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.

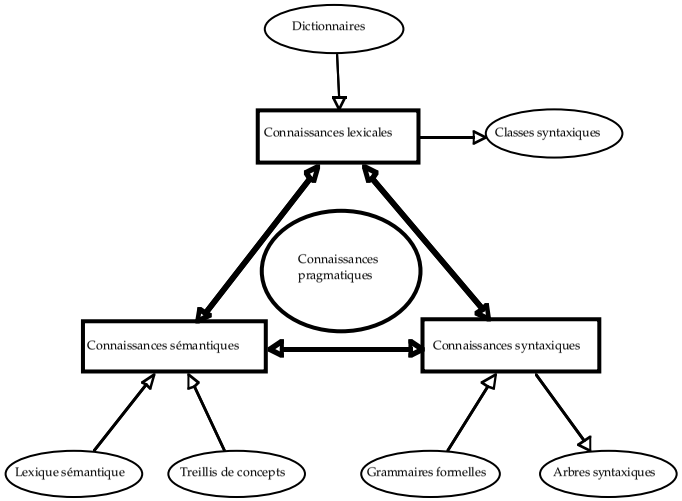


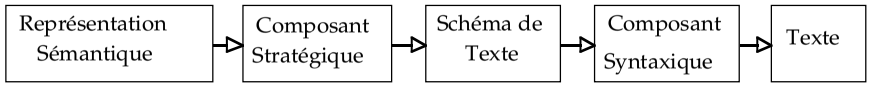
Figure 3.1 : Connaissances et TALN.

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.

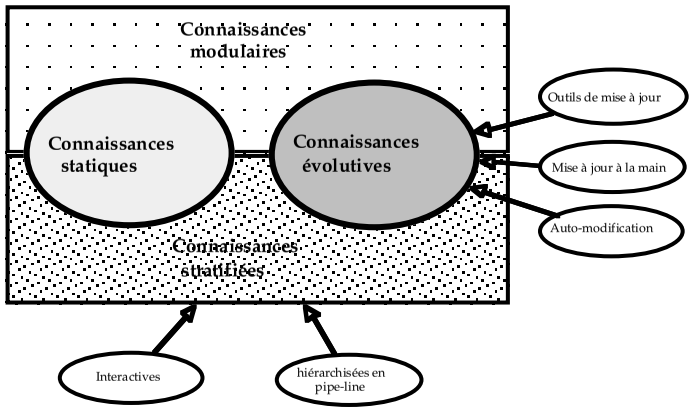


Figure 3.2 : Connaissance et utilisation des connaissances.

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 :

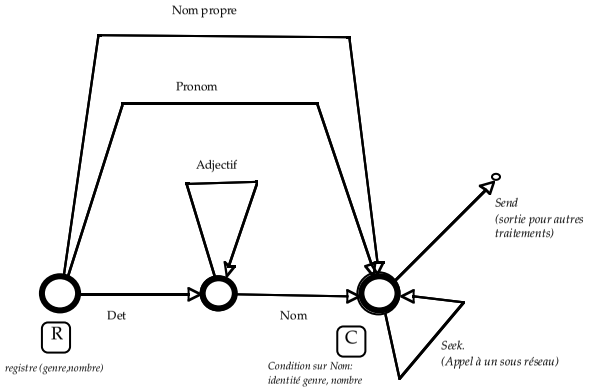


Figure 3.3 : ATN et contruction d'un groupe nominal.

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 | 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 | 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 :

$$A[...] \longrightarrow X_1[...]\ X_n[...]$$

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 :

$$A[...] \longrightarrow B[i...]$$

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 :

$$A[i...] \longrightarrow X_1[...]\ X_n[...]$$

(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 :

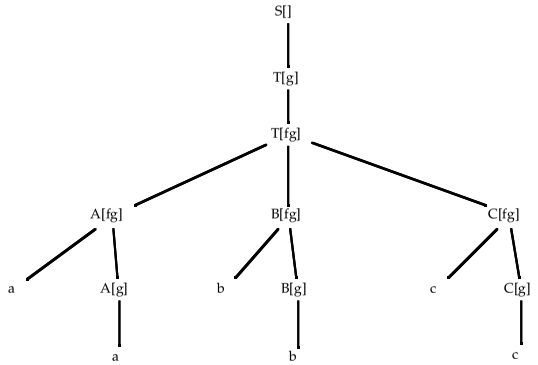


Figure 3.7 : Grammaire indexée : dérivation pour $a^n b^n c^n$.

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 :

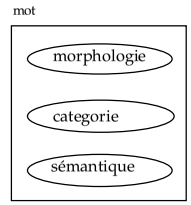


Figure 3.18 : Classe `mot`.

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.

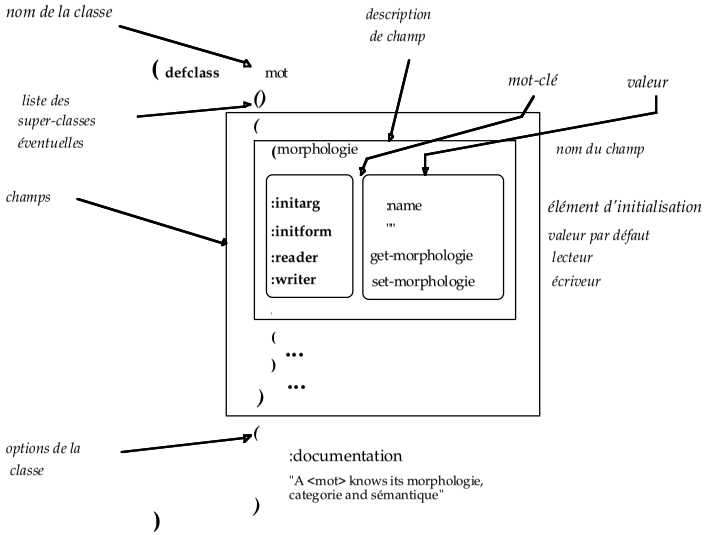


Figure 3.19 : Définition d'une classe.

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.

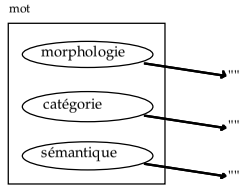


Figure 3.21 : Appel nu à `make instance`.

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 être 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[1] 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 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

[1] 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.

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 :

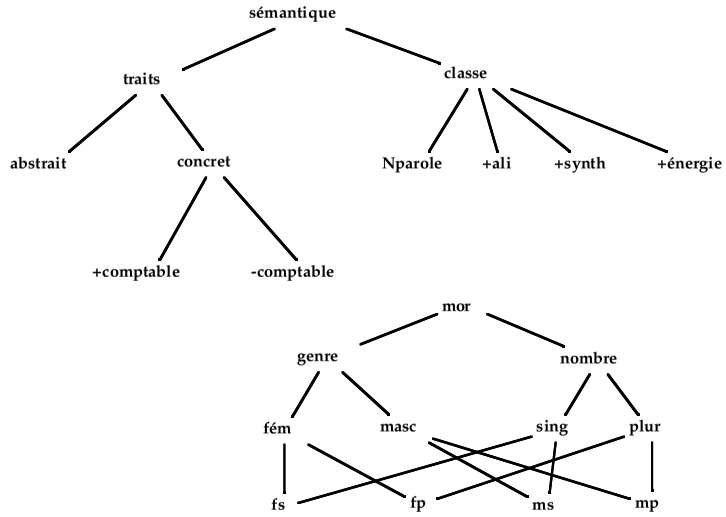


Figure 3.24 : Représentation morpho-sémantique de `N1 à N2`.

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 :

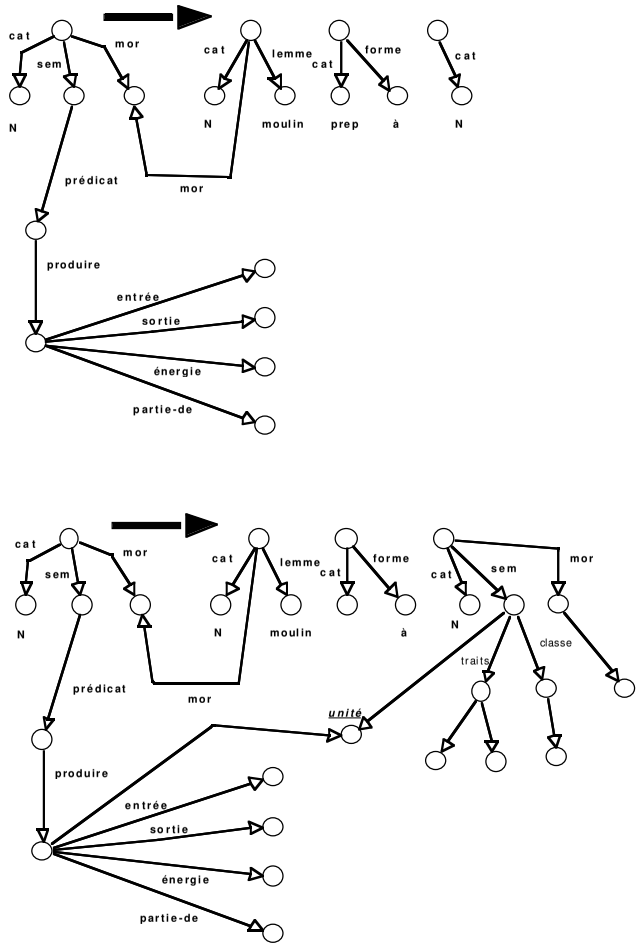


Figure 3.25 : `Moulin à N2`, représentation hiérarchisée des connaissances et construction du sens.

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.

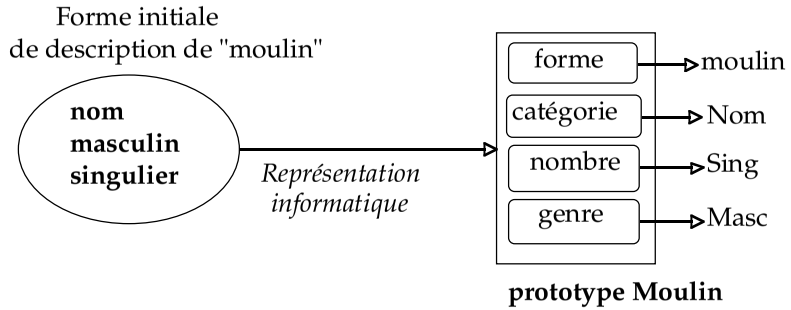


Figure 3.27 : De la description individuelle au prototype `Nom`.

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.

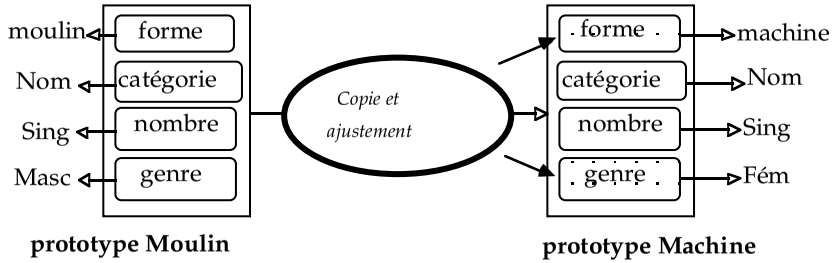


Figure 3.28 : Clonage des prototypes `Noms`.

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.

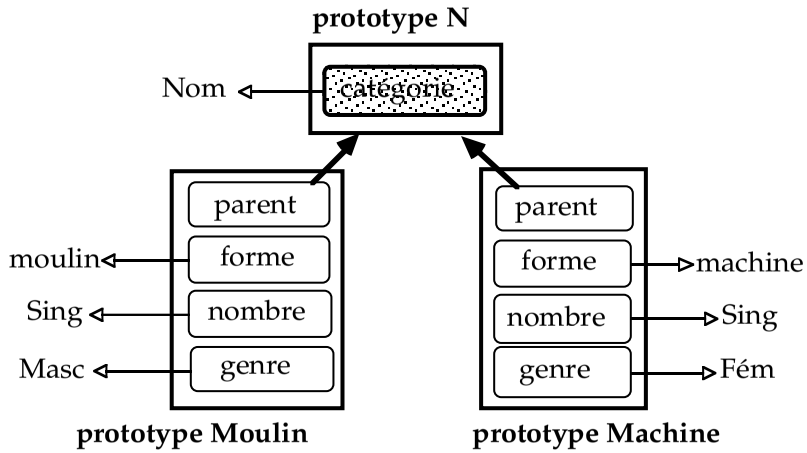


Figure 3.29 : Prototype et délégation chez les `Noms`.

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).

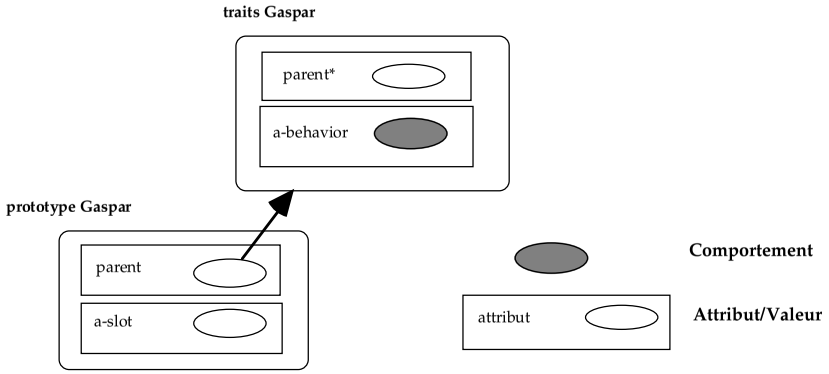


Figure 3.30 : `Traits` et prototypes.

Concept de Bases (Cointe & al 1991)

- 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.

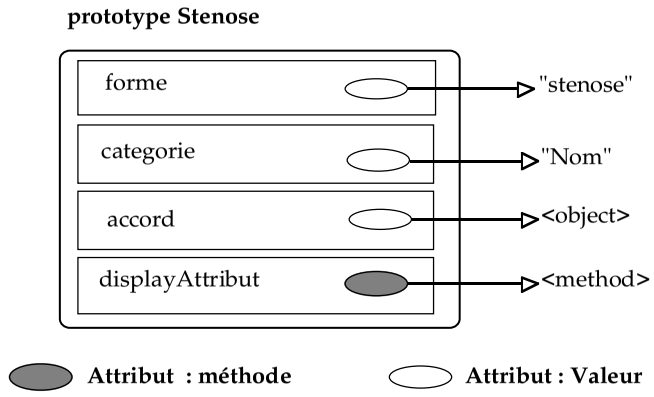


Figure 3.31 : Objet Self.

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[2] 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.

[2] 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.

- 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 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` :

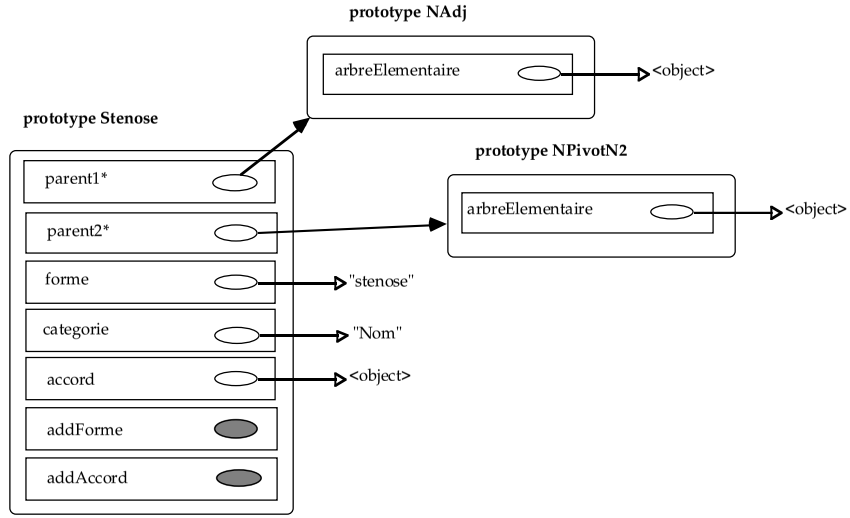


Figure 3.32 : Prototype et délégation en Self.

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 :

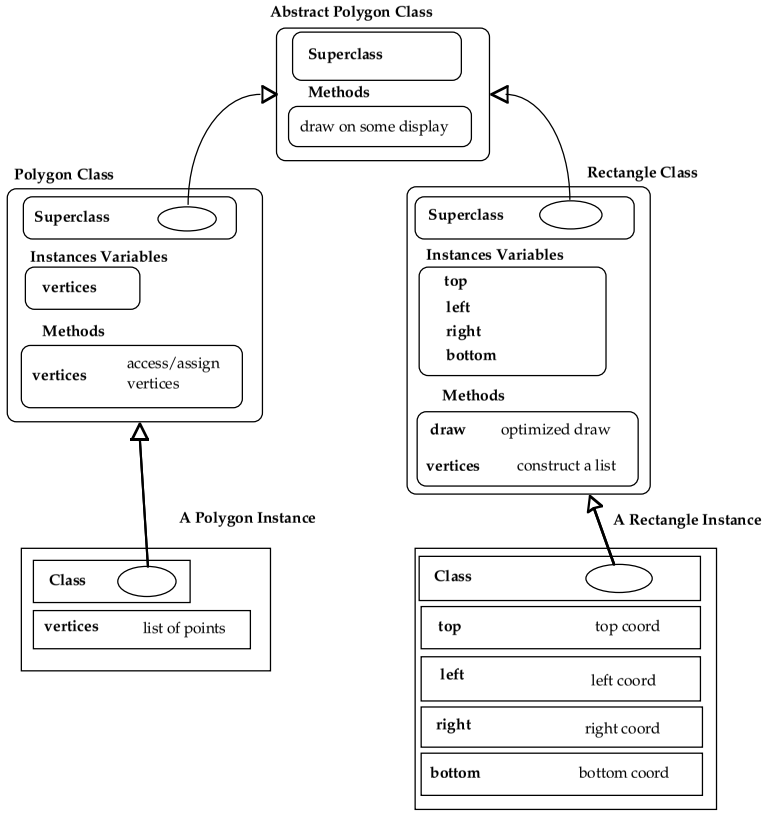


Figure 3.33 : Utilisation de classes abstraites pour prendre en compte des modifications.

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.

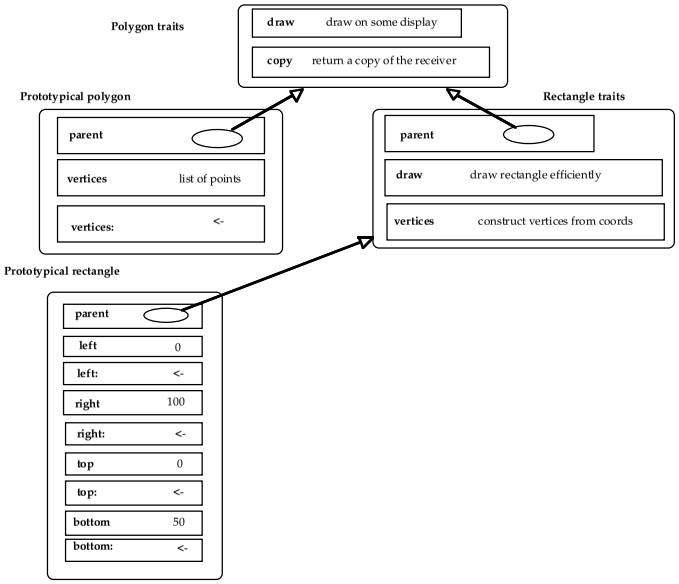


Figure 3.34 : Prototype polygone et modification de comportement.

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.

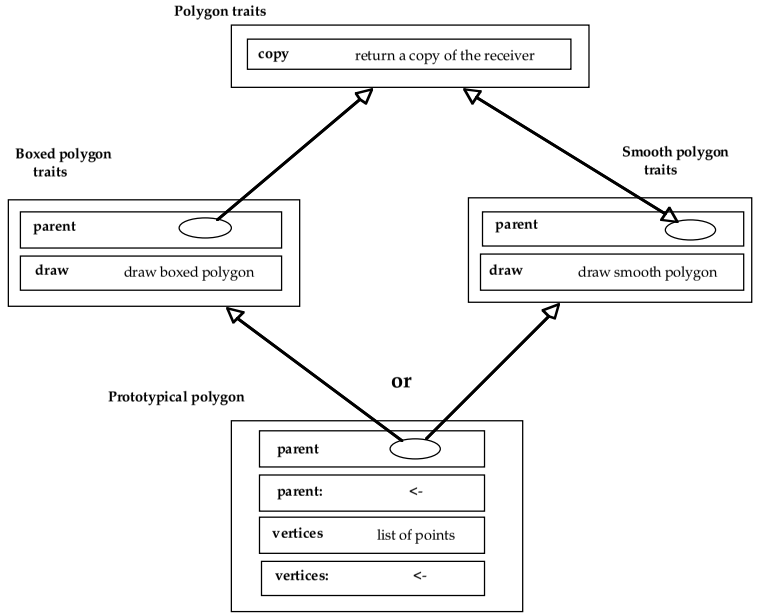


Figure 3.35 : Modification dynamique de comportement.

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 "*les 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.

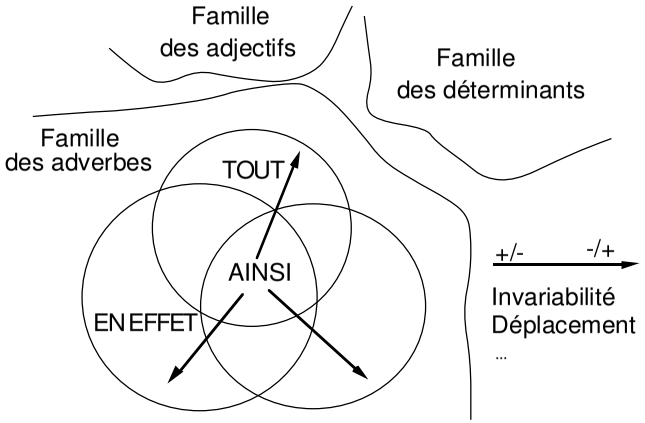
"Le prototype fournit en même temps le principe de catégorisation : les entités sont rangées dans une catégorie selon leur degré de ressemblance avec le prototype. Le degré de représentativité devient ainsi équivalent de degré d'appartenance dans des catégories conçues comme ayant des frontières floues. 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[3] pour évaluer la représentativité d'un exemplaire par rapport au prototype qui définit la classe.

Le niveau de base[4] 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*

[3] *"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).

[4] *"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).

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 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 :

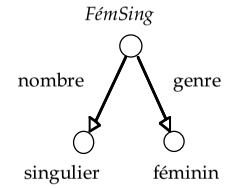


Figure 3.8 : `FémSing`

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

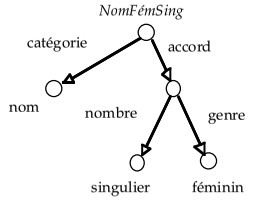


Figure 3.9 : `NomFémSing`

```
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.

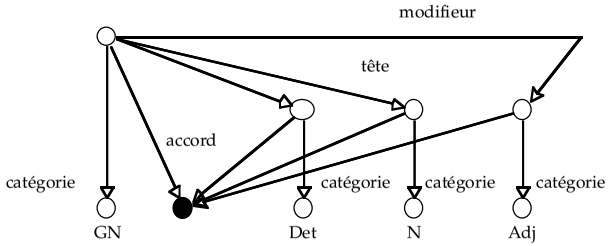


Figure 3.10 : Partage de valeurs entre DAGs

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 :

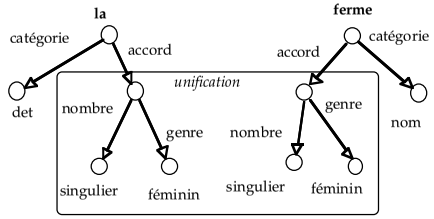


Figure 3.11 : Unification possible sur le trait `accord`

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` :

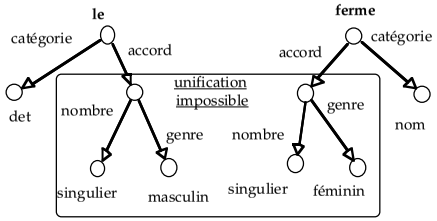


Figure 3.12 : Unification impossible sur le trait `accord`

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) :

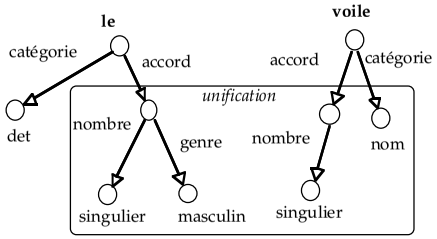


Figure 3.13 : Unification ajoutant de l'information

produit la structure suivante :

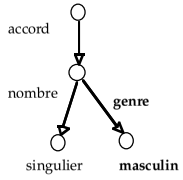


Figure 3.14 : Précision du trait `accord` de `voile` grâce à l'unification

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[5] 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

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

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
<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.

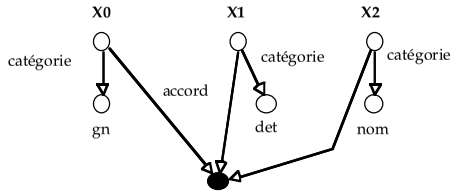


Figure 3.15 : Règle PATR-II "expansée"

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.

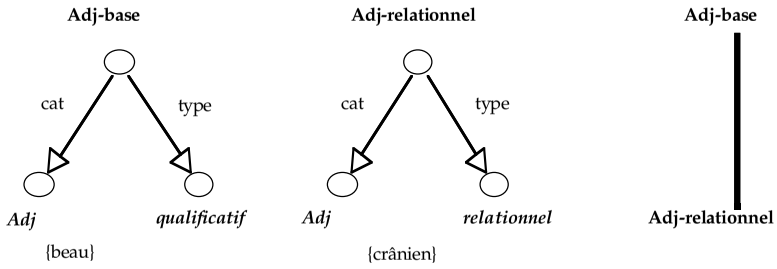


Figure 3.16 : Héritage et redéfinition.

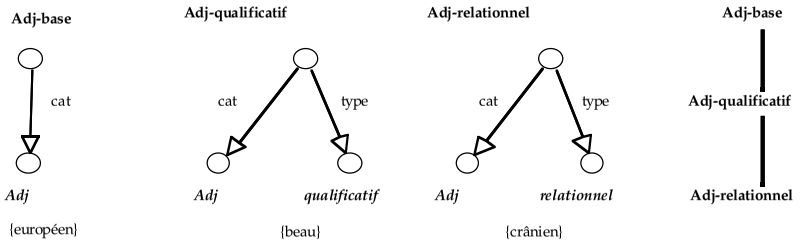
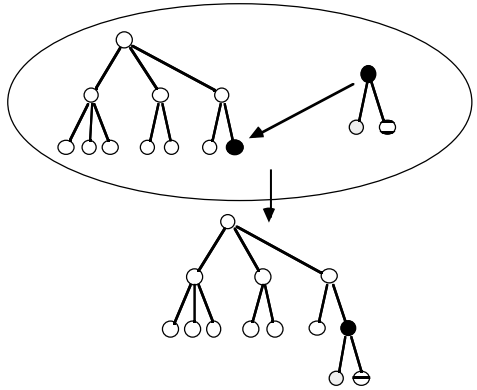


Figure 3.17 : Héritage et ambiguïté.

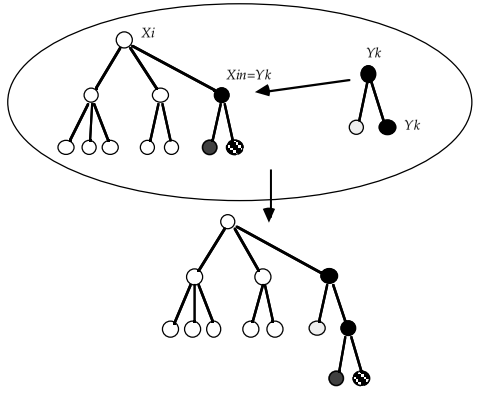
Incise 2. Les grammaires d'arbres adjoints (TAG)

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).