeurs pour les traits semantiques du N2
-----------------------------------------------------
Trait Concret :+
Trait Comptable :-
Trait Abstrait :-
Trait Nparole :+
Trait Energie :-
Trait humain :-
Trait vivant :-
Trait anime :-
Trait Synth :-
Trait deverbal :-
Trait Artefact :-
----------------------------------------
Categorie  :
(Type :'Nom')
Morph  :
 <Lemme : 'parole'><Forme : 'paroles'><Forme Plur : 'paroles'><Forme Sing : 'parole'>
 <Classe : 'nom-a-plur-en-s'>
Semantique  :
Trait :
 (comptable : '-')
 (humain : '-')
 (artefact : '-')
 (vivant : '-')
 (anime : '-')
 (abstrait : '-')
 (concret : '+')
 (Nparole : '+')
 (ali : '-')
 (synth : '-')
 (energie : '-')
 (deverbal : '-')
Classe :
''
Predicat :
Relation :
Accord  :
 <Genre : 'Fem'><Nombre : 'Plur'>

Pour Une nouvelle analyse, quitter GasparSemantique en repondant par <y>.
Nouvelle analyse :y
-------------------------------------------------------
<1 : Mod Semantique N1>
<2 : Mod Semantique Prep>
<3 : Mod semnatique N2>

Select a number or q to quit Gaspar-Mod-Semantique :q
-------------------------------------------------------
Gaspar-Modd-Semantique Shut Down
-------------------------------------------------------
```

```
<1 : Display Semantique>
<2 : Modify Semantique>
<3 : New Semantique>

Select a number or q to quit Gaspar-Semantique :q
-----------------------------------------------------
Gaspar-Semantique Shut Down

(reprise de l'analyse avec les nouvelles valeurs sémantiques pour paroles)

<NewParsing :'y'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Entree'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'POUR/Sortie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'AVEC/Energie'>
Ajout du N2 au prototype specialise MoulinaN2 <classe :'AVEC/Partie-de'>
( Prototype resultant :
----------------------
0-(Cat :'N1aN2'-Forme construite :'(moulin a paroles)'-NeededCat :'')-3)

...Fin Analyse
(0-( Cat : 'Nom'/'moulin')-1)
(1-( Cat : 'Prep'/'a')-2)
(2-( Cat : 'Nom'/'paroles')-3)
(0-( Cat : 'MoulinaN2'/'(moulin a paroles)')-3)
-0-
<Classe reconnue :'POUR/Sortie'>
<Forme :'(moulin a paroles)'>
Semantique : Comptable Concret Abstrait Energie Vivant Anime Humain Nparole Synth Ali Deverbal
<('-','+','-','-','','','+','+','-','-','')>
-3-

 Prototypes Initiaux :3


 Prototypes a Construire Declenches :1


 Prototypes Extraits :2


 Prototypes Construits :4

manager
Self 80>
```

Annexes Partie 2 Chapitre 7

A-2-7.1. Illustration de l'héritage avec Self sur un exemple

On donne dans le code présenté ci-dessous une illustration de l'héritage multiple et dynamique sur un exemple simple.

```
Self 1> 'heritageMultiple.self' _RunScript
reading ./heritageMultiple.self...
<object 3>
```

Héritage Multiple

On dispose dans cet exemple deux types de comportement pour l'affichage des attributs des prototypes définis.

```
Self 2>  inspect: traits parent1
( | _ parent* = traits clonable.
    ^ copy = ( clone).
    ^ display = (
        _AddSlotsIfAbsent: (| displayParent1 |).
        displayParent1:( '      (Attribut 1: ', slot1  printString, ')').
        displayParent1 printLine.
        self.
    ). | )
<object 5>
Self 3>  inspect: traits parent2
( | ^ display = (
        _AddSlotsIfAbsent: (| displayParent1 |).
        displayParent1:( '      (Attribut 1: ', slot1  printString, ')').
        displayParent1 printLine.
        _AddSlotsIfAbsent: (| displayParent2 |).
        displayParent2:( '      (Attribut 2: ', slot2  printString, ')').
        displayParent2 printLine.
        self.
    ). | )
<object 7>
```

On construit des prototypes qui vont déléguer un comportement à l'un ou à l'autre de ces objets ou aux deux.

```
Self 4> inspect: filsParent1
( | _ parent* <- <object 5>.
    ^ slot1 <- 'valeur attribut slot1'.
    ^ slot2 <- 'valeur attribut slot2'. | )
<object 9>
Self 5> inspect: filsParent1EtParent2
( | _ root2* <- <object 7>.
    _ root1* <- <object 5>.
    ^ slot1 <- 'valeur attribut slot1'.
    ^ slot2 <- 'valeur attribut slot2'. | )
<object 11>
Self 6> inspect: filsParent1EtParent2Bis
( | _ root2* <- <object 7>.
    _ root1* <- <object 5>.
    ^ slot1 <- 'valeur attribut slot1'.
    ^ slot2 <- 'valeur attribut slot2'.
    ^ display = (
        _AddSlotsIfAbsent: (| displayParent2 |).
        displayParent2:( '      (Attribut 2: ', slot2  printString, ')').
```

```
        displayParent2 printLine.
        _AddSlotsIfAbsent: (| displayParent1 |).
        displayParent1:( '      (Attribut 1: ', slot1  printString, ')').
        displayParent1 printLine.
         self.
    ). | )
<object 13>
Self 7> inspect: filsParent1EtParent2Ter
( | _ root2* <- <object 7>.
    _ parent* <- <object 5>.
    ^ slot1 <- 'valeur attribut slot1'.
    ^ slot2 <- 'valeur attribut slot2'.
    ^ displayFils = ( parent.display). | )
<object 3>
```

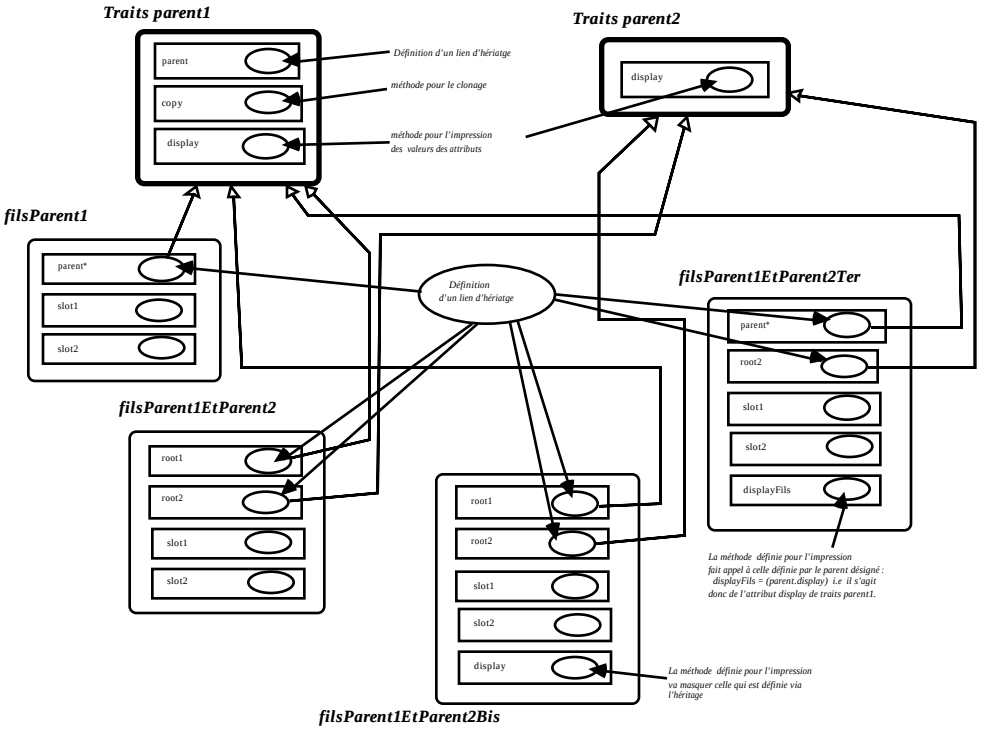


Figure a-2-7.1 : L'héritage et Self.

On regarde maintenant comment se comporte chacun de ces prototypes par rapport aux parents qui leurs sont attachés.

```
Self 8>  filsParent1 display
      (Attribut 1: 'valeur attribut slot1')
<object 9>
```

"Le prototype précédent délègue le comportement visé à un seul parent"

```
Self 9>  filsParent1EtParent2 display
More than one 'display' slot was found in filsParent1EtParent2 <11>.
Sending method holder is shell <17>.
The matching slots are: traits parent1 <5> and traits parent2 <7>.
## Stack trace for process <18>
##____Rcvr___Selector/Block__________File:line_____Method-holder/location_____
#5    <11>   display                  <error>:1       filsParent1EtParent2
#6    <17>   <top level expr>         <the prompt>:1 shell
```

Le prototype précédent délègue le même comportement à deux parents, d'où le message d'erreur produit. Pour éviter de tels conflits, il faut soit les résoudre au momemt de la définition des objets en prenant soin de ne pas générer de telles ambiguïtés. On dispose aussi de primitives qui permettent de selectionner le parent désiré pour la délégation.

```
Self 10>  filsParent1EtParent2Bis display
      (Attribut 2: 'valeur attribut slot2')
      (Attribut 1: 'valeur attribut slot1')
<object 8>
```

Le prototype précédent possède le même nom d'attribut que celui qu'il pourrait déléguer à ces deux parents, c'est donc cet attribut qui est activé (activation du slot de plus bas niveau).

```
Self 11>  filsParent1EtParent2Ter display
More than one 'display' slot was found in filsParent1EtParent2Ter <3>.
Sending method holder is shell <17>.
The matching slots are: traits parent1 <5> and traits parent2 <7>.
## Stack trace for process <21>
##____Rcvr___Selector/Block__________File:line_____Method-holder/location_____
#5    <3>    display                  <error>:1      filsParent1EtParent2Ter
#6    <17>   <top level expr>         <the prompt>:1 shell
```

Le prototype précédent délègue le même comportement à deux parents, d'où le message d'erreur produit. Ce même prototype permet aussi de passer outre cette ambiguïté: l'attribut `displayfils` permet de réaliser un renvoi dirigé vers un de ses parents.

```
Self 12>  filsParent1EtParent2Ter displayFils
      (Attribut 1: 'valeur attribut slot1')
filsParent1EtParent2Ter
```

De même on peut réaliser un renvoi de message pour les deux prototypes précédents auxquels les messages d'affichage qui leur étaient adressés ne pouvaient être exécutés. On utilise pour cela des primitives du langage de la manière suivante:

```
Self 13>  'display' sendTo:  filsParent1EtParent2 DelegatingTo: filsParent1EtParent2 root1
      (Attribut 1: 'valeur attribut slot1')
filsParent1EtParent2
Self 14>  'display' sendTo:  filsParent1EtParent2 DelegatingTo: filsParent1EtParent2 root2
      (Attribut 1: 'valeur attribut slot1')
      (Attribut 2: 'valeur attribut slot2')
filsParent1EtParent2
Self 15>  'display' sendTo:  filsParent1EtParent2Ter DelegatingTo: filsParent1EtParent2Ter root2
      (Attribut 1: 'valeur attribut slot1')
      (Attribut 2: 'valeur attribut slot2')
filsParent1EtParent2Ter
Self 16>  'display' sendTo:  filsParent1EtParent2Ter DelegatingTo: filsParent1EtParent2Ter parent
      (Attribut 1: 'valeur attribut slot1')
filsParent1EtParent2Ter
```

Héritage Dynamique

Il est possible de modifier les comportements attachés à un prototype donné. Il convient pour cela de définir le lien qui existe entre le prototype et son père avec un slot d'assignation: c'est le cas pour tous les prototypes définis ici. On peut donc modifier les liens qui existent entre les objets et leurs ancêtres.

```
Self 17>  filsParent1 parent
traits parent1
Self 18> filsParent1 parent: traits parent2
filsParent1
Self 19>  filsParent1 parent
traits parent2
Self 20>  filsParent1 display
      (Attribut 1: 'valeur attribut slot1')
      (Attribut 2: 'valeur attribut slot2')
filsParent1
Self 21>
```

Le comportement attaché au prototype précédent a été modifié.

A-2-7.2. Génération dynamique de prototypes lexicaux (1)

On donne ci-dessous une première illustration de la génération de contraintes associées à des représentations prototypiques d'éléments lexicaux. Les représentations pré-définies ici sont du même type que celles que nous avons déjà présentées. On présente tout d'abord le fichier qui contient les contraintes recueillies:

```
#Dc
pontage
Nom
Sn
(('Det' & 'Nom') asList)
N1prepN2
(('Nom' & 'Prep' & 'Nom') asList)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
AdjN
(('Adj' & 'Nom') asList)
NAdj
(('Nom' & 'Adj') asList)
PrepN
(('Prep' & 'Nom') asList)
Fc#
#Dc
lesion
Nom
Sn
(('Det' & 'Nom') asList)
N1prepN2
(('Nom' & 'Prep' & 'Nom') asList)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
AdjN
(('Adj' & 'Nom') asList)
NAdj
(('Nom' & 'Adj') asList)
PrepN
(('Prep' & 'Nom') asList)
Fc#
#EndOfContraintes#
```

La première contrainte sur `pontage` peut se lire de la manière suivante:

```
<pontage,Nom> -> {Sn (Det Nom), NPrepN (N1PrepN2), AdjN (Adj Nom), NAdj (Nom Adj), PrepN (Prep Nom)}
```

c'est-à-dire que le prototype lexical `pontage` de catégorie `Nom` peut initialiser la construction des prototypes `Sn`, `NPrepN`... avec les composants associés. Idem pour le prototype `lésion`. On présente ci-dessous la trace de la phase de lecture de ces contraintes puis de leurs affectations aux prototypes concernés.

```
-----------------------------------------------------
Gaspar-Load Shut Down


-----------------------------------------------------
<Gaspar Location:/home/info/fleury/Self/source/Self-3.0/Phase13/...>

-----------------------------------------------------
<1: Menu Load >
<2: Menu Parsing>
<3: Menu Lexical (Display-Create)>
<4: Menu Define Prototype From Corpus>
<5: Menu Help>
<6: PontageaN2-Semantic-Rules>
<7: ui start>
<8: ui quit>
<9: Tools>

Select a number or q to quit Gaspar:2
-----------------------------------------------------
-----------------------------------------------------
<1: GasparParser Menu>
<2: GasparContraints Menu>

Select a number or q to quit MenuParser:2
-----------------------------------------------------
-----------------------------------------------------
<1: Define Contraints From Corpus>
<2: Display All Defined Word-Contraints From Corpus>
<3: Display One Defined Word-Contraints From Corpus>
<4: Display Pre-Defined Cat-Contraints>

Select a number or q to quit MenuContraints:1
-----------------------------------------------------
```

```
     Vidage du dictionnaire des contraintes

     Recreation du dictionnaire des contraintes

     Dictionnaire Vide des contraintes disponible

 Lecture du Fichier Disponible de Contraintes sur les Mots.

/home/info/fleury/Self/source/Self-3.0/Phase13/work/inputContraintes.self


Contrainte[1] dans le fichier des contraintes :

---------------------------------------------------------------------------
Contrainte sur : <pontage>
Valeur categorielle du mot contraint : <Nom>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Nom>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Det' & 'Nom') asList)>
Categorie de la contrainte lue : <Sn>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <N1prepN2>
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1prepN2>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Prep' & 'Verbe') asList)>
Categorie de la contrainte lue : <N1prepV>
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1prepV>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Adj' & 'Nom') asList)>
Categorie de la contrainte lue : <AdjN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <AdjN>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Adj') asList)>
Categorie de la contrainte lue : <NAdj>
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <PrepN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <PrepN>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!


Creation des arbres elementaires associes a la contrainte.
---------------------------------------------------------

/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self...
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self...
Arbre elementaire en Cours de Construction :
<list{'pontage', 'Nom', list{catSn, catN1prepN2, catN1prepV, catAdjN, catNAdj, catPrepN, list{list, list, list, list,
list, list}}>


--> Verification de l'existence d'un prototype portant la valeur categorielle <'Sn'> et des composants associes
<list{'Det', 'Nom'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'N1prepN2'> et des composants associes
<list{'Nom', 'Prep', 'Nom'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'N1prepV'> et des composants associes
<list{'Nom', 'Prep', 'Verbe'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'SAdjN'> et des composants associes
<list{'Adj', 'Nom'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'NSAdj'> et des composants associes
<list{'Nom', 'Adj'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'PrepN'> et des composants associes
<list{'Prep', 'Nom'}>
Nombre de Prototype ayant un attribut forme :159
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
```

```
Contrainte[2] dans le fichier des contraintes :

---------------------------------------------------------------------------
Contrainte sur : <lesion>
Valeur categorielle du mot contraint : <Nom>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Nom>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Det' & 'Nom') asList)>
Categorie de la contrainte lue : <Sn>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <N1prepN2>
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1prepN2>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Prep' & 'Verbe') asList)>
Categorie de la contrainte lue : <N1prepV>
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1prepV>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Adj' & 'Nom') asList)>
Categorie de la contrainte lue : <AdjN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <AdjN>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Adj') asList)>
Categorie de la contrainte lue : <NAdj>
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <PrepN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <PrepN>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Creation des arbres elementaires associes a la contrainte.
-------------------------------------------------------

/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self...
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self...
Arbre elementaire en Cours de Construction :
<list{'lesion', 'Nom', list{catSn, catN1prepN2, catN1prepV, catAdjN, catNAdj, catPrepN}, list{list, list, list, list,
list, list}}>


--> Verification de l'existence d'un prototype portant la valeur categorielle <'Sn'> et des composants associes
<list{'Det', 'Nom'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'N1prepN2'> et des composants associes
<list{'Nom', 'Prep', 'Nom'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'N1prepV'> et des composants associes
<list{'Nom', 'Prep', 'Verbe'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'SAdjN'> et des composants associes
<list{'Adj', 'Nom'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'NSAdj'> et des composants associes
<list{'Nom', 'Adj'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!
--> Verification de l'existence d'un prototype portant la valeur categorielle <'PrepN'> et des composants associes
<list{'Prep', 'Nom'}>
Nombre de Prototype ayant un attribut forme :161
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

Fin de Lecture du Fichier des Contraintes.

     Contraintes Lues :
list{list{'pontage', 'Nom', list, list}, list{'lesion', 'Nom', list, list}}


Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 3> -> 'Sn'
     Composants:list{'Det', 'Nom'}


Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 4> -> 'N1prepN2'
     Composants:list{'Nom', 'Prep', 'Nom'}
```

```
Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 5> -> 'N1prepV'
     Composants:list{'Nom', 'Prep', 'Verbe'}


Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 6> -> 'SAdjN'
     Composants:list{'Adj', 'Nom'}


Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 7> -> 'NSAdj'
     Composants:list{'Nom', 'Adj'}


Nouvelle(s) contraintes sur le mot <'pontage' 'Nom'> :
    <object 8> -> 'PrepN'
     Composants:list{'Prep', 'Nom'}


Recherche et Mise a jour des prototypes concernes

-------------------------------------------------------------------------------------------
 Prototype : ['pontage'|'Nom']
--> Mise a jour Du Champ EltTree De Ce Prototype avec les arbres suivants :
list{catSn, catN1prepN2, catN1prepV, catAdjN, catNAdj, catPrepN,}
--> Mise a jour Ok Du Champ EltTree De Ce Prototype
-------------------------------------------------------------------------------------------

Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 3> -> 'Sn'
     Composants:list{'Det', 'Nom'}


Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 4> -> 'N1prepN2'
     Composants:list{'Nom', 'Prep', 'Nom'}


Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 5> -> 'N1prepV'
     Composants:list{'Nom', 'Prep', 'Verbe'}


Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 6> -> 'SAdjN'
     Composants:list{'Adj', 'Nom'}


Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 7> -> 'NSAdj'
     Composants:list{'Nom', 'Adj'}


Nouvelle(s) contraintes sur le mot <'lesion' 'Nom'> :
    <object 8> -> 'PrepN'
     Composants:list{'Prep', 'Nom'}

Recherche et Mise a jour des prototypes concernes

-------------------------------------------------------------------------------------------
 Prototype : ['lesion'|'Nom']
--> Mise a jour Du Champ EltTree De Ce Prototype avec les arbres suivants :
list{catSn, catN1prepN2, catN1prepV, catAdjN, catNAdj, catPrepN}
--> Mise a jour Ok Du Champ EltTree De Ce Prototype
-------------------------------------------------------------------------------------------


     Dictionnaire des contraintes disponible


-------------------------------------------------------
<1: Define Contraints From Corpus>
<2: Display All Defined Word-Contraints From Corpus>
<3: Display One Defined Word-Contraints From Corpus>
<4: Display Pre-Defined Cat-Contraints>

Select a number or q to quit MenuContraints:3
-------------------------------------------------------
Contraintes sur quel Mot :pontage
Categorie de ce Mot [Cat|*] :Nom
-----------------------------------
Word :pontage
     Cat : <Nom>
     Categories Syntaxiques Associees :
          'Sn'
          'N1prepN2'
          'N1prepV'
          'SAdjN'
          'NSAdj'
          'PrepN'
-------------------------------------------------------
<1: Define Contraints From Corpus>
<2: Display All Defined Word-Contraints From Corpus>
<3: Display One Defined Word-Contraints From Corpus>
<4: Display Pre-Defined Cat-Contraints>

Select a number or q to quit MenuContraints:q
-------------------------------------------------------
MenuContraints Shut Down
-------------------------------------------------------
<1: GasparParser Menu>
<2: GasparContraints Menu>

Select a number or q to quit MenuParser:q
-------------------------------------------------------
MenuParser Shut Down
-------------------------------------------------------
```

A-2-7.3. Génération dynamique de prototypes lexicaux (2)

On exécute une réorganisation des savoirs associés aux prototypes représentant la catégorie des `Noms`. On fait tout d'abord une recherche de tous ces prototypes et de leurs contraintes respectives (on retrouve les contraintes précédemment définies); on détermine ensuite les comportements communs: une classe de comportements partagés par tous les prototypes qui disposent de contraintes. Cette classe peut être associée à l'objet `traits` commun à tous ces prototypes. Les prototypes qui ne disposent pas de contraintes à l'issue de la définition de celles-ci disposeront ainsi d'un savoir par défaut. Cela ne veut pas dire que ce savoir sera utilisé pour les entités qui ne disposent pas de contraintes particulières. Il s'agit simplement de marquer que sur une famille donnée, certains prototypes possèdent des contraintes et que ces savoirs sont disponibles mais ne sont pas utilisables pour tous les éléments de la famille.

```
-----------------------------------------------------
<1: SearchAndDisplay Parent for all Prototypes>
<2: SearchAndDisplay Parent for one Prototype>
<3: SearchAndDisplay Sons for all Objects Traits>
<4: SearchAndDisplay Sons for one Object Traits>
<5: Modify Links Beetween Selected Prototypes>
<6: Modify Links Beetween All Prototypes>

Select a number or q to quit MenuSearchAndDestroy:5
-----------------------------------------------------

Modifier Liens sur quelle categorie :Nom

Nombre de Prototype ayant un attribut forme :190

Comportements Communs aux Prototypes  pontage et lésion :

  Sn N1prepN2 N1prepV AdjN NAdj PrepN

Prototype Lexical pontage :

 Ajout du comportement Sn a la liste des comportements individuels

 Ajout du comportement N1PrepN2 a la liste des comportements individuels

 Ajout du comportement N1PrepV a la liste des comportements individuels

 Ajout du comportement SAdjN a la liste des comportements individuels

 Ajout du comportement NSAdj a la liste des comportements individuels

 Ajout du comportement PrepN a la liste des comportements individuels

Prototype Lexical lesion:

 Ajout du comportement Sn a la liste des comportements individuels si celui-ci n'est pas deja recense

 Ajout du comportement N1PrepN2 a la liste des comportements individuels si celui-ci n'est pas deja recense

 Ajout du comportement N1PrepV a la liste des comportements individuels si celui-ci n'est pas deja recense

 Ajout du comportement SAdjN a la liste des comportements individuels si celui-ci n'est pas deja recense

 Ajout du comportement NSAdj a la liste des comportements individuels si celui-ci n'est pas deja recense

 Ajout du comportement PrepN a la liste des comportements individuels si celui-ci n'est pas deja recense


--> Comportements Syntaxiques Partages par les 2 Prototypes <pontage|Nom><lesion|Nom> :

  0. Sn

  1. N1PrepN2

  2. N1PrepV

  3. SAdjN

  4. SAdjN

  5. PrepN

--> Ensemble des Comportements Syntaxiques des 2 Prototypes <pontage|Nom><lesion|Nom> :

  0. Sn

  1. N1PrepN2

  2. N1PrepV

  3. SAdjN

  4. SAdjN

  5. PrepN

-----------------------------------------------------
<1: SearchAndDisplay Parent for all Prototypes>
<2: SearchAndDisplay Parent for one Prototype>
```

```
<3: SearchAndDisplay Sons for all Objects Traits>
<4: SearchAndDisplay Sons for one Object Traits>
<5: Modify Links Beetween Selected Prototypes>
<6: Modify Links Beetween All Prototypes>

Select a number or q to quit MenuSearchAndDestroy:q
-------------------------------------------------------
Menu SearchAndDestroy Shut Down
-------------------------------------------------------
```

A-2-7.4. Génération dynamique de prototypes lexicaux (3)

L'exemple qui suit est une illustration de la génération de nouvelles représentations prototypiques de catégories syntaxiques non encore reconnues. Le fichier de contraintes a désormais l'allure suivante:

```
#Dc
pontage
Nom
Sn
(('Det' & 'Nom') asList)
NPivotprepN2
(('NomPivot' & 'Prep' & 'Nom') asList)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
AdjN
(('Adj' & 'Nom') asList)
NAdj
(('Nom' & 'Adj') asList)
PrepN
(('Prep' & 'Nom') asList)
N1NPivot
(('Nom' & 'NomPivot') asList)
Fc#
#Dc
lesion
Nom
Sn
(('Det' & 'Nom') asList)
N1prepN2
(('Nom' & 'Prep' & 'Nom') asList)
N1prepV
(('Nom' & 'Prep' & 'Verbe') asList)
AdjN
(('Adj' & 'Nom') asList)
NAdj
(('Nom' & 'Adj') asList)
PrepN
(('Prep' & 'Nom') asList)
NPivotN2
(('NomPivot' & 'Nom') asList)
Fc#
#EndOfContraintes#
```

Le mot `pontage` peut construire un nouveau comportement du type `N1NPivot {Nom, NomPivot}`, de même le mot `lesion` peut construire un nouveau comportement du type `NPivotN2 {NomPivot, Nom2}`. Il n'existe pas de pré-représentations prototypiques pour ces comportements. Il faut donc définir les objets adéquats pour les représenter. La trace qui suit présente cette création dynamique d'objets et le rattachement de ceux-ci aux prototypes lexicaux concernés.

```
-----------------------------------------------------
<1: Define Contraints From Corpus>
<2: Display All Defined Word-Contraints From Corpus>
<3: Display One Defined Word-Contraints From Corpus>
<4: Display Pre-Defined Cat-Contraints>

Select a number or q to quit MenuContraints:1
-----------------------------------------------------

     Vidage du dictionnaire des contraintes

     Recreation du dictionnaire des contraintes

     Dictionnaire Vide des contraintes disponible

 Lecture du Fichier Disponible de Contraintes sur les Mots.
/home/info/fleury/Self/source/Self-3.0/Phase13/work/inputContraintes.self


Contrainte[1] dans le fichier des contraintes :
-------------------------------------------------------------------------

Contrainte sur : <pontage>
Valeur categorielle du mot contraint : <Nom>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Nom>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Det' & 'Nom') asList)>
Categorie de la contrainte lue : <Sn>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!
```

```
Constituants de la Contrainte : <(('NomPivot' & 'Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <NPivotPrepN2>
--> Verification de l'existence d'un prototype portant la valeur categorielle <NPivotPrepN2>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Prep' & 'Verbe') asList)>
Categorie de la contrainte lue : <NPivotPrepV>
--> Verification de l'existence d'un prototype portant la valeur categorielle <NPivotPrepV>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Adj' & 'Nom') asList)>
Categorie de la contrainte lue : <AdjN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <AdjN>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'Adj') asList)>
Categorie de la contrainte lue : <NAdj>
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Prep' & 'Nom') asList)>
Categorie de la contrainte lue : <PrepN>
--> Verification de l'existence d'un prototype portant la valeur categorielle <PrepN>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!

Constituants de la Contrainte : <(('Nom' & 'NomPivot') asList)>
Categorie de la contrainte lue : <N1NPivot>
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1NPivot>
Nombre de Prototype ayant un attribut forme :158
--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle, on va en creer un!
Creation du Prototype <'N1NPivot'>
/home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto1Output.self
---> Definition de l' objet Traits associe : En cours
---> Definition de l' objet Traits associe Ok.
---> Definition de l' objet Prototype : En Cours
---> Definition de l' objet Prototype : Ok
---> Lecture et Creation  de l' objet Prototype : En Cours
/home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto1Output.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto1Output.self...
---> Lecture et Creation  de l' objet Prototype Ok.


Creation des arbres elementaires associes a la contrainte.

-------------------------------------------------------

/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/contraintsOn_Corpus.self...
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
/home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/constituantOnContraints.self...

Arbre elementaire en Cours de Construction :
<list{'pontage',  'Nom',  list{catSn,  catNPivotPrepN2,  catNPivotPrepV,  catAdjN,  catNAdj,  catPrepN,  catN1NPivot},
list{list, list, list, list, list, list, list}}>

--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'Sn'> et des composants associes
<list{'Det', 'Nom'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Presence d'un pivot
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'NPivotPrepN2'> et des composants
associes <list{'NomPivot', 'Prep', 'Nom'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'NPivotPrepV'> et des composants associes
<list{'Nom', 'Prep', 'Verbe'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'SAdjN'> et des composants associes
<list{'Adj', 'Nom'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'NSAdj'> et des composants associes
<list{'Nom', 'Adj'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'PrepN'> et des composants associes
<list{'Prep', 'Nom'}>
```

```
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--> Presence d'un pivot
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <'N1NPivot'> et des composants associes
<list{'Nom', 'NomPivot'}>
Nombre de Prototype ayant un attribut forme :160
--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
Creation du Prototype <'N1NPivot'> --> <list{'Nom', 'Nom'}>
/home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto2Output.self
---> Definition de l' objet Traits associe : En cours
---> Definition de l' objet Traits associe Ok.
---> Definition de l' objet Prototype : En Cours
---> Definition de l' objet Prototype : Ok
---> Lecture et Creation  de l' objet Prototype : En Cours
/home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto2Output.self
reading /home/info/fleury/Self/source/Self-3.0/Phase13/work/defineProto2Output.self...
---> Lecture et Creation  de l' objet Prototype Ok.
--> Definition et Creation du prototype Ok.
--> Mise a jour de l'arbre elementaire : En cours.
--> Mise a jour de l'arbre elementaire Ok.
```

(... idem avec lesion)

```
Fin de Lecture du Fichier des Contraintes.


     Contraintes Lues :

list{list{'pontage', 'Nom', list, list}, list{'lesion', 'Nom', list, list}}


Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 3> -> 'Sn'
     Composants:list{'Det', 'Nom'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 4> -> 'NPivotPrepN2'
     Composants:list{'NomPivot', 'Prep', 'Nom'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 5> -> 'NPivotPrepV'
     Composants:list{'Nom', 'Prep', 'Verbe'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 6> -> 'SAdjN'
     Composants:list{'Adj', 'Nom'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 7> -> 'NSAdj'
     Composants:list{'Nom', 'Adj'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 8> -> 'PrepN'
     Composants:list{'Prep', 'Nom'}

Nouvelle contrainte sur le mot <'pontage' 'Nom'> :

    <object 9> -> 'N1NPivot'
     Composants:list{'Nom', 'NomPivot'}


Recherche et Mise a jour des prototypes concernes

-------------------------------------------------------------------------------------------
 Prototype : ['pontage'|'Nom']
--> Mise a jour Du Champ EltTree De Ce Prototype avec les arbres suivants :
list{catSn, catNPivotPrepN2, catNPivotPrepV, catAdjN, catNAdj, catPrepN, catN1NPivot}
--> Mise a jour Ok Du Champ EltTree De Ce Prototype
-------------------------------------------------------------------------------------------
```

(idem avec lesion)

```
     Dictionnaire des contraintes disponible

-------------------------------------------------------
<1: Define Contraints From Corpus>
<2: Display All Defined Word-Contraints From Corpus>
<3: Display One Defined Word-Contraints From Corpus>
<4: Display Pre-Defined Cat-Contraints>

Select a number or q to quit MenuContraints:
-------------------------------------------------------
```

On donne ci-dessous des exemples de prototypes construits; ces derniers sont présentés via l'interface graphique de Self:

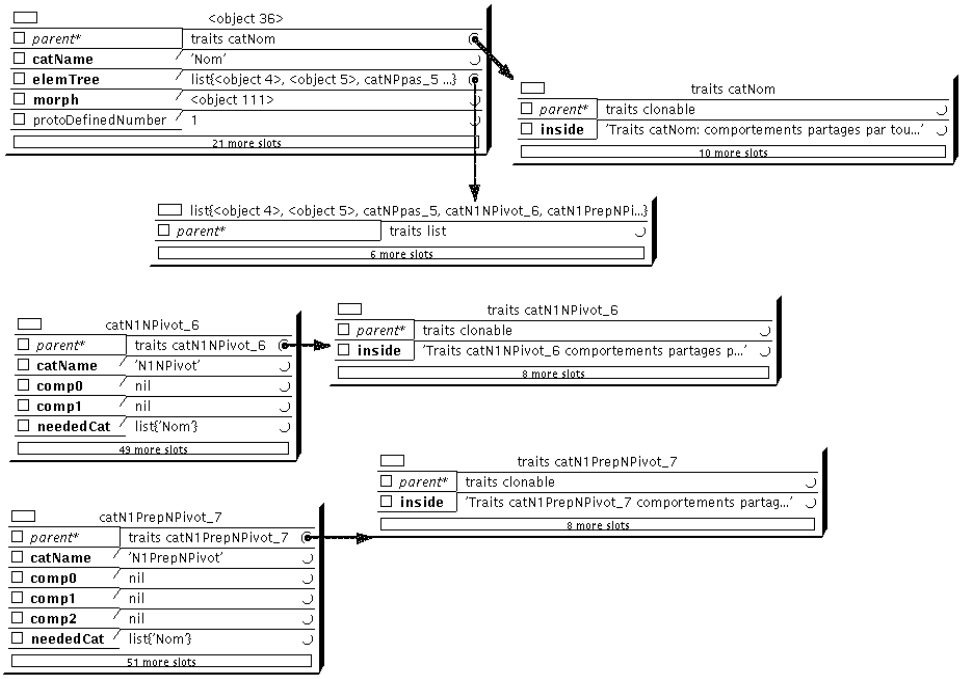


Figure a-2-7.2: Génération du prototype `pontage`

Cette première figure présente le prototype `pontage` construit ainsi que les arbres associés. La figure qui suit présente le prototype lexical `effort` construit par clonage et les arbres associés.

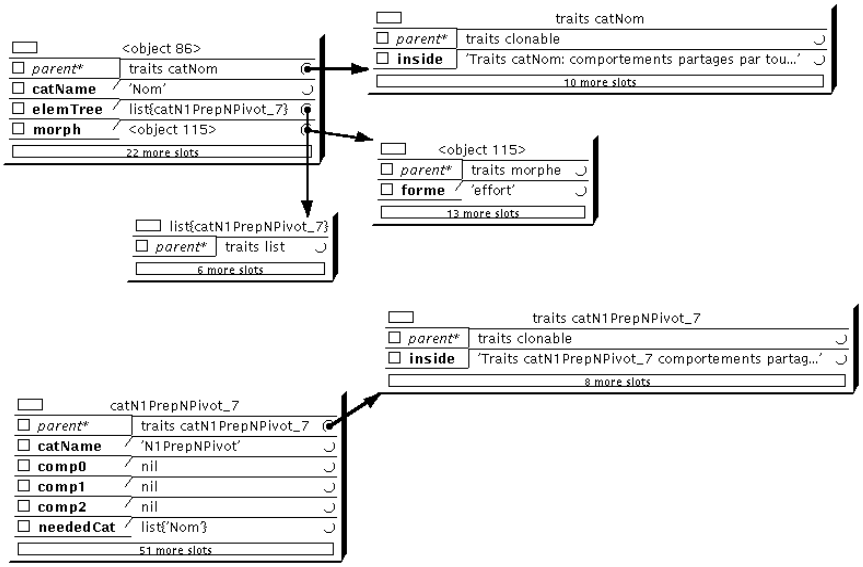


Figure a-2-7.3: Génération du prototype `effort`

A-2-7.5. Détermination des comportements partagés à une classe catégorielle donnée

Si on exécute de nouveau le classement des comportements associés aux noms, on obtient les résultats suivants.

```
-----------------------------------------------------
<1: SearchAndDisplay Parent for all Prototypes>
<2: SearchAndDisplay Parent for one Prototype>
<3: SearchAndDisplay Sons for all Objects Traits>
<4: SearchAndDisplay Sons for one Object Traits>
<5: Modify Links Beetween Selected Prototypes>
<6: Modify Links Beetween All Prototypes>

Select a number or q to quit MenuSearchAndDestroy:5
-----------------------------------------------------

Modifier Liens sur quelle categorie :Nom
Nombre de Prototype ayant un attribut forme :195
Comportements Communs aux Protototypes  pontage et lésion :
  Sn N1prepN2 N1prepV AdjN NAdj PrepN

Prototype Lexical pontage :
 Ajout du comportement Sn a la liste des comportements individuels
 Ajout du comportement N1PrepN2 a la liste des comportements individuels
 Ajout du comportement N1PrepV a la liste des comportements individuels
 Ajout du comportement SAdjN a la liste des comportements individuels
 Ajout du comportement NSAdj a la liste des comportements individuels
 Ajout du comportement PrepN a la liste des comportements individuels
 Ajout du comportement N1NPivot a la liste des comportements individuels

Prototype Lexical lesion:
 Ajout du comportement Sn a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement N1PrepN2 a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement N1PrepV a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement SAdjN a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement NSAdj a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement PrepN a la liste des comportements individuels si celui-ci n'est pas deja recense
 Ajout du comportement NPivotN2 a la liste des comportements individuels si celui-ci n'est pas deja recense

--> Comportements Syntaxiques partages par les 2 Prototypes <pontage|Nom><lesion|Nom> :
  0. Sn
  1. N1PrepN2
  2. N1PrepV
  3. SAdjN
  4. SAdjN
  5. PrepN

--> Ensemble des Comportements Syntaxiques des 2 Prototypes <pontage|Nom><lesion|Nom> :
  0. Sn
  1. N1PrepN2
  2. N1PrepV
  3. SAdjN
  4. SAdjN
  5. PrepN
  6. NPivotN2
  7. N1NPivot

-----------------------------------------------------
<1: SearchAndDisplay Parent for all Prototypes>
<2: SearchAndDisplay Parent for one Prototype>
<3: SearchAndDisplay Sons for all Objects Traits>
<4: SearchAndDisplay Sons for one Object Traits>
<5: Modify Links Beetween Selected Prototypes>
<6: Modify Links Beetween All Prototypes>

Select a number or q to quit MenuSearchAndDestroy:q
-----------------------------------------------------
Menu SearchAndDestroy Shut Down
-----------------------------------------------------
```

Le savoir attaché aux prototypes représentant `pontage` et `lésion` est plus précis que celui attaché à l'objet portant les comportements partagés par tous les noms. Avant la détermination des comportements partagés, les prototypes lexicaux construits ne partagent pas d'arbre en commun:

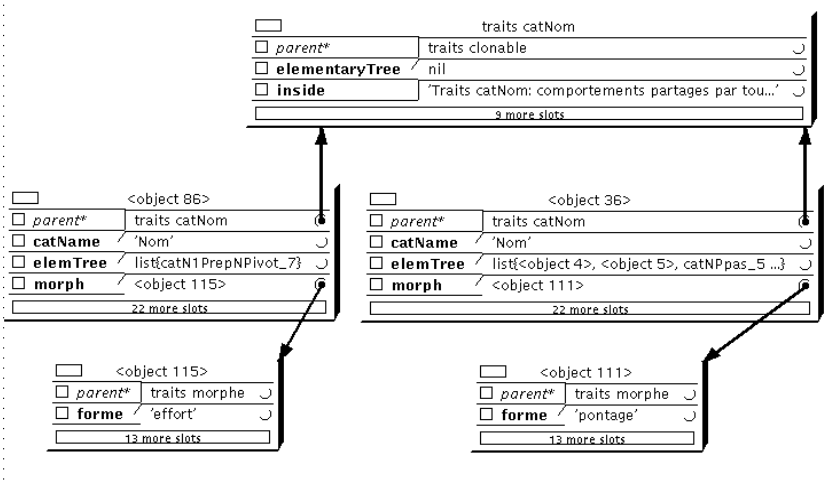


Figure a-2-7.4 : Pas de partage de comportement après génération

Après la détermination des comportements partagés, les prototypes lexicaux délèguent un comportement partagé :

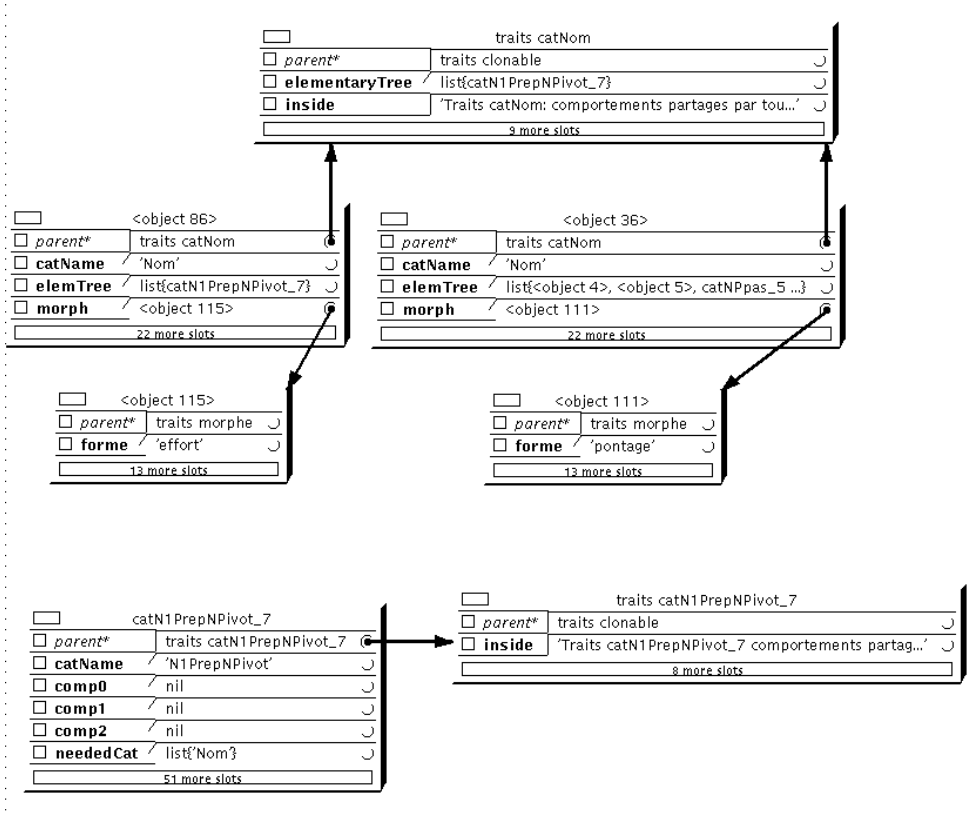


Figure a-2-7.5 : Le classement met en place un partage de comportement

Annexes Partie 2 Chapitre 8

A-2-8.1. `Gaspar` : un analyseur à mots implémenté avec Self-4.0

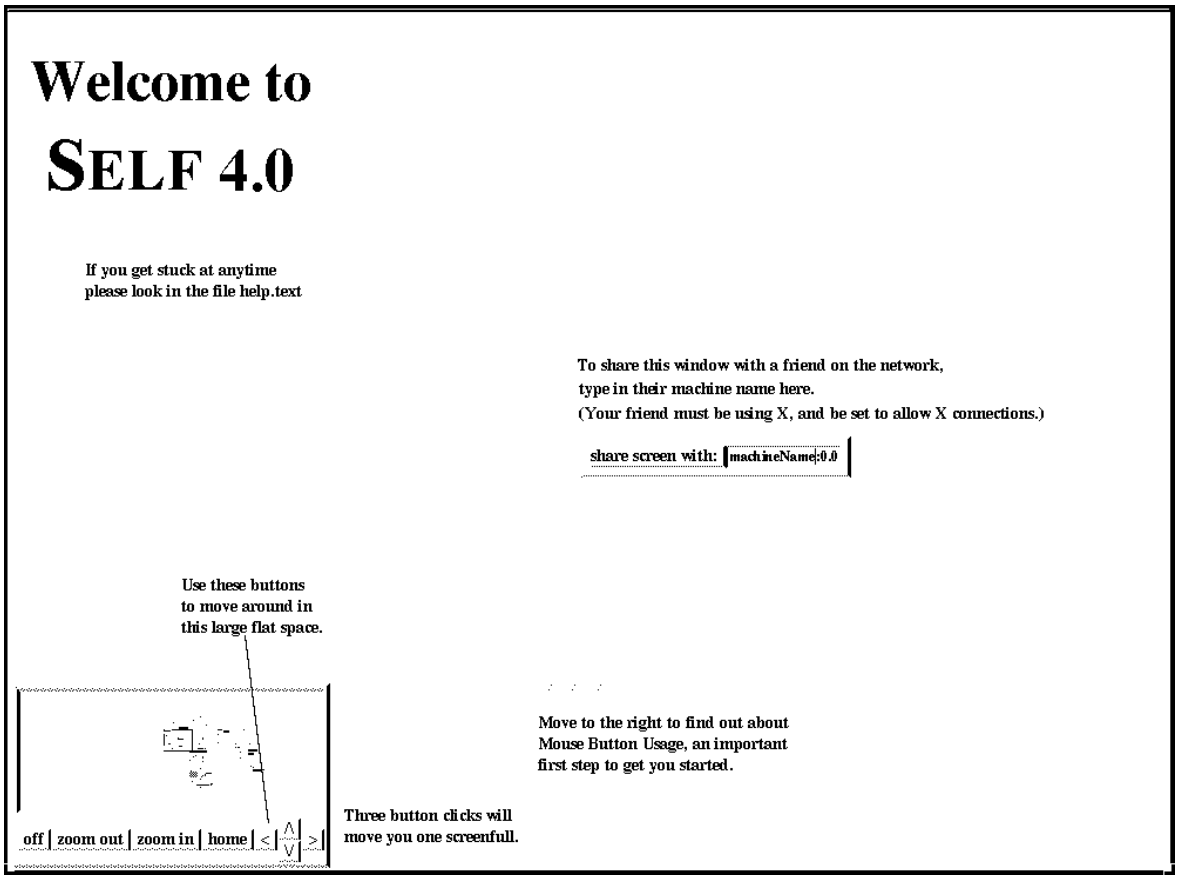


Figure a-2-8.1: "Welcome to Self-4.0"

A-2-8.2. Chargement du corpus et génération dynamique de prototypes lexicaux et de leurs contraintes (sur corpus de test)

Trace de la génération automatique des prototypes

Les figures qui suivent présentent des traces graphiques des prototypes construits lors du processus de génération dont le déroulement est donné infra. Dans la première figure on trouve tous les prototypes construits automatiquement par le processus de génération mis en place (tous ces objets , sur la gauche, ne sont pas expansés). On peut aussi y voir l'attribut de l'objet `Gaspar` qui porte tous les prototypes d'arbres construits (ces objets apparaissent sur la gauche). De même, on y voit (partiellement) un attribut de `Gaspar` portant tous les mots disponibles (`lexiqueGaspar`); les attributs nommés `proto*` correspondent aux prototypes des mots générés (ces objets apparaissent sur la gauche), et enfin on peut y voir une description détaillée du prototype construit associé au mot `pontage`.

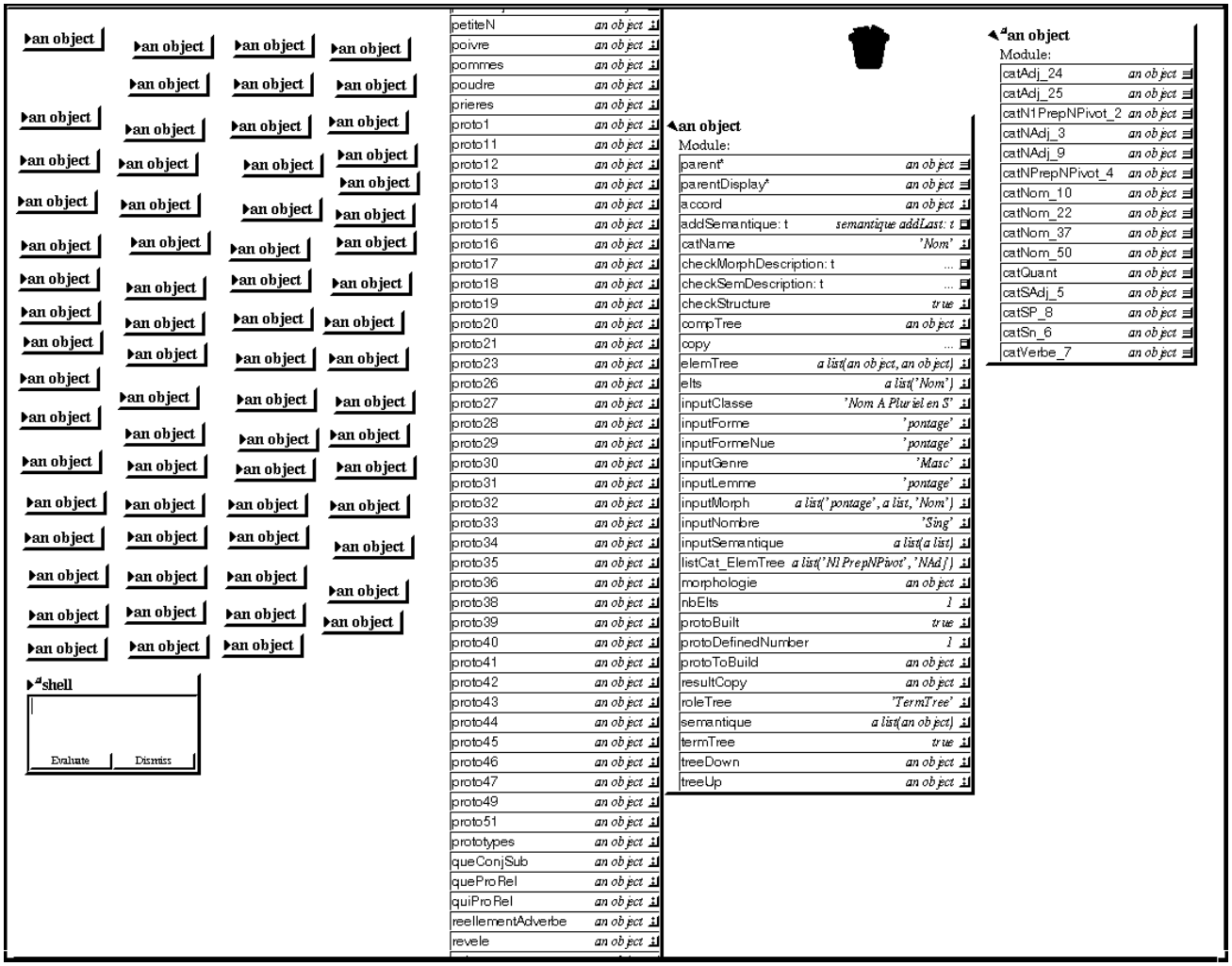


Figure a-2-8.2 : Gaspar et l'interface graphique de Self: prototypes construits (1)

Dans la seconde figure on trouve les deux prototypes d'arbre élémentaire associés au prototype `pontage` construits dans la contrainte 1 qui suit.

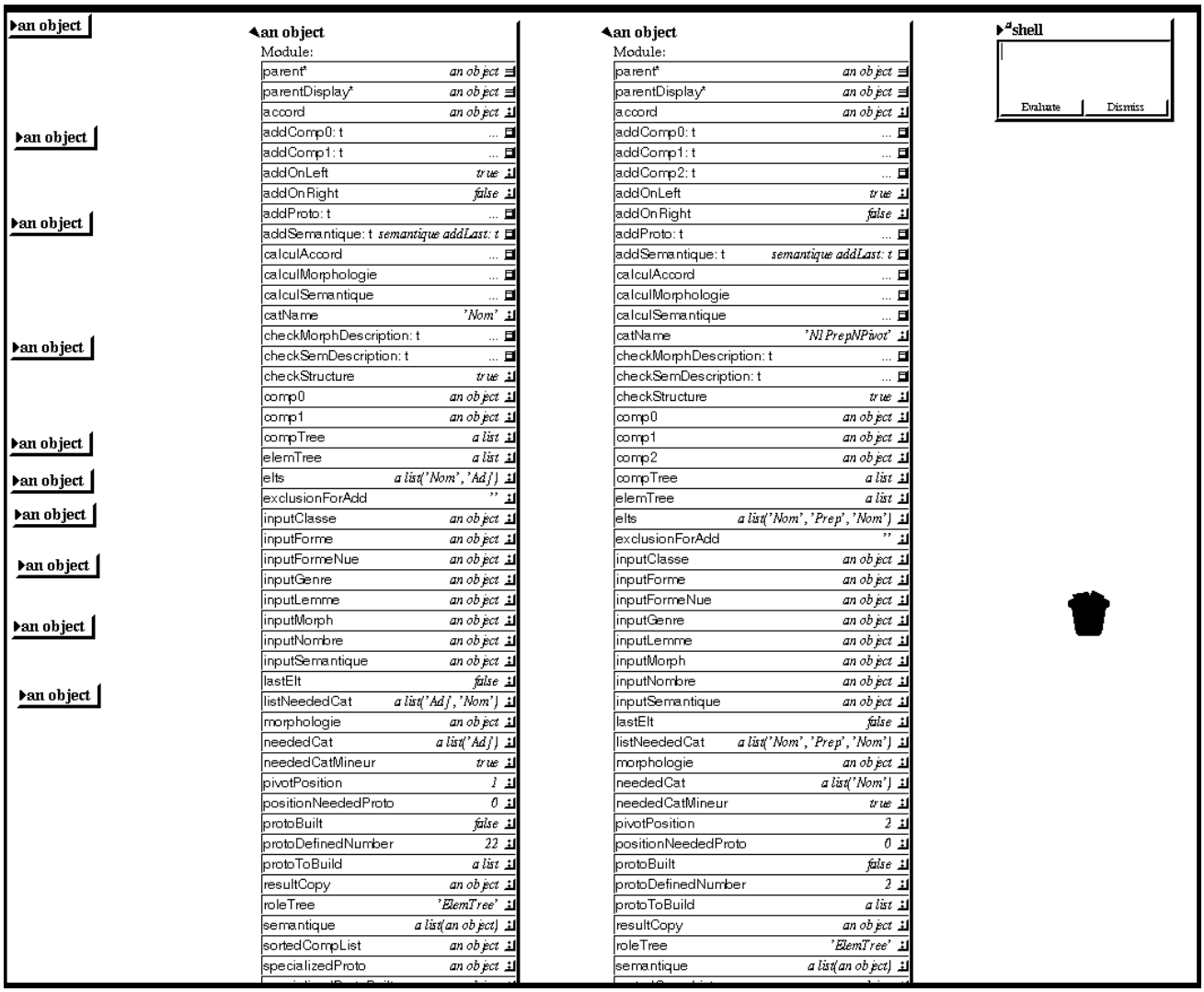


Figure a-2-8.3: Gaspar et l'interface graphique de Self: prototypes construits (2)

Génération dynamique ou Remodelage des prototypes lexicaux et des prototypes d'arbres associés sur la base des savoirs extraits à partir d'un corpus.

Avant cette phase de travail, il n'existe aucune représentation prototypique de mot ou d'arbre. Le processus de génération doit donc tout d'abord créer les objets de toutes pièces puis par clonage et ajustement. La trace ci-dessous ne présente que les deux premières contraintes traitées sur le corpus de test. Elle commence avec le mot pontage. Comme il n'existe aucune représentation prototypique de la catégorie des noms, le processus construit un prototype pour représenter cette catégorie, puis un prototype pour représenter le mot. Pour traiter la deuxième contrainte, le processus construit une représentation du mot lesion par clonage et ajustement en utilisant ces prototypes initialement générés. Il en va de même pour la génération des arbres.

```
Contrainte 1:
--------------
Contrainte sur <pontage|Nom>
--> Verification de l'existence d'un Prototype Lexical : <Nom|pontage>
Nombre de Prototype ayant un attribut <roleTree> :0
--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle, on va donc en creer un avec la forme adequate!
--> Ajout d'une nouvelle categorie mineure
Categorie deja recensee

-Ae-> Arbre Elementaire 1 sur cette Entree lexicale
---------------------------------------------------
--> Presence d'un pivot
Position du Pivot : 2.
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1PrepNPivot> et des composants associes
<a list('Nom', 'Prep', 'NomPivot')>
Nombre de Prototype ayant un attribut <roleTree> :2

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

--------------------------------------
     Prototype Construit
--------------------------------------

        -----------------------------------------------------
        Cat : N1PrepNPivot
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


--Ad--> Arbre Derive 1 sur Cet Arbre Elementaire
-----------------------------------------------
Verification: arbre auxiliaire a adjoindre
Position pour une adjonction eventuelle dans l'arbre elementaire initial : 2
--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj> et des composants associes <a
list('NomPivot', 'Adj')>
Nombre de Prototype ayant un attribut <roleTree> :3

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

--------------------------------------
     Prototype Construit
--------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Role : AuxTree
        Elts : a list('NomPivot', 'Adj')
        NeededCat : a list('NomPivot')
        ListNeededCat : a list('NomPivot', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


--Ad--> Arbre Derive 2 sur Cet Arbre Elementaire
-----------------------------------------------
Verification: arbre auxiliaire pour adjonction
--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NPrepNPivot> et des composants associes
<a list('Adj', 'NPrepNPivot')>
Nombre de Prototype ayant un attribut <roleTree> :4

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

--------------------------------------
     Prototype Construit
--------------------------------------

        -----------------------------------------------------
        Cat : NPrepNPivot
        Role : AuxTree
```

```
        Elts : a list('Adj', 'NPrepNPivot')
        NeededCat : a list('NPrepNPivot')
        ListNeededCat : a list('NPrepNPivot', 'Adj')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


-Ae-> Arbre Elementaire 2 sur cette Entree lexicale
-----------------------------------------------------
--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj> et des composants associes <a
list('Nom', 'Adj')>
Nombre de Prototype ayant un attribut <roleTree> :5

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

-------------------------------------
     Prototype Construit
-------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Role : ElemTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


--Ad--> Arbre Derive 1 sur Cet Arbre Elementaire
-------------------------------------------------
     Verification: arbre d'analyse
--> Cat declencheur: NAdj==>Sn
Arbre Montant a partir de l'arbre elementaire sur la tete NAdj
--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'NAdj')>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'NAdj')>
Nombre de Prototype ayant un attribut <roleTree> :6

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

-------------------------------------
     Prototype Construit
-------------------------------------

        -----------------------------------------------------
        Cat : Sn
        Role : ElemTree
        Elts : a list('Det', 'NAdj')
        NeededCat : a list('NAdj')
        ListNeededCat : a list('NAdj', 'Det')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


--Ad--> Arbre Derive 2 sur Cet Arbre Elementaire
-------------------------------------------------
Verification: arbre auxiliaire a adjoindre
Position pour une adjonction eventuelle dans l'arbre elementaire initial : 1
--> Presence d'un pivot
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <SAdj> et des composants associes <a
list('Adj', 'AdjPivot')>
Nombre de Prototype ayant un attribut <roleTree> :7

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

-------------------------------------
     Prototype Construit
-------------------------------------

        -----------------------------------------------------
        Cat : SAdj
        Role : AuxTree
        Elts : a list('Adj', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Adj', 'Adj')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


-------------------------------------------------------


    Nouvelle contrainte sur le mot <pontage|Nom>


-------------------------------------------------------

        -----------------------------------------------------
        Cat : N1PrepNPivot
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------
```

```
Contraintes sur cet arbre :
-------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot -> gradable:= -

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Prep -> pas de contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Prep -> pas de contrainte de traits definie
           Traits sur Arbre(feuille2) : NomPivot -> gradable:= -

-------------------------------------------------------


    Nouvelle contrainte sur le mot <pontage|Nom>


-------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Role : ElemTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


Contraintes sur cet arbre :
-------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

Recherche et Mise a jour des prototypes concernes

---------------------------------------------------------------------------
 Prototype : [pontage|Nom]
--> Mise a jour Du Champ EltTree De Ce Prototype avec les arbres suivants :
    <a list(an object, an object)>
    <a list('N1PrepNPivot', 'NAdj')>
--> Mise a jour Ok Du Champ EltTree De Ce Prototype
---------------------------------------------------------------------------


Contrainte 2:
--------------
Contrainte sur <lesion|Nom>
--> Verification de l'existence d'un Prototype Lexical : <Nom|lesion>
Nombre de Prototype ayant un attribut <roleTree> :8
--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle, on continue!
--> Mais Il n'existe pas de Prototype qui porte cette forme, on va donc en creer un!

-Ae-> Arbre Elementaire 1 sur cette Entree lexicale
--------------------------------------------------
--> Presence d'un pivot
Position du Pivot : 2.
--> Verification de l'existence d'un prototype portant la valeur categorielle <N1PrepNPivot> et des composants associes
<a list('Nom', 'Prep', 'NomPivot')>
Nombre de Prototype ayant un attribut <roleTree> :9

-----------------------------------
       Prototype Selectionne
-----------------------------------
        -----------------------------------------------------
        Cat : N1PrepNPivot
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


----------------------------------------
     Verification des contraintes
----------------------------------------


Contraintes associees a l'arbre a construire
---------------------------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Prep -> pas de contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Prep -> pas de contrainte de traits definie
           Traits sur Arbre(feuille2) : NomPivot -> pas de contrainte de traits definie

--------------------------------------------------
```

```
         Verification sur prototype0 selectionne
--------------------------------------------------


Contraintes associees a ce prototype
------------------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot -> gradable:= -

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Prep -> pas de contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Prep -> pas de contrainte de traits definie
           Traits sur Arbre(feuille2) : NomPivot -> gradable:= -

     Nouveau type de contraintes: on va ajuster les contraintes sur cet Arbre

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--Ad--> Arbre Derive 1 sur Cet Arbre Elementaire
-----------------------------------------------
Verification: arbre auxiliaire pour adjonction
--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NPrepNPivot> et des composants associes
<a list('Adj', 'NPrepNPivot')>
Nombre de Prototype ayant un attribut <roleTree> :10

-----------------------------------
       Prototype Selectionne
-----------------------------------
        -------------------------------------------------------
        Cat : NPrepNPivot
        Role : AuxTree
        Elts : a list('Adj', 'NPrepNPivot')
        NeededCat : a list('NPrepNPivot')
        ListNeededCat : a list('NPrepNPivot', 'Adj')
        Right-Direction : false
        Left-Direction : true
        -------------------------------------------------------


--------------------------------------
     Verification des contraintes
--------------------------------------


Contraintes associees a l'arbre a construire
--------------------------------------------

     ==> Type sur Tete(arbre) : NPrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NPrepNPivot -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Adj -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : NPrepNPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : NPrepNPivot -> pas de contrainte de traits definie

--------------------------------------------------
         Verification sur prototype0 selectionne
--------------------------------------------------


Contraintes associees a ce prototype
------------------------------------

     ==> Type sur Tete(arbre) : NPrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NPrepNPivot -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Adj -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : NPrepNPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : NPrepNPivot -> pas de contrainte de traits definie

     Prototype 0 adequat avec meme type de contrainte

     Contraintes similaires: on garde l'Arbre prototype0 tel quel

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--Ad--> Arbre Derive 2 sur Cet Arbre Elementaire
-----------------------------------------------
     Verification: arbre d'analyse
--> Cat declencheur: N1PrepNPivot==>Sn
Arbre Montant a partir de l'arbre elementaire sur la tete N1PrepNPivot
--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'N1PrepNPivot')>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'N1PrepNPivot')>
Nombre de Prototype ayant un attribut <roleTree> :10

--> Verification terminee
--> Il n'existe pas de Prototype qui porte cette valeur categorielle avec ces composants, on va en creer un!
--> Definition et Creation du prototype Ok.

--------------------------------------
     Prototype Construit
--------------------------------------
```

```
        ----------------------------------------------------
        Cat : Sn
        Role : ElemTree
        Elts : a list('Det', 'N1PrepNPivot')
        NeededCat : a list('N1PrepNPivot')
        ListNeededCat : a list('N1PrepNPivot', 'Det')
        Right-Direction : false
        Left-Direction : true
        ----------------------------------------------------


--Ad--> Arbre Derive 3 sur Cet Arbre Elementaire
-----------------------------------------------
Verification: arbre auxiliaire a adjoindre
Position pour une adjonction eventuelle dans l'arbre elementaire initial : 2
--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj> et des composants associes <a
list('NomPivot', 'Adj')>
Nombre de Prototype ayant un attribut <roleTree> :11

-----------------------------------
       Prototype Selectionne
-----------------------------------
        ----------------------------------------------------
        Cat : NAdj
        Role : AuxTree
        Elts : a list('NomPivot', 'Adj')
        NeededCat : a list('NomPivot')
        ListNeededCat : a list('NomPivot', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


----------------------------------------
     Verification des contraintes
----------------------------------------


Contraintes associees a l'arbre a construire
--------------------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : NomPivot -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : NomPivot -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

--------------------------------------------------
         Verification sur prototype0 selectionne
--------------------------------------------------


Contraintes associees a ce prototype
------------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> humain:= -

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : NomPivot -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : NomPivot -> humain:= -
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

     Nouveau type de contraintes: on va ajuster les contraintes sur cet Arbre

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

-Ae-> Arbre Elementaire 2 sur cette Entree lexicale
---------------------------------------------------
--> Pas de pivot detecte
Position du Pivot : 0.
--> Verification de l'existence d'un prototype portant la valeur categorielle <NAdj> et des composants associes <a
list('Nom', 'Adj')>
Nombre de Prototype ayant un attribut <roleTree> :12

-----------------------------------
       Prototype Selectionne
-----------------------------------
        ----------------------------------------------------
        Cat : NAdj
        Role : ElemTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


----------------------------------------
     Verification des contraintes
----------------------------------------


Contraintes associees a l'arbre a construire
--------------------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie
```

```
     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

--------------------------------------------------
         Verification sur prototype0 selectionne
--------------------------------------------------


Contraintes associees a ce prototype
------------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

     Prototype 0 adequat avec meme type de contrainte

     Contraintes similaires: on garde l'Arbre prototype0 tel quel

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--Ad--> Arbre Derive 1 sur Cet Arbre Elementaire
-----------------------------------------------
     Verification: arbre d'analyse
--> Cat declencheur: NAdj==>Sn
Arbre Montant a partir de l'arbre elementaire sur la tete NAdj
--> Pas de pivot detecte
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'NAdj')>
--> Verification de l'existence d'un prototype portant la valeur categorielle <Sn> et des composants associes <a
list('Det', 'NAdj')>
Nombre de Prototype ayant un attribut <roleTree> :12

-----------------------------------
       Prototype Selectionne
-----------------------------------
        ----------------------------------------------------
        Cat : Sn
        Role : ParseTree
        Elts : a list('Det', 'NAdj')
        NeededCat : a list('NAdj')
        ListNeededCat : a list('NAdj', 'Det')
        Right-Direction : false
        Left-Direction : true
        ----------------------------------------------------


---------------------------------------
     Verification des contraintes
---------------------------------------


Contraintes associees a l'arbre a construire
--------------------------------------------

     ==> Type sur Tete(arbre) : Sn -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Sn -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Det -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : NAdj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Det -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : NAdj -> pas de contrainte de traits definie

--------------------------------------------------
         Verification sur prototype0 selectionne
--------------------------------------------------


Contraintes associees a ce prototype
------------------------------------------

     ==> Type sur Tete(arbre) : Sn -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Sn -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Det -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : NAdj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Det -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : NAdj -> pas de contrainte de traits definie

     Prototype 0 adequat avec meme type de contrainte

     Contraintes similaires: on garde l'Arbre prototype0 tel quel

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

--Ad--> Arbre Derive 2 sur Cet Arbre Elementaire
-----------------------------------------------
Verification: arbre auxiliaire a adjoindre
Position pour une adjonction eventuelle dans l'arbre elementaire initial : 1
--> Presence d'un pivot
Position du Pivot : 1.
--> Verification de l'existence d'un prototype portant la valeur categorielle <SAdj> et des composants associes <a
list('Adj', 'AdjPivot')>
Nombre de Prototype ayant un attribut <roleTree> :12

-----------------------------------
```

```
       Prototype Selectionne
-----------------------------------
        -------------------------------------------------------
        Cat : SAdj
        Role : AuxTree
        Elts : a list('Adj', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Adj', 'Adj')
        Right-Direction : false
        Left-Direction : true
        -------------------------------------------------------


---------------------------------------
     Verification des contraintes
---------------------------------------


Contraintes associees a l'arbre a construire
---------------------------------------------

     ==> Type sur Tete(arbre) : SAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : SAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Adj -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : AdjPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : AdjPivot -> pas de contrainte de traits definie

-------------------------------------------------
         Verification sur prototype0 selectionne
-------------------------------------------------


Contraintes associees a ce prototype
-------------------------------------

     ==> Type sur Tete(arbre) : SAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : SAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Adj -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : AdjPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : AdjPivot -> pas de contrainte de traits definie

     Prototype 0 adequat avec meme type de contrainte

     Contraintes similaires: on garde l'Arbre prototype0 tel quel

--> Verification terminee
--> Il existe un Prototype qui porte cette valeur categorielle avec ces composants, on continue!

-------------------------------------------------------


    Nouvelle contrainte sur le mot <lesion|Nom>


-------------------------------------------------------

        -------------------------------------------------------
        Cat : N1PrepNPivot
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : false
        Left-Direction : true
        -------------------------------------------------------


Contraintes sur cet arbre :
-------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Prep -> pas de contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Prep -> pas de contrainte de traits definie
           Traits sur Arbre(feuille2) : NomPivot -> pas de contrainte de traits definie

-------------------------------------------------------


    Nouvelle contrainte sur le mot <lesion|Nom>


-------------------------------------------------------

        -------------------------------------------------------
        Cat : NAdj
        Role : ElemTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


Contraintes sur cet arbre :
-------------------------
```

```
     ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie

Recherche et Mise a jour des prototypes concernes

-------------------------------------------------------------------------
 Prototype : [lesion|Nom]
--> Mise a jour Du Champ EltTree De Ce Prototype avec les arbres suivants :
    <a list(an object, an object)>
    <a list('N1PrepNPivot', 'NAdj')>
--> Mise a jour Ok Du Champ EltTree De Ce Prototype
-------------------------------------------------------------------------



etc...
```

A-2-8.3. Classement automatique des prototypes lexicaux (sur corpus de test)

On donne ci-dessous une trace graphique de pôles de comportement construits.

- Il s'agit tout d'abord du pôle de comportement regroupant les adjectifs `marginale, circonflexe, coronaire` et `carotide` (prototypes situés au bas de la figure). L'objet portant l'arbre élémentaire partagé est situé en haut de celle-ci.

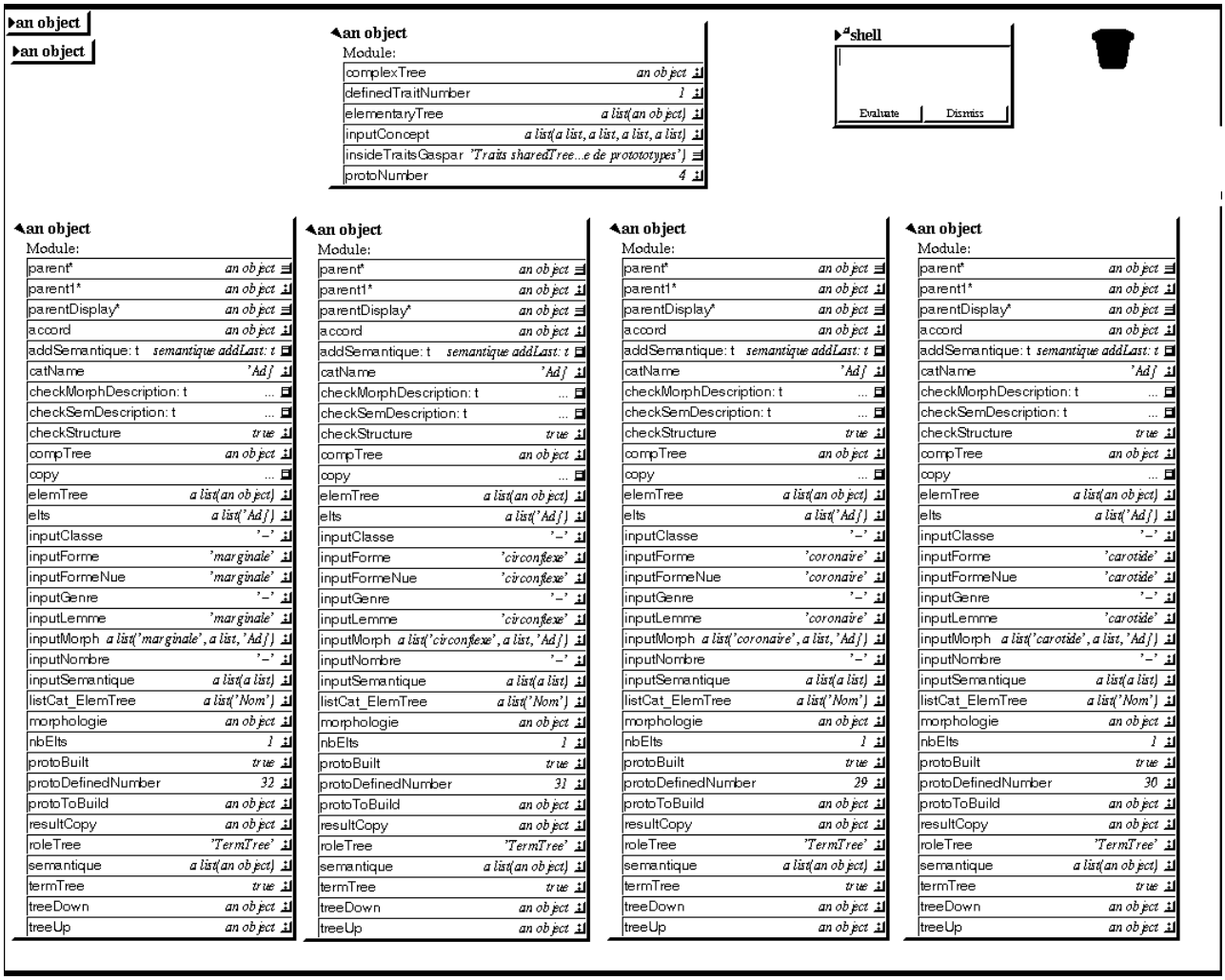


Figure a-2-8.4 : Gaspar et l'interface graphique de Self: prototypes classés (1)

- La figure qui suit présente une trace graphique du pôle de comportement regroupant les noms `risque`, `apparition`, `persistance`, `pas`, `presence`, `absence` et `constitution` (les prototypes associés et situés au bas de la figure sont expansés). L'objet portant l'arbre élémentaire partagé est situé en haut de celle-ci.

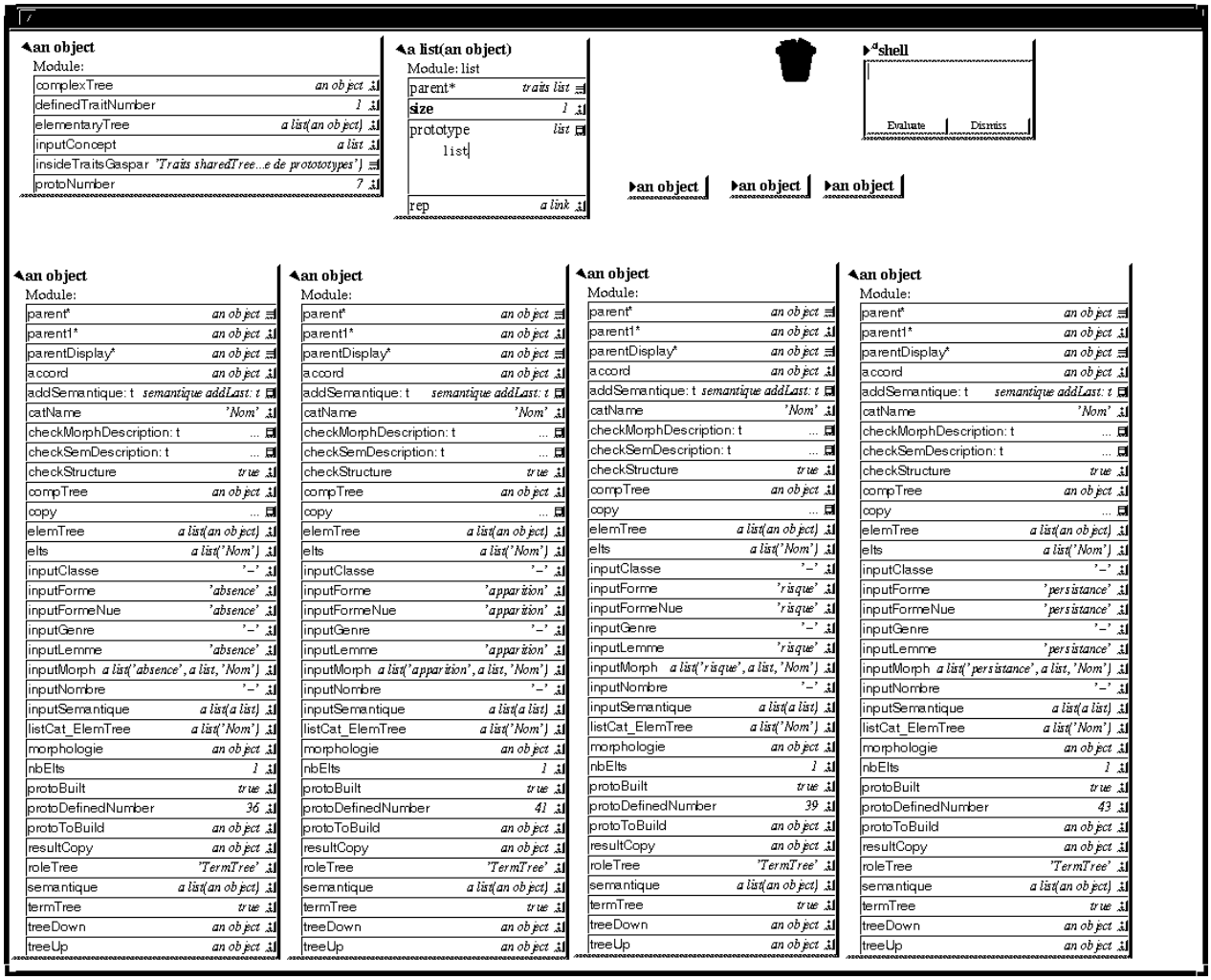


Figure a-2-8.5 : Gaspar et l'interface graphique de Self: prototypes classés (2)

Le mini-corpus de test que nous utilisons est le suivant:

lesion, vraiment, pontage, de, a, avec, sur, gauche, proximal, moyen, distal, anterieur, inferieur, superieur, serre, severe, tres, peu, pas, coronaire, carotide, circonflexe, marginale, long, ulcere, complexe, inhomogene, absence, pas, risque, presence, apparition, constitution, persistance, residuel, significatif, important, negligeable, predominant, premier, double.

Le classement des prototypes lexicaux présentés ci-dessous conduit à la mise en place d'un réseau de prototypes dans lequel certains objets sont porteurs de comportements partagés.

Construction d'un Réseau de Comportements partagés sur la catégorie: Nom

```
==============================================================
Liste des prototypes partageant les memes Comportements:
--------------------------------------------------------
     <Cat:Nom|Forme:persistance|Arbres elementaires:a list(an object)>
           Concept(0)->etat
     <Cat:Nom|Forme:apparition|Arbres elementaires:a list(an object)>
           Concept(0)->proc
           Concept(1)->evt
     <Cat:Nom|Forme:presence|Arbres elementaires:a list(an object)>
           Concept(0)->etat
     <Cat:Nom|Forme:risque|Arbres elementaires:a list(an object)>
           Concept(0)->etat
     <Cat:Nom|Forme:pas|Arbres elementaires:a list(an object)>
           Concept(0)->etat
     <Cat:Nom|Forme:absence|Arbres elementaires:a list(an object)>
           Concept(0)->etat
     <Cat:Nom|Forme:constitution|Arbres elementaires:a list(an object)>
           Concept(0)->proc
==============================================================

-------------------------------------------
    Description du pole de comportements
         partages construit
-------------------------------------------


==============================================================

-----------------------------------------------------------------
     Arbres d'Analyse associes aux Mots de ce Pole.
-----------------------------------------------------------------


-----------------------------------------------------------------
     Mot :persistance|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :apparition|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :presence|Nom
-----------------------------------------------------------------

              Arbre Elementaire
```

```
     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :risque|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :pas|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :absence|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie
```

```
     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe


-----------------------------------------------------------------
     Mot :constitution|Nom
-----------------------------------------------------------------

              Arbre Elementaire

     Ae Cat:Nom
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'SP')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
           Type sur Arbre(feuille1) : SP -> pas de contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
           Traits sur Arbre(feuille1) : SP -> pas de contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------


     Aucun Arbre d'Analyse associe

=================================================================
```

A-2-8.4. La classification évolutive est en marche

A-2-8.4.1. Recherche des comportements partagés et individuels par deux prototypes de mot sélectionnés

```
--> Comportements Syntaxiques Partages par les 2
Prototypes <pontage|Nom><lesion|Nom> :

--------<Comp0>----------------
  Arbre Cat : NAdj
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Nom', 'Adj')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de
contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------

============================================================
========

--------------------------------------------
    Description du pole de comportements
         partages construit
--------------------------------------------


============================================================
========

------------------------------------------------------------
----------
     Arbres d'Analyse associes aux Mots <lesion &
pontage>.
------------------------------------------------------------
----------
```

```
-------------------------------------------------------------
----------
     Mot :pontage|Nom
-------------------------------------------------------------
----------

             Arbre Elementaire

     Ae Cat:NAdj
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==>  Type  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------------
----------
     Al Cat:Sn
     Al Role:ParseTree
     Al Composants:a list('Det', 'NAdj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Sn -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Sn  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Det  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  NAdj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Det  ->  pas  de
contrainte de traits definie
```

```
           Traits sur Arbre(feuille1) : NAdj -> pas de
contraintе de traits definie
-------------------------------------------------------
----------
-------------------------------------------------------
----------
     Al Cat:SAdj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : SAdj -> pas de
contraintе de type definie
     ==> Traits sur Tete(arbre) : SAdj -> pas de
contraintе de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Adj -> pas de
contraintе de type definie
           Type sur Arbre(feuille1) : AdjPivot -> pas de
contraintе de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de
contraintе de traits definie
           Traits sur Arbre(feuille1) : AdjPivot -> pas
de contraintе de traits definie
-------------------------------------------------------
----------

---------------------------------------------
Premier type d'Arbre d'analyse sur ce Pole
---------------------------------------------

-------------------------------------------------------
----------
     Mot :lesion|Nom
-------------------------------------------------------
----------

              Arbre Elementaire

     Ae Cat:NAdj
     Ae Role:ElemTree
     Ae Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de
contraintе de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de
contraintе de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de
contraintе de type definie
```

```
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------------
----------
     Al Cat:Sn
     Al Role:ParseTree
     Al Composants:a list('Det', 'NAdj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Sn -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Sn  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Det  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  NAdj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Det  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  NAdj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------------
----------
-------------------------------------------------------------
----------
     Al Cat:SAdj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==>  Type  sur  Tete(arbre)  :  SAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  SAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Adj  ->  pas  de
contrainte de type definie
           Type sur Arbre(feuille1) : AdjPivot -> pas de
contrainte de type definie
```

```
     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Adj -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : AdjPivot -> pas
de contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------
=======================================================
========

-->  Ensemble  des  Comportements  Syntaxiques  des  2
Prototypes <pontage|Nom><lesion|Nom> :

--------<Comp0>----------------
  Arbre Cat : N1PrepNPivot
  Arbre Composants : a list('Nom', 'Prep', 'Nom')
  Arbre Ordre Composants : a list('Nom', 'Prep', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de
contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot ->
gradable:= -

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Prep -> pas de
contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Prep -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille2) : NomPivot ->
gradable:= -
-------------------------------


--------<Comp1>----------------
  Arbre Cat : NAdj
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Nom', 'Adj')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------
```

```
     ==>  Type  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------


--------<Comp2>----------------
  Arbre Cat : N1PrepNPivot
  Arbre Composants : a list('Nom', 'Prep', 'Nom')
  Arbre Ordre Composants : a list('Nom', 'Prep', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : N1PrepNPivot -> pas de
contrainte de type definie
     ==> Traits sur Tete(arbre) : N1PrepNPivot -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Prep  ->  pas  de
contrainte de type definie
           Type sur Arbre(feuille2) : NomPivot -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Prep  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille2)  :  NomPivot  ->  pas
de contrainte de traits definie
-------------------------------
```

A-2-8.5.2. Construction d'un réseau de comportements partagés sur les prototypes d'arbre élémentaire

Construction d'un Réseau de Comportements partagés sur les Ae de catégorie: Nom

```
==========================================================
========
Liste des prototypes partageant les memes Comportements:
----------------------------------------------------------

--------<Proto0>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= -
-------------------------------


--------<Proto1>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= -
-------------------------------
```

```
--------<Proto2>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= -
-------------------------------


--------<Proto3>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= -
-------------------------------

============================================================
========

-----------------------------------------
```

```
    Description du pole de comportements
         partages construit
-------------------------------------------


=========================================================
========

-------------------------------------------------------
----------
     Arbres d'Analyse associes aux Arbres de ce Pole.
-------------------------------------------------------
----------


-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Adj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Adj -> pas de contrainte
de type definie
     ==> Traits sur Tete(arbre) : Adj -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : AdjPivot -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : AdjPivot -> pas
de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

---------------------------------------------
Premier type d'Arbre d'analyse sur ce Pole
---------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Adj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')
```

```
---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Adj -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Adj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : AdjPivot -> pas de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : AdjPivot -> pas
de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Adj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Adj -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Adj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : AdjPivot -> pas de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : AdjPivot -> pas
de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
```

```
-----------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Adj
     Al Role:AuxTree
     Al Composants:a list('Adj', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Adj -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Adj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : AdjPivot -> pas de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : AdjPivot -> pas
de contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------
=======================================================
========


=======================================================
========
Liste des prototypes partageant les memes Comportements:
-------------------------------------------------------

--------<Proto0>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Quant', 'Nom')
  Arbre Ordre Composants : a list('Quant', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie
```

```
     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Quant -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Nom -> aff-loc

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Quant -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Nom -> pas de
contrainte de traits definie
--------------------------------


--------<Proto1>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Quant', 'Nom')
  Arbre Ordre Composants : a list('Quant', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Quant -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Nom -> aff-loc

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Quant -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Nom -> pas de
contrainte de traits definie
--------------------------------

============================================================
========

-------------------------------------------
    Description du pole de comportements
         partages construit
-------------------------------------------


============================================================
========

------------------------------------------------------------
----------
     Arbres d'Analyse associes aux Arbres de ce Pole.
------------------------------------------------------------
----------


-----------------------------------
```

```
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:NAdj
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==>  Type  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------
----------

---------------------------------------------
Premier type d'Arbre d'analyse sur ce Pole
---------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:NAdj
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==>  Type  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
```

```
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------
=======================================================
========


=======================================================
========
Liste des prototypes partageant les memes Comportements:
-------------------------------------------------------

--------<Proto0>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type sur Arbre(feuille1) : SP -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj ->
gradable:= -
-------------------------------


--------<Proto1>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
```

```
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= +
--------------------------------


--------<Proto2>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

----------------------------------
     Contraintes sur Arbre
----------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= +
--------------------------------


--------<Proto3>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

----------------------------------
     Contraintes sur Arbre
----------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
```

```
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= +
-------------------------------


--------<Proto4>----------------
  Arbre Cat : Nom
  Arbre Composants : a list('Nom', 'Adj')
  Arbre Ordre Composants : a list('Adj', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

----------------------------------
     Contraintes sur Arbre
----------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> aff-loc
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits   sur   Arbre(feuille1)   :   Adj   ->
gradable:= +
-------------------------------

============================================================
========

-------------------------------------------
    Description du pole de comportements
         partages construit
-------------------------------------------


============================================================
========

------------------------------------------------------------
----------
     Arbres d'Analyse associes aux Arbres de ce Pole.
------------------------------------------------------------
----------


-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------
```

```
-------------------------------------------------------
----------
     Al Cat:Nom
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------
----------

---------------------------------------------
Premier type d'Arbre d'analyse sur ce Pole
---------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Nom
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
```

```
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Nom
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==> Traits sur Tete(arbre) : Nom -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Nom
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
```

```
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:Nom
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : Nom -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  Nom  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------
=======================================================
========
```

Construction d'un Réseau de Comportements partagés sur les Ae de catégorie: SP

```
=========================================================
========
Liste des prototypes partageant les memes Comportements:
---------------------------------------------------------

--------<Proto0>----------------
  Arbre Cat : SP
  Arbre Composants : a list('Prep', 'Nom')
  Arbre Ordre Composants : a list('Prep', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

----------------------------------
     Contraintes sur Arbre
----------------------------------

     ==> Type sur Tete(arbre) : SP -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  SP  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Prep  ->  pas  de
contrainte de type definie
           Type sur Arbre(feuille1) : Nom -> loc-lieu ||
vaisseau

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Prep  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Nom  ->  pas  de
contrainte de traits definie
-------------------------------


--------<Proto1>----------------
  Arbre Cat : SP
  Arbre Composants : a list('Prep', 'Nom')
  Arbre Ordre Composants : a list('Prep', 'Nom')
  Arbre Coherence : true
  Arbre Role : ElemTree

----------------------------------
     Contraintes sur Arbre
----------------------------------

     ==> Type sur Tete(arbre) : SP -> pas de contrainte
de type definie
     ==>  Traits  sur  Tete(arbre)  :  SP  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Prep  ->  pas  de
contrainte de type definie
```

```
           Type sur Arbre(feuille1) : Nom -> loc-lieu ||
vaisseau

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Prep -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Nom -> pas de
contrainte de traits definie
-------------------------------

==========================================================
========

--------------------------------------------
    Description du pole de comportements
         partages construit
--------------------------------------------


==========================================================
========

----------------------------------------------------------
----------
     Arbres d'Analyse associes aux Arbres de ce Pole.
----------------------------------------------------------
----------


-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

----------------------------------------------------------
----------
     Al Cat:NAdj
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

---------------------------------
     Contraintes sur Arbre
---------------------------------

     ==> Type sur Tete(arbre) : NAdj -> pas de
contrainte de type definie
     ==> Traits sur Tete(arbre) : NAdj -> pas de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type sur Arbre(feuille0) : Nom -> pas de
contrainte de type definie
           Type sur Arbre(feuille1) : Adj -> pas de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits sur Arbre(feuille0) : Nom -> pas de
contrainte de traits definie
           Traits sur Arbre(feuille1) : Adj -> pas de
contrainte de traits definie
----------------------------------------------------------
----------
```

```
-------------------------------------------
Premier type d'Arbre d'analyse sur ce Pole
-------------------------------------------

-----------------------------------
      Arbres d'Analyse Associes
-----------------------------------

-------------------------------------------------------
----------
     Al Cat:NAdj
     Al Role:AuxTree
     Al Composants:a list('Nom', 'Adj')

--------------------------------
     Contraintes sur Arbre
--------------------------------

     ==>  Type  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de type definie
     ==>  Traits  sur  Tete(arbre)  :  NAdj  ->  pas  de
contrainte de traits definie

     ==> Types sur Feuilles(arbre) :
           Type  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de type definie
           Type  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de type definie

     ==> Traits sur Feuilles(arbre) :
           Traits  sur  Arbre(feuille0)  :  Nom  ->  pas  de
contrainte de traits definie
           Traits  sur  Arbre(feuille1)  :  Adj  ->  pas  de
contrainte de traits definie
-------------------------------------------------------
----------

-------------------------------------------
Arbre d'analyse deja rencontre sur ce Pole
-------------------------------------------
=======================================================
========
```

A-3-9.2.7. Analyse avec Gaspar

Parseur Gaspar

La figure qui suit présente les activités mises en place pour l'analyse.

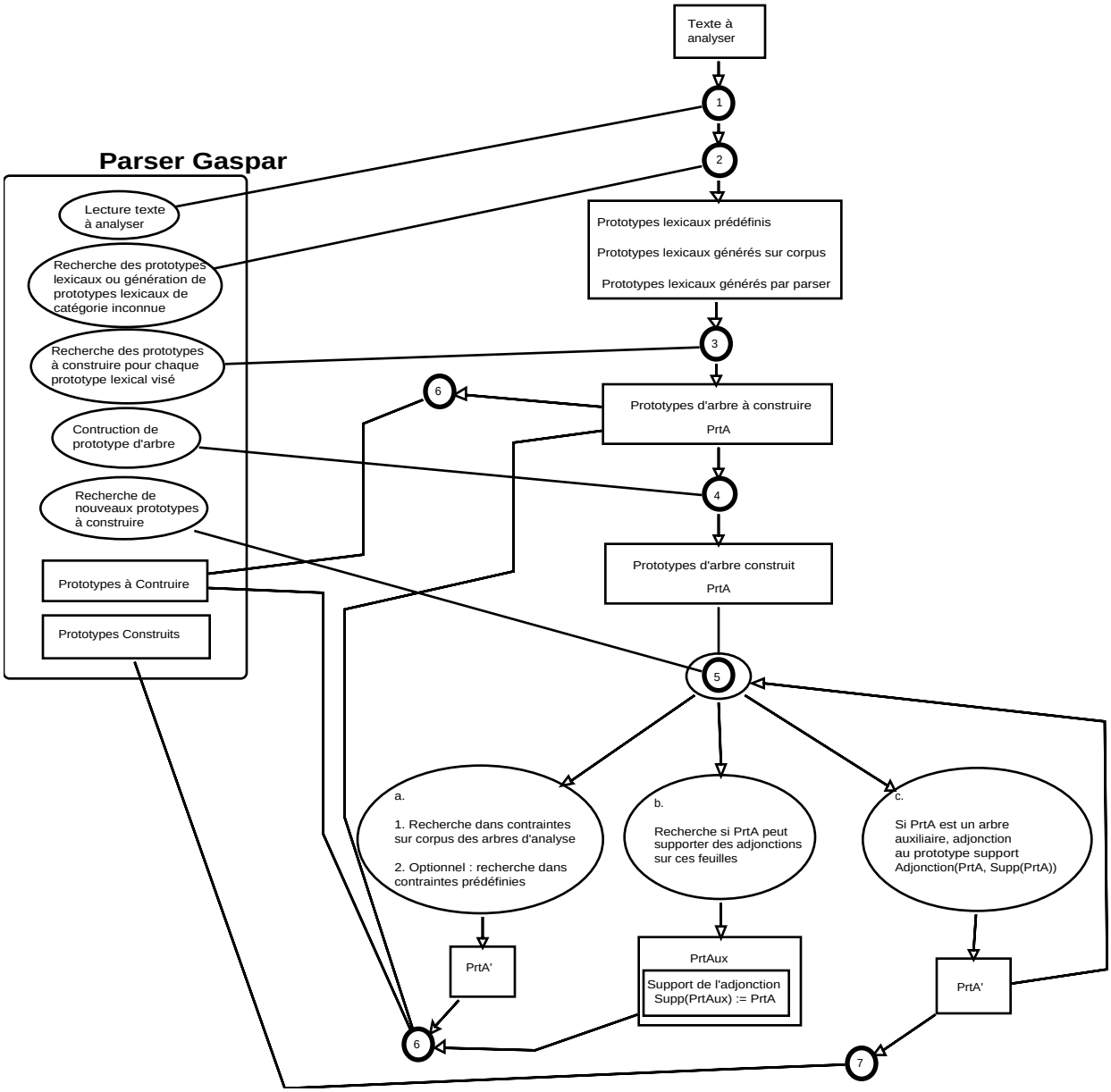


Figure a-2-9.1: Analyse avec Gaspar

Analyse sur "une petite brise la glace"

```
          Debut Analyse....

Noeuds initiaux:

Premier Type de Categorie a Cette Position

(0-(Det|une)-1)

Premier Type de Categorie a Cette Position

(1-(Adj|petite)-2)

Nouveau Type de Categorie a Cette Meme Position

(1-(Nom|petite)-2)

Premier Type de Categorie a Cette Position

(2-(Verbe|brise)-3)

Nouveau Type de Categorie a Cette Meme Position

(2-(Nom|brise)-3)

Premier Type de Categorie a Cette Position

(3-(Det|la)-4)

Nouveau Type de Categorie a Cette Meme Position

(3-(Clit|la)-4)

Premier Type de Categorie a Cette Position

(4-(Verbe|glace)-5)

Nouveau Type de Categorie a Cette Meme Position

(4-(Nom|glace)-5)

Recherche des prototypes a construire pour chacun
des prototypes initiaux construits.
-------------------------------------------------


-------------------------------------------------------------------------------
          Prototype initial:  Arbre <1>
                   <une|Det>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
          Prototype initial:  Arbre <2>
                   <petite|Adj>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object)

-------------------------------------------------------------------------------
          Prototype initial:  Arbre <3>
                   <petite|Nom>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(9 elements)

     Nouveau prototype a construire :

                                        Arbre <12>
     1-(Cat:Sn|Forme:(Det:*|Nom:petite)|Forme Nue:petite|RoleTree:ElemTree)-2

     Nouveau prototype a construire :

                                        Arbre <13>
     1-(Cat:Nom|Forme:(Nom:petite|PRel:*)|Forme Nue:petite|RoleTree:ElemTree)-2

-------------------------------------------------------------------------------
          Prototype initial:  Arbre <4>
                   <brise|Verbe>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object, an object)

     Nouveau prototype a construire :
```

```
                                        Arbre <14>
     2-(Cat:Sv|Forme:(Verbe:brise|Sn:*)|Forme Nue:brise|RoleTree:ElemTree)-3

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <5>
                     <brise|Nom>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(9 elements)

     Nouveau prototype a construire :

                                        Arbre <15>
     2-(Cat:AdjN|Forme:(Adj:*|Nom:brise)|Forme Nue:brise|RoleTree:ElemTree)-3

     Nouveau prototype a construire :

                                        Arbre <16>
     2-(Cat:Nom|Forme:(Nom:brise|PRel:*)|Forme Nue:brise|RoleTree:ElemTree)-3

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <6>
                     <la|Det>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <8>
                     <la|Clit>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <10>
                     <glace|Verbe>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object, an object)

     Nouveau prototype a construire :

                                        Arbre <17>
     4-(Cat:Sv|Forme:(Clit:*|Verbe:glace)|Forme Nue:glace|RoleTree:ElemTree)-5

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <11>
                     <glace|Nom>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(9 elements)

     Nouveau prototype a construire :

                                        Arbre <18>
     4-(Cat:Sn|Forme:(Det:*|Nom:glace)|Forme Nue:glace|RoleTree:ElemTree)-5

Fin de la recherche des prototypes a construire.
-----------------------------------------------


1.Recherche des prototypes a adjoindre aux prototypes
 a construire disponibles.
2.Initialisation des nouveaux prototypes a construire
 a partir des prototypes construits.


-----------------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 12)-2
-----------------------------------------------------------------------------

        -------------------------------------------------------
        Cat : Sn
        Forme : (Det:*|Nom:petite)
        Role : ElemTree
        Elts : a list('Det', 'Nom')
        NeededCat : a list('Det')
        ListNeededCat : a list('Nom', 'Det')
        Right-Direction : false
        Left-Direction : true
        -------------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
```

```
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <1>
     0-(Cat:Det|Forme construite:une|Forme nue:une|RoleTree:TermTree)-1


     Prototype resultant:

                                        Arbre <12>
     0-(Cat:Sn|Forme construite:(Det:une|Nom:petite)|Forme nue:une petite|NeededCat:a list|RoleTree:ElemTree)-2

   Prototype construit:

                                        Arbre <12>
     0-(Cat:Sn|Forme construite:(Det:une|Nom:petite)|Forme nue:une petite|NeededCat:a list|RoleTree:ElemTree)-2

   Ce prototype n'est pas declencheur de nouveau prototype.

Il existe des contraintes predefinies par defaut qui vont etre associees a ce nouveau prototype construit

     Nouveau prototype a construire:

                                        Arbre <19>
     0-(Cat:P|Forme:(Sn:(Det:une|Nom:petite)|Sv:*)|Forme Nue:une petite|RoleTree:ParseTree)-2


--------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
--------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-------------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 13)-2
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:petite|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 14)-3
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:brise|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 15)-3
-------------------------------------------------------------------------
```

```
        -----------------------------------------------------
        Cat : AdjN
        Forme : (Adj:*|Nom:brise)
        Role : ElemTree
        Elts : a list('Adj', 'Nom')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <2>
     1-(Cat:Adj|Forme construite:petite|Forme nue:petite|RoleTree:TermTree)-2


     Prototype resultant:

                                        Arbre <15>
     1-(Cat:AdjN|Forme construite:(Adj:petite|Nom:brise)|Forme nue:petite brise|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                        Arbre <15>
     1-(Cat:AdjN|Forme construite:(Adj:petite|Nom:brise)|Forme nue:petite brise|NeededCat:a list|RoleTree:ElemTree)-3

   Ce prototype n'est pas declencheur de nouveau prototype.

Il existe des contraintes predefinies par defaut qui vont etre associees a ce nouveau prototype construit

     Nouveau prototype a construire:

                                        Arbre <20>
     1-(Cat:Sn|Forme:(Det:*|AdjN:(Adj:petite|Nom:brise))|Forme Nue:petite brise|RoleTree:ParseTree)-3


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 16)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:brise|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 4-(Arbre 17)-5
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Clit:*|Verbe:glace)
        Role : ElemTree
        Elts : a list('Clit', 'Verbe')
        NeededCat : a list('Clit')
        ListNeededCat : a list('Verbe', 'Clit')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
```

```
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <8>
     3-(Cat:Clit|Forme construite:la|Forme nue:la|RoleTree:TermTree)-4


     Prototype resultant:

                                        Arbre <17>
     3-(Cat:Sv|Forme construite:(Clit:la|Verbe:glace)|Forme nue:la glace|NeededCat:a list|RoleTree:ElemTree)-5

   Prototype construit:

                                        Arbre <17>
     3-(Cat:Sv|Forme construite:(Clit:la|Verbe:glace)|Forme nue:la glace|NeededCat:a list|RoleTree:ElemTree)-5

   Ce prototype n'est pas declencheur de nouveau prototype.

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-----------------------------------------------------------------------
   Prototype extrait a construire : 4-(Arbre 18)-5
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sn
        Forme : (Det:*|Nom:glace)
        Role : ElemTree
        Elts : a list('Det', 'Nom')
        NeededCat : a list('Det')
        ListNeededCat : a list('Nom', 'Det')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <6>
     3-(Cat:Det|Forme construite:la|Forme nue:la|RoleTree:TermTree)-4


     Prototype resultant:

                                        Arbre <18>
     3-(Cat:Sn|Forme construite:(Det:la|Nom:glace)|Forme nue:la glace|NeededCat:a list|RoleTree:ElemTree)-5

   Prototype construit:

                                        Arbre <18>
     3-(Cat:Sn|Forme construite:(Det:la|Nom:glace)|Forme nue:la glace|NeededCat:a list|RoleTree:ElemTree)-5

   Ce prototype n'est pas declencheur de nouveau prototype.

Il existe des contraintes predefinies par defaut qui vont etre associees a ce nouveau prototype construit

-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 19)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|Nom:petite)|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
```

```
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 20)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sn
        Forme : (Det:*|AdjN:(Adj:petite|Nom:brise))
        Role : ParseTree
        Elts : a list('Det', 'AdjN')
        NeededCat : a list('Det')
        ListNeededCat : a list('AdjN', 'Det')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <1>
     0-(Cat:Det|Forme construite:une|Forme nue:une|RoleTree:TermTree)-1


     Prototype resultant:

                                        Arbre <20>
     0-(Cat:Sn|Forme    construite:(Det:une|AdjN:(Adj:petite|Nom:brise))|Forme    nue:une    petite    brise|NeededCat:a
list|RoleTree:ParseTree)-3

   Prototype construit:

                                        Arbre <20>
     0-(Cat:Sn|Forme    construite:(Det:une|AdjN:(Adj:petite|Nom:brise))|Forme    nue:une    petite    brise|NeededCat:a
list|RoleTree:ParseTree)-3

   Ce prototype n'est pas declencheur de nouveau prototype.

Il existe des contraintes predefinies par defaut qui vont etre associees a ce nouveau prototype construit

     Nouveau prototype a construire:

                                        Arbre <21>
     0-(Cat:P|Forme:(Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:*)|Forme Nue:une petite brise|RoleTree:ParseTree)-3


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 13)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:petite|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie
```

```
        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 14)-3
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:brise|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <18>
     3-(Cat:Sn|Forme construite:(Det:la|Nom:glace)|Forme nue:la glace|RoleTree:ElemTree)-5


     Prototype resultant:

                                        Arbre <14>
     2-(Cat:Sv|Forme     construite:(Verbe:brise|Sn:(Det:la|Nom:glace))|Forme     nue:brise     la     glace|NeededCat:a
list|RoleTree:ElemTree)-5

   Prototype construit:

                                        Arbre <14>
     2-(Cat:Sv|Forme     construite:(Verbe:brise|Sn:(Det:la|Nom:glace))|Forme     nue:brise     la     glace|NeededCat:a
list|RoleTree:ElemTree)-5

   Ce prototype n'est pas declencheur de nouveau prototype.

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


   Ce prototype n'est pas declencheur de nouveau prototype d'arbre auxiliaire.


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 16)-3
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:brise|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 21)-3
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
```

```
        ----------------------------------------------------

        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <17>
     3-(Cat:Sv|Forme construite:(Clit:la|Verbe:glace)|Forme nue:la glace|RoleTree:ElemTree)-5


     Prototype resultant:

                                        Arbre <21>
     0-(Cat:P|Forme  construite:(Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:(Clit:la|Verbe:glace))|Forme  nue:une  petite
brise la glace|NeededCat:a list|RoleTree:ParseTree)-5

   Prototype construit:

                                        Arbre <21>
     0-(Cat:P|Forme  construite:(Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:(Clit:la|Verbe:glace))|Forme  nue:une  petite
brise la glace|NeededCat:a list|RoleTree:ParseTree)-5

-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 22)-3
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:brise|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 19)-2
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|Nom:petite)|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <14>
     2-(Cat:Sv|Forme construite:(Verbe:brise|Sn:(Det:la|Nom:glace))|Forme nue:brise la glace|RoleTree:ElemTree)-5


     Prototype resultant:

                                        Arbre <19>
     0-(Cat:P|Forme   construite:(Sn:(Det:une|Nom:petite)|Sv:(Verbe:brise|Sn:(Det:la|Nom:glace)))|Forme   nue:une   petite
brise la glace|NeededCat:a list|RoleTree:ParseTree)-5

   Prototype construit:

                                        Arbre <19>
```

```
     0-(Cat:P|Forme  construite:(Sn:(Det:une|Nom:petite)|Sv:(Verbe:brise|Sn:(Det:la|Nom:glace)))|Forme  nue:une  petite
brise la glace|NeededCat:a list|RoleTree:ParseTree)-5

-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 13)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:petite|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 16)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:brise|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 23)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 24)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|Nom:petite)|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
```

```
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 22)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:brise|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 13)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:petite|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 16)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:brise|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 23)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|AdjN:(Adj:petite|Nom:brise))|Sv:*)
        Role : ParseTree
```

```
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 24)-2
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : P
        Forme : (Sn:(Det:une|Nom:petite)|Sv:*)
        Role : ParseTree
        Elts : a list('Sn', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('Sn', 'Sv')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 22)-3
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:brise|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 13)-2
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:petite|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------
```

```
-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 16)-3
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:brise|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------



            ...Fin Analyse

Statistiques
------------


 Prototypes Initiaux:9

 Prototypes a Construire Declenches:10

 Prototypes Extraits:27

 Prototypes Construits:17

Le fichier /home/fleury/Self/source/Self-4.0/Phase2/trace/TraceParser.self est disponible pour une lecture de la trace
de l'analyse.


Le fichier /home/fleury/Self/source/Self-4.0/Phase2/trace/TraceResult.self est disponible pour une lecture des resultats
de l'analyse.
```

Analyse sur "analyseur à prototypes" avec construction automatique de nouveaux prototypes de mot

Dans cette phase d'analyse, le parseur Gaspar doit analyser la séquence "analyseur à prototypes". Cette séquence de mots contient une forme lexicale à laquelle le parseur ne va pas pouvoir, dans un premier temps, associer une ou plusieurs représentations prototypiques de mot. Il va associer, par défaut, à cette forme autant de représentations prototypiques de mot qu'il existe de catégorie pouvant potentiellement être associée à un mot. Il associe de plus à cette forme une représentation prototypique de mot de catégorie inconnue. L'analyse va donc se dérouler en utilisant ces savoirs construits automatiquement par le parseur en utilisant les savoirs déjà représentés (via les objets construits).

```
          Debut Analyse....

Noeuds initiaux:

     ---------------------------------------------------------------------------
     Aucun prototype lexical ne porte une telle forme : analyseur.
     1. Creation de pseudo-prototypes lexicaux pour chaque categorie grammaticale connue
     2. Creation d'un prototype lexical de categorie inconnue
     ---------------------------------------------------------------------------

          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Nom|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Verbe|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Clit|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Det|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <ConjSub|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Adj|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Prep|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <ProRel|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Pro|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Vinf|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Adverbe|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <unknownCat|analyseur>
          ---------------------------------------------------------------------------
          Creation du prototype lexical : <Quant|analyseur>

Premier Type de Categorie a Cette Position

(0-(Nom|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Verbe|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Clit|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Det|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(ConjSub|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Adj|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Prep|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(ProRel|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Pro|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Adverbe|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(unknownCat|analyseur)-1)

Nouveau Type de Categorie a Cette Meme Position

(0-(Quant|analyseur)-1)
```

```
Premier Type de Categorie a Cette Position

(1-(Prep|a)-2)

Premier Type de Categorie a Cette Position

(2-(Nom|prototypes)-3)

Recherche des prototypes a construire pour chacun
des prototypes initiaux construits.
-------------------------------------------------


-------------------------------------------------------------------------------
           Prototype initial:  Arbre <1>
                     <analyseur|Nom>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(9 elements)

     Nouveau prototype a construire :

                                        Arbre <38>
     0-(Cat:NPivotPrepN2|Forme:(Nom:analyseur|Prep:*|Nom:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

     Nouveau prototype a construire :

                                        Arbre <39>
     0-(Cat:NPivotPrepV|Forme:(Nom:analyseur|Prep:*|Verbe:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

     Nouveau prototype a construire :

                                        Arbre <40>
     0-(Cat:Nom|Forme:(Nom:analyseur|PRel:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <5>
                     <analyseur|Verbe>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object, an object, an object)

     Nouveau prototype a construire :

                                        Arbre <41>
     0-(Cat:Sv|Forme:(Verbe:analyseur|Sn:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

     Nouveau prototype a construire :

                                        Arbre <42>
     0-(Cat:Sv|Forme:(Verbe:analyseur|SP:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <9>
                     <analyseur|Clit>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <11>
                     <analyseur|Det>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <13>
                     <analyseur|ConjSub>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <15>
                     <analyseur|Adj>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object)

-------------------------------------------------------------------------------
```

```
           Prototype initial:  Arbre <19>
                     <analyseur|Prep>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <23>
                     <analyseur|ProRel>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(an object, an object)

     Nouveau prototype a construire :

                                        Arbre <43>
     0-(Cat:PRel|Forme:(ProRel:analyseur|Sv:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

     Nouveau prototype a construire :

                                        Arbre <44>
     0-(Cat:PRel|Forme:(ProRel:analyseur|P:*)|Forme Nue:analyseur|RoleTree:ElemTree)-1

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <25>
                     <analyseur|Pro>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <27>
                     <analyseur|Adverbe>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <31>
                     <analyseur|unknownCat>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <33>
                     <analyseur|Quant>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)

Aucun Arbre Elementaire n'est a construire

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <35>
                     <a|Prep>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Liste des prototypes a construire: a list(an object)

     Nouveau prototype a construire :

                                        Arbre <45>
     1-(Cat:SP|Forme:(Prep:a)|Forme Nue:a|RoleTree:ElemTree)-2

-------------------------------------------------------------------------------
           Prototype initial:  Arbre <37>
                     <prototypes|Nom>
-------------------------------------------------------------------------------

Aucun arbre elementaire n'est associe a ce Prototype. Verification dans Le Dictionnaire des Contraintes.

Aucune contrainte n'est attachee a ce prototype dans le Dictionnaire des Contraintes

Il existe des contraintes predefinies par defaut qui vont etre associees a ce prototype lexical
Liste des prototypes a construire: a list(9 elements)

     Nouveau prototype a construire :
```

```
                                        Arbre <46>
     2-(Cat:SP|Forme:(Prep:*|Nom:prototypes)|Forme Nue:prototypes|RoleTree:ElemTree)-3

Fin de la recherche des prototypes a construire.
-----------------------------------------------


1.Recherche des prototypes a adjoindre aux prototypes
 a construire disponibles.
2.Initialisation des nouveaux prototypes a construire
 a partir des prototypes construits.


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 38)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepN2
        Forme : (Nom:analyseur|Prep:*|Nom:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Prep')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <35>
     1-(Cat:Prep|Forme construite:a|Forme nue:a|RoleTree:TermTree)-2


     Prototype resultant:

                                        Arbre <38>
     0-(Cat:NPivotPrepN2|Forme       construite:(Nom:analyseur|Prep:a|Nom:*)|Forme       nue:analyseur       a|NeededCat:a
list('Nom')|RoleTree:ElemTree)-2

     Prototype Disponible Retenu:

                                        Arbre <37>
     2-(Cat:Nom|Forme construite:prototypes|Forme nue:prototypes|RoleTree:TermTree)-3


     Prototype resultant:

                                        Arbre <38>
     0-(Cat:NPivotPrepN2|Forme        construite:(Nom:analyseur|Prep:a|Nom:prototypes)|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                        Arbre <38>
     0-(Cat:NPivotPrepN2|Forme        construite:(Nom:analyseur|Prep:a|Nom:prototypes)|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3


-------------------------------------------------------------------------------
Le prototype construit a aussi permis la construction de prototypes specialises.
Recherche des prototypes a construire a partir de ces derniers.
-------------------------------------------------------------------------------

-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 39)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepV
        Forme : (Nom:analyseur|Prep:*|Verbe:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Vinf')
        NeededCat : a list('Prep')
        ListNeededCat : a list('Nom', 'Prep', 'Vinf')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <35>
```

```
     1-(Cat:Prep|Forme construite:a|Forme nue:a|RoleTree:TermTree)-2

     Prototype resultant:

                                        Arbre <39>
     0-(Cat:NPivotPrepV|Forme      construite:(Nom:analyseur|Prep:a|Verbe:*)|Forme      nue:analyseur      a|NeededCat:a
list('Vinf')|RoleTree:ElemTree)-2

-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 40)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:analyseur|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 41)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 42)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|SP:*)
        Role : ElemTree
        Elts : a list('Verbe', 'SP')
        NeededCat : a list('SP')
        ListNeededCat : a list('Verbe', 'SP')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 43)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|Sv:*)
        Role : ElemTree
        Elts : a list('ProRel', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('ProRel', 'Sv')
        Right-Direction : true
```

```
        Left-Direction : false
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 44)-1
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|P:*)
        Role : ElemTree
        Elts : a list('ProRel', 'P')
        NeededCat : a list('P')
        ListNeededCat : a list('ProRel', 'P')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        --------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 45)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : SP
        Forme : (Prep:a)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : SP -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : SP -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Prep -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Nom -> aff-loc

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Prep -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Nom -> pas de contrainte de traits definie
        --------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <37>
     2-(Cat:Nom|Forme construite:prototypes|Forme nue:prototypes|RoleTree:TermTree)-3


     Prototype resultant:

                                        Arbre <45>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                        Arbre <45>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Ce prototype n'est pas declencheur de nouveau prototype.

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------


        Prototype d'arbre auxiliaire potentiel a construire :

        -----------------------------------------------------
```

```
        Cat : NAdj
        Forme : ?
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        Prototype d'arbre auxiliaire potentiel a construire :

        -------------------------------------------------------
        Cat : Nom
        Forme : ?
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        Prototype d'arbre auxiliaire potentiel a construire :

        -------------------------------------------------------
        Cat : NAdj
        Forme : ?
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


Arbre a0 Ch2 ===> Adjonction possible sur arbre construit en Position 1

-----------------------------------------------------------------------
1. Preconstruction de l'arbre auxiliaire attache au prototype construit
2. Si l'arbre auxiliaire est preconstruit, il sera ajoute a la liste
   des prototypes a construire. Il portera les prototypes dans lesquels
   il peut etre adjoint
-----------------------------------------------------------------------


       Sub >> Prototype support de la substitution
       -------------------------------------------
          Categorie : NAdj|Forme:?|Role:AuxTree|Elts:a list('Nom', 'Adj')

       Sub >> Prototype utilise pour la substitution
       ---------------------------------------------
          Categorie : SP|Forme:(Prep:a|Nom:prototypes)|Role:ElemTree|Elts:a list('Prep', 'Nom')

        Sub >> Prototype Resultant de la substitution
        ---------------------------------------------
          Categorie : NAdj|Forme:(Nom:prototypes)|Role:AuxTree|Elts:a list('Nom', 'Adj')

     Nouveau Prototype (arbre auxiliaire) a construire
     -------------------------------------------------
                                        Arbre <50>
     2-(Cat:NAdj|Forme:(Nom:prototypes)|Forme Nue:prototypes|RoleTree:AuxTree)-3


Arbre a1 Ch2 ===> Adjonction possible sur arbre construit en Position 1

-----------------------------------------------------------------------
1. Preconstruction de l'arbre auxiliaire attache au prototype construit
2. Si l'arbre auxiliaire est preconstruit, il sera ajoute a la liste
   des prototypes a construire. Il portera les prototypes dans lesquels
   il peut etre adjoint
-----------------------------------------------------------------------


       Sub >> Prototype support de la substitution
       -------------------------------------------
          Categorie : Nom|Forme:?|Role:AuxTree|Elts:a list('Nom', 'Adj')

       Sub >> Prototype utilise pour la substitution
       ---------------------------------------------
          Categorie : SP|Forme:(Prep:a|Nom:prototypes)|Role:ElemTree|Elts:a list('Prep', 'Nom')

        Sub >> Prototype Resultant de la substitution
        ---------------------------------------------
          Categorie : Nom|Forme:(Nom:prototypes)|Role:AuxTree|Elts:a list('Nom', 'Adj')

     Nouveau Prototype (arbre auxiliaire) a construire
     -------------------------------------------------
                                        Arbre <52>
     2-(Cat:Nom|Forme:(Nom:prototypes)|Forme Nue:prototypes|RoleTree:AuxTree)-3


Arbre a2 Ch2 ===> Adjonction possible sur arbre construit en Position 1

-----------------------------------------------------------------------
1. Preconstruction de l'arbre auxiliaire attache au prototype construit
2. Si l'arbre auxiliaire est preconstruit, il sera ajoute a la liste
   des prototypes a construire. Il portera les prototypes dans lesquels
   il peut etre adjoint
-----------------------------------------------------------------------


       Sub >> Prototype support de la substitution
       -------------------------------------------
          Categorie : NAdj|Forme:?|Role:AuxTree|Elts:a list('Nom', 'Adj')
```

```
       Sub >> Prototype utilise pour la substitution
       ---------------------------------------------
          Categorie : SP|Forme:(Prep:a|Nom:prototypes)|Role:ElemTree|Elts:a list('Prep', 'Nom')

        Sub >> Prototype Resultant de la substitution
        ---------------------------------------------
          Categorie : NAdj|Forme:(Nom:prototypes)|Role:AuxTree|Elts:a list('Nom', 'Adj')

     Nouveau Prototype (arbre auxiliaire) a construire
     -------------------------------------------------
                                         Arbre <54>
     2-(Cat:NAdj|Forme:(Nom:prototypes)|Forme Nue:prototypes|RoleTree:AuxTree)-3



-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 46)-3
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : SP
        Forme : (Prep:*|Nom:prototypes)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Prep')
        ListNeededCat : a list('Nom', 'Prep')
        Right-Direction : false
        Left-Direction : true
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                         Arbre <35>
     1-(Cat:Prep|Forme construite:a|Forme nue:a|RoleTree:TermTree)-2


     Prototype resultant:

                                         Arbre <46>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                         Arbre <46>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Ce prototype n'est pas declencheur de nouveau prototype.

Ce prototype n'est pas declencheur de nouveau prototype (cf. contraintes predefinies)


-------------------------------------------------------------------------------
Recherche si le prototype construit peut etre le support d'une adjonction.
i.e. Recherche des arbres d'analyse auxiliaires associes au prototype construit.
-------------------------------------------------------------------------------




-------------------------------------------------------------------------------
Le prototype construit a aussi permis la construction de prototypes specialises.
Recherche des prototypes a construire a partir de ces derniers.
-------------------------------------------------------------------------------

-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 47)-2
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepN2
        Forme : (Nom:analyseur|Prep:a|Nom:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
```

```
   Prototype extrait a construire : 1-(Arbre 48)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : SP
        Forme : (Prep:a)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : SP -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : SP -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Prep -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Nom -> aff-loc

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Prep -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Nom -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 50)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 52)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 54)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------
```

```
        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NomAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NomAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 39)-2
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepV
        Forme : (Nom:analyseur|Prep:a|Verbe:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Vinf')
        NeededCat : a list('Vinf')
        ListNeededCat : a list('Nom', 'Prep', 'Vinf')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 40)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:analyseur|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 41)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 42)-1
-------------------------------------------------------------------------
```

```
        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|SP:*)
        Role : ElemTree
        Elts : a list('Verbe', 'SP')
        NeededCat : a list('SP')
        ListNeededCat : a list('Verbe', 'SP')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <46>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|RoleTree:ElemTree)-3


     Prototype resultant:

                                        Arbre <42>
     0-(Cat:Sv|Forme        construite:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                        Arbre <42>
     0-(Cat:Sv|Forme        construite:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3

-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 43)-1
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|Sv:*)
        Role : ElemTree
        Elts : a list('ProRel', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('ProRel', 'Sv')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 44)-1
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|P:*)
        Role : ElemTree
        Elts : a list('ProRel', 'P')
        NeededCat : a list('P')
        ListNeededCat : a list('ProRel', 'P')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 55)-1
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|SP:*)
```

```
        Role : ElemTree
        Elts : a list('Verbe', 'SP')
        NeededCat : a list('SP')
        ListNeededCat : a list('Verbe', 'SP')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


     Prototype Disponible Retenu:

                                        Arbre <45>
     1-(Cat:SP|Forme construite:(Prep:a|Nom:prototypes)|Forme nue:a prototypes|RoleTree:ElemTree)-3


     Prototype resultant:

                                        Arbre <55>
     0-(Cat:Sv|Forme        construite:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3

   Prototype construit:

                                        Arbre <55>
     0-(Cat:Sv|Forme        construite:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))|Forme        nue:analyseur        a
prototypes|NeededCat:a list|RoleTree:ElemTree)-3

-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 47)-2
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepN2
        Forme : (Nom:analyseur|Prep:a|Nom:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 48)-2
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : SP
        Forme : (Prep:a)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : SP -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : SP -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Prep -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Nom -> aff-loc

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Prep -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Nom -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 50)-3
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
```

```
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 52)-3
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 54)-3
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NomAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NomAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 39)-2
-------------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NPivotPrepV
        Forme : (Nom:analyseur|Prep:a|Verbe:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Vinf')
        NeededCat : a list('Vinf')
        ListNeededCat : a list('Nom', 'Prep', 'Vinf')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
```

```
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 40)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:analyseur|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 41)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 43)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|Sv:*)
        Role : ElemTree
        Elts : a list('ProRel', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('ProRel', 'Sv')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 44)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|P:*)
        Role : ElemTree
        Elts : a list('ProRel', 'P')
        NeededCat : a list('P')
        ListNeededCat : a list('ProRel', 'P')
        Right-Direction : true
        Left-Direction : false
```

```
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 56)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|SP:*)
        Role : ElemTree
        Elts : a list('Verbe', 'SP')
        NeededCat : a list('SP')
        ListNeededCat : a list('Verbe', 'SP')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 47)-2
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NPivotPrepN2
        Forme : (Nom:analyseur|Prep:a|Nom:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 48)-2
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : SP
        Forme : (Prep:a)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : SP -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : SP -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Prep -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Nom -> aff-loc

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Prep -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Nom -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
```

```
   Prototype extrait a construire : 2-(Arbre 50)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 52)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 54)-3
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : NomAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NomAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 39)-2
-----------------------------------------------------------------------

        -----------------------------------------------------
        Cat : NPivotPrepV
        Forme : (Nom:analyseur|Prep:a|Verbe:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Vinf')
        NeededCat : a list('Vinf')
        ListNeededCat : a list('Nom', 'Prep', 'Vinf')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------
```

```
        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 40)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:analyseur|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 41)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 43)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|Sv:*)
        Role : ElemTree
        Elts : a list('ProRel', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('ProRel', 'Sv')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 44)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : PRel
```

```
        Forme : (ProRel:analyseur|P:*)
        Role : ElemTree
        Elts : a list('ProRel', 'P')
        NeededCat : a list('P')
        ListNeededCat : a list('ProRel', 'P')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 56)-1
-----------------------------------------------------------------------

        -------------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|SP:*)
        Role : ElemTree
        Elts : a list('Verbe', 'SP')
        NeededCat : a list('SP')
        ListNeededCat : a list('Verbe', 'SP')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 47)-2
-----------------------------------------------------------------------

        -------------------------------------------------------
        Cat : NPivotPrepN2
        Forme : (Nom:analyseur|Prep:a|Nom:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Nom', 'Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------


-----------------------------------------------------------------------
   Prototype extrait a construire : 1-(Arbre 48)-2
-----------------------------------------------------------------------

        -------------------------------------------------------
        Cat : SP
        Forme : (Prep:a)
        Role : ElemTree
        Elts : a list('Prep', 'Nom')
        NeededCat : a list('Nom')
        ListNeededCat : a list('Prep', 'Nom')
        Right-Direction : true
        Left-Direction : false
        -------------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : SP -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : SP -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Prep -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Nom -> aff-loc

        ==> Traits sur Feuilles(arbre) :
```

```
              Traits sur Arbre(feuille0) : Prep -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Nom -> pas de contrainte de traits definie
        --------------------------------------------


--------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 50)-3
--------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : NAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        --------------------------------------------


--------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 52)-3
--------------------------------------------------------------------

        ----------------------------------------------------
        Cat : Nom
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : Nom -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : Nom -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        --------------------------------------------


--------------------------------------------------------------------
   Prototype extrait a construire : 2-(Arbre 54)-3
--------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NAdj
        Forme : (Nom:prototypes)
        Role : AuxTree
        Elts : a list('Nom', 'Adj')
        NeededCat : a list('Adj')
        ListNeededCat : a list('Nom', 'Adj')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        --------------------------------------------
        Contraintes associees a l'arbre a construire
        --------------------------------------------
        ==> Type sur Tete(arbre) : NomAdj -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : NomAdj -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
              Type sur Arbre(feuille0) : Nom -> pas de contrainte de type definie
              Type sur Arbre(feuille1) : Adj -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
              Traits sur Arbre(feuille0) : Nom -> pas de contrainte de traits definie
              Traits sur Arbre(feuille1) : Adj -> pas de contrainte de traits definie
        --------------------------------------------


--------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 39)-2
--------------------------------------------------------------------

        ----------------------------------------------------
        Cat : NPivotPrepV
        Forme : (Nom:analyseur|Prep:a|Verbe:*)
        Role : ElemTree
        Elts : a list('Nom', 'Prep', 'Vinf')
```

```
        NeededCat : a list('Vinf')
        ListNeededCat : a list('Nom', 'Prep', 'Vinf')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 40)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Nom
        Forme : (Nom:analyseur|PRel:*)
        Role : ElemTree
        Elts : a list('Nom', 'PRel')
        NeededCat : a list('PRel')
        ListNeededCat : a list('Nom', 'PRel')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 41)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : Sv
        Forme : (Verbe:analyseur|Sn:*)
        Role : ElemTree
        Elts : a list('Verbe', 'Sn')
        NeededCat : a list('Sn')
        ListNeededCat : a list('Verbe', 'Sn')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



-------------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 43)-1
-------------------------------------------------------------------------

        -----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|Sv:*)
        Role : ElemTree
        Elts : a list('ProRel', 'Sv')
        NeededCat : a list('Sv')
        ListNeededCat : a list('ProRel', 'Sv')
        Right-Direction : true
        Left-Direction : false
        -----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------
```

```
-----------------------------------------------------------------------
   Prototype extrait a construire : 0-(Arbre 44)-1
-----------------------------------------------------------------------

        ----------------------------------------------------
        Cat : PRel
        Forme : (ProRel:analyseur|P:*)
        Role : ElemTree
        Elts : a list('ProRel', 'P')
        NeededCat : a list('P')
        ListNeededCat : a list('ProRel', 'P')
        Right-Direction : true
        Left-Direction : false
        ----------------------------------------------------


        ---------------------------------------------
        Contraintes associees a l'arbre a construire
        ---------------------------------------------
        ==> Type sur Tete(arbre) : -> pas de contrainte de type definie
        ==> Traits sur Tete(arbre) : -> pas de contrainte de traits definie

        ==> Types sur Feuilles(arbre) :
            -> pas de contrainte de type definie

        ==> Traits sur Feuilles(arbre) :
            -> pas de contrainte de traits definie
        ---------------------------------------------



            ...Fin Analyse

Statistiques
------------


 Prototypes Initiaux:14

 Prototypes a Construire Declenches:12

 Prototypes Extraits:53

 Prototypes Construits:19

Le fichier /home/fleury/Self/source/Self-4.0/Phase3/trace/TraceParser.self est disponible pour une lecture de la trace
de l'analyse.


Le fichier /home/fleury/Self/source/Self-4.0/Phase3/trace/TraceResult.self est disponible pour une lecture des resultats
de l'analyse.



Phrase initiale:  'analyseur' 'a' 'prototypes'
----------------


Prototype construit couvrant la phrase initiale:<Cat:NPivotPrepN2|Forme:(Nom:analyseur|Prep:a|Nom:prototypes)>

Prototype construit couvrant la phrase initiale:<Cat:Sv|Forme:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))>

Prototype construit couvrant la phrase initiale:<Cat:Sv|Forme:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes))>


Liste des prototypes construits a l'issue de l'analyse:
-------------------------------------------------------


Prototype construit au cours de l'analyse:
[1]: <Arbre : 1> TermTree
(0-(Cat:Nom|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[2]: <Arbre : 5> TermTree
(0-(Cat:Verbe|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[3]: <Arbre : 9> TermTree
(0-(Cat:Clit|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[4]: <Arbre : 11> TermTree
(0-(Cat:Det|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[5]: <Arbre : 13> TermTree
(0-(Cat:ConjSub|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[6]: <Arbre : 15> TermTree
(0-(Cat:Adj|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[7]: <Arbre : 19> TermTree
(0-(Cat:Prep|Forme:analyseur)-1)

<analyseur>
```

```
Prototype construit au cours de l'analyse:
[8]: <Arbre : 23> TermTree
(0-(Cat:ProRel|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[9]: <Arbre : 25> TermTree
(0-(Cat:Pro|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[10]: <Arbre : 27> TermTree
(0-(Cat:Adverbe|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[11]: <Arbre : 31> TermTree
(0-(Cat:unknownCat|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[12]: <Arbre : 33> TermTree
(0-(Cat:Quant|Forme:analyseur)-1)

<analyseur>

Prototype construit au cours de l'analyse:
[13]: <Arbre : 35> TermTree
(1-(Cat:Prep|Forme:a)-2)

<a>

Prototype construit au cours de l'analyse:
[14]: <Arbre : 37> TermTree
(2-(Cat:Nom|Forme:prototypes)-3)

<prototypes>

Prototype construit au cours de l'analyse:
[15]: <Arbre : 38> ElemTree
(0-(Cat:NPivotPrepN2|Forme:(Nom:analyseur|Prep:a|Nom:prototypes))-3)

<analyseur a prototypes>

Prototype construit au cours de l'analyse:
[16]: <Arbre : 45> ElemTree
(1-(Cat:SP|Forme:(Prep:a|Nom:prototypes))-3)

<a prototypes>

Prototype construit au cours de l'analyse:
[17]: <Arbre : 46> ElemTree
(1-(Cat:SP|Forme:(Prep:a|Nom:prototypes))-3)

<a prototypes>

Prototype construit au cours de l'analyse:
[18]: <Arbre : 42> ElemTree
(0-(Cat:Sv|Forme:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes)))-3)

<analyseur a prototypes>

Prototype construit au cours de l'analyse:
[19]: <Arbre : 55> ElemTree
(0-(Cat:Sv|Forme:(Verbe:analyseur|SP:(Prep:a|Nom:prototypes)))-3)

<analyseur a prototypes>


Statistiques
------------


 Prototypes Initiaux:14

 Prototypes a Construire Declenches:12

 Prototypes Extraits:53

 Prototypes Construits:19
```