UNIVERSITÉ DU QUÉBEC

MEMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAITRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
NICOLE MUNYANA

LE TEXT MINING ET XML

AOÛT 2007

LE TEXT MINING ET XML

Nicole MUNYANA

SOMMAIRE

L'extraction des termes complexes est une étape de prétraitement importante pour des opérations informatiques ou d'informatique linguistique complexes comme : la terminologie, le résumé automatique, mais aussi le text-mining et la classification textuelle. Nous présentons dans ce mémoire un filtre linguistique pour l'extraction des termes complexes. Ce filtre linguistique est fondé sur un modèle catégoriel : la Grammaire Catégoriel Combinatoire Applicative.

C'est grâce à ce modèle catégoriel que nous avons pu identifier les termes complexes à partir d'une liste des termes candidats. Une analyse syntaxique des formes phénotypiques qui est obtenue par un calcul sur les types syntaxiques, nous a permis de vérifier si un terme candidat est du groupe nominal. Ce sont les termes candidats ayant comme catégorie le groupe nominal qui sont préservés par notre filtre linguistique. Les termes candidats n'ayant pas comme catégorie le groupe nominal sont rejetés. L'expression applicative obtenue après l'analyse syntaxique a donné après réduction des combinateurs la forme normale du terme complexe. Les résultats du traitement des termes complexes sont stockés dans une base de données XML. XML offre des possibilités intéressantes pour des manipulations ultérieures de ces résultats par des requêtes XQuery.

Notre filtre linguistique est différent de la plupart des autres filtres linguistiques d'identification des termes complexes, parce qu'il tend à être multilingue. Avec la croissance du Web et des bases de données textuelles multilingues, cet aspect est significatif. Tout ce que nous avons besoin pour adapter l'approche à une nouvelle langue est un dictionnaire des types catégoriels avec les entrées lexicologiques de cette langue.
L'approche théorique a été implémentée en C++ pour un corpus important du français.

REMERCIEMENTS

Je remercie très sincèrement le professeur Ismaïl BISKRI, directeur de ce mémoire, pour avoir accepté de diriger mes travaux, pour l'aide et le temps qu'il a bien voulu me consacrer et pour ses conseils et ses remarques précieux.

Je tiens aussi à remercier les professeurs Linda Badri et François Meunier qui avec le professeur Ismaïl Biskri ont évalué ce mémoire.

J'adresse enfin mes plus sincères remerciements à tous mes proches et amis qui m'ont toujours soutenue et encouragé au cours de la réalisation de ce mémoire.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

L'extraction des termes complexes est une étape de prétraitement importante pour des opérations informatiques ou d'informatique linguistique complexes comme : la terminologie, le résumé automatique, mais aussi le text-mining et la classification textuelle. Ces dernières années, un certain nombre d'outils permettant d'identifier des termes complexes ont été développé et proposé dans la littérature scientifique. Ces outils acceptent en entrée un texte ou un corpus, l'un ou l'autre prétraité (étiqueté par exemple) ou non, et produisent automatiquement une liste de termes candidats, souvent par l'intermédiaire d'une approche statistique (bayésienne) ou d'une approche linguistique. Les approches statistiques peuvent être multilingues, mais elles sont cependant bruyantes. Les approches linguistiques sont moins bruyantes, cependant elles ne peuvent pas traiter les corpus multilingues ou certains néologismes dans des domaines spécifiques. Ces approches semblent adaptées aux textes bien stéréotypés.

Le multilinguisme est devenu, avec la montée du Web, l'une des contraintes les plus significatives dans le développement des outils pour le traitement des langues naturelles. Il est donc important de considérer les approches qui tiennent compte de cet aspect. Notre approche a des possibilités d'être multilingue. Nous employons un filtre linguistique informatique peu coûteux à appliquer. Ce filtre est favorable au traitement d'autres langues que le français et l'anglais bien que le français soit la seule langue considérée dans ce travail.

Nous proposons, pour l'extraction des termes complexes une approche multilingue. Nous utilisons un filtre linguistique fondé sur un modèle catégoriel : la Grammaire Catégorielle Combinatoire Applicative de Biskri (1995). L'entrée de ce filtre est une liste de termes candidats à valider.

Par ailleurs, les Bases de données relationnelles traditionnelles reposent sur une structuration formelle solide mais aussi manquant de flexibilité. Aussi l'utilisation de telles bases de données pour recueillir des termes complexes diminue les possibilités de manipulation de ces derniers dans des applications ultérieures. XML (Extensible Markup Language) semble permettre une flexibilité intéressante pour le stockage de ces termes. C'est pour cette raison que nous avons préféré stocker les termes complexes validés par notre filtre linguistique dans une base de données XML.

Nous avons organisé ce mémoire en huit chapitres (dont l'introduction pour le premier chapitre).

Nous présentons au deuxième chapitre les Grammaires Catégorielles. Nous suivons les étapes chronologiques depuis les fondements philosophiques de Husserl jusqu'à la Grammaire Catégorielle Combinatoire de Steedman.

Le troisième chapitre est consacré à la Grammaire Applicative Universelle de Shaumyan (1965, 1977, 1987) et à sa version étendue, la Grammaire Applicative et Cognitive (Desclés, 1990).

Nous consacrons le quatrième chapitre au modèle de la Grammaire Catégorielle Combinatoire Applicative de Biskri (1995).
Ce modèle d'analyse concrétise deux objectifs principaux : l'analyse syntaxique des textes et la construction d'une interprétation sémantique fonctionnelle. A travers ces deux objectifs l'idée est de matérialiser le passage d'une structure phrastique linéaire concaténée à une structure opérateur/opérande prédicative.

Le cinquième chapitre est consacré à notre recherche théorique. Nous présentons dans ce chapitre les détails théoriques d'un filtre linguistique dont le but est l'identification et la validation des termes complexes. Ce filtre linguistique est fondé sur le modèle de la Grammaire Catégorielle Combinatoire Applicative. Ce filtre prend en entrée une liste de termes candidats. Ce filtre linguistique :

- reconnaît les termes candidats syntaxiquement corrects par un calcul sur les types syntaxiques.

- engendre une structure opérateur/opérande prédicative représentant l'interprétation sémantique fonctionnelle du terme candidat. La construction de cette structure prédicative se fera au moyen des combinateurs de la logique combinatoire introduits progressivement.

Dans un premier temps nous présenterons tous les résultats théoriques auxquels nous avons abouti. Nous illustrerons ces résultats dans un second temps par des exemples. Nous exposerons enfin des solutions à des problèmes posés par certaines constructions, en particulier des termes avec modifieurs arrières, un agencement des mots tel que « Nom - Nom ».

Le sixième chapitre sera consacré à l'implantation informatique du filtre linguistique. L'objectif principal de cette implémentation est de prouver que nous pouvons concevoir un programme informatique qui met en pratique les résultats théoriques auxquels nous sommes arrivés.

Nous présentons au septième chapitre, les résultats d'analyse de cent termes candidats. Nous proposons aussi un traitement complet de quelques termes candidats.

Le huitième et dernier chapitre est réservé à la conclusion de notre travail.

CHAPITRE 2

ÉTAT DE L'ART

Ce chapitre présente un panorama des Grammaires catégorielles. Il expose tout d'abord leurs origines philosophiques et logiques. Il décrit ensuite les systèmes d'Ajdukiewicz (1935), de Bar-Hillel (1953), et enfin le calcul de Lambek (1958, 1961).

La dernière partie de ce chapitre est consacrée à la présentation de la Grammaire Catégorielle Combinatoire de Steedman (1982, 1987, 1989).

2.1 Origines philosophiques et logiques des Grammaires Catégorielles

Les origines philosophiques des Grammaires catégorielles sont issues des travaux du philosophe Husserl (1913). Ce dernier, reprend une opposition traditionnelle depuis les Grecs et distingue les expressions catégorématiques des expressions syncatégorématiques.
Les expressions catégorématiques sont celles qui sont pourvues d'un sens, elles apparaissent sous formes de syntagmes nominaux ou d'énoncés. Les expressions qui ne sont pas pourvues d'un sens complet sont les expressions syncatégorématiques. Avec ces expressions une signification complète n'est perçue que conjointement avec les significations partielles d'autres parties du discours.

La conception des catégories sémantiques du logicien Lesniewski (1922) reprend la tradition des catégories aristotéliciennes, celle des parties du discours de la grammaire traditionnelle et celle enfin des catégories de signification développée par Husserl. Il a retenu deux sortes d'expressions: les noms et les propositions.
En dehors de ces deux sortes de catégories fondamentales, les autres expressions du langage seront des syncatégorèmes.

Cette classification permet de constater que les expressions syncatégorématiques fonctionnent comme des opérateurs qui contribuent à constituer des expressions catégorématiques.

En reprenant cette classification, Ajdukiewicz puis Bar-Hillel et enfin Lambek ont proposé différents systèmes formels pour vérifier la bonne connexion syntaxique des langues naturelles. Ces systèmes formels ont pour nom : **les Grammaires Catégorielles**.

2.2 Le Modèle de Kazimierz Ajdukiewicz

Ajdukiewicz (1935) propose un modèle qui comporte un ensemble de catégories divisées en deux classes : les catégories de base et les catégories opérateurs.
Les catégories de base sont aux nombres de deux : les catégories syntagmes nominaux et les catégories phrases, qui sont représentées respectivement par les notations N ('noun') et S ('sentence').
Les catégories opérateurs sont récursivement obtenues à partir des catégories de base et d'un symbole d'application.

Toutes les catégories de ce modèle sont récursivement obtenues à travers l'application des règles suivantes :

1. Les catégories de base du modèle sont des catégories du modèle.
2. Si X et Y sont des catégories du modèle alors $\frac{X}{Y}$ est une catégorie du modèle.

$\frac{X}{Y}$ est une catégorie foncteur.

La barre de fraction symbolise l'application.
Le dénominateur symbolise le type de l'argument.
Le numérateur symbolise le type du résultat de l'application du foncteur de type $\frac{X}{Y}$ à l'argument de type Y.
Nous constatons qu'à partir du triplet < S, N, –>, nous pouvons générer un ensemble infini de catégories.

Partant de cela, l'idée de Ajdukiewicz est d'associer aux mots et aux expressions, en fonction des rapports qu'ils entretiennent entre eux dans la phrase, des types qui vont indiquer quelles expressions données peuvent se combiner entre elles. Il restera après à établir quand une suite de mots est correctement formée d'un point de vue syntaxique.
Les deux règles de réduction suivantes vont permettre de vérifier cette connexion syntaxique.

Règles de réduction

$$\frac{X}{Y} \quad Y \quad \text{---> } X$$

$$Y \quad \frac{X}{Y} \quad \text{---> } X$$

Pour l'exemple suivant : *Jean marche.*

Jean	*marche*
N	S
	N
S	

En appliquant les règles de réduction, on obtient : (*Jean marche*) avec le type S.
Ajdukiewicz dit que si au bout d'un nombre fini d'applications des règles de réduction nous obtenons une expression qui a pour type l'une des deux catégories de base, alors nous pouvons conclure que l'expression en question est bien formée.

2.3 Le modèle de Yeoshua Bar-Hillel

Le modèle de Bar-Hillel (1953) ne présente que des règles de réduction (règles d'application). Ce modèle bidirectionnelle contient les deux notions de division-droite et division-gauche. Ces deux notions impliquent l'idée que les réductions se font dans les deux sens selon l'ordre dans lequel un opérateur attend ses arguments.

Ainsi dans le système de Bar-Hillel il y a :

- Deux catégories de base S et N.

- Deux règles de formation des catégories complexes :

  - Les catégories de base sont des catégories.

  - Si X et Y sont des catégories alors X/Y (respectivement X\Y) sont des catégories.

  X/Y est la catégorie d'un opérateur ayant comme argument un opérande de type Y positionné à droite. X\Y est la catégorie d'un opérateur ayant comme argument un opérande de type X positionné à gauche.

- Deux règles de réduction des types :

  Règle de réduction droite :

  X/Y Y ---> X

  Règle de réduction gauche :

  Y Y\X ---> X

Prenons un exemple : *Jean regarde Marie*.

```
Jean        regarde      Marie
------      ---------    --------
N           (N\S)/N      N
                ----------------------------Règle droite
                N\S
            ----------------------Règle gauche
            S
```

En appliquant la règle droite suivie de la règle gauche nous obtenons le résultat :

(*Jean regarde Marie*) de type S

Ce qui signifie que la phrase est syntaxiquement bien construite.

2.4 Le Calcul de Lambek

Le calcul de Lambek sur les types des grammaires catégorielles est apparu dans deux articles de Lambek en 1958 et 1961. Dans son formalisme Lambek introduit des types fonctionnels X/Y et X\Y qui représentent des types d'opérateurs appliqués à des opérandes à droite ou à gauche.

Dans le calcul de Lambek, on peut à partir d'un ensemble fini des types primitifs, définir un ensemble infini de types complexes à l'aide des règles récursives suivantes :
1. Tout type primitif est un type syntaxique.
2. Si X et Y sont des types syntaxiques alors X/Y et X\Y sont des types syntaxiques.
Nous introduisons aussi une relation, notée '—>' qui est appelée **relation de réduction**; on écrira 'X —> Y' pour signifier que "le type X se réduit au type Y". La relation 'X <—> Y' signifie que l'on a à la fois 'X —> Y' et 'Y —> X'.

Le système formel du calcul de Lambek peut être donc considéré comme un calcul syntaxique généré par :
- Un ensemble fini de types syntaxiques primitifs.
- Un symbole de concaténation '-'.

  Définitions :

  1. Si 'u_1' et 'u_2' sont des expressions, la concaténation de 'u_1' et 'u_2' est notée 'u_1-u_2'. 'u_1-u_2' est une expression.

  2. Si une expression 'u_1' a pour type X, et une expression 'u_2' a pour type Y alors l'expression 'u_1-u_2' a pour type X-Y.
- La division à droite '/'.
- La division à gauche '\'.
- La relation de réduction '—>'.

2.5 La Grammaire Catégorielle Combinatoire

La Grammaire Catégorielle Combinatoire (Steedman 1987, 1989) est une généralisation combinatoire des Grammaires Catégorielles de Ajdukiewicz et de Bar-Hillel. Cette généralisation partage d'une part avec les Grammaires Catégorielles l'idée que les catégories des langues naturelles comprennent des catégories foncteurs et des catégories arguments; et permet d'autre part à des opérations autres que l'application fonctionnelle de combiner des catégories des langues naturelles.

Les opérations décrites plus loin sont :
- La composition fonctionnelle et le changement de type issus des travaux de Lambek (1958, 1961) et Geach (1972).
- La substitution fonctionnelle proposée par Szabolcsi (1987).

Ces nouvelles opérations permettent d'analyser des énoncés d'une façon incrémentale. L'idée intuitive de cette approche est que notre compréhension des phrases est incrémentale, en ce sens que chaque terme successif contribue à l'accumulation graduelle du sens. La stratégie d'analyse incrémentale se présente comme une solution pratique au problème de la pseudo ambiguïté. Ce problème réside dans le fait que plusieurs analyses syntaxiques d'une phrase sont possibles, toutefois, ils ne correspondent qu'à une seule interprétation sémantique.

2.5.1 Analyse catégorielle combinatoire

Jusqu'à présent nous n'avons présenté que deux catégories de base S et N. Steedman (1987, 1989) introduit d'autres catégories dans ces travaux :
- Une catégorie "syntagme nominal" notée par NP.
- Une catégorie "syntagme verbal" notée VP.
- Une catégorie "groupe prépositionnel" notée PP.

etc.

Les catégories complexes sont récursivement produites à partir de ces catégories de base

et des opérateurs de division à gauche et à droite.

Des éléments comme les verbes par exemple sont associés à une catégorie syntaxique qui leur attribue le statut de foncteur. Par exemple *aimer* se voit associer le type syntaxique (S\NP)/NP, où NP est le type des arguments du foncteur *aimer*. En effet, *aimer* opère sur un syntagme nominal en position objet pour produire un syntagme verbal qui à son tour va s'appliquer à un syntagme nominal en position sujet pour former la phrase (Steedman, 1989).

Certains auteurs préconisent des catégories qui soient à la fois des objets syntaxiques et sémantiques. D'ailleurs, ils les représentent par une structure de donnée informatique unique dans le but de réaliser une implémentation d'un analyseur, basée sur l'unification (Zeevat, Klein, Calder, 1986) ; (Pareschi, Steedman, 1987).

Dans cette optique la catégorie étendue du verbe *aimer* est la suivante :

$$(S:aimer'\ np_2\ np_1\backslash NP:np_1)/NP:np_2$$

Avec cette notation de Steedman (1989), les types syntaxiques sont en lettres majuscules, les constantes sémantiques portent des symboles ('), Les lettres minuscules sont des variables sémantiques.

Remarque :

L'expression (aimer' np_2 np_1) est équivalente à l'expression ((aimer' np_2) np_1).

Cette nouvelle approche a des répercussions sur l'utilisation des règles d'application[1] qui sont les règles de base dans le cadre des Grammaires Catégorielles Combinatoires.

X/Y	Y	—>	X	(>)	*application fonctionnelle avant*
Y	X\Y	—>	X	(<)	*application fonctionnelle arrière*

Chaque règle est en même temps syntaxique et sémantique. Cela se traduit pour l'analyse de la phrase *Jean--aime--Marie* par :

---

[1]L'utilisation des règles d'application fonctionnelle est identique à l'utilisation des règles de réduction de Bar-Hillel décrites au paragraphe **2.3**. Pour reprendre les notations de Steedman, l'application fonctionnelle avant est symbolisée par > et l'application fonctionnelle arrière est symbolisée par <.

Jean-	*aime-*	*Marie*
-----	-----	-------
NP: *Jean'*	(S: *aime'* np_2 np_1\NP: np_1)/NP: np_2	NP: *Marie'*
	---------- (>)	
	S: *aime' Marie'* np_1\NP: np_1	
---------- (<)		
S:*aime' Marie' Jean'*		

Ainsi, dans un premier temps, on associe à chaque unité linguistique un type syntaxique et une interprétation sémantique. Dans un deuxième temps, on effectue une réduction sur les types, complétée par l'unification nécessaire pour retrouver la bonne interprétation sémantique des "variables sémantiques" np_1 et np_2.

Pour appliquer la règle (>) aux catégories (S: *aime'* np_2 np_1\NP: np_1)/NP: np_2 et NP: *Marie'*, il est nécessaire que la variable np_2 s'unifie avec la constante *Marie'*. Dès lors le résultat de l'application de la règle (>) est représenté par la catégorie S: *aime' Marie'* np_1\NP: np_1. L'application de la règle (<) aux catégories NP: *Jean'* et S: *aime' Marie'* np_1\NP: np_1, unifie np_1 avec *Jean'* et donne la catégorie résultat S: *aime' Marie' Jean'*. La dérivation construit ainsi une interprétation composée qu'on considère comme structure standard de la phrase *Jean-aime-Marie*.

2.5.2 Les règles combinatoires

Pour enrichir le système des grammaires catégorielles, Steedman propose:

- Des règles de composition fonctionnelle.
- Des règles de changement de type.
- Des règles de substitution fonctionnelle.

Dans chaque règle proposée par Steedman, les types syntaxiques sont associés à des interprétations sémantiques qui permettent de rendre compte de l'aspect fonctionnel des énoncés. Dans les paragraphes qui suivent, Steedman propose une approche basée sur le lambda-calcul et l'unification pour construire l'interprétation sémantique.

2.5.2.1 La composition fonctionnelle

Le principe de la composition fonctionnelle est simple, on l'interprète par :
Une fonction de Y dans X, de type syntaxique X/Y ou X\Y et d'interprétation sémantique F, peut être combinée avec une fonction de Z dans Y de type syntaxique Y/Z ou Y\Z et d'interprétation G. Le résultat de cette opération est une fonction composée de Z dans X, de type X/Z ou X\Z et d'interprétation sémantique λx(F(Gx)).
Ce principe donne lieu à la production des quatre règles combinatoires dites *règles de composition fonctionnelle*[2].

1)	X/Y:F-Y/Z:G	—>	X/Z:λx(F(Gx))	**(>B)**
2)	X/Y: F-Y\Z: G	—>	X\Z: λx(F(Gx))	**(>Bx)**
3)	Y\Z: G-X\Y: F	—>	X\Z: λx(F(Gx))	**(<B)**
4)	Y/Z: G-X\Y: F	—>	X/Z: λx(F(Gx))	**(<Bx)**

Avec ces règles de composition fonctionnelle nous pouvons combiner par exemple les catégories des verbes *peut* et *faire* dans la phrase *Jean-peut-faire-ce-travail,* via l'unification des catégories (S: pred np_1\NP: np_1) et (S: *faire'* np_2 np_3\NP: np_3). Le résultat de l'unification est traduit par le remplacement de "pred" par "*faire'* np_2" et de "np3" par "np1", ce que nous donnons avec la figure suivante :

peut- *faire*

----- -----

(S: *peut'* (pred np_1)\NP: np_1)/(S: pred np_1\NP: np_1) (S:*faire'* np_2 np_3\NP:np_3)/NP:np_2

----------**>B**

(S:*peut'* (*faire'* np_2 np_1)\NP:np_1)/NP:np_2

---

[2]Le symbole (-) désigne ici aussi la concaténation. [3]Remarquons que Steedman ne représente la concaténation dans ses règles par aucun symbole. Nous voulons par l'introduction de ce symbole faciliter la lecture de ce document et donc éviter au lecteur de confondre une structure concaténée et une structure applicative.

2.5.2.2 Le changement de type

Le changement de type permet à des arguments de devenir des fonctions. Il permet ainsi à ces entités de composer avec d'autres fonctions.

Quatre règles de changement de type sont proposées par Steedman (1989):

5) X:x —> Y/(Y\X): λF(Fx) **(>T)**

6) X:x —> Y/(Y/X): λF(Fx) **(>Tx)**

7) X:x —> Y\(Y/X): λF(Fx) **(<T)**

8) X:x —> Y\(Y\X): λF(Fx) **(<Tx)**

Pour illustrer l'intérêt des règles de changement par un exemple, donnons pour la phrase *Jean-aime-Marie* l'analyse suivante:

Jean-	*aime-*	*Marie*
------	------	-------
N:*Jean'*	(S:*aime'* np_2 np_1\NP:np_1)/NP:np_2	NP:*Marie'*
---------**>T**		
S:pred *Jean'*/(S:pred *Jean'*\NP:*Jean'*)		
----------**>B**		
S:*aime'* np_2 *Jean'*/NP:np_2		
---------->		
S:*aime' Marie' Jean'*		

L'effet immédiat de la règle de changement de type est de permettre à la catégorie opérateur obtenue à partir de la catégorie de l'argument *Jean* de composer avec la catégorie du prédicat *aime* via une opération d'unification.

2.5.2.3 La substitution fonctionnelle

Le principe de la substitution fonctionnelle est interprété par :

Une fonction principale de second ordre de type (X/Y)/Z, (X/Y)\Z, (X\Y)/Z ou (X\Y)\Z

et d'interprétation F peut se combiner avec une fonction de premier ordre dans Y de catégorie Y/Z ou Y\Z et d'interprétation G. Le résultat est une fonction X/Z ou X/Z de Z dans X avec une interprétation λx(Fx(Gx)).

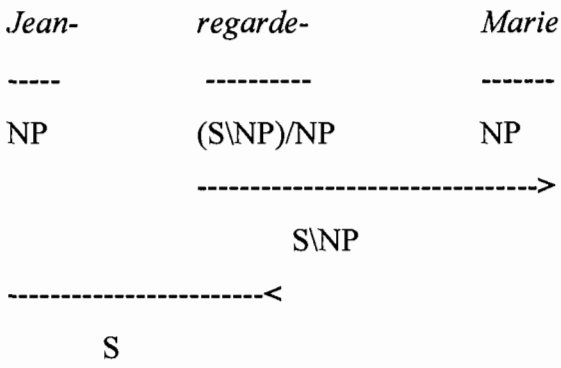
Ce principe donne lieu à la production des quatre règles combinatoires dites *règles de substitution fonctionnelle.*

9)	(X/Y)/Z:F-Y/Z:G	—>	X/Z: λx(Fx(Gx))	**(>S)**
10)	(X/Y)\Z:F-Y\Z:G	—>	X\Z: λx(Fx(Gx))	**(>Sx)**
11)	Y\Z:G-(X\Y)\Z:F	—>	X\Z: λx(Fx(Gx))	**(<S)**
12)	Y/Z:G-(X\Y)/Z:F	—>	X/Z: λx(Fx(Gx))	**(<Sx)**

2.5.3 Analyse incrémentale

L'analyse syntaxique présente un problème majeur dans le cadre des grammaires catégorielles : la pseudo ambiguïté. Un énoncé simple non ambiguë peut avoir plusieurs analyses syntaxiques possibles qui ne correspondent qu'à une seule interprétation sémantique.

Prenons l'exemple de la phrase *jean-regarde-marie.* Plusieurs analyses syntaxiques sont possibles. La première étant celle qui consiste à appliquer la catégorie du verbe à celle de l'objet puis appliquer la catégorie du prédicat complexe obtenu à celle du sujet:



La deuxième consiste à appliquer un changement de type à la catégorie du sujet puis à opérer une composition fonctionnelle sur la catégorie résultant du changement de type et la catégorie du verbe pour enfin terminer par l'application fonctionnelle :

Jean- *regarde-* *Marie*
----- ---------- -------
NP (S\NP)/NP NP
---->**T**
S/(S\NP)
----------->**B**
S/NP
------------------------------------>
S

D'autres analyses sont encore possibles. L'interprétation sémantique associée à toutes les analyses syntaxiques est unique, elle est sous la forme applicative suivante :

((*regarde' Marie'*) *Jean'*).

Pour résoudre ce problème Haddock (1987), Pareschi (1987) et Steedman (1987, 1989) proposèrent une stratégie d'analyse syntaxique incrémentale de gauche à droite. Cette stratégie favorise une seule analyse syntaxique et donc élimine les autres analyses. Dans le cas de la phrase *Jean-regarde-Marie,* c'est la deuxième analyse proposée ci dessus qui est retenue.

2.6 Conclusion

Nous venons à travers ce chapitre d'apprécier les étapes fondamentales qui nous ont permis de passer de concepts purement philosophiques à une réalité nettement concrète et relativement puissante pour l'analyse syntaxique des langues. Ce que nous appelons réalité se trouve matérialisée dans l'approche de la Grammaire Catégorielle Combinatoire.

Les travaux de Ajdukiewicz, de Bar-Hillel, de Lambek et de Steedman ont grandement contribué à la mise au point de ces grammaires.

Les travaux d'autres chercheurs ont beaucoup contribué au développement des Grammaires Catégorielles. Nous pouvons citer le modèle de Biskri (1995). C'est un modèle quasi-incrémentale basé essentiellement sur l'utilisation des Grammaires Catégorielle Combinatoire de Steedman. Ce modèle sera présenté en détail au chapitre 4. D'autres travaux n'ont pas été cités dans ce chapitre car ils n'ont pas une influence directe sur notre travail.

CHAPITRE 3

LES GRAMMAIRES APPLICATIVES.

Nous consacrons ce chapitre à la Grammaire Applicative Universelle de Shaumyan (1965, 1977, 1987) et à son extension la Grammaire Applicative et Cognitive (Desclés, 1990). Ainsi, nous aborderons ce qui motive l'utilisation d'une représentation prédicative fonctionnelle construite autour de l'emploi du principe applicatif ainsi que des combinateurs de la logique combinatoire de H.B. Curry (1958).

Sebastian Konstantinovitch Shaumyan a présenté en 1965 la Grammaire Applicative qu'il voulait "universelle". Selon Desclés (1990), le terme universel est pris dans un double sens :

1. d'une part il vise à caractériser les invariants du langage et à établir des formulations universelles des catégories grammaticales,
2. d'autre part, il vise à représenter d'une manière idéale le langage sous-jacent aux langues naturelles et à établir un morphisme du langage génotype vers les langues phénotypes.

Si nous reprenons Biskri (1995), Shaumyan veut prouver, à travers le modèle de la Grammaire Applicative, que les relations syntaxiques ne sont pas fonctions de leurs représentations en ordre linéaire. Il montre aussi que la structure grammaticale d'une langue est indépendante de la manière dont elle est représentée à travers les expressions. Il parle enfin de deux niveaux pour la grammaire :

1. L'étude de la structure grammaticale elle-même.
2. L'étude de la manière dont la structure grammaticale est représentée dans les expressions.

Cette distinction laisse place à un langage abstrait dont Shaumyan dit qu'il est par hypothèse un système de déduction universel[3], autrement dit, qu'il représente un calcul déductif sur les types possibles des structures d'une langue.

Shaumyan distingue deux niveaux de base dans sa théorie :
- Le niveau des observables.
- Le niveau des construits.

Il finit ensuite par conclure que sa théorie distingue deux niveaux:
- Le niveau phénotypique ou phénotype.
- Le niveau génotypique ou génotype.

Le niveau phénotypique est un niveau concret, contrairement au niveau génotypique qui est abstrait.

Nous allons dans ce qui suit présenter le modèle de la Grammaire Applicative Universelle de Shaumyan, les outils qu'elle utilise, et enfin sa version étendue : la Grammaire Applicative et Cognitive (Desclés, 1990).

3.1 Le Modèle de la Grammaire Applicative universelle

Le modèle de la Grammaire Applicative Universelle s'articule autour de deux niveaux de représentation des langues naturelles :
1. Le niveau phénotypique.
2. Le niveau génotypique.

Le niveau phénotypique est représenté par les langues phénotypiques qui sont les langues naturelles telles que nous pouvons les observer. Le niveau génotypique est représenté par un langage abstrait qui va permettre de décrire les langues phénotypiques. Ce langage qui n'appartient à aucune langue naturelle a ses propres règles de calcul qui sont assez

[3] On entend par "système de déduction universel", un système indépendant de la représentation concaténée des termes.

générales pour servir à la description de toutes les langues phénotypiques[4]. Il est basé sur le principe "opérateur/opérande". Il est organisé autour de l'utilisation des outils de la logique combinatoire de Curry (1958). On l'appelle **langage génotypique.**

Pour abstraire toutes les entités du langage génotypique, Shaumyan distingue trois classes d'expressions linguistiques essentielles :

- Les noms des objets qui sont appelés "termes" (terms).
- Les noms des situations qui sont appelés "phrases" (sentences).
- Les moyens nécessaires pour construire les noms des objets et les noms des situations, qui sont appelés "opérateurs".

Un opérateur s'applique à une ou plusieurs expression(s) appelée(s) opérande(s) pour produire un "résultat". Il agit sur un opérande par l'opération "application".

L'application d'un opérateur "f" à un opérande "x" est notée par la simple juxtaposition "f x". L'application est associative à gauche, "f x y" est équivalente à "((f x) y)". Par conséquent il est possible de réécrire des fonctions n-aires en plusieurs fonctions unaires construites progressivement.

Exemples :

- f(x,y,z) est équivalente à ((f x)y)z).
- aime (Marie, Jean) est équivalente à ((aime Marie) Jean)

L'opération applicative est utilisée pour construire toutes les expressions du génotype. C'est d'ailleurs pour cela que la grammaire du génotype est appelée **Grammaire Applicative**. Par conséquent, le langage génotypique est un langage applicatif.

La construction récursive des types syntaxiques à partir des types de base est obtenue par les règles récursives suivantes :

1. Les types de base sont des types; s (sentences) et t (terms) sont des types de base.
2. Si x et y sont des types alors F xy est un type.

F est un opérateur pour la simplification des types, Fxy est un type fonctionnel, il représente le type de toutes les fonctions allant de x vers y.

---

[4]C'est d'ailleurs la raison principale qui fait que l'approche de la Grammaire Applicative est supposée être universelle.

Pour ((*aime Marie*) *Jean*), le type de *Marie* et de *Jean* est t, le type du verbe transitif *aime* est FtFts.

La simplification des types est obtenue avec la règle applicative suivante :

$$\frac{Fxy \quad , \quad x}{y}$$

Nous dirons que x est le type de l'opérande et que Fxy est le type de l'opérateur et le type de l'application de l'opérateur à l'opérande est y.

3.2 La Grammaire Applicative et Cognitive

La Grammaire Applicative et Cognitive est issue des travaux de Jean Pierre Desclés et ses collaborateurs. Elle est une version étendue de la grammaire applicative universelle de Shaumyan.

Cette version reprend les niveaux de représentation des langues naturelles, que sont le phénotype et le génotype. Elle introduit un troisième niveau où seront représentées les significations des prédicats par des schèmes qui sont les générateurs des représentations sémantiques des phrases : le niveau cognitif (Desclés, 1990).

Le système génotype englobe un langage génotype basé sur la notion d'application, une règle de simplification et des combinateurs de la logique combinatoire.

3.2.1 La logique combinatoire

La notion de combinateur est introduite pour la première fois par Schönfinkel (1924), elle est reprise par Curry et Feys (1958). La logique combinatoire a été développée dans le but de trouver une solution logico-mathématique à des paradoxes logiques tels que celui de Russell. Elle est aussi en rapport avec le λ-calcul de Church (1941). Ces deux systèmes d'ailleurs sont les moyens utilisés par les informaticiens pour analyser les propriétés sémantiques des langages de programmation de haut niveau.

Les combinateurs sont des opérateurs abstraits qui permettent de construire, à partir d'opérateurs, des opérateurs de plus en plus complexes. L'action d'un combinateur sur un argument est définie par une règle spécifique appelée **β-réduction**. Cette règle établit une relation entre une expression avec un combinateur et une expression équivalente sans combinateur.

Shaumyan dit que les combinateurs ont une fonction purement syntaxique. Ils forment des combinaisons complexes d'opérateurs indépendamment de leurs significations.

Desclés (1990) prétend que les combinateurs introduisent aussi une sémantique intrinsèque à cause de la β-réduction qui définit en quelque sorte la signification du combinateur.

3.2.1.1 Le combinateur "B" de composition

Soient deux opérateurs complexes f et g. Le combinateur **B** leur associe un opérateur complexe **B** f g tel que pour un argument x on ait la règle de réduction[5] suivante:

$$\mathbf{B}\ f\ g\ x \geq f\ (g\ x).$$

Le combinateur **B** est représenté dans le λ -calcul par la λ -expression : λ x y z x (y z).

[5] β-réduction.

3.2.1.2 Le combinateur "S"de substitution

Le principe de l'action du combinateur **S** est le suivant : soient f et g deux opérateurs, f est binaire et g est unaire. Le combinateur **S** leur associe un opérateur complexe **S** f g. L'action de cet opérateur à un opérande x est spécifiée par la règle de β-réduction suivante :

$$\mathbf{S}\ f\ g\ x \geq f\ x\ (g\ x)$$

La représentation dans le λ-calcul du combinateur **S** est donnée par la λ-expression suivante: λ x y z x z (x z)

3.2.1.3 Le combinateur "C_*" de changement de type

La notion de changement de type introduite au chapitre précèdent est représentée dans la logique combinatoire par le combinateur $\mathbf{C}_*$. L'action de ce combinateur est définie par la β-réduction suivante :

$$\mathbf{C}_*\ X\ Y \geq Y\ X$$

Le combinateur $\mathbf{C}_*$ a pour but de changer le statut de l'opérateur et d'en faire un opérande. Ainsi, si X est un opérateur ayant pour opérande Y alors X devient opérande de l'opérateur ($\mathbf{C}_*$ Y).

Le combinateur $\mathbf{C}_*$ correspond à la λ-expression suivante : λ x y y x.

3.2.1.4 Le Combinateur "C" de permutation

Ce combinateur associe à un opérateur f un opérateur complexe **C** f telle que si f agit sur les opérandes x et y pris dans cet ordre, **C** f agira sur les opérandes y et x pris dans cet autre ordre. L'action du combinateur **C** est définie par la β-réduction suivante :

$$\mathbf{C}\ f\ y\ x \geq f\ x\ y$$

Le combinateur **C** est représentée par la λ-expression suivante : λ x y z x z y.

3.2.1.5 Les combinateurs complexes

Outre les combinateurs élémentaires que nous venons de voir, il existe des combinateurs complexes construits à partir des combinateurs élémentaires. Nous avons par exemple :

B C C ; **B B C_*** ; **B C_***

L'action de ces combinateurs est déterminée par l'application enchaînée des combinateurs élémentaires à partir du combinateur élémentaire le plus à gauche.

Prenons l'expression **B B C_***xyz. On peut considérer que la réduction du combinateur **B B C_*** est exprimée à travers la réduction de **B** puis de **B** puis de **C_***.

1	**B B C_*** x y z	
2	**B** (**C_*** x) y z	élimination-B
3	(**C_*** x) (y z)	élimination-B
4	y z x	élimination- **C_***

3.2.2 Calcul sur les types

La Grammaire Applicative et Cognitive, comme sa devancière la Grammaire Applicative Universelle, combine le calcul des Grammaires Catégorielles avec le calcul des combinateurs développé par Curry dans la logique combinatoire (Curry, Feys, 1958). Elle reprend les types de base s et t.

La construction des types complexes est faite récursivement par les règles :
- s et t sont des types.
- Si x et y sont des types alors Fxy est un type.

Nous pouvons ainsi construire les types suivants :

Ftt : type des déterminants.

Fts : type des verbes intransitifs.

FtFts : type des verbes transitifs.

Etc...

L'opération de base pour la réduction des types est l'opération applicative que nous noterons de la manière suivante :

$$\frac{Fxy \quad x}{y}$$

Le numérateur représente les types en entrée, autrement dit le type de l'opérateur (Fxy) et le type de l'opérande (x). Le dénominateur représente le type résultant de l'opération d'application.

Prenons un exemple :
Soit la notation préfixée applicative de la phrase *Jean-aime-Marie* : aime Marie Jean.
Au prédicat *aime* nous associons le type FtFts. A *Marie* et à *Jean* nous associons le type t.
La réduction des types pour la structure "*aime Marie Jean*" se fait de la manière suivante :

aime	*Marie*	*Jean*
FtFts	t	t
Fts		
s		

3.3 Formes normales

3.3.1 Définitions

Prenons les trois expressions suivantes :

1. **B C** a b c d
2. **B B** $\mathbf{C}_*$ abc
3. a (b c)

Les expressions 1 et 2 peuvent être réduites car elles contiennent des combinateurs. L'expression 3 ne peut pas être réduite car elle ne contient pas de combinateurs.

Définition 1 :

Une expression est dite sous forme normale lorsqu'elle ne peut plus être réduite.

Ainsi l'expression 3 est sous forme normale. Les expressions 1 et 2 ne sont pas sous forme normale.

Définition 2:

Soit E' une expression sous forme normale. Soit E une expression qui n'est pas sous forme normale. Si E est réduite à E' alors nous dirons que E' est la forme normale de E.

Pour les expressions 1 et 2 les formes normales sont respectivement :

a b d c et b c a

3.3.2 Théorème de Church-Rosser [6]

Si une expression combinatoire X se réduit en une expression combinatoire Y_1 et si X se réduit en une autre expression combinatoire Y_2 alors il existe une expression combinatoire Z tel que Y_1 et Y_2 se réduisent en Z.

---

[6] Pour la démonstration de ce théorème se reporter à celle de Martin-Löf et de Tait dans (Hindley, Seldin, 1986).

Ce théorème révèle le fait que la logique combinatoire possède la propriété de Church-Rosser pour les relations transitives.

L'intérêt de ce théorème est dans ses corollaires[7].

Corollaire 1: *Une expression combinatoire a au plus une seule forme normale.*

Corollaire 2 : *Si X=Y alors il existe un Z tel que X —> Z et Y—> Z.*

Corollaire 3 : *Si X = Y et Y est une forme normale, alors X —> Y.*

Corollaire 4 : *Si X = Y alors soit X et Y n'ont pas de forme normale, soit X et Y ont la même forme normale.*

Corollaire 5 : *Si X et Y sont deux expressions combinatoires ayant des formes normales différentes alors X ≠ Y.*

Nous pouvons donc résumer la signification du théorème de Church-Rosser par :

Si une expression combinatoire donnée peut se réduire en une expression sous forme normale, alors cette expression est unique.

3.4 Conclusion

Nous avons présenté dans ce chapitre la Grammaire Applicative Universelle de Shaumyan et son extension la Grammaire Applicative et Cognitive (Desclés, 1990). Le modèle de la Grammaire Applicative Universelle distingue deux niveaux de représentation des langues naturelles : le niveau phénotypique et le niveau génotypique. Avec sa Grammaire Applicative et Cognitive, Desclés ajoute un troisième niveau : le niveau cognitif.

La Grammaire Applicative Universelle combine le calcul des Grammaires Catégorielles avec le calcul des combinateurs développé par Curry dans la logique combinatoire. Les

---

[7]Ces corollaires sont pris dans (Desclés, 1990).

combinateurs sont des opérateurs abstraits qui permettent la construction des opérateurs plus complexes. Chaque combinateur est associé à une règle de β-réduction. Cette règle établit une relation entre une expression avec un combinateur et une expression équivalente sans combinateur.

Nous présenterons au prochain chapitre une méthode qui nous permet de relier des expressions concaténées du niveau phénotypique aux expressions prédicatives du niveau génotypique. Cela suppose évidemment une analyse syntaxique des formes phénotypiques.

CHAPITRE 4

LA GRAMMAIRE CATÉGORIELLE COMBINATOIRE APPLICATIVE.

Nous présentons dans ce chapitre un modèle d'analyse quasi-incrémentale : la Grammaire Catégorielle Combinatoire Applicative de Biskri (1995). Ce modèle qui établit une association canonique entre les règles catégorielles combinatoires de Steedman (1987,1989) et les combinateurs de la logique combinatoire de Curry (1958), évolue dans le cadre des Grammaires Applicatives et Cognitives de Desclés (1990).

La Grammaire Catégorielle Combinatoire Applicative relie explicitement les expressions du phénotype à leurs représentations fondamentales dans le génotype.
Au niveau du phénotype, les caractéristiques particulières des langages naturels sont exprimées (morphologie, syntaxe, etc....). Les expressions linguistiques de ce niveau sont des unités linguistiques concaténées selon les règles syntagmatiques de la langue. Nous écrirons la représentation de concaténation dans le phénotype des unités linguistiques u1, u2, u3, u4, comme suit : u1-u2-u3-u4. En appliquant des règles combinatoires à des unités concaténées, nous construisons une structure applicative renfermant des combinateurs. Cette structure applicative appartient au génotype.

Au niveau du génotype, des invariants grammaticaux et structures qui sont fondamentales aux phrases du niveau du phénotype sont exprimés. Le niveau du génotype utilise un langage formel appelé « le langage génotype ». Dans ce niveau, des interprétations sémantiques fonctionnelles sont exprimées au moyen des combinateurs.

Une analyse fondée sur la Grammaire Catégorielle combinatoire Applicative s'effectue en deux grandes étapes :

1. La première étape consiste à une vérification de la bonne connexion syntaxique des unités linguistiques et la construction des structures applicatives avec l'introduction progressive des combinateurs. L'expression obtenue appartient au langage génotype.
2. La deuxième étape consiste à utiliser les règles de β-réduction de la logique combinatoire de façon à construire une structure applicative en éliminant successivement les combinateurs introduits à la première étape. L'expression obtenue constitue « la forme normale ». Cette dernière exprime l'interprétation sémantique fonctionnelle. Ce dernier calcul s'effectue entièrement dans le génotype.

La construction de l'interprétation sémantique fonctionnelle se fait dans la continuité du calcul syntaxique avec le passage du phénotype au génotype. Ce passage du phénotype au génotype s'apparente au concept de la compilation comme l'illustre la figure 4.1 suivante[8] :

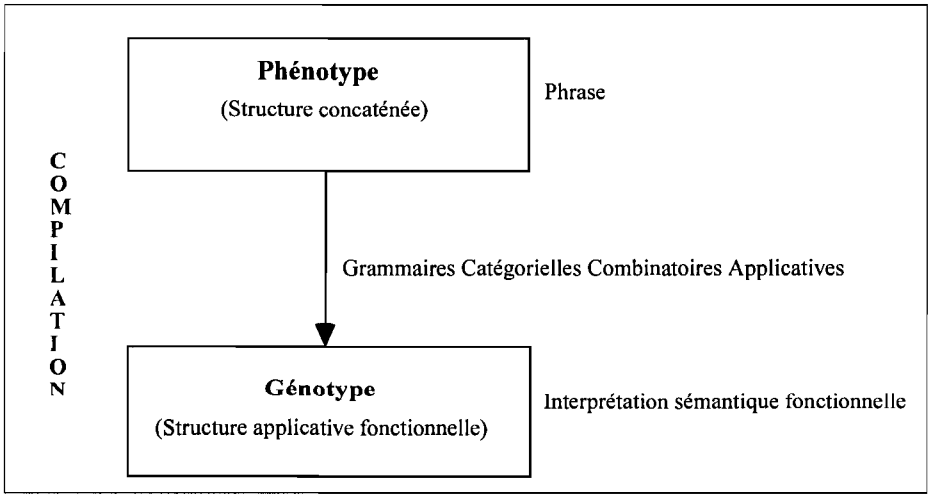


Figure 4.1 : Le passage du phénotype au génotype.

[8] Prise dans Biskri et Desclés, 1997

4.1 Les types d'unités linguistiques

La notion de type dans le cadre de la Grammaire Catégorielle Combinatoire Applicative reste sensiblement la même que pour les modèles de Ajdukiewicz, Bar-Hillel, Lambek et Steedman.

Les Grammaires Catégorielles comportent un ensemble des catégories divisées en deux classes : les catégories de base et les catégories opérateurs. Les Grammaires Catégorielles assignent des catégories aux unités linguistiques. La bonne connexion syntaxique est vérifiée en simplifiant successivement les catégories. Une phrase est une unité linguistique ayant une bonne connexion syntaxique qui se simplifie en une catégorie de phrases. Une Grammaire Catégorielle est donc définie pour la donnée d'une assignation des catégories à ses unités linguistiques. La simplification des catégories revient à effectuer un calcul inférentiel sur les catégories et non pas sur les unités linguistiques. On conceptualise donc les Grammaires Catégorielles comme des systèmes inférentiels de types en considérant chaque catégorie comme un type.

Une Grammaire Catégorielle est donc déterminée par différents types d'unités linguistiques. Toute l'information pour vérifier la bonne connexion des syntagmes est donnée par les types assignés aux unités linguistiques de la grammaire.

A partir des types de base et de symboles opératoires "/" et "\", on peut construire les types complexes.

- Le type X/Y désigne les fonctions de Y vers X sachant que l'opérande de type Y est placé à droite de l'opérateur de type X/Y. Ainsi, un foncteur de type X/Y a l'argument de type Y positionné à droite du foncteur et l'application du foncteur à l'argument donne une unité de type X.
- Le type X\Y désigne les fonctions de Y vers X sachant que l'opérande de type Y est placé à gauche de l'opérateur de type X\Y. Ainsi, un foncteur de type X\Y a l'argument de type Y, positionné à gauche du foncteur et l'application du foncteur à l'argument donne une unité de type X.

Ainsi, dans la notation du type d'un foncteur, le type de l'argument est toujours à droite du symbole opératoire "/" ou "\". Le type de l'application du foncteur à l'argument se trouve quant à lui toujours positionné à gauche du symbole opératoire.

Dans le système de Biskri (1995), les types de base sont au nombre de deux : 'S' pour sentence qui est le terme anglais pour désigner une phrase et 'N' pour noun qui est le terme anglais pour désigner un nom. On peut donc construire des types complexes à partir de ces deux types de base et de symboles opératoires "/" et "\".

Voici quelques types syntaxiques :

N	:Syntagme nominal
S	:Phrase
N/N	:Article ou adjectif
N\N	:Complément de nom
S\N	:Verbe intransitif
(S\N)/ N	:Verbe transitif

4.2 Les règles de la Grammaire Catégorielle Combinatoire Applicative

a. Les règles d'application :

Les règles d'application représentent le concept de base des grammaires catégorielles. Ces règles sont présentées comme suit :

$$\frac{[Y/X : u_1]\text{-}[X : u_2]}{[Y : (u_1\ u_2)]}\ (>)$$

$$\frac{[X : u_1]\text{-}[Y\backslash X : u_2]}{[Y : (u_2\ u_1)]}\ (<)$$

Les lettres en majuscule représentent les types syntaxiques et les lettres en minuscule l'interprétation sémantique fonctionnelle. Les prémisses dans chaque règle sont des expressions concaténées typées considérées comme étant des opérateurs ou des opérandes, le résultat de chaque règle est une expression applicative typée.

b. Les règles combinatoires applicatives [9]:

Les règles combinatoires applicatives introduisent les combinateurs B, S, C, C*.

Les règles de composition fonctionnelle :

$[X/Y : u_1]$-$[Y/Z : u_2]$
----------->**B**
$[X/Z : (\mathbf{B}\ u_1\ u_2)]$

$[X/Y : u_1]$-$[Y\backslash Z : u_2]$
----------->**Bx**
$[X\backslash Z : (\mathbf{B}\ u_1\ u_2)]$

$[Y\backslash Z : u_1]$-$[X\backslash Y : u_2]$
----------<**B**
$[X\backslash Z : (\mathbf{B}\ u_2\ u_1\ )]$

$[Y/Z : u_1]$-$[X\backslash Y : u_2]$
----------<**Bx**
$[X/Z : (\mathbf{B}\ u_2\ u_1)]$

Les règles de composition distributive :

$[(X/Y)/Z : u_1]$-$[Y/Z : u_2]$
----------->**S**
$[X/Z : (\mathbf{S}\ u_1\ u_2)]$

$[X/Y)\backslash Z : u_1]$-$[Y\backslash Z : u_2]$
----------->**Sx**
$[X\backslash Z : (\mathbf{S}\ u_1\ u_2)]$

$[Y\backslash Z : u_1]$-$[(X\backslash Y)\backslash Z : u_2]$
----------<**S**
$[X\backslash Z : (\mathbf{S}\ u_2\ u_1)]$

[9] Présentées par Biskri et Desclés (1997)

$[Y/Z : u_1]$-$[(X\backslash Y)/Z : u_2]$
----------**<Sx**
$[X/Z : (\mathbf{S}\ u_2\ u_1)]$

Les règles de permutation :

$[(X\backslash Y)/Z: u]$
----------**>C**
$[(X/Z)\backslash Y : (\mathbf{C}\ u)]$

$[(X/Y)/Z : u]$
----------**>Cx**
$[(X/Z)/Y : (\mathbf{C}\ u)]$

$[(X/Y)\backslash Z : u]$
----------**<C**
$[(X\backslash Z)/Y : (\mathbf{C}\ u\ )]$

$[(X\backslash Y)\backslash Z : u]$
----------**<Cx**
$[(X\backslash Z)\backslash Y : (\mathbf{C}\ u)]$

c-Les règles de changement de type :

$[X : u]$
----------**>T**
$[Y/(Y\backslash X) : (\mathbf{C}_*\ u)]$

$[X : u]$
----------**>Tx**
$[Y/(Y/X) : (\mathbf{C}_*\ u)]$

$[X : u]$
----------**<T**
$[Y\backslash(Y/X) : (\mathbf{C}_*\ u)]$

$[X : u]$
----------**<Tx**
$[Y\backslash(Y\backslash X) : (\mathbf{C}_*\ u)]$

Les règles combinatoires applicatives font apparaître des combinateurs. La permutation d'une unité u introduit le combinateur **C**; le changement de type d'une unité u introduit le combinateur $\mathbf{C}_*$; la composition fonctionnelle de deux unités concaténées introduit le combinateur **B** et enfin la composition distributive de deux unités concaténées introduit le combinateur **S**. Ces règles permettent de construire une structure applicative faisant apparaître des combinateurs.

Au fur et à mesure de son déroulement, l'analyse syntaxique construit une structure applicative où les combinateurs ont le rôle de manipuler les unités linguistiques, de les composer entre elles et de décomposer les résultats intermédiaires auxquels on peut arriver. Le résultat de cette analyse est une structure applicative qui après réduction des combinateurs donne la forme normale de l'énoncé initiale. Cette forme normale constitue l'interprétation sémantique fonctionnelle de l'énoncé initiale. Avec les règles combinatoires et les règles d'application, nous pouvons analyser une phrase au moyen d'une stratégie quasi incrémentale de gauche à droite.

Prenons l'exemple : *Jean aime Marie.*

Jean - *aime* - *Marie*	
------- ------- --------	
[N:*Jean*] [(S\N)/N:*aime*] [N:*Marie*]	étape 0
------->T	
[S/(S\N):(**C**$_*$ *Jean*)]	étape1
----------->B	
[S/N:(**B** (**C**$_*$ *Jean*) *aime*)]	étape 2
---------->	
[S:((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]	étape 3

A l'étape 0 les unités linguistiques sont associées aux types syntaxiques qui leur conviennent. A l'étape 1, la règle(>T) appliquée à l'unité typée [*N : Jean*] transforme l'opérande en opérateur et génère l'expression ($\mathbf{C}_*$ *Jean*) ayant pour type S/(S\N). La composition avant (>B) est appliquée à ($\mathbf{C}_*$ *Jean*) et *aime* à l'étape 2. Ceci donne comme résultat une structure applicative (B($\mathbf{C}_*$ *Jean*) *aime*) ayant pour type S/N. A la troisième étape, l'application avant(>) s'applique aux entités (B($\mathbf{C}_*$ *Jean*) *aime*) et

Marie et construit l'expression applicative finale (B($\mathbf{C}_*$ *Jean*) *aime*) *Marie*) de type S. Le type S indique que l'énoncé est syntaxiquement correct. Les combinateurs de l'expression applicative finale seront réduits pour donner la forme normale de l'énoncé. Cette dernière représente l'interprétation sémantique de l'énoncé initiale. C'est ce que nous présentons avec la suite de réduction suivante :

[S:((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]

[S:((**C**$_*$ *Jean*) (*aime Marie*))] **B**

[S:((*aime Marie*) *Jean*)] **C**$_*$

Nous pouvons présenter ce passage du système concaténationnel au système applicatif d'une autre manière. Avec cette présentation le passage du phénotype au génotype est plus clairement mis en avant.

1	[N:*Jean*]-[(S\N)/N:*aime*]-[N:*Marie*]		**Structure concaténée typée du phénotype**
2	[S/(S\N):(**C**$_*$ *Jean*)]-[(S\N)/N:*aime*]-[N:*Marie*]		(>**T**)
3	[S/N:(**B** (**C**$_*$ *Jean*) *aime*)]-[N:*Marie*]		(>**B**)
4	[S:((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]		(>)
5	[S : ((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]		**Structure applicative typée du génotype**
6	[S : ((**C**$_*$ *Jean*) (*aime Marie*))]	(**B**)	
7	[S : ((*aime Marie*) *Jean*)]	(**C**$_*$)	**Forme normale du génotype**

A l'étape 1 nous nous situons dans le phénotype et au fur et à mesure que nous appliquons les règles combinatoires applicatives, nous nous rapprochons du génotype. À l'étape 5 nous entrons dans le génotype. La réduction des combinateurs s'effectue dans les deux dernières étapes. Elle donne l'interprétation sémantique de l'énoncé initiale.

4.3 Modifieurs arrières et réorganisation structurelle.

L'analyse syntaxique « de gauche à droite » soulève le problème de non-déterminisme introduit par la présence dans la langue, de modifieurs arrières. Ces derniers sont des opérateurs qui s'appliquent à l'ensemble ou à une partie d'une structure préalablement construite. Nous avons dans la phrase *Jean- frappa-marie-hier*, un modifieur arrière *hier* qui opère sur l'ensemble de la phrase *Jean-frappa-Marie* de type S. Le modifieur arrière *hier* se voit attribuer le type S\S, on peut donc poursuivre l'analyse syntaxique par une règle d'application arrière.

Pour la phrase *Jean-aime-Marie-tendrement* l'analyseur produit dans un premier temps le "faux constituant" [S : ((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]. Le type S de ce faux constituant ne couvre pas tout l'énoncé et n'est pas combinable avec le type de *tendrement*. L'adverbe *tendrement* porte le type (S\N)\(S\N). En effet, *tendrement* est un opérateur ayant comme opérande (*aime Marie*) positionné à sa gauche. Une analyse quasi incrémentale de gauche à droite favorise l'application d'une règle combinatoire dès que possible. Ce facteur a pour conséquence directe de « noyer » certains constituants (en particulier ceux qui doivent être opérande du modifieur arrière) dans d'autre constituant. Dans notre exemple *aime* et *Mari*e sont noyés dans ((**B** (**C**$_*$ *Jean*) *aime*) *Marie*), ce qui évidemment ne nous permet pas de construire directement l'opérande (*aime Marie*).

Le problème posé revient à la possibilité d'un retour arrière. Mais ce retour en arrière est de nature à accroître le coût « computationnel » (mémoire et temps d'exécution) d'une analyse syntaxique. Pour résoudre ce problème, Biskri (1995) propose un retour en arrière qui permet de réduire considérablement ce coût computationnel en construisant des analyses sémantiques correctes et en éliminant les pseudo-ambiguités. Un tel retour en arrière décomposera le constituant déjà construit en deux entités dont une se combine forcément avec le modifieur arrière. Cette opération proposée par Biskri est la réorganisation structurelle. Elle s'effectue en deux étapes successives suivantes :

a. La réorganisation du constituant déjà construit isole deux entités, et teste si le modifieur arrière se combine à gauche avec l'une de ces deux entités. Si le test est négatif, on procède à la réduction des combinateurs jusqu'à ce que le test donne une valeur positive. À la fin du processus, on récupère une nouvelle structure équivalente à la première.

Exemple : Dans le cas de l'énoncé *Jean aime Marie tendrement*, les étapes de la réorganisation sont :

Le constituant construit : [S : ((**B** (**C**$_*$ *Jean*) *aime*) *Marie*)]

Les deux sous-catégories sont : [S/N : (**B** (**C**$_*$ *Jean*) *aime*)] ; [N : *Marie*]

Test : [S/N : (**B** (**C**$_*$ *Jean*) *aime*)] ne se combine pas à gauche avec

[(S\N)\(S\N) : *tendrement*]

[N : *Marie*] ne se combine pas à gauche avec [(S\N)\(S\N) : *tendrement*]

Réduction du combinateur **B** : [S : ((**C**$_*$ *Jean*) (*aime Marie*))]

Les deux sous-catégories sont : [S/(S\N) : (**C**$_*$ *Jean*)] ; [S\N : (*aime Marie*)]

Test : [S/(S\N) : (**C**$_*$ *Jean*)] ne se combine pas à gauche avec

[(S\N)\(S\N) : *tendrement*]

[S\N : (*aime Marie*)] se combine à gauche avec

[(S\N)\(S\N) : *tendrement*]

Arrêt du processus de réduction des combinateurs. Nous récupérons en sortie la catégorie:

[S : ((**C**$_*$ *Jean*) (*aime Marie*))].

b. La décomposition réalisée grâce aux deux règles :

[X : (u_1 u_2)]		[X : (u_1 u_2)]
----------**>dec**	;	----------**<dec**
[X/Y : u_1]-[Y : u_2]		[Y : u_2]-[X\Y : u_1]

Nous lisons ces règles comme suit :

- Pour (**>dec**): Si nous avons une structure applicative (u_1 u_2) de type X, avec u_1

de type X/Y et u_2 de type Y, alors nous pouvons construire une nouvelle expression concaténée formée des deux catégories [X/Y : u_1] et [Y : u_2].

- Pour (**<dec**): Si nous avons une structure applicative (u_1 u_2) de type X, avec u_1 de type X\Y et u_2 de type Y, alors nous pouvons construire une nouvelle expression concaténée formée des deux catégories [Y : u_2] et [X\Y : u_1].

Ces deux règles nous permettent de reconstruire un nouvel agencement concaténé de la structure opérateur/opérande issue de la réorganisation.

Pour la phrase *Jean aime Marie tendrement* la décomposition est appliquée à la structure qui résulte de la réorganisation :

[S : (($\mathbf{C}_*$ *Jean*) (*aime Marie*))]

Avec la règle (**>dec**), nous produisons l'agencement concaténé :

[S/(S\N) : ($\mathbf{C}_*$ *Jean*)] - [S\N : (*aime Marie*)].

Le passage entre un système concaténationnel et un système applicatif pour la phrase *Jean-aime-Marie-tendrement* se présente comme suit :

1	[N : *Jean*]-[(S\N)/N : *aime*]-[N : *Marie*]-[(S\N)\(S\N) : *tendrement*]	
...		
4	[S : ((**B** (**C**$_*$ *Jean*) *aime*)*Marie*)]-[(S\N)\(S\N) : *tendrement*]	
5	[S : (($\mathbf{C}_*$ *Jean*) (*aime Marie*))]-[(S\N)\(S\N) : *tendrement*]	**B**
6	[S/(S\N) : ($\mathbf{C}_*$ *Jean*)]-[S\N : (*aime Marie*)]-[(S\N)\(S\N) : *tendrement*]	(**>dec**)
7	[S/(S\N) : ($\mathbf{C}_*$ *Jean*)]-[S\N : (tendrement (*aime Marie*))]	*(<)*
8	[S : (($\mathbf{C}_*$ *Jean*) (*tendrement* (*aime Marie*)))]	*(>)*
9	[S : (($\mathbf{C}_*$ *Jean*) (*tendrement* (*aime Marie*)))]	*Niveau génotypique*
10	[S : ((*tendrement* (*aime Marie*)) *Jean*)]	$\boldsymbol{C}_*$

Les deux étapes de la réorganisation structurelle entrent dans l'analyse complète de la phrase *Jean-aime-Marie-tendrement* (étape 5 de l'analyse précédente pour la réorganisation et l'étape 6 pour la décomposition).

4.4 La coordination.

La Grammaire Catégorielle Combinatoire Applicative s'est aussi intéressée à la coordination. La coordination est l'action de joindre deux mots ou deux expressions du même genre ou ayant la même fonction. Le but du travail que nous allons présenter au prochain chapitre n'est pas d'analyser la coordination, c'est pourquoi nous ne nous attarderons pas à en parler en détail[10].

4.5 Les métarègles.

Le formalisme de la Grammaire Catégorielle Combinatoire est enrichi par différentes métarègles qui contrôlent le changement de type. Ces métarègles d'une part définissent les situations d'application des règles de changement de type, et d'autre part choisissent la bonne règle de changement de type à appliquer.

Nous n'allons pas non plus présenter en détail les métarègles. Nous avons opté à ne pas utiliser les règles de changement de type dans notre travail pour des raisons que nous allons clarifier au chapitre suivant.

4.6 Conclusion

Nous avons présenté dans ce chapitre un modèle d'analyse : la Grammaire Catégorielle Combinatoire Applicative. Ce modèle évolue dans le cadre des Grammaires Applicatives

[10] Pour en savoir plus consultez : Biskri, La Grammaire Catégorielle Combinatoire Applicative dans le cadre des Grammaires Applicatives et cognitives, 1995

et Cognitives de Desclés. Il est basé sur l'utilisation des Grammaires Catégorielles Combinatoires de Steedman et des combinateurs de Curry.

La Grammaire Catégorielle Combinatoire Applicative relie un énoncé du niveau phénotypique à une analyse applicative sous-jacente exprimée dans le génotype. Ce traitement s'effectue en deux grandes étapes :

1. La vérification de la bonne connexion syntaxique et la construction de structures applicatives avec l'introduction progressive des combinateurs à certaines positions de la chaîne syntagmatique.

2. L'utilisation des règles de β-réduction pour construire une autre structure applicative en éliminant successivement les combinateurs introduits à la première étape.

L'objectif du travail que nous présenterons au prochain chapitre est d'extraire les termes complexes d'une liste des termes candidats. Pour y parvenir nous utilisons un filtre linguistique fondé sur la Grammaire Catégorielle combinatoire Applicative. Les résultats théoriques auxquels nous aboutissons sont appliqués au français.

CHAPITRE 5

EXTRACTION DES TERMES COMPLEXES.

Nous allons consacrer ce chapitre à la présentation d'un filtre linguistique dont le rôle est l'extraction des termes complexes.

Contrairement au terme simple qui est formé d'un seul mot (par exemple, *base*), **un terme complexe** comprend deux ou plusieurs mots séparés par des espaces blancs (par exemple, *base de données*). Dans notre travail nous définissons un **terme complexe** comme étant une unité linguistique qui a une bonne connexion syntaxique qui se simplifie en une catégorie groupe nominal.

Notre filtre linguistique qui est fondé sur la Grammaire Catégorielle Combinatoire Applicative va permettre de :

- Identifier à partir d'une liste des termes candidats, les termes dont la catégorie grammaticale est du groupe nominal.
- Préserver ces termes.
- Rejeter les termes n'ayant pas comme catégorie grammaticale le groupe nominal.

L'identification des termes complexes consiste à trouver dans un corpus les termes candidats dont la catégorie grammaticale est du groupe nominal.
L'analyse que nous proposons pour identifier les termes complexes consiste à appliquer les règles combinatoires aux unités concaténées. Cela a pour conséquence la construction d'une structure applicative renfermant des combinateurs. Dans cette optique deux parties se dégagent :

1. La partie qui consiste à analyser syntaxiquement les termes candidats et qui est obtenue par un calcul sur les types syntaxiques.

2. La partie qui consiste à construire l'interprétation sémantique fonctionnelle et qui est réalisée grâce premièrement à la génération de structures applicatives renfermant des combinateurs puis à la réduction des combinateurs.

Dans ce qui suit, et dans un premier temps, nous présentons les règles combinatoires applicatives que nous allons utiliser dans notre système. Dans un deuxième temps nous présentons le traitement qui nous permet d'identifier les termes complexe à partir d'une liste des termes candidats.

Avant d'aborder cela, nous introduisons la notion de type dans le cadre de notre projet.
La Grammaire Catégorielle Combinatoire Applicative assigne des catégories à chaque unité linguistique. La bonne connexion syntaxique est vérifiée en simplifiant successivement les catégories. Les catégories syntaxiques sont les types orientés, développés à partir des types de base et de deux symboles opératoires " / " et " \ ".
Une unité linguistique " u " avec le type fonctionnel X/Y est considérée comme opérateur (ou fonction) à qui l'opérande de type Y est placé à sa droite.
Une unité linguistique " v " avec le type fonctionnel X\Y est considérée comme opérateur (ou fonction) à qui l'opérande de type Y est placé à sa gauche.

Au vu de ce que nous avons présenté dans les chapitres précédents, le lecteur aura compris que l'idée de base des Grammaires Catégorielles est de penser les mots comme des fonctions. Cette idée de fonction est symbolisée par les types syntaxiques qui sont présentés pour décrire les différentes relations qu'une catégorie syntaxique donnée peut entretenir avec les autres catégories.

Nous rencontrons dans la Grammaire Catégorielle Combinatoire Applicative deux catégories de base : 'S' pour les catégories phrases et 'N' pour les catégories des unités nominales. Puisque le but de notre travail est de trouver dans des corpus les termes complexes (groupes nominaux), il n'est pas utile de considérer les catégories phrases. Nous aurons donc dans les traitements que nous allons effectuer le type de base 'N' pour

les catégories des unités nominales. Nous introduisons un autre type de base 'T' pour les catégories quelconques.

5.1 Les règles combinatoires applicatives utilisées

La vérification de la connexion syntaxique appropriée du terme candidat est obtenue par le calcul sur les types syntaxiques. Ce calcul est basé sur l'utilisation des règles catégorielles combinatoires. Les règles[11] que nous allons utiliser pour analyser les termes candidats sont les règles d'application, les règles de composition fonctionnelle et les règles de permutation. La composition fonctionnelle de deux unités concaténées introduit le combinateur **B** ; la permutation d'une unité introduit le combinateur **C.** Rappelons que les combinateurs sont des opérateurs abstraits qui permettent la construction des opérateurs plus complexes. Chaque combinateur est associé à une règle de β-réduction. Par exemple, nous présentons les combinateurs **B**, **C** avec les règles suivantes (U1, U2, U3 sont des expressions applicatives typées) :

((**B** U1 U2) U3)-> (U1 (U2 U3))
((**C** U1) U2) U3)-> ((U1 U3) U2)

Nous avons opté de ne pas utiliser les règles de changement de type dans notre travail car elles nécessitent toute une série des métarègles pour leurs utilisations. Nous avons par contre préféré l'utilisation de la règle de permutation. En effet, cette règle n'exige aucune métarègle d'une part, d'autre part, elle permet une analyse logique avec une stratégie de gauche à droite. En plus du point de vue technique, le combinateur C, qui est introduit dans l'expression syntagmatique par la règle de permutation, peut être équivalent à une combinaison des combinateurs $\mathbf{C}_*$ et **B**, respectivement introduits dans l'expression syntagmatique par le changement de type et par les règles de composition fonctionnelle. En fait, les expressions combinatoires (a) et (b) suivantes sont équivalentes selon le théorème de Church-Rosser.

---

[11] Ces règles sont exposées au paragraphe 4.2 du chapitre précèdent

a) (((**C** X) Y) Z)

b) (((**B** (**C**$_*$ Y)) X) Z)

En effet,

L e procédé de β -réduction de (a) est

(((**C** X) Y) Z)

((X Z) Y)

L e procédé de β -réduction de (b) est

(((**B** (**C**$_*$ Y)) X) Z)

((**C**$_*$ Y) (X Z))

((X Z) Y)

La forme normale de (a) est identique à la forme normale de (b). Selon le théorème de Church-Rosser. Les expressions combinatoires (a) et (b) sont alors équivalentes.

En appliquant les règles que nous venons de présenter nous pouvons analyser des termes candidats au moyen d'une stratégie quasi-incrémentale de gauche à droite.

1. Traitons le premier exemple suivant :

Théorie complexe

1. [N: *Théorie*] - [N\N: *complexe*]
2. [N: *(complexe Théorie)*] (<)
3. *(complexe Théorie)*

A la première étape, nous attribuons les types aux unités linguistiques, N pour *Théorie* et N\N pour *complexe*. La règle d'application arrière à la deuxième étape nous donne l'expression (*complexe Théorie*) de type N. Le type N est celui qu'il faut pour que le terme candidat soit validé. Donc nous pouvons conclure que le terme candidat *Théorie complexe* est un terme complexe. L'expression à l'étape 3 représente la forme normale du terme complexe validé.

5.2 Identification des termes complexes.

Cette section est consacrée à l'analyse des termes candidats. Comme nous l'avons mentionné au début de ce chapitre, le traitement que nous proposons est basé sur la Grammaire Catégorielle Combinatoire Applicative.

Une analyse syntaxique des formes phénotypiques qui est obtenue par un calcul sur les types syntaxiques va nous permettre de vérifier si un terme candidat est un groupe nominal. Ce calcul est basé sur l'application des règles combinatoires.
L'expression applicative avec combinateurs obtenue après cette analyse syntaxique va donner, après réduction des combinateurs, la forme normale du terme candidat. Ce calcul s'effectue dans le génotype.

Le traitement prend la forme d'une "compilation". Nous illustrons cela par la figure 5.1 :

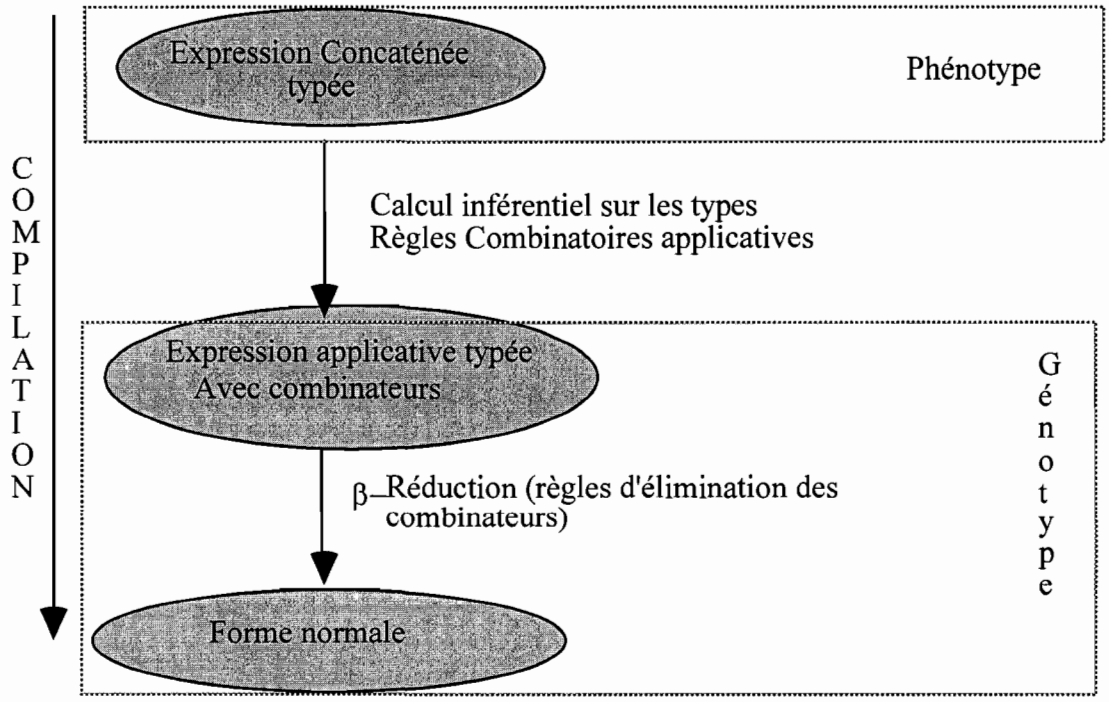


Figure 5.1 : Traitement pour identifier les termes complexes.

Le traitement permettant d'identifier les termes complexes va donc s'effectuer en trois étapes principales suivantes :

1. La première étape consiste à attribuer des catégories aux unités linguistiques.
2. La deuxième étape est illustrée par la vérification de la connexion syntaxique appropriée. Dans le cas qui nous concerne, nous vérifions si le terme candidat est un groupe nominal.
3. La troisième étape consiste à construire la forme normale.

Analysons cet exemple : *Théorie de nombres*

1	[N: *Théorie*]-[(N\N)/T : *de*]-[T: *nombre*]		**Structure concaténée typée du phénotype**
2	[N : *Théorie*]-[(N/T)\N : (***C*** *de)*]-[T: *nombres*]	(>**C**)	
3	[(N/T : ((***C*** *de*) *Théorie*)]-[T: *nombres*]	(<)	**Compilation**
4	[N : (((***C*** *de*) *Théorie*) *nombres*)]	(>)	
		V	
5	(((***C*** *de*) *Théorie*) *nombres*)		**Structure applicative typée du génotype**
6	((*de nombres*) *Théorie*)	(**C**)	**Forme normale du génotype**

Pour cet exemple, l'étape 1 assigne les types catégoriels aux unités linguistiques. Puisque le type de *Théorie* ne peut pas se composer avec le type de *de*, ce dernier subit dans l'étape 2, une opération de permutation qui introduit le combinateur **C**. Les étapes 3 et 4 respectivement actionnent les règles d'application arrière et avant pour donner l'expression (((**C** *de*) *Théorie*) *nombres*) de type N. À l'étape 5 nous entrons dans le génotype. La réduction du combinateur **C** à l'étape 6, construit la forme normale du terme candidat : ((*de nombres*) *Théorie*).

5.2.1 Le modifieur arrière.

L'analyse syntaxique « de gauche à droite » soulève le problème de non-déterminisme introduit par la présence dans la langue, de modifieurs arrières qui se tiennent comme

des opérateurs appliqués à la totalité ou à une partie d'une structure préalablement construite. Comme nous l'avons présenté au chapitre précèdent, il y a des situations où la présence du modifieur arrière ne pose pas de problèmes. Cependant, il existe d'autres situations où l'analyse "bloque". Pour résoudre ce problème, nous faisons appel à la réorganisation structurelle.

Deux étapes successives caractérisent cette réorganisation structurelle :

- La réorganisation du faux constituant.
- La décomposition.

La réorganisation du constituant déjà construite isole à chaque fois deux sous-catégories et teste si le modifieur arrière peut être combiné à gauche ou pas avec un de ces deux sous-catégories. Si aucune des sous-catégories ne se combine avec le modifieur arrière, le terme candidat est considéré comme syntaxiquement incorrecte et il sera rejeté. Dans le cas qui nous concerne nous n'aurons pas besoin de réduction de combinateurs. A la fin de la réorganisation une nouvelle structure applicative typée équivalente à la première est récupérée.

La décomposition permet de reconstruire un nouvel agencement concaténé de la structure opérateur/opérande issu de la réorganisation.

Traitons deux exemples qui illustrent les propos exposés ci-dessus :

Exemple 1 : *Théorie de nombres complexes*

1. [N: *Théorie*] – [(N\N)/T: *de*] – [T: *nombres*] – [T\T: *complexes*]	
2. [N: *Théorie*] – [(N/T)\N: (**C** *de*)] – [T: *nombres*] – [T\T: *complexes*]	(>C)
3. [(N/T) : ((**C** *de*) *Théorie*)] – [T: *nombres*] – [T\T: *complexes*]	(<)
4. [N: ((**C** *de*) *Théorie*) *nombres*)] – [T\T: *complexes*]	(>)
5. [(N/T) : ((**C** *de*) *Théorie*)] – [T: *nombres*] – [T\T: *complexes*]	(Réorganisation structurale)
6. [(N/T) : ((**C** *de*) *Théorie*)] – [T: *complexes nombres*)]	(<)
7. [N: (((**C** *de*) *Théorie*) *(complexes nombres*))]	(>)
8. (((**C** *de*) *Théorie*) *(complexes nombres*))	*Niveau génotypique*
9. ((*de (complexes nombres*)) *Théorie*)	**C**

Après avoir attribué les types aux unités linguistiques à l'étape1, nous construisons par la permutation suivie de l'application arrière et avant, l'expression
((**C** *de*) *Théorie*) *nombres*) de type N. À l'étape 5 l'analyse « bloque ». Pour un terme comme *Théorie de nombres complexes* l'analyseur crée au début le faux constituant *Théorie de nombres*. Ce constituant n'est pas combinable avec *complexes*, puisque le type de *complexes* est T\T tandis que le type de *Théorie de nombres* est N et aucune règle catégorielle ne peut par conséquent être appliquée. En fait, *complexes* est un opérateur ayant comme opérande *nombres* qui se tient à sa gauche. C'est pourquoi nous faisons appel à la réorganisation structurelle. La décomposition est appliquée à la structure qui résulte de la réorganisation de [N: ((**C** *de*) *Théorie*) *nombres*)]. Avec la règle de décomposition (>dec), nous produisons la commande concaténée
[(N/T) : ((**C** *de*) *Théorie*)] – [T: *nombres*]. Nous pouvons maintenant continuer l'analyse. En utilisant les règles d'application arrière et avant aux étapes 6 et 7 nous construisons l'expression typée [N: (((**C** *de*) *Théorie*) *(complexes nombres*))]. Les étapes 8 et 9 du génotype nous mènent à la forme normale ((*de (complexes nombres*)) *Théorie*) après réduction du combinateur C.

Exemple 2 : *Base de la théorie des nombres*

1. [N: *Base*] – [(N\N)/N: *de*] – [N/T: *la*] – [T: *théorie*] – [(T\T)/T: *des*] – [T: *nombres*]	
2. [N: *Base*] – [(N/N)\N: (**C** *de*)] – [N/T: *la*] – [T: *théorie*] – [(T\T)/T: *des*] – [T: *nombres*]	(>C)
3. [(N/N) : ((**C** *de*) *Base*)] – [N/T: *la*] – [T: *théorie*] – [(T\T)/T: *des*] – [*T: nombres*]	(<)
4. [(N/T) : (**B** ((**C** *de*) *Base*) *la*)] – [T: *théorie*] – [(T\T)/T: *des*] – [T: *nombres*]	(>B)
5. [N: ((**B** ((**C** *de*) *Base*) *la*) *théorie*)] – [(T\T)/T: *des*] – [T: *nombres*]	(>)
6. [(N/T) : (**B** ((**C** *de*) *Base*) *la*)] – [T: *théorie*] – [(T\T)/T: *des*] – [T: *nombres*]	(Réorganisation Structurelle)
7. [(N/T) : (**B** ((**C** *de*) *Base*) *la*)] – [T: *théorie*] – [(T/T)\T: (**C** *des*)] – [T: *nombres*]	(>C)
8. [(N/T) : (**B** ((**C** *de*) *Base*) *la*)] – [(T/T) : ((**C** *des*) *théorie*)] – [T: *nombres*]	(<)
9. [(N/T) : (**B** (**B** ((**C** *de*) *Base*) *la*) ((**C** *des*) *théorie*))] – [T: *nombres*]	(>B)
10. [N: ((**B** (**B** ((**C** *de*) *Base*) *la*) ((**C** *des*) *théorie*)) *nombres*)]	(>)
11. ((**B** (**B** ((**C** *de*) *Base*) *la*) ((**C** *des*) *théorie*)) *nombres*)	*Niveau génotypique*
12. ((**B** ((**C** *de*) *Base*) *la*) (((**C** *des*) *théorie*) *nombres*))	**B**
13. (((**C** *de*) *Base*) *(la* (((**C** *des*) *théorie*) *nombres*)))	**B**
14. ((*de (la* (((**C** *des*) *théorie*) *nombres*))) *Base*)	**C**
15. ((*de (la* ((*des nombres*) *théorie*))) *Base*)	**C**

Nous rencontrons à l'exemple 2 le processus observé à l'exemple précédent. On ne peut pas continuer l'analyse du terme candidat *Base de la théorie des nombres* à l'étape 6. En effet, le constituant ((**B** ((**C** *d*e) *Base*) *l*a) *théori*e) est faux puisque *théorie* est opérande du modificateur *des nombres*. Nous faisons appel à la réorganisation structurelle comme dans l'exemple précédent pour pouvoir continuer l'analyse. Nous aboutissons à l'étape 10 à l'expression typée [N: ((**B** (**B** ((**C** *d*e) *Base*) *l*a) ((**C** *d*es) *théori*e)) *nombre*s)]. Nous entrons dans le génotype à l'étape 11. La réduction des combinateurs dans les quatre dernières étapes nous permet d'obtenir la forme normale ((*de (la* ((*des nombre*s) *théori*e))) *Base*).

5.2.2 Commentaires

Dans nos exemples nous avons procédé aux analyses incrémentales de gauche à droite. Ces genres d'analyses éliminent le phénomène de pseudo-ambiguïté qui consiste à construire plusieurs arbres de la dérivation syntaxique correspondant seulement à une interprétation sémantique. Ces analyses syntaxiques nous ont permis de constater que tous les termes candidats analysés jusqu'à présent dans ce chapitre sont de la catégorie N (groupe nominal). C'est cette catégorie dont a besoin un terme candidat pour être validé. Les quatre termes candidats :

1. Théorie complexe.
2. Théorie de nombres.
3. Théorie de nombres complexes.
4. Base de la théorie des nombres.

Sont de la catégorie N. Ils sont identifiés comme étant des termes complexes, ils sont donc préservés par le filtre linguistique.

Ces termes candidats analysés suivent certains modèles français décrits dans (Daille, 1994), (Sta, 1998) :

- Adjectif de nom (1).
- Nom « de » Nom (2).
- Nom « de » l'adjectif du nom (3).
- Nom « de » « la » Nom « des » Nom (4).

5.2.3 Sont-ils à préserver ou à rejeter ?

L'usage de la langue française permet un agencement des mots tel que « Nom - Nom ». L'analyse d'un terme candidat comprenant un tel agencement pose problème car aucune règle combinatoire ne peut combiner deux unités linguistiques de type N juxtaposées. Clarifions cela par un exemple.

Considérons les termes candidats suivants :

- Fonction membre virtuelle.
- Liens hypertextes.

Nous attribuons les types aux unités linguistiques du terme *Fonction membre virtuelle* de la manière suivante :

[*N : Fonction*]-[*N : membre*]- [*N\N : virtuelle*]

L'analyse de ce terme va « bloquer » dès le début du traitement, car il n y a aucune règle combinatoire qui peut combiner [*N : Fonction*] et [*N : membre*]. Dans cette situation le terme candidat : *Fonction membre virtuelle* sera rejeté car il est considéré comme syntaxiquement incorrect.

Si nous considérons *membre* comme modifieur de *Fonction*, nous aurons le traitement suivant :

1 [*N : Fonction*] - [*N\N : membre*] – [*N\N : virtuelle*]
2 [*N : (membre Fonction)*] - [*N\N : virtuelle*] *(<)*
3 [*N : (virtuelle(membre Fonction))*] *(<)*
4 *(virtuelle(membre Fonction))*

Cette deuxième situation nous permet de procéder à l'analyse complète du terme candidat. Après avoir attribuer les types aux unités linguistiques à l'étape 1, nous utilisons deux fois la règle d'application arrière aux étapes 2 et 3 et nous obtenons une expression typée [*N : (virtuelle(membre Fonction))*]. N est le type qu'il faut pour que le terme candidat soit préservé par le filtre linguistique. Ainsi donc *Fonction membre virtuelle* est un terme complexe. Nous procédons de la même façon pour l 'autre exemple.

Nous ne pouvons pas procéder à une analyse complète du terme candidat : *Liens hypertextes* que si nous considérons *hypertextes* comme modifieur de *Liens*. Dans ce cas nous aurons le traitement qui suit :

1. [N: *Liens*] - [N\N: *hypertextes*]
2. [N: *(hypertextes Liens)*] (<)
3. *(hypertextes Liens)*

A l'étape 1, nous attribuons les types syntaxiques aux unités linguistiques. Nous construisons une expression typée [N: *(hypertextes Liens)*] à l'étape 2 par l'utilisation de la règle d'application arrière. L'étape 3 nous donne la forme normale du terme analysé. *Liens hypertextes* est de type N. C'est donc un terme complexe.

5.3. Conclusion

Nous avons présenté dans ce chapitre les détails théoriques d'un filtre linguistique dont le but est l'identification des termes complexes. Ce filtre linguistique est fondé sur un modèle de Grammaire Catégorielle Combinatoire Applicative. Ce dernier prend la forme d'une compilation. La construction de l'interprétation sémantique fonctionnelle se fait dans la continuité du calcul syntaxique avec le passage du phénotype au génotype. C'est grâce à ce modèle catégoriel que nous avons pu identifier les termes complexes à partir d'une liste des termes candidats. Une analyse syntaxique des formes phénotypiques qui est obtenue par un calcul sur les types syntaxiques nous a permis de vérifier si un terme candidat est du groupe nominal. Ce sont les termes candidats ayant comme catégorie le groupe nominal qui sont préservés par notre filtre linguistique.

L'expression applicative obtenue après l'analyse syntaxique nous a donné après réduction des combinateurs la forme normale du terme complexe.

Un problème s'est posé concernant l'analyse des termes candidats ayant la forme :
"nom - nom". Dans cette situation, le deuxième terme ne se combine pas avec le premier parce qu'aucune règle combinatoire applicative ne peut être utilisée. Nous avons résolu le

problème en considérant le deuxième terme comme modifieur du premier, ce qui a permis à l'analyse de continuer en utilisant une règle d'application.

Nous allons consacrer le chapitre suivant à l'implémentation d'un prototype permettant une analyse basée sur les résultats théoriques que nous venons de présenter.

CHAPITRE 6

IMPLÉMENTATION

Ce chapitre est consacré à l'implémentation d'un filtre linguistique dont le but est l'identification des termes complexes. Ce filtre linguistique est fondé sur un modèle de Grammaire Catégorielle Combinatoire Applicative.

L'objectif principal que nous visons à travers cette implémentation est de prouver que nous pouvons concevoir un programme informatique qui met en pratique les résultats théoriques auxquels nous sommes arrivés.
L'implémentation a été faite en C++. Nous avons choisi le langage C++ parce qu'il permet la programmation orientée objet. Nous souhaitons profiter de toutes sortes d'avantages qu'offre ce genre de paradigme de programmation. Il nous offre la possibilité de définir nos propres types de données pour les variables qui vont rentrer en jeu dans l'implémentation de notre projet.

6.1 Le langage C++

Le C++ est un langage de programmation. Ce langage permet la programmation sous de multiples paradigmes comme, par exemple, la programmation procédurale, la programmation générique et la programmation orientée objet.

La programmation orientée objet a pour ambition, en particulier de faciliter l'activité de programmation, elle permet notamment :

- Le développement des composants réutilisables.

- L'abstraction entre l'implémentation des modules et leur utilisation, apportant ainsi un plus grand confort dans la programmation. Elle s'intègre donc parfaitement dans le cadre de la modularité.
- L'encapsulation des données pour une meilleure protection et donc une plus grande fiabilité des programmes.
- La définition des types de données par l'utilisateur : **les classes**.

Les classes constituent une sorte de type de données pour lesquelles l'accès aux informations n'est possible que via cette classe. Dans une classe se trouve associées à la fois des **données** (on parle des membres donnés) et des **fonctions** (on parle de fonctions membres ou de méthodes).

Un **objet** est un ensemble de données sur lesquelles des méthodes peuvent être appliquées. Les objets d'une classe sont des instances de cette classe. La classe définit donc la structure de données, appelées **champs** ou **variables d'instances**, que les objets correspondants auront, ainsi que les méthodes de l'objet. À chaque instanciation, une allocation de mémoire est faite pour les données du nouvel objet créé. L'initialisation de l'objet nouvellement créé est faite par une méthode spéciale, le **constructeur**. Lorsque l'objet est détruit, une autre méthode est appelée : le **destructeur**. L'utilisateur peut définir ses propres constructeurs et destructeurs d'objets.

Les données des objets peuvent être protégées : c'est-à-dire que seules les méthodes de l'objet peuvent y accéder. Ce n'est pas une obligation, mais cela accroît la fiabilité des programmes. Si une erreur se produit, seules les méthodes de l'objet doivent être vérifiées. Cette notion de protection des données et de masquage de l'implémentation interne aux utilisateurs de l'objet constitue ce que l'on appelle l'**encapsulation**.

Un programme C++ se compose d'un ensemble de fichiers contenant des textes source compilés séparément. Un fichier peut contenir des définitions des classes, des fonctions et des variables.

Nous ne nous étalerons pas sur la présentation du langage C++. Il existe plusieurs ouvrages qui permettent aux lecteurs intéressés d'approfondir leur culture du C++.

6.2 Les éléments de l'implémentation

L'implémentation d'un filtre linguistique nécessite un certain nombre d'éléments en particulier le choix des structures de données pour les variables qui vont nécessairement rentrer en jeu.

Nous attachons plus d'importance à celles qui ont un rôle incontournable dans notre implémentation. Les principales données que nous véhiculons entre les différentes classes du programme sont de deux genres :

- Le premier genre de données correspond à la représentation des types syntaxiques.
- Le deuxième genre de données correspond à la représentation des constructions prédicatives.

6.2.1 Structure de données pour les types syntaxiques.

Nous rencontrons dans une analyse syntaxique deux sortes de types syntaxiques : des types syntaxiques foncteurs et des types syntaxiques de base. Dans cette section nous allons voir comment organiser les structures de données pour les uns et les autres. La structure de données qui a été choisie est l'arbre binaire car nous estimons qu'elle nous offre la facilité de représenter les deux types syntaxiques.

La structure de données "arbre binaire" :

En informatique, un **arbre binaire** est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé nœud, le nœud initial étant appelé **racine**. Dans un arbre binaire, chaque élément possède au plus deux éléments fils

au niveau inférieur, habituellement appelés **gauche** et **droit**. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé **père**.

Au niveau le plus élevé, il y a donc un nœud racine. Au niveau directement inférieur, il y a au plus deux nœuds fils. En continuant à descendre aux niveaux inférieurs, on peut avoir quatre, puis huit, puis seize, etc. C'est-à-dire la suite des puissances de deux. L'ensemble des nœuds situé à gauche de la racine forme le sous arbre binaire gauche et celui situé à droite de la racine constitue le sous arbre binaire droit. Un nœud n'ayant aucun fils est appelé **feuille**. Le nombre de niveaux total, autrement dit la distance entre la feuille la plus éloignée et la racine, est appelé **hauteur** de l'arbre. Le niveau d'un nœud est appelé **profondeur**. La figure 6.1 illustre le vocabulaire qui vient d'être décrit.

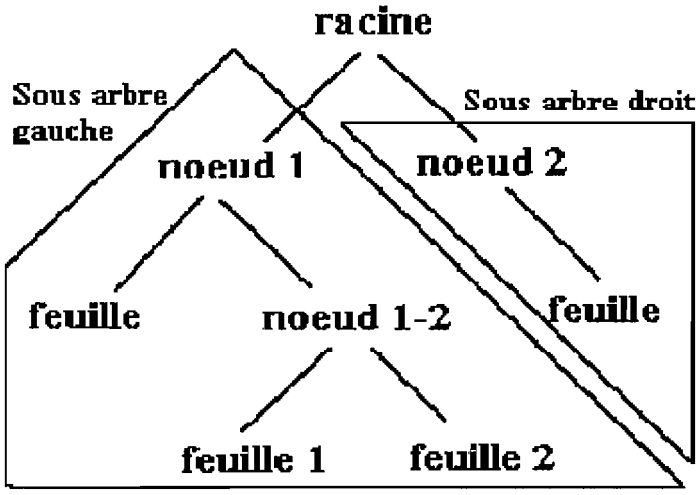


Figure 6.1 : Arbre binaire.[12]

6.2.1.1 Les types de base

Les types de base n'introduisent aucun symbole d'opération et donc aucun opérande. Nous avons choisi de les représenter par une seule lettre :

---

[12] Tiré de : http://www.dailly.info/algorithmes-de-tri/abr.php

- 'N' pour les syntagmes nominaux.

- 'T' pour les unités linguistiques quelconques.

Pour représenter un type de base nous concevons un arbre binaire contenant un seul nœud (racine). Chaque élément du nœud est un caractère. Nous aurons :

• Dans le cas d'un syntagme nominal, le type syntaxique :

arbre1->racine->element = 'N'

• Dans le cas d'une unité linguistique quelconque, le type syntaxique :

arbre2-> racine->element = 'T'.

6.2.1.2 Les types foncteurs

Les types foncteurs sont représentés en Grammaire Catégorielle Combinatoire Applicative par X/Y et X\Y.

Nous représentons ces types foncteurs par un arbre binaire.

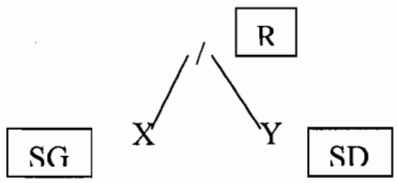
Si nous représentons

- La racine de l'arbre binaire par R,

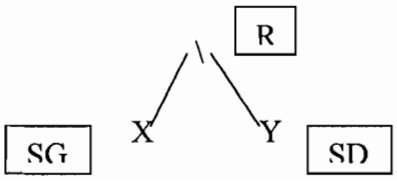
- le sous arbre binaire gauche par SG,

- le sous arbre binaire droit par SD.

Le type syntaxique X/Y peut être représenté par SG R SD. Avec une représentation en arbre généalogique nous aurons :



Le type syntaxique X\Y peut être représente par SG R SD. Avec une représentation en arbre généalogique nous aurons :



R correspondant à la racine de l'arbre représente le symbole opératoire,

SD correspondant au sous arbre droit représente le type de l'opérande,

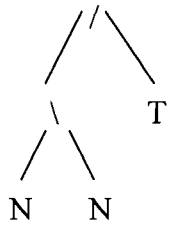
SG correspondant au sous arbre gauche représente le type du résultat de l'application de l'opérateur à l'opérande.

Donnons quelques exemples concrets :

Prenons les unités concaténées typées suivantes :

[N : Théorie]–[(N\N)/T : de]-[T : nombres]-[T\T : complexes]

• Le type syntaxique foncteur (N\N)/T de "de" est représenté comme suit : arbre2



L'instruction suivante nous permet d'accéder à l'élément R et de ce fait, connaître sa valeur : arbre2-> racine-> element[13] ; pour cet exemple la valeur retournée est '/'.

Nous accédons à l'élément SD, par l'instruction : arbre2->racine->filsd[14]. La valeur retournée est 'T'.

Nous accédons à l'élément SG, par l'instruction : arbre2->racine->filsg[15]. La valeur retournée est N\N.

6.2.2 Structure de données pour les constructions prédicatives

Nous avons eu l'occasion dans les chapitres précédents, de présenter deux aspects principaux qui caractérisent les expressions applicatives :

---

[13] Nous donnerons la définition de l'arbre binaire de notre programme dans les paragraphes suivants

[14] filsd : pointe vers le sous arbre situé à droite de la racine de l'arbre

[15] filsg : pointe vers le sous arbre situé à gauche de la racine de l'arbre

- Le rapport opérateur/opérande que peuvent entretenir les unités linguistiques[16].
- L'utilisation des combinateurs[17].

Dans notre programme, nous avons représenté ces deux aspects par des chaînes de caractères. Nous représentons le facteur opérateur/opérande par la concaténation de deux chaînes de caractères. Nous nous servons de l'opération de concaténation pour les chaînes de caractères de la classe Cstring qui est le signe " + ".
L'expression applicative (x y) est une chaîne de caractères résultant de la concaténation de l'opérateur x à l'opérande y : "x "+ "y" ;
Nous procédons de la même manière pour rendre compte de la portée des combinateurs. Nous concaténons le combinateur aux autres éléments ; les opérateurs qui sont soumis à l'action du combinateur pour former l'opérateur complexe.

• Pour le combinateur **B**, nous concaténons le combinateur **B** à x et y sur lesquels il agit pour former l'opérateur complexe (**B** x y) : '**B**' + "x" + "y"; Cette instruction nous donne une chaîne de caractères "**B** x y" que nous mettons entre parenthèses pour obtenir l'opérateur complexe (**B** x y).

• Pour le combinateur **C,** nous concaténons le combinateur **C** à x sur lequel il agit pour former l'opérateur complexe (**C** x). L'instruction '**C**' + "x" ; nous donne une chaîne de caractères "Cx", que nous mettons entre parenthèses pour former l'opérateur complexe (**C** x).

6.3 Les classes du programme

Il n'est peut être pas utile de le rappeler, mais un programme en informatique doit être construit de façon modulaire, chaque module ayant un rôle bien établi à remplir.

---

[16]Rappelons que le rapport opérateur/opérande est introduit par les deux règles d'application fonctionnelle. Les expressions applicatives sous forme normale adoptent un rapport opérateur/opérande strict. Elles n'utilisent pas de combinateurs.
[17]Rappelons que des combinateurs sont introduit au niveau des expressions applicatives lorsque dans l'analyse syntaxique, on applique une règle combinatoire autre que les règles d'application fonctionnelle.

Nous avons défini plusieurs classes (types de donnée) afin de représenter certaines variables importantes de notre programme. Dans une classe se trouvent associées à la fois des données et des méthodes. Seules les méthodes de la classe peuvent accéder aux données de l'objet. La notion de classe s'intègre donc dans le cadre de la modularité car il y a aussi une abstraction entre l'implémentation d'une classe et son utilisation. Nous exposons dans ce paragraphe les différentes classes de notre programme. Elles sont utilisées par certaines classes de notre programme et par le programme principal. Notons toutefois que nous ne présentons que les classes les plus importantes.

6.3.1 Lecture des termes candidats et enregistrement des termes complexes

La classe CFichier :

Les méthodes définies dans cette classe nous permettent de :

- Sélectionner un fichier contenant des termes candidats.
- Ouvrir ce fichier.
- Lire les données.

Dans la même classe nous définissons la méthode qui permet de sauvegarder les termes complexes résultats de l'analyse syntaxique dans une base de données XML.

6.3.2 Définition de la structure des données pour les types syntaxiques

La classe Cnoeud :

Comme nous l'avons mentionné plus haut, nous utilisons les arbres binaires pour représenter les types syntaxiques. La classe Cnoeud définit une structure de données arbre binaire. Elle définit les données membres suivantes :

- char element ;
- Cnoeud * filsg;

- Cnoeud * filsd;

Cnoeud définit donc un élément de type caractère et deux pointeurs :

- *filsg* : un pointeur vers une donnée de type Cnoeud située à gauche du nœud *element*
- *filsd* : un pointeur vers une donnée de type Cnoeud située à droite du nœud *element*

Nous représentons schématiquement à la figure 6.2 les données de la classe Cnoeud de la manière suivante :

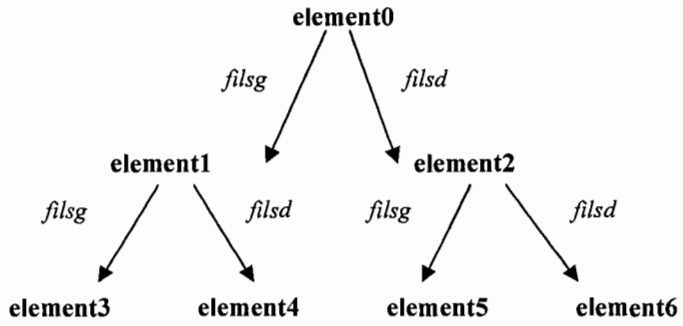


Figure 6.2 : Arbre schématisant la structure de la classe Cnoeud.

6.3.3 Insertions d'éléments dans l'arbre et parcours de l'arbre

La classe Carbre :

Les données membres de cette classe sont les suivantes :

- Cnoeud * racine ;
- Cnoeud * courant ;
- int nbr_noeud;

La donnée *racine* est un pointeur vers la racine de l'arbre, *courant* est un pointeur vers le nœud courant enfin *nbr_nœud* est un entier qui nous renseigne sur le nombre d'éléments contenus dans l'arbre.

Les fonctions membres importantes sont :

- Les fonctions qui permettent d'insérer des éléments dans l'arbre.
- Les fonctions pour parcourir l'arbre
    - Se positionner à la racine.
    - Se positionner à gaucher du nœud courant.
    - Se positionner à droite du nœud courant.
- La fonction pour détruire l'arbre.

6.3.4 Conversion des types syntaxiques chaînes de caractères en arbres binaires

La classe Ctype :

Les types syntaxiques sont attribués manuellement aux unités lexicales. Ils sont saisis un par un. Pour faciliter la saisie des types syntaxiques à un utilisateur donné, nous avons préféré que la variable qui les représente soit organisée en chaîne de caractères. La classe Ctype définit des méthodes qui sont appliquées aux types syntaxiques chaînes de caractères pour les convertir en types syntaxiques arbres binaires. La plus importante de ces méthodes est la méthode Type_en_arbre.

Exemple : le type syntaxique (X\Y)\Z est représenté comme tel à la saisie. La méthode Type_en _arbre va le convertir en arbre binaire pour donner l'arbre binaire X\Y\Z.

6.3.5. Les règles combinatoires applicatives.

La classe CRegle

Cette classe implémente les règles combinatoires applicatives que nous avons présentées au chapitre précédent. La classe CRegle définit les règles d'application fonctionnelle, les

règles de composition et les règles de permutation. Nous reviendrons sur l'implémentation de règles combinatoires applicatives au prochain paragraphe. Toutefois, nous présentons ci-dessous les fonctions qui implémentent les règles suivantes :

- Y – X\Y==> X,
- X/Y-Y/Z==>X/Z,
- (X\Y)/Z==> (X/Z)\Y.

Rappelons que les types syntaxiques sont organisés en arbre binaire.

La règle Y – X\Y==> X

Les règles d'application mettent en rapport un opérateur avec son opérande.

Nous devons déclarer deux variables en paramètre au minimum pour construire la règle Y – X\Y==> X :

- Un arbre binaire qui représente un type syntaxique foncteur.
- Un arbre binaire qui représente l'opérande.

Y est représenté par arbre1 et X\Y par arbre2

```
Si arbre2→ racine→ element = = '\\'
  et Si arbre1→racine = =arbre2→racine→filsd  alors
       retourner arbre2->racine->filsg
```

Exemple : T – N\T

arbre1 = T ; arbre2 = N \ T

test :

```
|          arbre2→racine→élément = = '\\'
et |       arbre1→racine = = T
   |       arbre2 →racine→filsd = = T
```

Retourner Arbre2->racine->filsg c à d N pour cet exemple.

La règle X/Y-Y/Z==>X/Z

Les règles de composition sont différentes des règles d'application en ce sens que les règles d'application mettent en rapport un opérateur avec son opérande alors que les règles de composition mettent en rapport deux opérateurs, le résultat de l'application d'un des deux opérateurs à son opérande donne l'argument du deuxième opérateur. Rappelons toutefois que l'opérateur complexe construit par la composition héritera d'un type opérateur construit à partir des types des opérateurs initiaux.

Nous avons parmi les variables définies pour construire les règles de composition fonctionnelle :

- Un arbre binaire *arbre1* pour représenter le premier type syntaxique foncteur.
- Un arbre binaire *arbre2* pour le deuxième type syntaxique foncteur.
- Un arbre binaire *arbreRes* pour représenter l'opérateur complexe construit par la composition

X/Y est représenté par arbre1 et Y/Z par arbre2

```
Si arbre1→racine→element = = '/'
   et Si arbre2→racine→element = = '/'
     et Si arbre1→racine→filsd = =arbre2→racine→filsg alors
          arbreRes→racine→element = '/' ;
          arbreRes→racine→filsg  = arbre1→racine→filsg ;
          arbreRes→racine→filsd  = arbre2→racine→filsd ;
```

Exemple : N/T– T/N

Test :

Arbre1 = N/T ; arbre2 = T/N

arbre1→racine→element = = '/'

et | et | arbre2→racine→element = = '/'

et | | arbre1→racine→filsd = = T

et | et | arbre2→racine→filsg = = T

Retourner : arbreRes = N/N

La règle (X\Y)/Z==> (X/Z)\Y

Les règles de permutation sont caractérisées par la permutation des positions des opérandes. Nous définissons la variable l'arbre binaire *arbre1* pour pouvoir construire cette règle.

(X\Y)/Z est représenté par arbre1

Si arbre1→racine→element = = '/'

et Si arbre1→racine→filsg->element= = '\\' alors

arbre1→racine→element = '\\' ;

arbre1→racine→filsg->element = '/' ; et

Nous permutons les noeuds Arbre1→racine→filsg→filsd et

arbre1→racine→filsd

Exemple : (N\N)/T

test :

arbre1 == N\N/T

| | arbre1→racine→element = = '/'

| et | arbre1→racine→filsg->element = = '\\'

Resultat : Arbre1= N/T\N

6.3.6 Réduction des combinateurs.

La classe CCombinateur

C'est la classe CCombinateur qui définit les méthodes permettant de construire les formes normales.

La première méthode teste la présence d'un combinateur dans la structure applicative générée par l'analyse syntaxique. Si un combinateur est détecté, la deuxième méthode de la classe vérifie si c'est le combinateur "**C**" engendré par l'application de la permutation ou le combinateur "**B**" engendré par l'application de la composition fonctionnelle. Si le combinateur détecté est **C**, la méthode Reduire_C qui implémente la règle de β-réduction pour éliminer le combinateur **C** de la structure applicative est appelée. Par contre si c'est le combinateur **B** qui est détecté, c'est la méthode Reduire_B qui implémente la règle de β-réduction pour éliminer le combinateur **B** de la structure applicative qui est appelée.

6.4 Programme principal.

Notre programme est une application de type "Dialogue" en Visual C++. Le programme principal est donc une classe qui dérive de la classe CDialog. CDialog fait partie des classes de Microsoft Foundation Class (MFC) qui est un ensemble des classes prédéfinies de Visual C++. Nous allons présenter dans cette section les méthodes importantes du programme principal. Ces méthodes sont classifiées dans les catégories suivantes :

- Lecture des données.
- Attribution des types.
- Analyse syntaxique.
- Affichage des termes complexes.
- Présentation des termes complexes dans un document XML (ou base de données XML).

Nous présentons les principales méthodes dans la figure 6.3. Cette figure donne une idée générale sur le déroulement de notre programme principale.

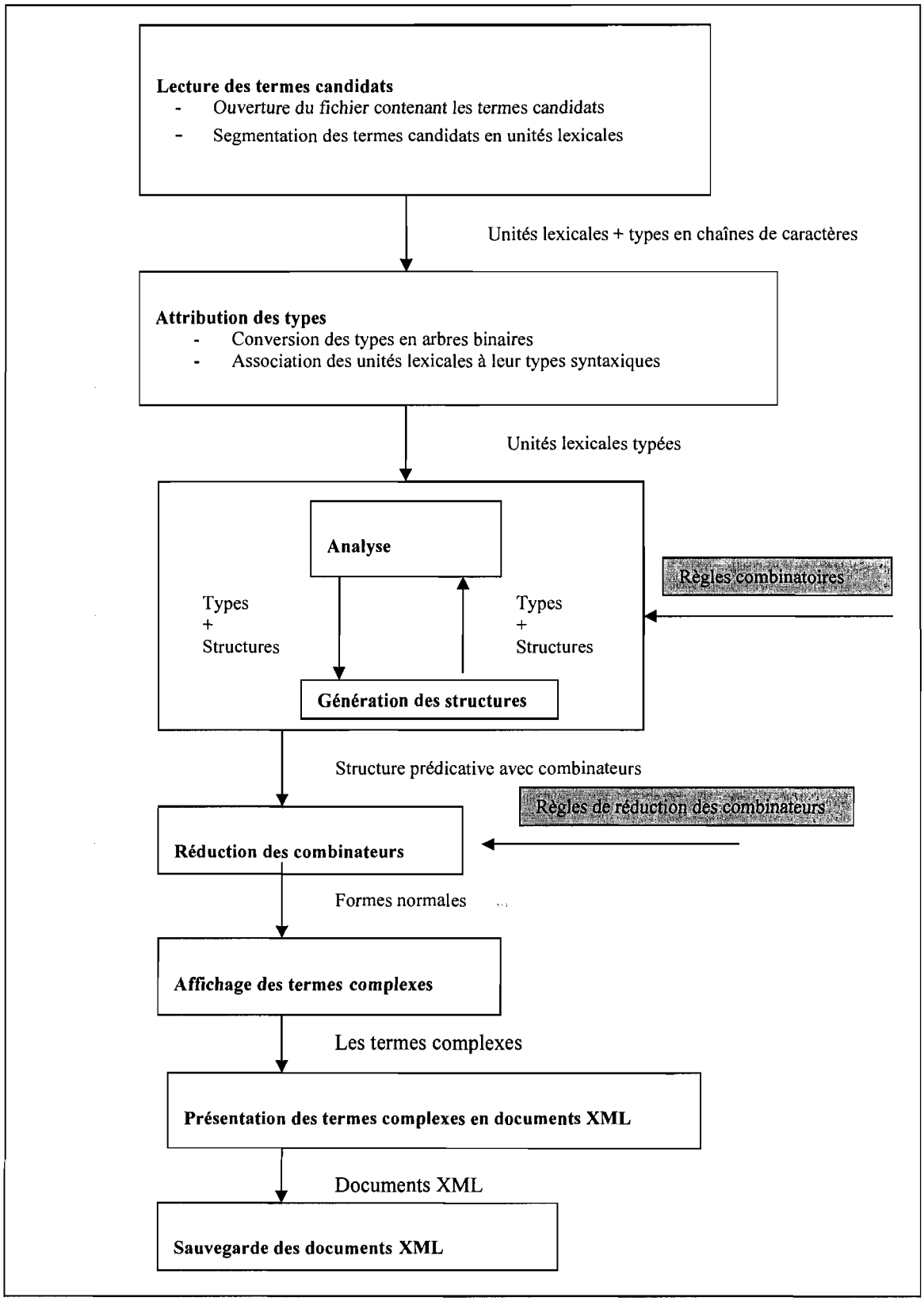


Figure 6.3 : Structure générale du programme principal.

6.4.1 Lecture des données

Nous avons classé dans cette catégorie les méthodes qui permettent la lecture des termes candidats et des types syntaxiques des unités lexicales qui forment ces termes candidats.

Pour analyser les termes candidats, il faut au préalable les lire. Grâce aux fonctions membres de la classe Cfichier nous pouvons sélectionner le fichier contenant les termes candidats, l'ouvrir et l'afficher dans une boîte de dialogue. Chaque terme candidat est découpé en mots permettant ainsi de former les unités lexicales auxquelles nous allons attribuer un type syntaxique. Les types syntaxiques sont attribués manuellement. Ils sont saisis un par un.

Exemple :

Le terme candidat *Théorie de nombres complexes* est découpé en mots (unités lexicales). Ces mots sont rangés dans un List Box représenté par la variable *m_listmots*. On associe à chaque mot sélectionné dans la liste son type syntaxique comme l'illustre le tableau 6.1. Chaque mot et son type sont rangés dans un List Control représenté par la variable : *ListMotType*

Types syntaxiques	**Unités lexicales**
N	Théorie
(N\N)/T	de
T	nombres
T\T	complexes

Tableau 6.1 : La variable *ListMotType*.

Le premier élément de la première colonne représente le type syntaxique du premier élément de la deuxième colonne, le deuxième élément de la première colonne représente le type syntaxique du deuxième élément de la deuxième colonne, etc.

Dans notre implémentation, nous réservons la variable *m-type* à la saisie des types syntaxiques.

6.4.2 L'attribution des types

Nous avons construit à partir de la List Control *ListMotType* la variable *Terme* et la variable *arbre*. La variable *Terme* est le tableau 6.2 contenant des chaînes de caractères. Ce tableau contient les unités lexicales de l'énoncé à analyser. La valeur attribuée à l'indice du tableau nous permet d'accéder à chaque terme de l'énoncé. L'élément à l'indice 0 correspond au premier terme de l'énoncé, l'élément à l'indice 1 correspond au deuxième terme, etc....

Comme mentionné précédemment le types syntaxiques sont organisés en chaîne de caractères. La procédure Type_En_Arbre de conversion transforme la chaîne de caractères en arbre binaire. La variable "arbre" est le tableau 6.3 d'arbres binaires qui représentent les types syntaxiques des unités lexicales à analyser. Nous nous servons de la valeur de l'indice du tableau pour accéder à chaque type syntaxique.

Pour associer une unité lexicale à son type syntaxique, autrement dit pour forme le couple (unité lexicale, type syntaxique) nous associons un élément du tableau *Terme* à un élément du tableau *arbre* ayant le même indice.

Exemple :

Le tableau *Terme*

O	**1**	**2**
Théorie	**de**	**nombres**

Le tableau 6.2.

Le tableau *arbre*

0	1	2
N	N\N/N	N

Tableau 6.3.

L 'élément "arbre" à l'indice 0 est le type syntaxique de l'élément "Terme" à l'indice 0 [N : Théorie]
L 'élément "arbre" à l'indice 1 est le type syntaxique de l'élément "Terme" à l'indice 1 [N\N/N : de]
L 'élément "arbre" à l'indice 2 est le type syntaxique de l'élément "Terme" à l'indice 2 [N :nombres]

6.4.3 L'analyse syntaxique

Nous avons déjà mentionné dans les chapitres précédents que l'analyse que nous effectuons pour identifier les termes complexes est basée sur l'utilisation des règles combinatoires applicatives.

Pour commencer l'analyse nous entrons d'abord le nombre de termes que nous souhaitons avoir par termes complexes à la fin de l'analyse.

Exemple : si pour le terme candidat *Théorie de nombres complexes* nous souhaitons que notre terme complexe soit composé de trois termes nous aurons le traitement suivant :

Première étape :

[N: *Théorie*]-[(N\N)/N : *de*]-[N: *nombres*]
[N : *Théorie*]-[(N/N)\N *: (**C** de)*]-[N: *nombres*] (>C)
[(N/N : ((***C*** *de*) *Théorie*)]-[N: *nombres*] (<)
[N : (((***C*** *de*) *Théorie*) *nombres*)] (>)
(((***C*** *de*) *Théorie*) *nombres*)
((*de nombres*) *Théorie*) **C**

Deuxième étapes :

1. [(N\N)/N: *de*] – [N: *nombres*] – [N\N: *complexes*]
2. [N\N : (*de nombres*)] – [N\N: *complexes*] (>)
3. [N\N: (**B** *complexes* (*de nombres*))] (<**B**)

Pour cet exemple nous analysons trois termes de gauche à droite pour chaque étape. A la première étape nous commençons par le premier terme à gauche. A la deuxième étape nous commençons par le deuxième terme. Il n'y a pas de troisième étape parce que si nous commençons par le troisième terme en allant vers la droite nous constatons qu'il nous reste que deux termes alors que pour cet exemple il faut traiter trois termes à chaque étape.

Nous obtenons à la première étape une expression applicative typée, [N: (((**C** *de*) *Théorie*) nombres). Le type N (groupe nominal) est le type qu'il faut pour que le terme soit préservé. À la deuxième étape nous obtenons une expression de type N\N. Cette dernière sera rejetée car il n'est pas de la catégorie N.

Après avoir entrer le nombre des termes que nous souhaitions avoir, nous passons à l'analyse syntaxique des termes candidats. Pour construire l'analyseur syntaxique nous devons déclarer quelques variables importantes.

En appliquant des règles combinatoires à des unités concaténées nous construisons une structure applicative typée par un calcul sur les types syntaxiques. Nous devons donc déclarer les variables qui représentent :

- les unités concaténées,
- les types syntaxiques.

L'analyse syntaxique que nous effectuons est incrémentale ; cela fait que un résultat intermédiaire obtenu à une étape peut servir à construire un autre résultat à l'étape suivante. D'une façon concrète, l'application au niveau syntaxique d'une règle combinatoire génère une structure applicative typée. Or cette structure applicative est considérée comme étant une représentation applicative partielle de l'énoncé que nous voulons analyser. Partant de là cette représentation partielle constitue l'un des deux

éléments qui seront soumis à l'action d'une des règles combinatoires. Nous avons donc déclaré d'autres variables pour représenter :

- La structure applicative (représentant un résultat intermédiaire).
- Le type syntaxique de cette structure applicative.

Avant de présenter les variables citées ci-dessus telles qu'elles sont implémentées dans notre programme, nous présentons d'abord les variables : *nCount* et la variable *n*. La variable *nCount* représente le nombre d'unités lexicales que contient l'énoncé à analyser. *nCount* constitue alors la dimension des tableaux *Terme* et *arbre* que nous avons présentés dans la section précédente. On peut attribuer à la variable *n* (indice des tableaux *Terme* et *arbre*) une valeur qui varie de 0 à nCount-1. La valeur attribuée à *n* nous permet d'accéder aux différents éléments des tableaux *Terme* et *arbre*.

Nous présentons ci-dessous les variables importantes déclarées pour construire un analyseur :

• *Terme[n]* pour représenter l'élément courant. Cet élément est une unité lexicale.
• *arbre[n]* pour représenter le type syntaxique du constituant courant.
• *Terme[n+1]* pour représenter le constituant suivant. Ce constituant est une unité lexicale.
• *arbre[n+1]* pour représenter le type syntaxique du constituant suivant.

• *Resultat_interm* pour représenter le résultat intermédiaire de l'application d'une règle combinatoire. Ce résultat est une structure applicative.

• *arbreRes* pour représenter le type syntaxique du résultat intermédiaire.

Nous illustrons notre propos par un exemple.

A l'étape 1 de l'analyse du terme candidat *Théorie des nombres*, nous sommes dans la situation suivante :

Terme[n] = "*Théorie*"
Arbre [n] = N

Terme[n+1] = "*des*"
Arbre[n+1] = N\N/T

La deuxième étape nous donne la situation suivante:

Terme[n] = "*Théorie*"
Arbre [n] = N
Terme[n+1] = " (C *des)* "
Arbre[n+1] = N/T\N

La variable *n* est incrémentée.

L'utilisation de la règle d'application nous amène à la situation suivante :

Resultat_interm = " ((**C** *des*) *Théorie*) "
arbreRes = N/T
Terme[n+1] = "*nombres*"
arbre[n+1] = T

La variable *n* est incrémentée.

Enfin l'utilisation de la règle d'application nous donne :
Resultat_interm = " (((**C** *des*) *Théorie*) *nombres)* "
arbreRes = N
Terme[n+1] = ""
arbre[n+1] = NULL

Les variables "Resultat_interm" et "arbreRes" nous permettent de stocker momentanément les différents résultats partiels de l'analyse puis d'une façon définitive le résultat final. Ces six variables nous permettent de représenter les règles combinatoires applicatives de la manière suivante :

Pour la première étape

1. [arbre[n] : Terme[n]]- [arbre[n+1] : Terme[n+1]]

[arbreRes : Resultat_interm]

2. [arbre[n] : Terme[n]] ou [arbre[n+1] : Terme[n+1]]

[arbre[n] : Terme[n]] [arbre[n+1] : Terme[n+1]]

Pour les étapes suivantes

1. [arbreRes : Resultat_interm] - [arbre[n+1] : Terme[n+1]]

[arbreRes : Resultat_interm]

2. [arbreRes : Resultat_interm] ou [arbre[n+1] : Terme[n+1]]

[arbreRes : Resultat_interm] [arbre[n+1] : Terme[n+1]]

Les règles en 1. sont réservées au cas de composition et d'application.
Les règles en 2. sont réservées au cas de permutation.

Avec un test appliqué aux valeurs des variables "arbre[n] ", "arbre[n+ 1]" et "arbreRes" nous arrivons à savoir quelle règle combinatoire déclencher parmi les règles d'application fonctionnelle et de composition fonctionnelle.

6.4.3.1 Règles d'application et de composition fonctionnelle.

Les règles d'application

La règle Y – X\Y==> X

Si arbre[n+1]-> racine-> element = = '\\'
et Si arbre[n]->racine = = arbre[n+1]->racine->filsd alors
Appeler la fonction "Application_arr" qui implémente la règleY-X\Y==>X

Pour l'application de la règle Y-X\Y==>X, il faut qu'arbre[n]->racine = =Y et arbre[n+1]->racine = = X\Y.

Le test arbre[n+1]-> racine-> element = = '\' est fait pour savoir si l'élément *Terme[n+1]* compose par l'application à gauche. arbre[n]->racine = = arbre[n+1]-> racine->filsd vérifie que *Terme[n]* a le type de l'opérande de *Terme[n+1]*. Le type du résultat de l'application de *Terme[n+1]* à son opérande est donné par le sous arbre gauche de l'arbre[n+1]. C'est ce type que nous récupérons puis affectons à la variable *arbreRes*. La construction de la structure applicative est simple : l'opérateur est placé à gauche de l'opérande. Ce qui se traduit par :

Resultat_interm = (Terme[n+ 1] + Terme[n]).

La règle X/Y-Y==>X

Si arbre[n]->racine->element = = '/'
et Si arbre[n]->racine->filsd = = arbre[n+1]->racine alors
Appeler la fonction "Application_av" qui implémente la règle X/Y-Y==>X

La règle X/Y-Y==>X est déclenchée si le type de *Terme[n]* est un opérateur s'appliquant à un opérande placé à droite. Cela est vérifié par le test arbre[n]->racine->élément = = '/'. Autrement dit arbre[n] = = X/Y et arbre[n+1] = = Y. Le deuxième test arbre[n]->racine-> filsd = = arbre[n+1]->racine vérifie si l'élément suivant a le type de l'opérande de l'élément courant.

La fonction afficher (arbre[n]->racine-> filsg) nous permet de récupérer la valeur du sous arbre gauche. Cette dernière est le type de l'application de *Terme[n]* à son opérande *Terme[n+1]*. Le type obtenu est affecté à la variable *arbreRes*. L'instruction Resultat_interm = (Terme[n] +Terme[n+1]) construit l'expression applicative.

Présentons deux exemples :

Exemple1 : la règle Y-X\Y==>X

[N : *Base*] – [N\N : *fondamentale*]

arbre [n] = N ; arbre[n+ 1] = N \ N;
Terme[n]= "*Base*"; Terme[n+1] = " *fondamentale* ";
arbreRes->racine = arbre[n+1]->racine->filsg = N;

Resultat_interm = (Terme[n+ 1] +Terme[n]) = " (*fondamentale Base*) " ;

Exemple 2 : la règle X/Y-Y==>X

[N/N : *La*] – [N : *base*]

arbre[n] = N/N ; arbre[n+ 1] = N ;
Terme[n]= "*la*"; Terme[n+1] = " *base* ";

arbreRes->racine = arbre[n]->racine->filsg = N;

Resultat_interm = (Terme[n] +Terme[n+ 1]) = " (*La base*) " ;

Les règles de composition fonctionnelle

La règle X/Y-Y/Z==>X/Z

```
Si arbre[n]->racine->element = = '/'
  et Si arbre[n+1]->racine->element = = '/'
    et Si arbre[n]->racine->filsd = =arbre[n+ 1]->racine->filsg alors
        Appeler la fonction "Comp_FonctAvB" qui implémente la règle
        X/Y-Y/Z==>X/Z
```

L'application de la règle X/Y-Y/Z==>X/Z, nécessite la situation où arbre[n] = X /Y et arbre[n+ 1] = Y/Z. Les tests arbre[n]->racine->element = = '/' et

arbre[n+1] ->racine->element = = '/' vérifient que *Terme[n]* et *Terme[n+1]* composent par l'application à droite. Le test arbre[n]->racine->filsd = = arbre[n+1]>racine->filsg

permet de savoir si l'application de *Terme[n+ 1]* à son opérande construit une expression ayant le type de l'opérande de *Terme[n]*.

Les règles de composition mettent en rapport deux opérateurs. Le résultat de l'application d'un des deux opérateurs à son opérande donne l'argument du deuxième opérateur. L'opérateur complexe construit par la composition héritera d'un type opérateur construit à partir des types des opérateurs initiaux. Dans le cas de la règle X/Y-Y/Z==>X/Z cet opérateur complexe prend un opérande qui est placé à sa droite.
Les affectations suivantes permettent de construire le type de l'opérateur complexe :

```
arbreRes->racine->element = '/' ;
arbreRes->racine->filsg  = arbre[n]->racine->filsg ;
arbreRes->racine->filsd  = arbre [n+1]->racine->filsd ;
```

L'expression applicative obtenue contient le combinateur **B**. Nous la construisons par l'instruction : *Resultat_interm* = (‘**B**’ +Terme[n]+ Terme[n+1]).

La règle X/Y-Y\Z==>X\Z

```
Si arbre[n]->racine->element = = '/'
   et Si arbre[n+ 1]->racine->element = = '\\'
      et Si arbre[n]->racine->filsd = =arbre[n+ 1]->racine->filsg  alors
           Appeler la fonction "Comp_FonctAvBx" qui implémente la règle
           X/Y-Y\Z==>X\Z
```

L'application de la règle X/Y-Y\Z==>X\Z est possible si arbre[n] = = X/Y et arbre[n+1] = = Y\Z. Il faut aussi vérifier que *Terme[n]* compose par l'application à droite et *Terme[n+ 1]* compose par l'application à gauche, avec les tests arbre[n]->racine-> element = = '/' et arbre[n+ 1]-> racine->element = = '\\'. Le dernier test est de prouver par le test arbre[n]->racine-> filsd = = arbre[n+1]->racine->filsg si l'application de *Terme[n+ 1]* à son argument construit une expression ayant le type de l'argument de *Terme[n]*.

Cette règle X/Y-Y\Z ==>X\Z de composition présente les mêmes caractéristiques que la règle précédente. Cependant, l'opérateur complexe obtenu lorsque la première est déclenchée prendra un opérande placé à sa gauche. Nous construisons le type de l'opérateur complexe *arbreRes* avec les instructions suivantes :

arbreRes-> racine->element = '\\' ;

arbreRes->racine->filsg = arbre[n] ->racine->filsg ;

arbreRes->racine->filsd = arbre[n +1] ->racine->filsd ;

L'instruction *Resultat_interm* = (**'B'** + Terme[n]+Terme[N+1]) construit l'expression applicative.

La règle Y\Z-X\Y==>X\Z

Si arbre[n]->racine->element = = '\\'
et Si arbre[n+ 1] ->racine->element = = '\\'
et Si arbre[n] ->racine->filsg = =arbre[n+ 1] ->racine->filsd alors
Appeler la fonction "Comp_FonctArrB" qui implémente la règle
Y\Z-X\Y==>X\Z

Nous appliquons la règle Y\Z-X\Y==>X\Z quand arbre[n] = = Y\Z et arbre[n+1] = = X\Y. Ainsi, les tests arbre[n] ->racine->element = = '\\' et arbre[n+1] = = '\\ ' vérifient que *Terme[n]* et *Terme[n+1]* composent par l'application à gauche. Le test arbre[n] ->racine->filsg = = arbre[n+ 1] ->racine->filsd indique que le type de l'application de *Terme[n]* à son argument doit produire une expression ayant un type identique à celui de l'argument de *Terme[n+1]*.
L'opérateur complexe obtenu par composition dans le cas de la règle Y\Z-X\Y==>X\Z prendra un opérande qui sera placé à sa gauche. Ce qui s'exprime par l'instruction arbreRes->racine->element = '\\';

Avec les instructions :

arbreRes->racine->filsg = arbre[n+1] ->racine->filsg ;

arbreRes->racine->filsd=arbre[n] ->racine->filsd ; nous récupérons respectivement :
- le type résultat de l'application de l'opérateur complexe à son opérande qui est celui de l'application du deuxième opérateur initial à son argument (c'est à dire le type X),
- et le type de l'argument de l'opérateur complexe qui est celui de l'argument du premier opérateur initial (c'est à dire le type Z).

L'expression applicative est obtenue par l'instruction : Resultat_interm = (‘**B’**+ terme[n+1] + terme[n])

La règle Y/Z-X\Y==>X/Z

```
Si  arbre [n] ->racine->element = = '/'
  et Si arbre[n+1] ->racine->element = = '\\'
    et Si arbre[n] ->racine->filsg = =arbre[n+1] ->racine->filsd alors
        Appeler la fonction "Comp_FonctArrBx" qui implémente la règle
        Y/Z-X\Y==>X/Z
```

Pour appliquer la règle Y/Z-X\Y==>X/Z il faut qu'arbre[n] = = Y/Z et arbre[n+ 1] = = X\Y. Autrement dit *Terme[n]* compose par l'application à droite et *Terme[n+1]* compose par l'application à gauche. Ceci justifie les deux tests arbre [n] ->racine-> element = = '/' et arbre[n+1] ->racine->element = = '\\.'. Le test arbre[n] -> racine->filsg = =arbre[n+1] ->racine->filsd indique que le type de l'application de *Terme[n]* à son argument doit produire une expression ayant un type identique à celui de l'argument de *Terme[n+1].*

Comp_FonctArrBx, s'exprime avec :

arbreRes->racine->element = '/' ;

arbreRes->racine->filsg = arbre[n+1] ->racine->filsg ;

arbreRes->racine->filsd = arbre[n] ->racine->filsd ;

Resultat_interm =('**B**' + terme[n+1]+ terme[n]) ;

Cette règle présente les même caractéristiques que la règle précédente. Cependant pour la règle précédente, l'opérateur complexe construit a un opérande placé à sa gauche alors que pour ce cas-ci l'opérateur prend un opérande placé à sa droite.

Nous présentons deux exemples :

Exemple1 : la règle X/Y-Y/Z==>X/Z

[N/N: *la*] – [N/N : *fondamentale*]
arbre[n] = N/N ; arbre[n+1]= N/N;
Terme[n]= "*la*"; Terme[n+1] = " *fondamentale* ";
arbreRes->racine->element = '/' ;
arbreRes->racine->filsg = N ;
arbreRes->racine->filsd = N ;
arbreRes = N/N;
Resultat_interm = " (***B*** *la fondamentale*) " ;

Exemple2 : la règle Y\Z-X\Y==>X\Z

[N\N: *relationnelles*] – [N\N : *hiérarchiques*]
arbre[n] = N\N ; arbre[n+1]= N\N;
Terme[n]= " *relationnelles* "; Terme[n+1] = " *hiérarchiques* ";
arbreRes->racine->element = '\\' ;
arbreRes->racine->filsg = N ;
arbreRes->racine->filsd = N ;
arbreRes = N\N;
Resultat_interm= " (***B*** *hiérarchiques relationnelles*) " ;

Par ailleurs si aucun des tests présentés n'est valide, nous orientons notre analyse vers la permutation.

6.4.3.2 Les règles de permutation

Ces règles sont caractérisées par la permutation des positions des opérandes et par l'introduction du combinateur C dans l'expression applicative qui est construite lorsqu'une règle de permutation est appliquée. Avant de déclencher les règles de permutation nous posons la question suivante :

Si nous permutons les opérandes d'un opérateur cela va-t-il nous permettre d'utiliser les règles d'application ou de composition fonctionnelle à la prochaine étape de l'analyse ?

Nous présenterons ces règles de permutation en deux parties : la première concernera des tests appliqués aux variables *arbre[n]* et *arbre[n+1].* Ces tests nous renseignent sur les situations de déclenchement des règles de permutation et sur les règles (application ou composition) à utiliser directement après cette permutation. La deuxième concernera des modifications apportées aux valeurs des variables *arbre[n], arbre[n+1], Terme[n], terme[n+1], arbreRes,* et *Resultat_interm.*

La règle (X/Y)\Z==> (X\Z)/Y

Nous avons trois situations où cette règle peut être déclenchée :

```
1.Si arbre[n] = = X/Y\Z  et  arbre[n+ 1] = = Y
   et Si arbre[n]->racine->filsg->element = = '/'
     et Si  arbre[n] ->racine->filsg->filsd= = arbre[n+1] ->racine alors
        Appeler la fonction "Permutation_ArrC" qui implémente la règle
        (X/Y)\Z==> (X\Z)/Y.
```

L'appel de cette fonction nous permet de construire le type syntaxique X\Z/Y et de produire l'expression applicative (C Terme[n]). Ceci nous mène à une situation où : arbre[n] = = X\Z/Y, arbre[n+1] = = Y et Terme[n] = = (C Terme[n]). Nous pouvons alors appliquer la règle d'application à la prochaine étape d'analyse. Le résultat obtenu

X\Z est affecté à la variable *arbreRes*. L'instruction, *Resultat_interm* = (('C'+Terme[n]) +Terme[n+ 1]) construit l'expression applicative.

2. Si arbre[n] = = X/Y\Z et arbre[n+ 1] = = Y\Z
   et Si arbre[n]->racine->filsg->element = = '/'
   et Si arbre[n+1]->racine->elemen t= = '\\'
   et Si arbre[n] ->racine->filsg->filsd = = arbre[n+1] ->racine->filsg alors
   Appeler la fonction "Permutation_ArrC" qui implémente la règle (X/Y)\Z==> (X\Z)/Y.

Le type syntaxique X\Z/Y retourné par la fonction "Permutation_ArrC" est affecté à arbre[n]. Avec arbre[n] = = X\Z/Y et arbre [n+ 1] = = Y\Z nous utilisons la règle de composition (X/Y-Y\Z==>X\Z) comme présenté au paragraphe précédent. L'opérateur complexe X\Z\Z obtenu est affecté à la variable *arbreRes*. L'expression applicative ('B'+('C'+ Terme[n])+Terme[n+1]) construite est affectée à la variable *Resultat_interm*.

3. Si arbre[n] = = X/Y\Z et arbre[n+ 1] = = Y/Z
   et Si arbre[n]->racine->filsg->element = = '/'
   et Si arbre[n+1]->racine->element = = '/'
   et Si arbre[n] ->racine->filsg->filsd = = arbre[n+1] ->racine->filsg alors
   Appeler la fonction "Permutation_ArrC" qui implémente la règle (X/Y)\Z==> (X\Z)/Y.

Nous affectons le type syntaxique X\Z/Y retourné par la fonction "Permutation_ArrC" à arbre[n]. Nous obtenons une situation qui nous permet d'utiliser la règle de composition (X/Y-Y/Z==>X/Z) car arbre[n] = = X\Z/Y et arbre[n+1] = = Y/Z. Le résultat de l'application de cette règle est affecté à la variable *arbreRes*. Ainsi arbreRes = = X\Z/Z et l'expression applicative produite est ('B' +('C'+ Terme[n])+Terme[n+1]).

La règle (X/Y)/Z==>(X/Z)/Y

Nous avons deux situations où cette règle peut être déclenchée :

1. Si arbre[n] = = X/Y/Z et arbre[n+ 1] = = Y
   et Si arbre[n]->racine->element = = '/'
   et Si arbre[n] ->racine->filsg->filsd = = arbre[n+1] ->racine alors
   Appeler la fonction "Permutation_AvCx" qui implémente la règle (X/Y)/Z==> (X/Z)/Y.

Cette fonction retourne le type syntaxique que nous affectons à arbre[n]. Nous nous retrouvons avec arbre[n]= = X/Z/Y et arbre[n+1] = = Y, ce qui nous permet d'appliquer une règle d'application (X/Y-Y==>X) à la prochaine étape de l'analyse. Nous obtenons le type X/Z et une expression applicative (('C' + Terme[n])+ Terme[n+1]). X/Z est affecté à la variable *arbreRes* et l'expression applicative est affectée à la variable *Resultat_interm.*

2. Si arbre[n] = = X/Y/Z et arbre[n+ 1] = = Y/Z
   et Si arbre[n]->racine-->element = = '/'
   et Si arbre[n+1]->racine->element= = '/'
   et Si arbre[n] ->racine->filsg->filsd = = arbre[n+1] ->racine->filsg alors
   Appeler la fonction "Permutation_AvCx" qui implémente la règle (X/Y)/Z==> (X/Z)/Y.

Le type syntaxique retourné par la fonction "Permutation_AvCx" est affecté à la variable arbre[n]. Nous avons ainsi arbre[n] = = X/Z/Y et arbre[n+1] = = Y/Z, ce qui nous permet d'utiliser la règle de composition (X/Y-Y/Z ==>X/Z) à l'étape suivante de l'analyse. X/Z/Y-Y/Z==>X/Z/Z. Nous affectons X/Z/Z à *arbreRes*. L'expression applicative ('B'+('C'+ Terme[n]) + Terme[n+1]) produite est affectée à la variable *Resultat_interm.*

La règle (X\Y)/Z==> (X/Z)\Y

Trois situations où cette règle peut être déclenchée :

1. Si arbre[n] = =Y et arbre[n+ 1] = = X\Y/Z
   et Si arbre[n+1]->racine->filsg->element = = '\\'
   et Si arbre[n] ->racine= = arbre[n+ 1] ->racine->filsg->filsd alors
   Appeler la fonction "Permutation_AvC" qui implémente la règle
   (X\Y)/Z==> (X/Z)\Y.

L'appel de cette fonction nous permet de construire le type syntaxique (X/Z)\Y et de produire l'expression applicative (C Terme[n+1]). Ceci nous mène à une situation où : arbre[n] = = Y et arbre[n+1] = = X/Z\Y Nous pouvons alors appliquer à l'étape suivante de l'analyse la règle d'application(Y- X\Y==> X). Le résultat obtenu X/Z est affecté à la variable *arbreRes*. L'instruction *Resultat_interm* = (('C'+Terme[n+ 1])+Terme[n]) construit l'expression applicative.

2. Si arbre[n] = = Y/Z et arbre[n+ 1] = = X\Y/Z
   et Si arbre[n+1]->racine->filsg->element = = '\\'
   et Si arbre[n]->racine->élément = = '/'
   et Si arbre[n] ->racine->filsg = = arbre[n+1] ->racine->filsg->filsd alors
   Appeler la fonction "Permutation_AvC" qui implémente la règle
   (X\Y)/Z==> (X/Z)\Y.

Le type syntaxique X/Z\Y retourné par la fonction "Permutation_AvC" est affecté à arbre[n+ 1]. Avec arbre[n] = = Y/Z et arbre [n+ 1] = = X/Z\Y nous utilisons la règle de composition (Y/Z-X\Y==>X/Z) comme présenté au paragraphe précédent. L'opérateur complexe X/Z/Z obtenu est affecté à la variable *arbreRes*. L'expression applicative ('B'+('C'+ Terme[n+1])+Terme[n]) construite est affectée à la variable *Resultat_interm*.

3. Si arbre[n] = = Y\Z et arbre[n+ 1] = = X\Y/Z
   et Si arbre[n]->racine->element = = '\\'
      et Si arbre[n+1]->racine->filsg->element = = '\\'
         et Si arbre[n] ->racine->filsg = = arbre[n+1] ->racine->filsg->filsd alors
            Appeler la fonction "Permutation_AvC" qui implémente la règle (X\Y)/Z==> (X/Z)\Y.

Nous affectons le type syntaxique X/Z\Y retourné par la fonction "Permutation_AvC" à arbre[n+1]. Nous obtenons une situation qui nous permet d'utiliser la règle de composition (Y\Z-X\Y==>X\Z) Car arbre[n] = = Y\Z et arbre[n+1] = = X/Z\Y. Le résultat de l'application de cette règle (X/Z\Z) est affecté à la variable *arbreRes* et l'expression applicative ('B'+('C'+Terme[n+1])+Terme[n]) produite est affectée à la variable *Resultat_interm.*

La règle (X\Y)\Z==>(X\Z)\Y

Deux situations où cette règle peut être déclenchée :

1. Si arbre[n] = =Y et arbre[n+ 1] = = X\Y\Z
   et Si arbre[n+1]->racine->element = = '\\'
      et Si arbre[n] ->racine= = arbre[n+ 1] ->racine->filsg->filsd alors
         Appeler la fonction "Permutation_ArrCx" qui implémente la règle (X\Y)\Z==> (X\Z)\Y.

Cette fonction retourne le type syntaxique que nous affectons à arbre[n+1]. Nous nous retrouvons avec arbre[n] = = Y et arbre[n+1] = = X\Z\Y, ce qui nous permet d'appliquer une règle d'application (Y-X\Y==>X) à la prochaine étape de l'analyse. Nous obtenons le type X\Z et une expression applicative (('C'+ Terme[n+1])+ Terme[n]). X\Z est affecté à la variable *arbreRes* et l'expression applicative est affectée à la variable *Resultat_interm.*

2. Si arbre[n] = = Y\Z et arbre[n+ 1] = = X\Y\Z
et Si arbre[n]->racine->element = = '\\'
et Si arbre[n+1]->racine->element= = '\\'
et Si arbre[n] ->racine->filsg = = arbre[n+1] ->racine->filsg->filsd alors
Appeler la fonction "Permutation_ArrCx" qui implémente la règle
(X\Y)\Z==> (X\Z)\Y.

Le type syntaxique retourné par la fonction "Permutation_ArrCx" est affecté à la variable arbre[n+1]. Nous avons ainsi arbre[n] = = Y\Z et arbre[n+1] = = X\Z\Y, ce qui nous permet d'utiliser la règle de composition (Y\Z- X\Y ==>X\Z) à l'étape suivante de l'analyse. Y\Z-X\Z\Y==>X\Z\Z. Nous affectons X\Z\Z à *arbreRes.* L'expression applicative ('B'+('C'+ Terme[n+ 1]) + Terme[n]) produite est affectée à la variable *Resultat_interm.*

Nous présentons deux exemples qui illustrent l'utilisation de la règle
(X\Y)/Z==> (X/Z)\Y

Exemple1

[N: *Base*] – [(N\N)/T : *de*]
arbre[n] = N ; arbre[n+1]= N\N/T;
Terme[n] = "*Base*"; Terme[n+1] = "*de*" ;
Étape : 1
Nous appliquons la règle de permutation (X\Y)/Z==> (X/Z)\Y
Arbre[n] = N ; arbre[n+1] = N/T\N
Terme[n] = "*Base*"; Terme[n+1] = " (*C de*) " ;
Étape :2
L'utilisation de la règle d'application (Y-X\Y==>X) nous permet d'obtenir
ArbreRes = N/T;
Resultat_interm= " ((*C de*) *Base*) " ;

Exemple 2

[N\N : *relationnelle*] – [(N\N)/T : *de*]

arbre[n] = N\N ; arbre[n+1]= N\N/T;

Terme[n] = "*relationnelle*"; Terme[n+1] = "*de*" ;

Étape : 1

Nous appliquons la règle de permutation (X\Y)/Z==> (X/Z)\Y

Arbre[n] = N\N ; arbre[n+1] = N/T\N

Terme[n] = "*relationnelle*"; Terme[n+1] = " (*C de*) " ;

Étape :2

Nous utilisons la règle de composition (Y\Z-X\Y==>X\Z) pour d'obtenir

ArbreRes = N/T\N;

Resultat_interm= " (*B* (*C de*) *relationnelle*) " ;

Si aucune des règles combinatoires applicatives ne peut être appliquée, nous orientons notre analyse vers la réorganisation structurelle.

6.4.3.3 La réorganisation structurelle

Deux étapes successives caractérisent la réorganisation structurelle[18] :

- La réorganisation du faux constituant.
- La décomposition.

D'un point de vue pratique la réorganisation structurelle présente un certain nombre de contraintes. Nous avons introduit des nouvelles variables qui vont nous aider à limiter le poids de ces contraintes.

Au départ nous avons un tableau de chaînes de caractères *Terme* où sont répertoriées les unités lexicales formant l'énoncé et un tableau d'arbre binaire *arbre* qui contient les types syntaxiques de ces unités lexicales. La variable *n* est un entier qui représente l'indice de ces deux tableaux. Nous avons déclaré trois nouvelles variables. Un tableau de chaînes de caractères *Tab3*, un tableau d'arbre binaire *arbre2*, un entier *k*. On retrouve dans *Tab3* des expressions applicatives construites pendant l'analyse. *Arbre2* renferme

[18] Lire le paragraphe 5.2.1 du chapitre précèdent

les types syntaxiques de ces structures applicatives. La variable k est l'indice des tableaux *Tab3* et *arbre2*. Elle varie de 0 au nombre d'expressions applicatives moins un.

Nous présentons ces variables par un exemple :

Prenons le terme candidat : *Base de données*

[N : *Base*]-[(N\N)/T : *de*]-[T :*données*]

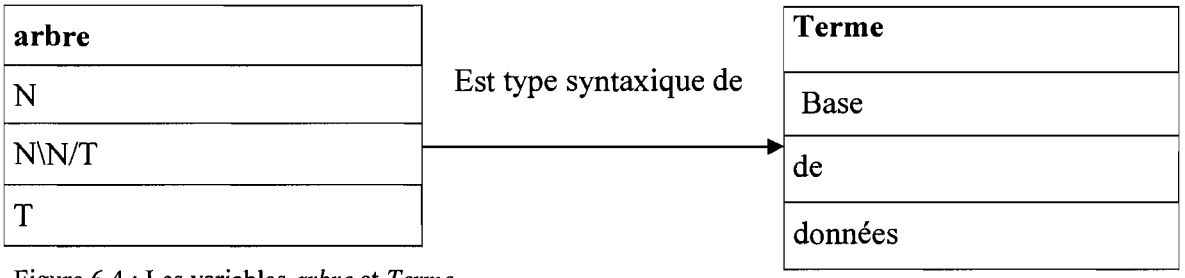


Figure 6.4 : Les variables *arbre* et *Terme.*

La variable n varie de 0 à 2

[N : *Base*]-[(N\N)/T : de]-[T :*données*]
[(N/T) : ((**C** *de*) *Base*)] – [T: *données*] (<)
[N: (((**C** *de*) *Base*) *données*)] (>)

Nous avons les expressions applicatives typées suivants :
[(N/T) : ((**C** *de*) *Base*)]
[N: (((**C** *de*) *Base*) *données*)]

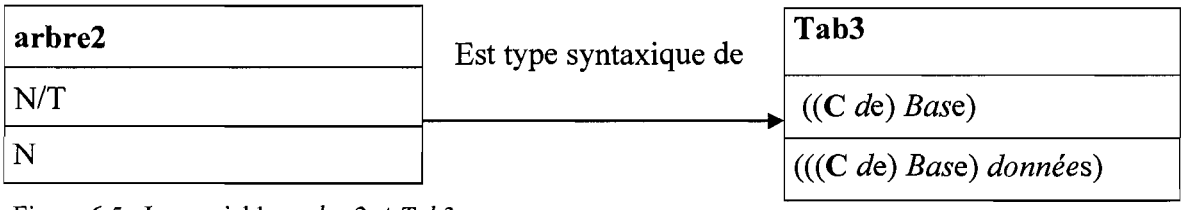
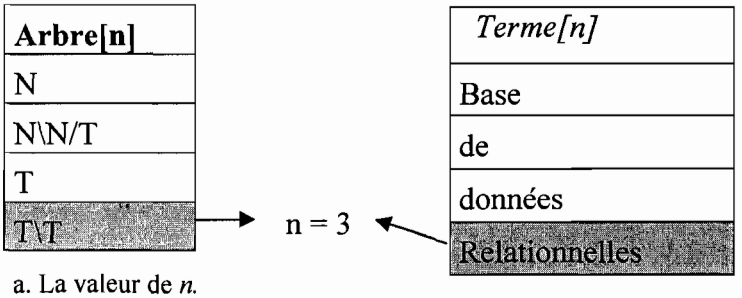


Figure 6.5 : Les variables *arbre2* et *Tab3.*

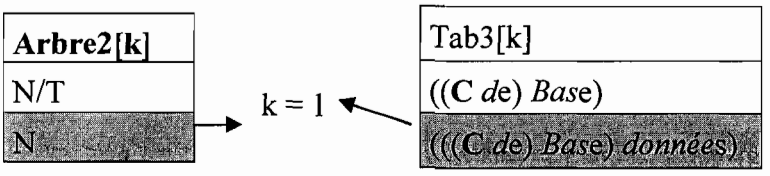
La variable k varie de 0 à 1

La réorganisation du constituant isole à chaque fois deux sous-catégories. La valeur attribuée à *k* va nous permettre d'accéder à l'une des sous-catégories c.à.d l'expression applicative et son type syntaxique. Pour trouver l'autre sous-catégorie (l'opérande) nous allons recourir à la valeur de *n*. Nous pouvons alors tester si l'un ou l'autre des sous-catégories se combine avec le modifieur arrière. Si aucune de deux sous-catégories ne se combine à gauche avec le modifieur arrière, le terme candidat à analyser est considéré comme syntaxiquement incorrect. Si ce n'est pas le cas, nous appliquons une règle combinatoire applicative. Le résultat obtenu est combiné avec l'autre sous-catégorie. Nous pouvons ainsi reprendre l'analyse par l'application d'une règle combinatoire à l'élément suivant et l'élément à droite obtenue après la réorganisation structurelle.

Exemple : Dans le cas du terme candidat *Base de données relationnelles* l'analyseur produit le constituant [N: (((**C** *de*) *Base*) *données*)]. Ce dernier ne se combine pas avec [T\T :*relationnelles*]. A ce niveau n = 3 et k = 1. Nous le représentons par le schéma présenté à la figure 6.6 :



a. La valeur de *n*.



b. La valeur de *k*.

Figure 6.6 : La valeur de *n* et la valeur de *k* quand l'analyse « bloque »

Le constituant construit : [N: (((**C** *de*) *Base*) *données*)]

Les deux sous catégories sont : [N/T : ((***C*** *de*) *Base*)] ; [T : *données*]. Avec arbre2[k-1] et Tab3[k-1] nous récupérons la première sous catégorie [N/T : ((***C*** *de*) *Base*)] et avec

arbre[n-1] et Terme[n-1] nous récupérons le deuxième sous-catégorie [T : *données*]. Nous pouvons maintenant tester si l'une ou l'autre de ces sous-catégories se combine avec [T\T : *relationnelles*]. [T :*données*] se combine à gauche avec [T\T : *relationnelles*]. Le résultat obtenu [T : (*relationnelles données*)] est combiné avec l'autre sous-catégorie issue de la réorganisation [N/T : ((**C** de) Base)]. Nous obtenons [N: (((**C** *de*) *Base*) (*relationnelles données*))].

6.4.4 La construction de la forme normale

Nous construisons la forme normale en éliminant les combinateurs d'une structure applicative. Rappelons que c'est la classe CCombinateur qui définit les fonctions implémentant la réduction des combinateurs. Lorsque la fonction Reduire_comb est appelée dans le programme principal, elle prend en entrée une structure applicative avec combinateurs, générée par l'analyseur syntaxique. Cette fonction applique la β-réduction adéquate au premier combinateur détecté. La β-rédution est appliquée directement à la variable qui contient la structure applicative.
Les instructions suivantes sont itérées jusqu'à ce qu'il ne reste plus de combinateurs dans la structure applicative :

- appel de la fonction Comb_test qui détecte le combinateur **B** ou **C,**
- appel de la fonction Reduire_B si le combinateur détecté est **B,**
- appel de la fonction Reduire _C si le combinateur détecté est **C.**

La réduction du combinateur **B** : ((**B** x y) z) ==> (x (y z))

Nous construisons le membre de droite (x (y z)) à partir du membre de gauche ((**B** x y) z) de la manière suivante :

1. Nous éliminons la lettre **B,** la parenthèse ouvrante qui la précède et la parenthèse fermante correspondante. Nous obtenons (x y z).
2. Nous mettons y z entre parenthèses. Ce qui nous donne (x (y z)).

La réduction du combinateur **C** : (((**C** x) y) z) ==> ((x z) y)

Nous construisons le membre de droite ((x z) y) à partir du membre de gauche (((**C** x) y) z) de la manière suivante :

1. Nous éliminons la lettre **C**, la parenthèse ouvrante qui la précède et la parenthèse fermante correspondante. Ce qui nous donne ((x y) z).
2. Nous permutons y et z pour obtenir ((x z) y).

6.4.5 Affichage des termes complexes

A l'issue de l'analyse syntaxique des termes candidats, nous obtenons des structures applicatives des termes validés (les termes complexes). Certaines de ces structures applicatives peuvent être sous la forme normale, et d'autres peuvent contenir des combinateurs. Ces dernières structures applicatives subissent la réduction des combinateurs pour construire les formes normales. Nous affectons ces formes normales à la variable *m_LesTermes*. Celle-ci est une variable Cstring qui est associée à une boîte de dialogue. Ce qui permet d'afficher ces termes complexes (formes normales) dans une boîte de dialogue.

Les termes complexes auxquels nous aboutissons à la fin du traitement des termes candidats sont aussi affichés sous forme de document XML ou base de données XML. Avant de donner les détails de cet affichage, nous exposons quelques concepts de base de XML qui sont utilisés dans notre travail.

6.4.5.1 Quelques concepts de base XML

XML est un métalangage permettant de représenter des documents sous forme d'éléments balisés imbriqués. Nous en décrivons les concepts de base dans cette section.

Un document XML, est constitué de deux parties principales : le prologue et l'élément document; ce dernier est également connu sous la désignation d'élément racine.

Exemple de document XML.

```
<?xml version="1.0" enconding="UTF-8" standalone="no"?>

<LESMOTS>
        <ENREGISTREMENT>
                <NUMERO>1</NUMERO>
                <EXPRESSION>Base de la th&#233;orie</EXPRESSION>
        </ENREGISTREMENT>

        <ENREGISTREMENT>
                <NUMERO>2</NUMERO>
                <EXPRESSION> la th&#233;orie des  nombres</EXPRESSION>
        </ENREGISTREMENT>

</LESMOTS>
```

Le prologue

Le prologue de XML est une instruction de traitement servant à identifier la version du langage XML et doit se trouver obligatoirement tout en haut du document XML.

Cette instruction est utilisée également pour déclarer le jeu de caractères d'encodage du document XML. Enfin, elle permet de spécifier si le document est autonome (standalone="yes") ou s'il dépend, pour son fonctionnement, d'autres fichiers ou de toutes autres ressources externes (standalone="no").

Exemple :

< ?xml version="1.0" encoding="UTF-8" standalone="no"?>
L'instruction XML spécifie la version 1.0 de XML avec un encodage *Unicode compressé* et requiert des documents externes.

L'élément document

Les éléments indiquent la structure logique du document et contiennent l'information de celui-ci.

Un élément type comprend un marqueur d'ouverture, le contenu de l'élément et le marqueur de fermeture. Le contenu de l'élément peut être constitué de données caractères, d'autres éléments (imbriqués) ou d'une combinaison des deux.

Dans notre exemple de document, l'élément document est LESMOTS. Son marqueur d'ouverture est <LESMOTS>, son marqueur de fermeture est </LESMOTS>
et son contenu est constitué de deux éléments ENREGISTREMENT.

<ENREGISTREMENT>
    <NUMERO>1</NUMERO>
    <EXPRESSION>Base de la théorie</EXPRESSION>
</ENREGISTREMENT>

Chaque élément ENREGISTREMENT, à son tour, contient deux éléments emboîtés NUMERO et EXPRESSION.

Chacun des éléments insérés dans l'élément ENREGISTREMENT, comme l'élément EXPRESSION, ne contient que des données caractères (figure 6.7).

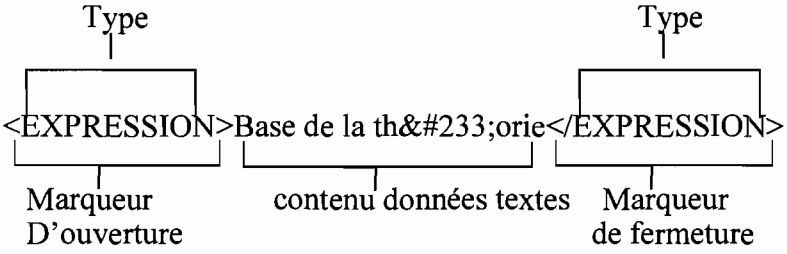


Figure 6.7 : L'élément EXPRESSION.

Affichage du document XML

Un document XML peut être ouvert directement dans Internet Explorer. Si le document XML ne contient pas de lien à une feuille de styles, Internet Explorer se contente d'afficher le code source du document, y compris son marquage et les données caractères. Cependant, si le document XML contient un lien vers une feuille de styles, Internet Explorer n'affichera que les données caractères des éléments du document, et il formatera ces données selon les règles spécifiées dans la feuille de styles. On peut utiliser soit une feuille de styles en cascade (une feuille de styles CSS- la même feuille de styles utilisée pour les pages HTML) ou une feuille de styles XSL (Extensible Stylesheet Language), c'est-à-dire, un type de feuille de styles plus puissant employant la syntaxe XML.

Affichage de document XML à l'aide de feuilles de styles XSL[19]

Pour l'affichage XML, une feuille de styles XSL est considérablement plus puissante et souple qu'une feuille de styles CSS. En effet, cette dernière permet simplement de définir le formatage de chaque élément XML, tandis qu'une feuille de styles XSL offre un contrôle complet du rendu. XSL permet de sélectionner précisément les données XML qu'on veut afficher, pour les présenter dans l'ordre ou l'arrangement souhaité et pour modifier ou ajouter de l'information. XSL donne un accès à tous les composants XML (tels que les éléments, les attributs, les commentaires et les instructions de traitement), permet de trier et de filtrer aisément les données XML, autorise à inclure des scripts dans la feuille de styles et fournit un jeu de méthodes utiles pour manipuler l'information.

Le langage XSL, par une série de règles de transformation, remplace les éléments XML et leurs attributs en balisage HTML (HyperText Markup Language) ou en d'autres marqueurs XML.

---

[19] A part la feuille de style "XslGCCA.xsl", cette section est prise dans YOUNG, M.J., XML étape par étape, 201, 337-344.

Utilisation d'une feuille de styles XSL :

Les principes de base

L'utilisation d'une feuille de styles XSL pour afficher un document XML comporte deux étapes :

1. *La création de la feuille de styles XSL*. XSL est une application XML. Autrement dit, une feuille de styles XSL est un document XML bien formé (qui satisfait un minimum de critères de conformité XML) qui se conforme aux règles de XSL. Comme tout document XML, une feuille de styles XSL est en mode texte. On peut la créer à l'aide d'un éditeur de texte.
2. *La liaison de la feuille de styles XSL au document XML*. Bien que XML soit utilisé pour créer à la fois le document XML et la feuille de styles XSL, le document et la feuille de styles doivent se trouver dans des fichiers séparés : un fichier pour le document XML (avec l'extension.xml) et un fichier pour la feuille de styles XSL (avec l'extension.xsl). On relie la feuille de styles XSL au document XML en incorporant, dans ce document, une instruction de traitement xml-stylesheet, qui a la forme générale suivante :

   <?xml-stylesheet type="text/xsl" href="XSLFichierChemin"?>

   Ici XSLFichierChemin est une URL entre guillemets, indiquant l'emplacement du fichier de la feuille de styles.

   On ajoute normalement l'instruction de traitement xml-stylesheet dans le prologue du document XML, à la suite de la déclaration XML.

Si on lie une feuille de styles XSL à un document XML, on peut l'ouvrir directement dans Internet Explorer; le navigateur affichera le document XML à l'aide des instructions de transformation de la feuille de styles.

Utilisation d'un template XSL unique

Une feuille de styles XSL contient un ou plusieurs templates (modèles), chacun renfermant l'information pour l'affichage d'une branche particulière des éléments du document XML. Dans cette section, nous présentons une simple feuille de styles XSL qui inclut uniquement un seul template. Ce template contient l'information pour l'affichage du document entier

La feuille de style "XslGCCA.xsl"

```
<?xml version="1.0"?>
<!--Nom de fichier: XslGCCA.xsl-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <H2>Les termes complexes.</H2>
    <TABLE BORDER="1" CELLPADDING="5">
      <THEAD>
        <TH>Numéro</TH>
        <TH>Les Termes</TH>
      </THEAD>
      <xsl:for-each select="LESMOTS/ENREGISTREMENT">
        <TR ALIGN="LEFT">
          <TD STYLE="font-weight:bold">
            <xsl:value-of select="NUMERO"/>
          </TD>

          <TD>
            <xsl:value-of select="EXPRESSION"/>
          </TD>

        </TR>
```

```
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

Chaque feuille de styles XSL doit comporter un élément document illustré ci-dessous.

```
<xsl: stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <!--un ou plusieurs éléments  de modèles...-->
</xsl:stylesheet>
```

L'élément document xsl:stylesheet ne sert pas seulement à recueillir les autres éléments, mais il identifie également le document comme feuille de styles XSL. C'est l'un des éléments XSL multifonctions utilisés dans une feuille de styles. L'ensemble des éléments XSL appartient à l'espace de noms xsl et celui-ci est placé dans le marqueur d'ouverture de l'élément xsl:stylesheet, comme ceci :

xmlns:xsl="http://www.w3.org/TR/WD-xsl"

Cette définition permet d'utiliser l'espace de noms dans n'importe quel élément d'une feuille de styles.

Le template

```
<xsl:template match="/">
    <!--éléments enfants ...-->
</xsl:template>
```

Le navigateur utilise un modèle pour afficher une branche particulière de la hiérarchie de l'élément dans le document XML auquel la feuille de styles est liée. L'attribut *match* indique la branche spécifique. La valeur de l'attribut *match* est connue sous le nom de pattern. Le pattern dans cet exemple ("/") représente la racine du document XML entier. Ce modèle particulier contient ainsi des instructions pour l'affichage du document XML complet.

Une template contient deux sortes d'élément XML :

1. *Des éléments représentant le marquage HTML.*

Voici un exemple de ce type d'élément XML, tirés de la feuille de styles précédente :

<H2>Les termes complexes.</H2>

affiche un titre de second niveau.

2. *Des éléments XSL.*

L'élément XSL *value-of* ajoute le contenu texte de l'élément XML spécifié - et de tous les éléments de sa filiation – à la sortie HTML, rendue affichée par le navigateur. On spécifie l'élément XML particulier via le pattern affecté à l'attribut select de l'élément XSL value-of.. Dans l'exemple précédent, on accède à chacun des éléments de la filiation de ENREGISTREMENT par un pattern contenant le nom de l'élément comme dans cet exemple :

<xsl:value-of select="NUMERO"/>

L'élément *for-each* fournit une sortie à partir du bloc d'éléments contenu dans l'élément for-each qui se répète une fois pour chaque élément XML du document correspondant au pattern, associé à l'attribut select de l'élément for-each. Dans cet exemple la boucle répète la procédure une fois pour chaque élément ENREGISTREMENT trouvé dans l'élément document LESMOTS

```
<xsl:for-each select="LESMOTS/ENREGISTREMENT">
  <TR ALIGN="LEFT">
    <TD STYLE="font-weight:bold">
      <xsl:value-of select="NUMERO"/>
    </TD>

    <TD>
      <xsl:value-of select="EXPRESSION"/>
    </TD>
```

```
        </TR>
    </xsl:for-each>
```

En résumé, une feuille de styles XSL indique au navigateur comment afficher un document XML.

6.4.5.2 Affichage du document XML de base

La fonction AfficheXMLDeBase de la classe Cfichier implémente la présentation des termes complexes dans un document XML. Ce document ne renferme pas des instructions de formatage c'est pourquoi à l'ouverture de celui-ci Internet Exporer ne va qu'afficher le code source du document. AfficheXMLDeBase prend en entrée la liste des termes complexes et donne en sortie un document XML de base. Ce dernier est affecté à la variable m_fichierXMLDeBase qui va permettre son affichage dans une boîte de dialogue.

Exemple : l'analyse syntaxique du terme candidat *Base de données relationnelles,* nous donne la forme normale *((de (relationnelles données)) base)*. Ce dernier est présenté dans un document XML de la manière suivante :

```
<?xml version="1.0"?>
<LESMOTS>
  <ENREGISTREMENT>
      <NUMERO>1</NUMERO>
      <EXPRESSION>((de  (relationnelles donn&#233;es)) base)</EXPRESSION>
  </ENREGISTREMENT>
</LESMOTS>
```

6.4.5.3 Affichage du document XML avec un lien à un document XSL

Les termes complexes sont aussi présentés dans un document XML ayant un lien à une feuille de style XSL. Cette dernière va nous permettre d'afficher les termes complexes selon les instructions de formatage spécifiées. En ce qui nous concerne les termes sont affichés dans un tableau.

Exemple : prenons encore le terme candidat *Base de données relationnelles*. Après l'analyse nous avons le terme complexe ((de(relationnelles données)) base). A l'ouverture du fichier XML qui est lié à la feuille de style "xslGCCA", nous obtenons le tableau 6.4 suivant ayant le titre : "Les Termes Complexes"

Les termes complexes

Numéros	Les Termes
1	((de (relationnelles données)) base)

Tableau 6.4 : Affichage du terme complexe dans un fichier XML lié à une feuille de style.

C'est la fonction AfficherXML de la classe Cfichier qui implémente le document XML ayant les instructions de formatage. Cette fonction prend en entrée la liste des termes complexes et donne en sortie un document XML ayant dans son prologue une instruction de traitement qui le lie au fichier XSL "xslGCCA". Le document est ensuite affecté à la variable m_monfichier qui permet son affichage dans une boîte de dialogue.

Le document XML qui nous a permis d'afficher le tableau 6.4 est le suivant :

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="XslGCCA.xsl"?>
<LESMOTS>
  <ENREGISTREMENT>
    <NUMERO>1</NUMERO>
    <EXPRESSION>((de  (relationnelles donn&#233;es)) base)</EXPRESSION>
```

</ENREGISTREMENT>

</LESMOTS>

Ce document est lié à la feuille de styles "XslGCCA.xsl" présentée dans ce chapitre.

6.4.6 Sauvegarde des documents XML ou base de données XML

La fonction SauverFichier de la classe Cfichier implémente la sauvegarde des données dans un fichier. Cette fonction est appelée dans le programme principal, pour permettre la sauvegarde de deux documents XML dans des fichiers XML.
Le fichier XML ayant un lien à une feuille de styles doit être dans le même dossier que le fichier XSL "xslGCCA".

6.5 Traitement des termes candidats

Traitons les deux termes candidats suivants :

1. base de la théorie des nombres ;
2. langage de définition de modèles de processus d'affaire.

Le terme candidat à analyser est sélectionné dans une liste des termes candidats. Si le terme est préservé par le filtre linguistique, sa forme normale est affichée dans une boîte de dialogue. Cette forme normale est ensuite présentée dans des documents XML ou base de données XML. Les documents XML sont sauvegardés et forment des fichiers XML.

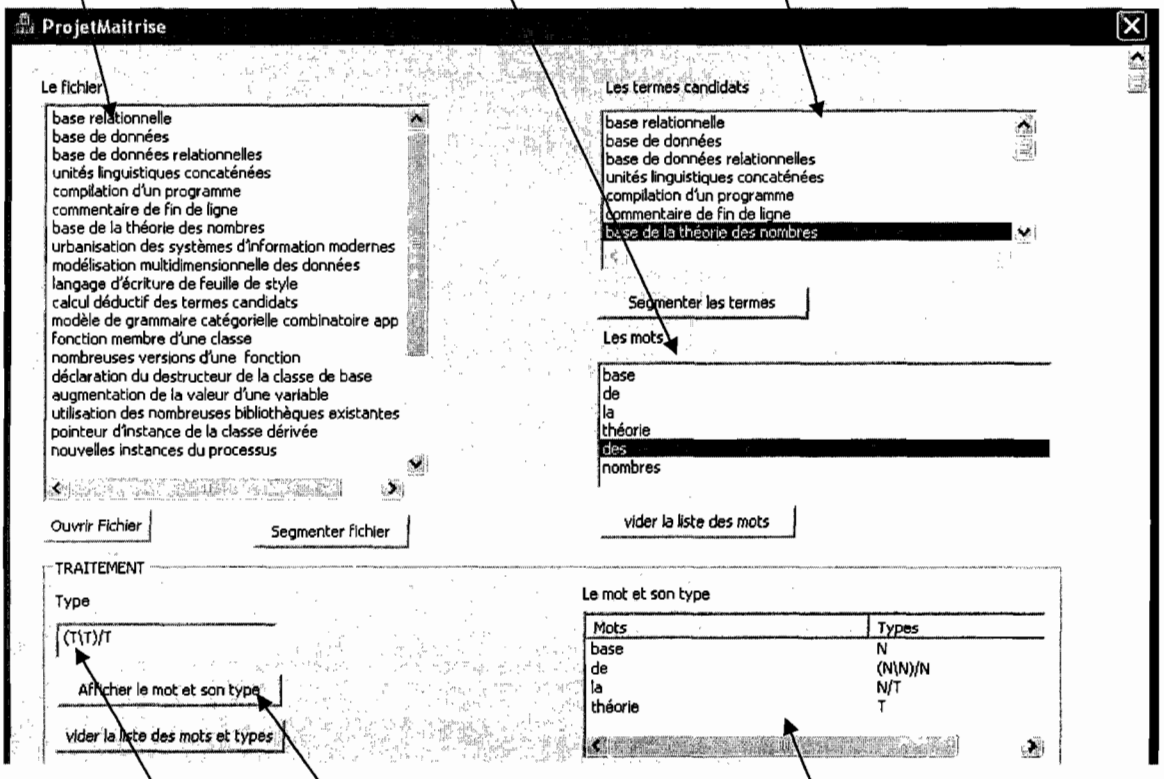
Nous présentons dans la figure 6.8 (a) et (b) le traitement du premier terme : *base de la théorie des nombres.*

(a)

Le fichier qui contient les termes candidats à Traiter

Unités linguistiques (mots) formant le terme à traiter.

La liste des termes candidats.
Le terme : *base de la théorie des nombres* est sélectionné.



Saisie des types syntaxiques

Bouton associé à la fonction qui affiche le mot et son type

Unités linguistiques(mots) associées à leurs types syntaxiques

(b)

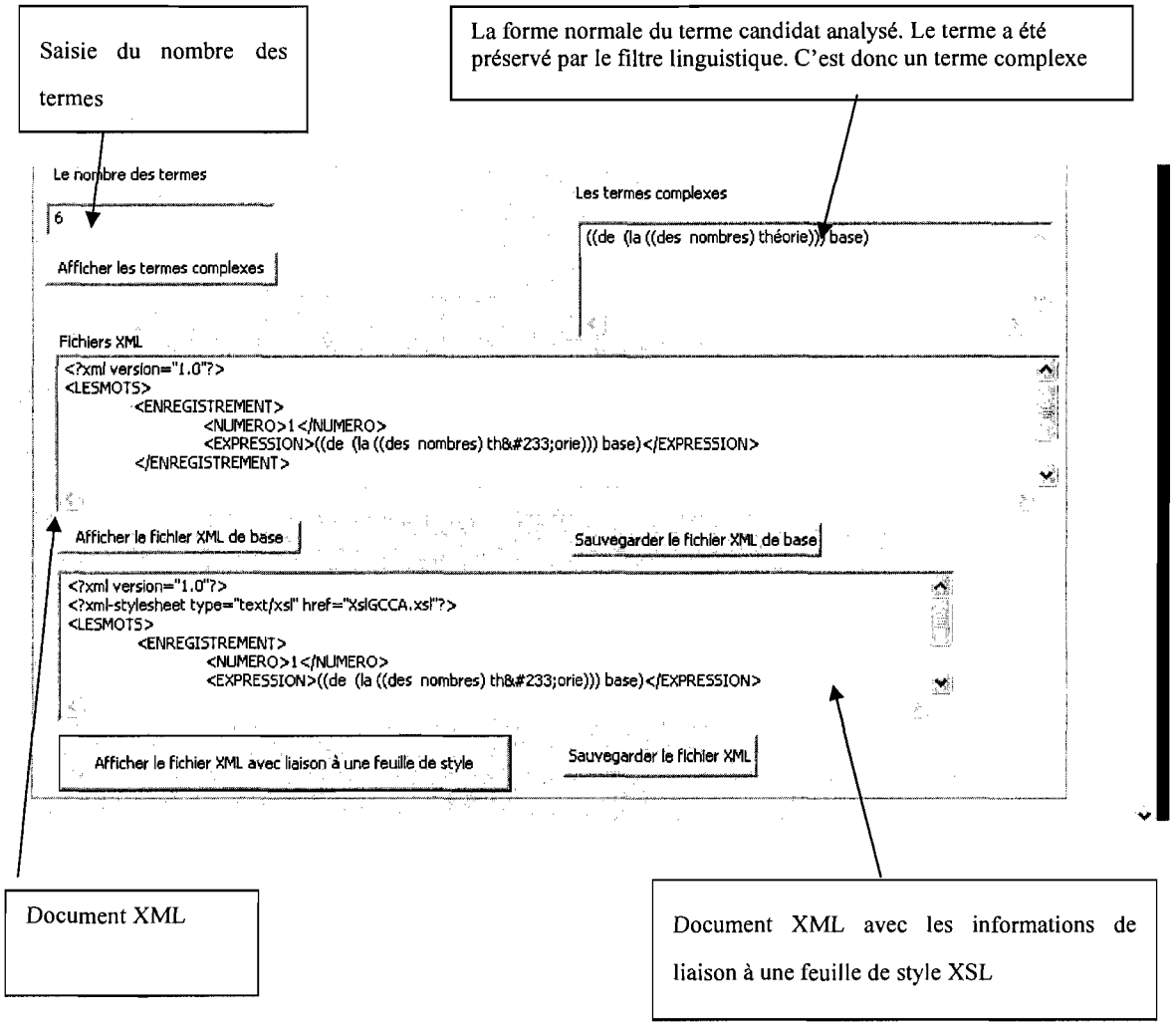


Figure 6.8 : Traitement du terme candidat : *base de la théorie des nombres.*

(a) sélection du terme candidat et attribution des types aux termes.

(b) Affichage des résultats.

Lorsque nous ouvrons le fichier XML, Internet Exporer affiche le code source du document XML présentant le terme complexe. Le terme complexe est donc enregistré dans une base de données XML, comme le montre la figure 6.9.

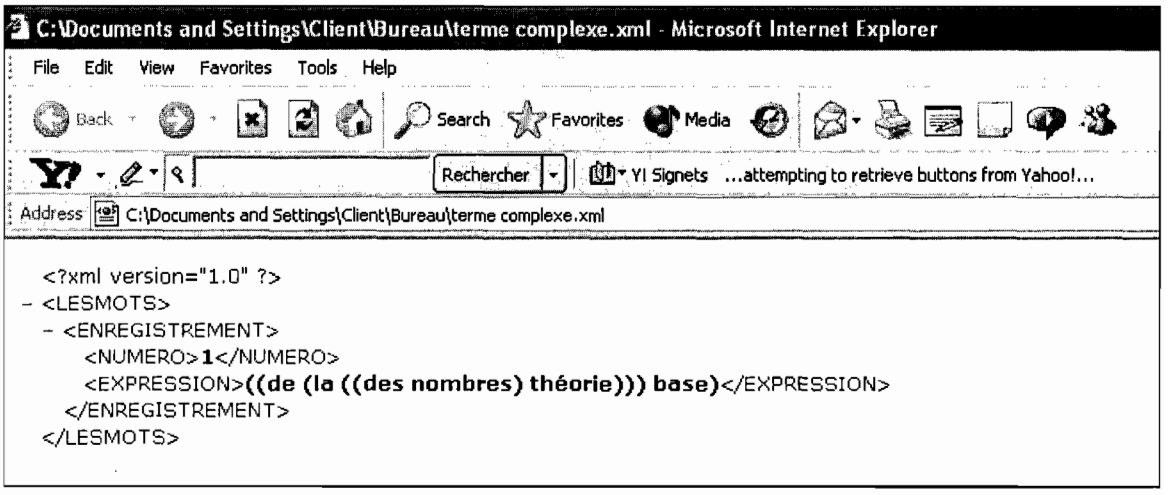


Figure 6.9 : Fichier XML contenant le premier terme validé par notre filtre.

A l'ouverture du fichier XML lié à une feuille de styles XSL, Internet Exporer affiche un document selon les règles spécifiées dans la feuille de styles (figure 6.10). Dans le cas qui nous concerne, les termes complexes sont affichés dans un tableau.

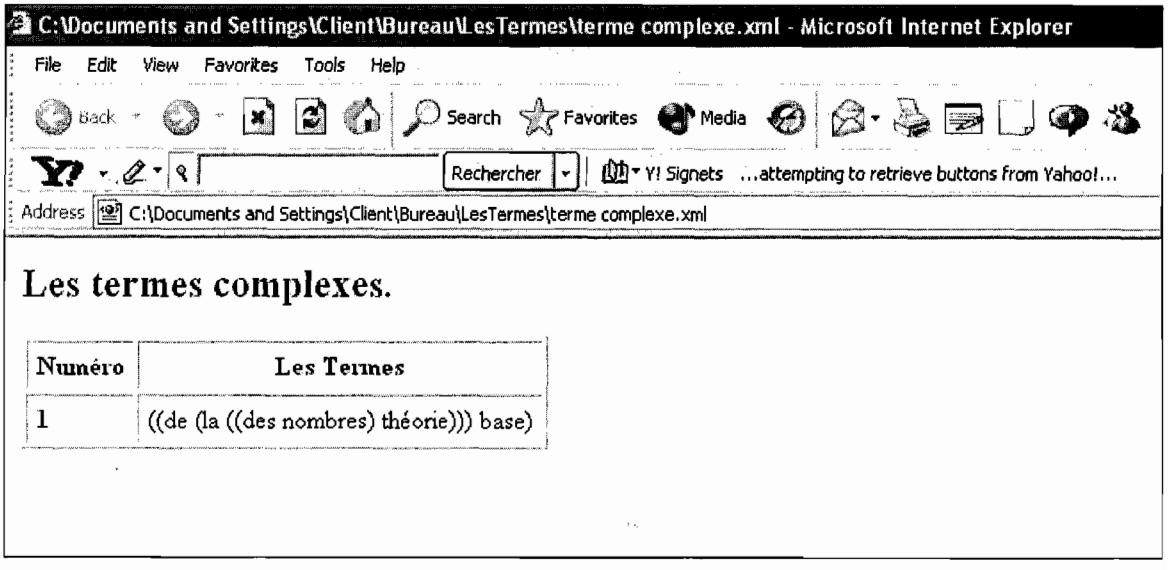


Figure 6.10 : Fichier XML lié à une feuille de styles XSL et contenant le premier terme validé.

Nous présentons à la figure 6.11 le traitement du deuxième terme candidat : *langage de définition de modèles de processus d'affaire.*

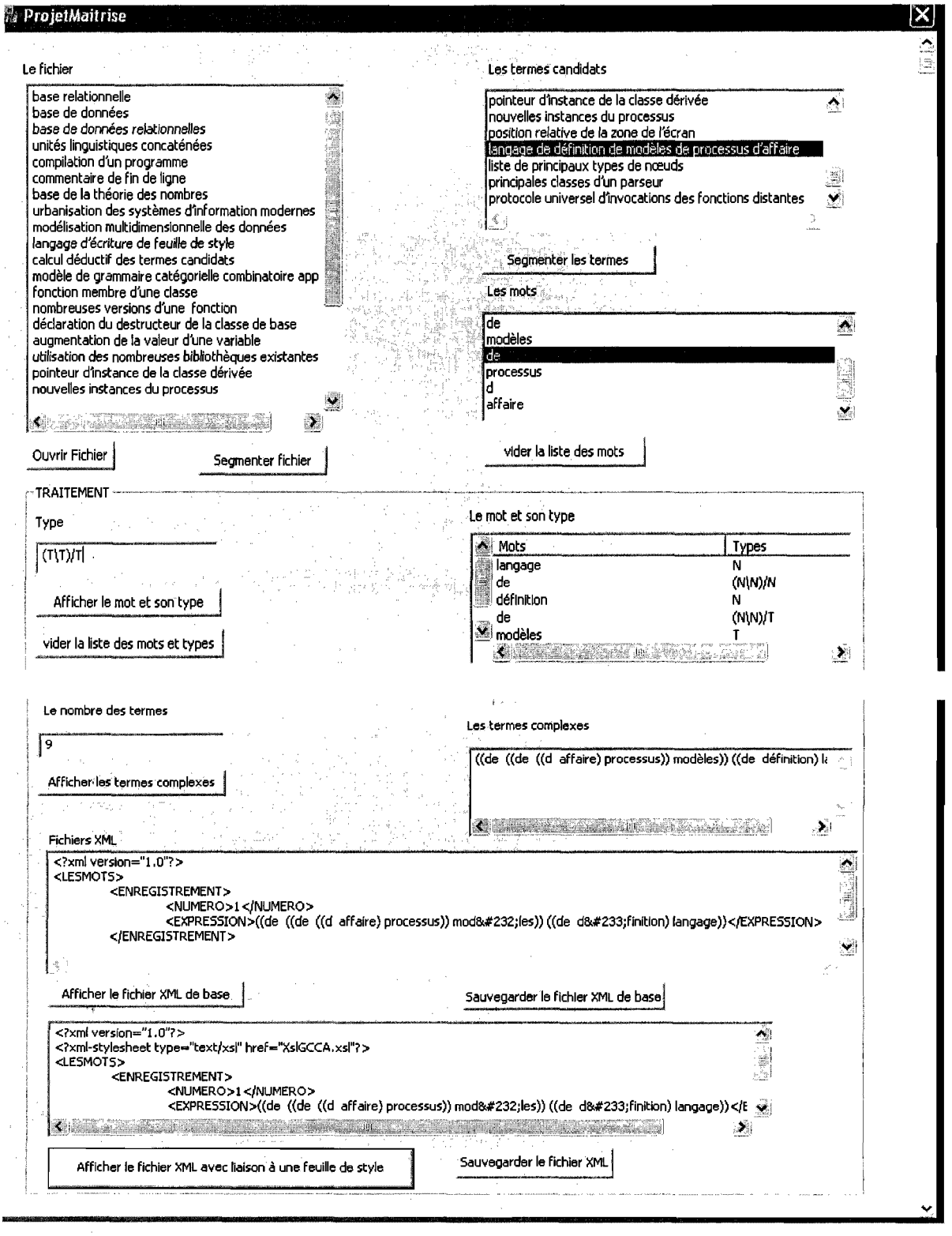


Figure 6.11 : Traitement du terme : *langage de définition de modèles de processus d'affaire.*

Le terme complexe est affiché dans la base de données XML (figure 6.12)

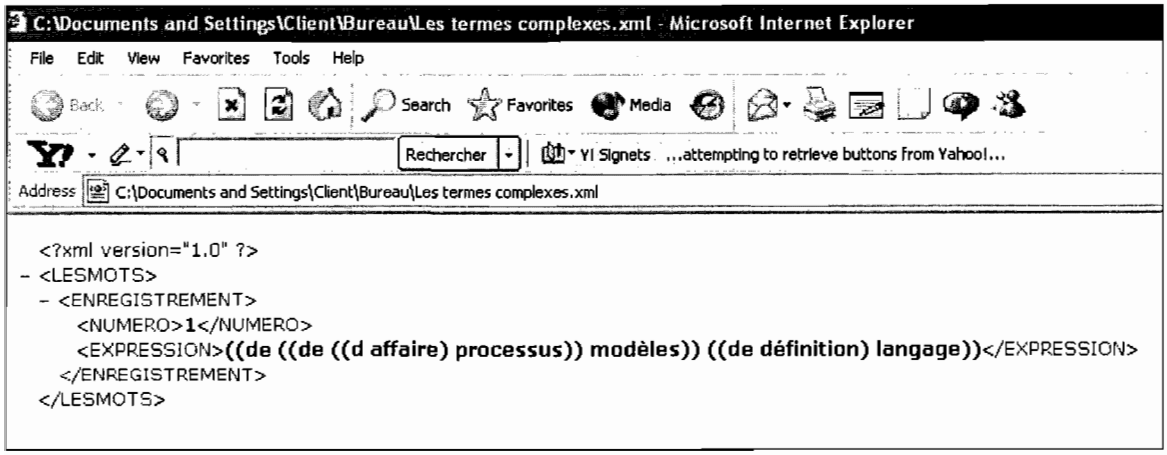


Figure 6.12 : Fichier XML contenant le deuxième terme validé par notre filtre.

Le terme complexe est affiché dans la base de données lié à une feuille de styles (figure 6.13).

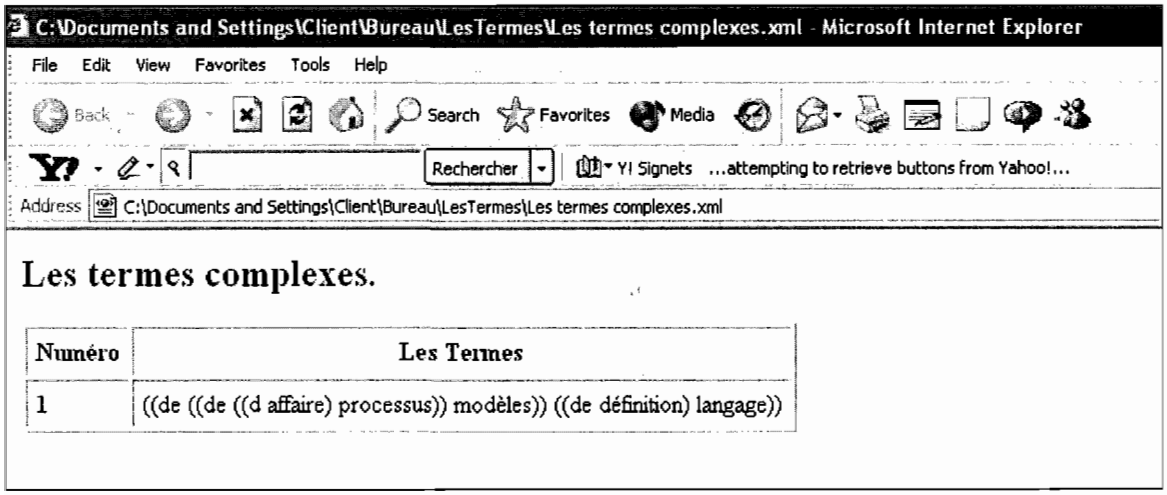


Figure 6.13 : Fichier XML lié à une feuille de styles XSL et contenant le deuxième terme validé.

6.6 Conclusion

Nous sommes arrivés à travers ce chapitre, à présenter l'implémentation d'un filtre linguistique dont le but est l'identification des termes complexes. A travers cette implémentation, nous avons pu prouver qu'il était possible de concevoir un programme informatique qui mettait en pratique les résultats théoriques présentés au chapitre précèdent.

L'implémentation a été faite en C++. Ce langage permet la programmation orientée objet. Nous avons défini plusieurs classes afin de représenter certaines variables importantes de notre programme. Ces classes sont utilisées par d'autres classes du programme et par le programme principal. Les principales données soumises à notre filtre linguistique sont de deux genres : les types syntaxiques et les constructions prédicatives. Les types syntaxiques sont organisés en arbre binaire et les constructions prédicatives sont représentées par des chaînes de caractères.

Le programme principal de notre application est une classe qui dérive de la classe CDialog. Les méthodes du programme principal permettent de lire les données, d'attribuer les types syntaxiques aux mots qui composent le terme candidat à analyser, de procéder à l'analyse syntaxique des termes candidats et d'afficher le résultat de cette analyse.

Les résultats de l'analyse syntaxique sont des structures applicatives des termes validés. Ces structures applicatives subissent la réduction des combinateurs pour construire les formes normales. Ces dernières sont affichées dans une boîte de dialogue. Les termes complexes (formes normales) sont aussi affichés sous forme de document XML et sous forme de document XML ayant un lien à une feuille de style XSL. Les documents XML sont ensuite stockés dans des fichiers XML, car XML offre des possibilités intéressantes pour des manipulations ultérieures des ces termes complexes..

CHAPITRE 7

EVALUATION

Nous avons effectué une évaluation de 100 termes candidats. Nous présentons dans ce chapitre les résultats de cette évaluation.

Nous présentons dans chaque tableau dont le titre est "les termes candidats à traiter" les termes candidats ayant le même modèle. Les unités lexicales d'une colonne ont le même type syntaxique placé sur la première ligne de la colonne. Par exemple, l'unité lexicale *base* appartenant à la première colonne est de type N, et *relationnelle* de la deuxième colonne est de type N\N. Nous avons ainsi les unités typées concaténées [N : *base*]-[N\N : *relationnelle*]. Les tableaux que nous désignons par "Les termes complexes" affichent les résultats du traitement des termes candidats. Les formes normales des termes qui ont été préservés par notre filtre linguistique sont dans la colonne "Les termes". Ce tableau contient aussi une colonne des prédicats et une colonne des arguments des termes complexes.

7.1 Traitement des termes candidats

Nous présentons les 100 termes candidats à traiter en plusieurs groupes.

7.1.1 Les termes validés

Premier groupe

Ce groupe comprend les termes candidats composés de deux termes que nous représentons de la manière suivante : N : terme1- N\N : terme2. Chaque terme1 de type N est concaténé au terme2 correspondant de type N\N. Les termes1 font parti de la

première colonne du tableau 7.1 et les termes2 font parti de la deuxième colonne.

Les termes candidats à traiter

N	-	N\N
base	-	relationnelle
nombre	-	complexe
unités	-	linguistiques
forme	-	normale
structure	-	applicative
grammaire	-	catégorielle
programmation	-	procédurale
fonctions	-	virtuelles
méthodes	-	publiques
bibliothèque	-	standard
architectures	-	multiniveaux
processus	-	unique
arbre	-	binaire

Tableau 7.1 : Premier groupe : N : terme1-N\N : terme2.

Résultats : Tous les termes candidats de ce groupe sont validés par notre filtre linguistique. Le tableau 7.2 présente la structure applicative de chaque terme complexe dans la colonne "Les termes". Il présente également le prédicat et l'argument de chaque terme complexe.

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	(relationnelle base)	relationnelle	base
2	(complexe nombre)	complexe	nombre
3	(linguistiques unités)	linguistiques	unités
4	(normale forme)	normale	forme
5	(applicative structure)	applicative	structure
6	(catégorielle grammaire)	catégorielle	grammaire
7	(procédurale programmation)	procédurale	programmation
8	(virtuelles fonctions)	virtuelles	fonctions
9	(publiques méthodes)	publiques	méthodes
10	(standard bibliothèque)	standard	bibliothèque
11	(multiniveaux architectures)	multiniveaux	architectures
12	(unique processus)	unique	processus
13	(binaire arbre)	binaire	arbre

Tableau 7.2 : Résultats du traitement des termes candidats du premier groupe.

Deuxième groupe

Nous présentons dans ce groupe (tableau 7.3) les termes candidats à trois termes. Nous les représentons de la manière suivantes : N : terme1- N\N : terme2 – N\N : terme3
Exemple : N : unités - N\N : linguistiques – N\N : concaténées.

Les termes candidats à traiter

N	-	N\N	-	N\N
unités	-	linguistiques	-	concaténées
règles	-	combinatoires	-	applicatives
type	-	catégoriel	-	spécifique

Tableau 7.3 : Deuxième groupe : N : terme1- N\N : terme2 – N\N : terme3.

Résultats : les 3 termes candidats présentés dans ce groupe sont préservés par notre filtre linguistique. Nous exposons le résultat des traitements dans le tableau 7.4.

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	(concaténées (linguistiques unités))	concaténées	(linguistiques unités)
2	(applicatives (combinatoires règles))	applicatives	(combinatoires règles)
3	(spécifique (catégoriel type))	spécifique	(catégoriel type)

Tableau 7.4 : Résultats du traitement des termes candidats du deuxième groupe.

Troisième groupe

Nous présentons dans le tableau 7.5 deux termes candidats ayant la forme :

N : terme1 - N\N : terme2 – (N\N)/N : terme3 – N : terme4

Exemple : N : modélisation - N\N : multidimensionnelle – (N\N)/N : de – N : données

Les termes candidats à traiter

N	-	N\N	-	(N\N)/N	-	N
modélisation	-	multidimensionnelle	-	des	-	données
scénario	-	typique	-	de	-	fonctionnement

Tableau 7.5 : Troisième groupe : N : terme1 - N\N : terme2 – (N\N)/N : terme3 – N : terme4

Résultats : Nous présentons les résultats du traitement dans le tableau 7.6.

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((des données) (multidimensionnelle modélisation))	(des données)	(multidimensionnelle modélisation)
2	((de fonctionnement) (typique scénario))	(de fonctionnement)	(typique scénario)

Tableau 7.6 : Résultats du traitement des termes candidats du troisième groupe

Quatrième groupe

Les termes candidats du tableau 7.7 sont représentés comme suit :

N : terme1 – (N\N)/T : terme2 – T/T : terme3 – T : terme4

Exemple : N : déclaration – (N\N)/T : de – T/T : la – T : classe

Les termes candidats à traiter

N	-	(N\N)/T	-	T/T	-	T
déclaration	-	d'	-	une	-	notation
compilation	-	d'	-	un	-	programme
déclaration	-	de	-	la	-	classe
valeur	-	d'	-	une	-	variable
affichage	-	de	-	l'	-	objet
structure	-	de	-	l'	-	élément

Tableau 7.7 : Quatrième groupe : N : terme1 – (N\N)/T : terme2 – T/T : terme3 – T : terme4

Résultats : Notre filtre a validé les 6 termes traités, les résultats sont affichés dans le tableau 7.8.

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((d (une notation)) déclaration)	(d (une notation))	déclaration
2	((d (un programme)) compilation)	(d (un programme))	compilation
3	((de (la classe)) déclaration)	(de (la classe))	déclaration
4	((d (une variable)) valeur)	(d (une variable))	valeur
5	((de (l objet)) affichage)	(de (l objet))	affichage
6	((de (l élément)) structure)	(de (l élément))	structure

Tableau 7.8 : Résultats du traitement des termes candidats du quatrième groupe.

Cinquième groupe

Le tableau 7.9 renferme les termes candidats que nous représentons comme suit :

N : Terme1 – (N\N)/N :terme2 – N : terme3

Exemple : N : Base – (N\N)/N : de – N : données

Les termes candidats à traiter

N		(N\N)/N		N
Base	-	de	-	données
Règle	-	de	-	permutation
Langage	-	de	-	programmation
Type	-	de	-	données
fonction	-	de	-	tri
fonction	-	de	-	recherche
déclaration	-	du	-	destructeur
gestion	-	des	-	exceptions
surcharge	-	des	-	opérations
opérateur	-	d'	-	incrémentation
espaces	-	de	-	noms
pointeur	-	d'	-	instance
environnements	-	de	-	développement
chaîne	-	de	-	caractères
intégration	-	de	-	données
architecture	-	de	-	système
revue	-	de	-	technologies
spécifications	-	d'	-	échanges
connexion	-	des	-	applications
techniques	-	de	-	base
zone	-	d'	-	affichage
processus	-	d'	-	affaire
transaction	-	d'	-	affaires
problème	-	de	-	fiabilité
recherche	-	d'	-	information
schéma	-	de	-	collaboration

Tableau 7.9 : Cinquième groupe : N : terme1 – (N\N)/N :terme2 – N : terme3.

Résultats : Les 26 termes traités par notre filtre sont validés. Nous présentons dans le tableau 7.10 leurs structures applicatives, leurs prédicats et leurs arguments.

Les termes complexes

Numéro	Les Termes	Prédicats	Arguments
1	((de données) Base)	(de données)	Base
2	((de permutation) Règle)	(de permutation)	Règle
3	((de programmation) langage)	(de programmation)	langage
4	((de données) type)	(de données)	type
5	((de tri) fonction)	(de tri)	fonction
6	((de recherche) fonction)	(de recherche)	fonction
7	((du destructeur) déclaration)	(du destructeur)	déclaration
8	((des exceptions) gestion)	(des exceptions)	gestion
9	((des opérations) surcharge)	(des opérations)	surcharge
10	((d incrémentation) opérateur)	(d incrémentation)	opérateur
11	((de noms) espaces)	(de noms)	espaces
12	((d instance) pointeur)	(d instance)	pointeur
13	((de développement) environnements)	(de développement)	environnements
14	((de caractères) chaînes)	(de caractères)	chaînes
15	((de données) intégration)	(de données)	intégration
16	((de système) architecture)	(de système)	architecture
17	((de technologies) revue)	(de technologies)	revue
18	((d échanges) spécifications)	(d échanges)	spécifications
19	((des applications) connexion)	(des applications)	connexion
20	((de base) techniques)	(de base)	techniques
21	((d affichage) zone)	(d affichage)	zone
22	((d affaire) processus)	(d affaire)	processus
23	((d affaires) transaction)	(d affaires)	transaction
24	((de fiabilité) problème)	(de fiabilité)	problème

Numéro	Les Termes	Prédicats	Arguments
25	((d information) recherche)	(d information)	recherche
26	((de collaboration) schéma)	(de collaboration)	schéma

Tableau 7.10 : Résultats du traitement des termes candidats du cinquième groupe.

Sixième groupe

Les termes candidats de ce groupe (tableau 7.11) ont la forme :

N : terme1- (N\N)/T : terme2 – T : terme3 – T\T : terme4

Exemple : N : Base - (N\N)/T : de – T : données – T\T : relationnelles

Les termes candidats à traiter

N	-	(N\N)/T	-	T	-	T\T
Base	-	de	-	données	-	relationnelles
Théorie	-	de	-	nombres	-	complexes
Validation	-	de	-	termes	-	complexes
Traitement	-	des	-	langues	-	naturelles
Règles	-	d'	-	application	-	fonctionnelle
Mécanisme	-	des	-	méthodes	-	virtuelles
Téléchargement	-	d'	-	applications	-	portables
Exécution	-	du	-	code	-	applicatif
Ensemble	-	d'	-	action	-	atomique
Format	-	de	-	données	-	riches
Assignation	-	de	-	variables	-	globales
Valeurs	-	d'	-	attributs	-	légales
Nombre	-	de	-	caractéristiques	-	optionnelles
Création	-	des	-	programmes	-	spécialisées

Tableau 7.11 : Sixième groupe : N : terme1- (N\N)/T : terme2 – T : terme3 – T\T : terme4.

Résultats : Le tableau 7.12 affiche les structures applicatives, les prédicats ainsi que les arguments des termes candidats précédents qui sont tous préservés par notre filtre.

Les termes complexes

Numéro	Les Termes	Prédicats	Arguments
1	((de (relationnelles données)) Base)	(de (relationnelles données))	Base
2	((de (complexes nombres)) Théorie)	(de (complexes nombres))	Théorie
3	((de (complexes termes)) Validation)	(de (complexes termes))	Validation
4	((des (naturelles langues)) Traitement)	(des (naturelles langues))	Traitement
5	((d (fonctionnelle application)) Règles)	(d (fonctionnelle application))	Règles
6	((des (virtuelles méthodes)) Mécanisme)	(des (virtuelles méthodes))	Mécanisme
7	((d (portables applications)) Téléchargement)	(d (portables applications))	Téléchargement
8	((du (applicatif code)) Exécution)	(du (applicatif code))	Exécution
9	((d (atomique action)) Ensemble)	(d (atomique action))	Ensemble
10	((de (riches données)) Format)	(de (riches données))	Format
11	((de (globales variables)) Assignation)	(de (globales variables))	Assignation
12	((d (légales attributs)) Valeurs)	(d (légales attributs))	Valeurs
13	((de (optionnelles caractéristiques)) Nombre)	(de (optionnelles caractéristiques))	Nombre
14	((des (spécialisées programmes)) Création)	(des (spécialisées programmes))	Création

Tableau 7.12 : Résultats du traitement des termes candidats du sixième groupe.

Septième groupe

Nous représentons les termes candidats de ce groupe (Tableau7.13) comme suit :

N : terme1 – (N\N)/T : terme2 – T : terme3 – (T\T)/T : terme4 – T : terme5 Exemple :

N : commentaire – (N\N)/T : de – T : fin – (T\T)/T : de – T : ligne

Les termes candidats à traiter

N	-	(N\N)/T	-	T	-	(T\T)/T	-	T
Commentaire	-	de	-	fin	-	de	-	ligne
Langage	-	de	-	description	-	de	-	données
Langage	-	d'	-	échange	-	de	-	données
Langage	-	de	-	transformation	-	des	-	documents
Langage	-	d'	-	orchestration	-	de	-	services
Évaluation	-	des	-	expressions	-	de	-	chemin
Séquences	-	de	-	processus	-	d'	-	affaire
Support	-	de	-	modèles	-	de	-	transaction
Activité	-	de	-	production	-	de	-	message
Définition	-	de	-	flux	-	de	-	contrôle

Tableau 7.13 : Septième groupe : N : terme1 – (N\N)/T : terme2 – T : terme3 – (T\T)/T : terme4 – T : terme5.

Résultats : Tous les termes candidats du septième groupe sont validés par notre filtre.
Les résultats du traitement sont affichés dans le tableau 7.14

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((de ((de ligne) fin)) commentaire)	(de ((de ligne) fin))	commentaire
2	((de ((de données) description)) langage)	(de ((de données) description))	langage
3	((d ((de données) échange)) langage)	(d ((de données) échange))	langage
4	((de ((des documents) transformation)) langage)	(de ((des documents) transformation))	langage
5	((d ((de services) orchestration)) langage)	(d ((de services) orchestration))	langage
6	((des ((de chemin) expressions)) évaluation)	(des ((de chemin) expressions))	évaluation
7	((de ((d affaire) processus)) séquences)	(de ((d affaire) processus))	séquences
8	((de ((de transaction) modèles)) support)	(de ((de transaction) modèles))	support
9	((de ((de message) production)) activité)	(de ((de message) production))	activité
10	((de ((de contrôle) flux)) définition)	(de ((de contrôle) flux))	définition

Tableau 7.14 : Résultats du traitement des termes candidats du septième groupe.

Huitième groupe

Les trois termes candidats de ce groupe (tableau 7.15) sont représentés comme suit :
N : terme1- (N\N)/N : terme2 – N/T : terme3 – T : terme4 – (T\T)/T : terme5 – T : terme6
Exemple : N : base - (N\N)/N : de – N/T : la – T : théorie – (T\T)/T : des – T : nombres

Les termes candidats à traiter

N	-	(N\N)/N	-	N/T	-	T	-	(T\T)/T	-	T
base	-	de	-	la	-	théorie	-	des	-	nombres
utilisation	-	de	-	la	-	règle	-	de	-	permutation
largeur	-	de	-	la	-	zone	-	d'	-	affichage

Tableau 7.15 : Huitième groupe : N : terme1- (N\N)/N : terme2 – N/T : terme3 – T : terme4 – (T\T)/T : terme5 – T : terme6.

Résultats : Les trois termes sont validés par notre filtre. Les résultats du traitement sont présentés dans le tableau 7.16 :

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((de (la ((des nombres) théorie))) base)	(de (la ((des nombres) théorie)))	base
2	((de (la ((de permutation) règle))) utilisation)	(de (la ((de permutation) règle)))	utilisation
3	((de (la ((d affichage) zone))) largeur)	(de (la ((d affichage) zone)))	largeur

Tableau 7.16 : Résultats du traitement des termes candidats du huitième groupe.

Neuvième groupe

Le tableau 7.17 renferme les termes candidats à 6 termes que nous représentons comme suit : N : terme1 – (N\N)/T : terme2 – T : terme3 – (T\T)/T : terme4 – T : terme5 T\T : terme6.
Exemple : N : urbanisation – (N\N)/T : des – T :système – (T\T)/T : d – T :information
T\T : moderne

Les termes candidats à traiter

N	-	(N\N)/T	-	T	-	(T\T)/T	-	T	-	T\T
Urbanisation	-	des	-	systèmes	-	d'	-	informations	-	modernes
Technologies	-	de	-	génération	-	de	-	pages	-	dynamiques
Variété	-	de	-	types	-	d'	-	éléments	-	significatifs
Coordination	-	de	-	processus	-	d'	-	affaires	-	collaboratifs

Tableau 7.17 : Neuvième groupe : N : terme1 – (N\N)/T : terme2 – T : terme3 – (T\T)/T : terme4 – T : terme5 - T\T : terme6.

Résultats : Les 4 termes candidats sont préservés par notre filtre linguistique, les résultats du traitement sont affichés dans le tableau 7.18 :

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((des ((d (modernes informations)) systèmes)) urbanisation)	(des ((d (modernes informations)) systèmes))	urbanisation
2	((de ((de (dynamiques pages)) génération)) technologies)	(de ((de (dynamiques pages)) génération))	technologies
3	((de ((d (significatifs éléments)) types)) variété)	(de ((d (significatifs éléments)) types))	variété
4	((de ((d (collaboratifs affaires)) processus)) coordination)	(de ((d (collaboratifs affaires)) processus))	coordination

Tableau 7.18 : Résultats du traitement des termes candidats du neuvième groupe

Dixième Groupe

Les 15 termes candidats de ce groupe sont formés d'unités lexicales typées. Aucun terme candidat de ce groupe n'a la forme des termes candidats présentés dans les tableaux précédents. Ces 15 termes ont été préservés par notre filtre linguistique. Les résultats du traitement sont affichés dans le tableau 7.19.

1. [N :langage]-[(N\N)/N :d]-[N :écriture]-[(N\N)/T :de]-[T :feuille]-[(T\T)/T :de]-[T :style]
2. [N :calcul]-[N\N : déductif]-[(N\N)/T :des]-[T :termes]-[T\T : candidats]
3. [N :modèle]-[(N\N)/N :de]-[N :grammaire]-[N\N : catégorielle]-[N\N : combinatoire]-[N\N :applicative]
4. [N/N :nombreuses]-[N : versions]-[(N\N)/N : d]-[N/N :une]-[N :fonction]
5. [N :déclaration]-[(N\N)/N :du]-[N :destructeur]-[(N\N)/N :de]-[N/T :la]-[T :classe]-[(T\T)/T :de]-[T :base]
6. [N :augmentation]-[(N\N)/N :de]-[N/N :la]-[N :valeur]-[(N\N)/T :d]-[T/T :une]-[T :variable]
7. [N :utilisation]-[(N\N)/N :des]-[N/N :nombreuses]-[N :bibliothèques]-[N\N :existantes]
8. [N :pointeur]-[(N\N)/N :d]-[N :instance]-[(N\N)/T :de]-[T/T :la]-[T :classe]-[T\T :dérivée]
9. [N/N :nouvelles]-[N :instances]-[(N\N)/N :du]-[N :processus]
10. [N :position]-[N\N :relative]-[(N\N)/N :de]-[N/T :la]-[T :zone]-[(T\T)/T :de]-[T/T :l]-[T :écran]
11. [N :langage]-[(N\N)/N :de]-[N :définition]-[(N\N)/T :de]-[T :modèles]-[(T\T)/T :de]-[T :processus]-[(T\T)/T :d]-[T :affaire]
12. [N :liste]-[(N\N)/N :de]-[N/T :principaux]-[T :types]-[(T\T)/T :d]-[T :éléments]
13. [N/N :principales]-[N :classes]-[(N\N)/N :d]-[N/N :un]-[N :parseur]
14. [N :protocole]-[N\N :universel]-[(N\N)/T :d]-[T :invocation]-[(T\T)/T :des]-[T :fonctions]-[T\T :distantes].
15. [N :ensemble]-[(N\N)/N :de]-[N :déclarations]-[(N\N)/T :d]-[T :espace]-[(T\T)/T :de]-[T : noms]

Les termes complexes.

Numéro	Les Termes	Prédicats	Arguments
1	((de ((de style) feuille)) ((d écriture) langage))	(de ((de style) feuille))	((d écriture) langage)
2	((des (candidats termes)) (déductif calcul))	(des (candidats termes))	(déductif calcul)
3	(applicative (combinatoire (catégorielle ((de grammaire) modèle))))	(applicative (combinatoire (catégorielle)))	((de grammaire) modèle)
4	((d (une fonction)) (nombreuses versions))	(d (une fonction))	(nombreuses versions)
5	((de (la ((de base) classe))) ((du destructeur)	(de (la ((de base) classe)))	((du

Numéro	Les Termes	Prédicats	Arguments
	déclaration))		destructeur) déclaration)
6	((d (une variable)) ((de (la valeur)) augmentation)))	(d (une variable))	((de (la valeur)) augmentation)
7	(existantes ((des (nombreuses bibliothèques)) utilisation)))	existantes	((des (nombreuses bibliothèques)) utilisation))
8	((de (la (dérivée classe))) ((d instance) pointeur))	(de (la (dérivée classe)))	((d instance) pointeur))
9	((du processus) (nouvelles instances))	(du processus)	(nouvelles instances)
10	((de (la ((de (l écran)) zone))) (relative position))	(de (la ((de (l écran)) zone)))	(relative position)
11	((de ((de ((d affaire) processus)) modèles)) ((de définition) langage))	(de ((de ((d affaire) processus)) modèles))	((de définition) langage)
12	((de (principaux ((d éléments) types))) liste)	(de (principaux ((d éléments) types)))	liste
13	((d (un parseur)) (principales classes))	(d (un parseur))	(principales classes)
14	((d ((des (distantes fonctions)) invocations)) (universel protocole))	(d ((des (distantes fonctions)) invocations))	(universel protocole)
15	((d ((de noms) espace)) ((de déclarations) ensemble))	(d ((de noms) espace))	((de déclarations) ensemble)

Tableau 7.19 : Résultats du traitement des termes candidats du dixième groupe

7.1.2 Les termes candidats rejetés

Les termes candidats suivants ont été rejetés par notre filtre linguistique :

- fonction membre d'une classe,
- fonction membre virtuelle,
- interface utilisateur,
- liens hypertextes multivalués

1. [N : fonction] – [N : membre] – [(N\N)/N : d] – [N/N : une] – [N : classe]
2. [N : fonction] – [N : membre] – [N\N : virtuelle]
3. [N : interface]- [N : utilisateur]
4. [N : liens] – [N : hypertextes] – [N\N : multivalués]

Nous retrouvons dans ces quatre termes l'agencement "Nom – Nom". Notre filtre linguistique n'est pas capable de reconnaître ce genre de structure parce que, dans cette situation, aucune règle combinatoire ne peut être appliquée. Toutefois si nous considérons le deuxième terme "Nom" comme modifieur du premier terme, cela nous permettra d'appliquer la règle d'application arrière (N- N\N) et de continuer l'analyse. Dans ce cas les quatre termes seront acceptés.

7.1.3 Constatation

Nous avons soumis à notre filtre linguistique 100 termes candidats à traiter.
96 termes candidats ont été validés par notre filtre et 4 termes ont été rejetés. Nous avons alors : 96 % de termes candidats préservés par notre filtre et 4% de termes candidats rejetés.

$$\text{Le taux des termes qui auraient dû être rejetés} = \frac{\text{Termes qui auraient dû être rejetés}}{\text{Termes préservés}} = \frac{0}{96} = 0$$

$$\text{Le taux des termes qui auraient dû être acceptés} = \frac{\text{Termes qui auraient du être acceptés}}{\text{Termes rejetés}} = \frac{4}{4} = 1$$

Nous avons dans notre corpus quatre termes candidats qui sont rejetés alors qu'ils auraient dû être acceptés. Notre filtre ne reconnaît pas l'agencement "Nom – Nom", car notre analyseur considère les termes ayant cet agencement comme syntaxiquement incorrects. Pour que ces termes soient validés nous considérons le deuxième terme

"Nom" comme modifieur du premier terme. Nous aurons dans ce cas un agencement "Nom - adjectif" qui est accepter par notre filtre.

7.2 Traitement complet des termes candidats

Nous présentons un traitement détaillé des termes suivants :

1. commentaire de fin de ligne,
2. utilisation de la règle de permutation,
3. protocole universel d'invocation des fonctions distantes,
4. position relative de la zone de l'écran,
5. langage de définition de modèles de processus d'affaire.

Analyse du terme 1

[N : "commentaire"] - [N\N/T : "de"] - [T : "fin"] - [T\T/T : "de"] - [T : "ligne"]
[N : "commentaire"] - [N/T\T : " (C de) "] - [T : "fin"] - [T\T/T : "de"] - [T : "ligne"]
[N/T : " ((C de) commentaire) "] - [T : "fin"] - [T\T/T : "de"] -[T : "ligne"]
[N : " (((Cde) commentaire) fin) "] - [T\T/T : " de"] - [T : "ligne"]
[N/T : " ((C de) commentaire) "] - [T : "fin"] - [T\T/T : "de"] - [T : "ligne"]
[N/T : " ((C de) commentaire) "] - [T : "fin"] - [T/T\T : " (C de) "] - [T : "ligne"]
[N/T : " ((C de) commentaire) "] - [T/T : " ((C de) fin) "] - [T : "ligne"]
[N/T : " (B ((C de) commentaire) ((C de) fin)) "] - [T ":ligne"]
[N : " ((B ((C de) commentaire) ((C de) fin)) ligne) "]

Réduction des combinateurs

"((B ((C de) commentaire) (((C de) fin) ligne)"
"(((C de) commentaire) (((C de) fin) ligne))"
"((de (((C de) fin) ligne)) commentaire)"
"((de ((de ligne) fin)) commentaire)"

Le terme candidat est accepté par notre filtre linguistique. Nous aboutissons à la fin de l'analyse syntaxique à l'expression applicative de type N. N est le type qu'il faut pour qu'un terme soit validé. Nous obtenons, après la réduction des combinateurs, la forme normale : "((de ((de ligne) fin)) commentaire)"

Analyse du terme 2

[N : "utilisation"] - [N\N/N : "de"] - [N/T : "la"] - [T : "règle"] - [T\T/T : "de"] - [T : " permutation"]
[N : "utilisation"] - [N/N\N : " (C de) "] - [N/T : "la"] - [T : "règle"] - [T\T/T : "de"] - [T : " permutation"]
[N/N : " ((C de)utilisation) "] - [N/T : "la"] - [T : "règle"] - [T\T/T : "de"] - [T : "permutation"]
[N/T : " (B ((C de)utilisation) la) "] - [T : "règle"] - [T\T/T : "de"] - [T : " permutation"]
[N : " ((B ((C de)utilisation) la) règle) "] - [T\T/T : "de"] - [T : "permutation"]
[N/T : " (B ((C de)utilisation) la) "] - [T : "règle"] - [T\T/T : "de"] - [T : "permutation"]
[N/T : " (B ((C de)utilisation) la) "] - [T : "règle"] - [T/T\T : " (C de) "] - [T : "permutation"]
[N/T : " (B ((C de)utilisation) la) "] - [T/T : " ((C de) règle) "] - [T : " permutation"]
[N/T : " (B (B ((C de)utilisation) la) ((C de) règle)) "] - [T : "permutation"]
[N : " ((B (B ((C de)utilisation) la) ((C de) règle)) permutation) "]

Réduction des combinateurs

"((B (B ((C de) utilisation) la) ((C de) règle)) permutation)"
"((B ((C de) utilisation) la) (((C de) règle) permutation))"
"(((C de) utilisation) (la (((C de) règle) permutation)))"
"((de (la (((C de) règle) permutation))) utilisation)"
"((de (la ((de permutation) règle))) utilisation)"

Le terme candidat traité est validé par le filtre car nous avons, à la fin de l'analyse syntaxique, une expression applicative de type N. Nous obtenons après la réduction des combinateurs la forme normale : "((de (la ((de permutation) règle))) utilisation)"

Analyse du terme 3

[N : "protocole"] - [N\N : "universel"] - [N\N/T : "d"] - [T : "invocation"] - [T\T/T : "des"] -
[T : "fonctions"] - [T\T : "distantes"]
[N : " (universel protocole) "] - [N\N/T : "d"] - [T : "invocation"] - [T\T/T : "des"] - [T : "fonctions"] -
[T\T : "distantes"]
[N : " (universel protocole) "] - [N/T/N : " (C d) "] - [T : "invocation"] - [T\T/T : "des"] -
[T : "fonctions"] - [T\T : "distantes"]
[N/T : " ((C d)(universel protocole)) "] - [T : "invocation"] - [T\T/T : "des"] - [T : "fonctions"] -
[T\T : "distantes"]
[N: " (((C d)(universel protocole)) invocation) "] - [T\T/T : "des"] - [T : "fonctions"] -
[T\T : "distantes"]
[N/T : " ((C d)(universel protocole)) "] - [T : "invocation"] - [T\T/T : "des"] - [T : "fonctions"] -
[T\T : "distantes"]
[N/T : " ((C d)(universel protocole)) "] - [T : "invocation"] - [T/T\T : " (C des) "] - [T : "fonctions"] -
[T\T : "distantes"]
[N/T : " ((C d)(universel protocole)) "] - [T/T : " ((C des) invocation) "] - [T : "fonctions"] -
[T\T : "distantes"]
[N/T : " (B ((C d)(universel protocole)) ((C des) invocation)) "] - [T : "fonctions"] -
[T\T : "distantes"]
[N: " ((B ((C d)(universel protocole)) ((C des) invocation)) fonctions) "] - [T\T : "distantes"]
[N/T : " (B ((C d)(universel protocole)) ((C des) invocation)) "] - [T : "fonctions"] -
[T\T : "distantes"]
[N/T : " (B ((C d)(universel protocole)) ((C des) invocation)) "] - [T : " (distantes fonctions) "]
[N : " ((B ((C d)(universel protocole)) ((C des) invocation)) (distantes fonctions)) "]

Réduction des combinateurs

"((B ((C d) (universel protocole)) ((C des) invocations)) (distantes fonctions))"
"(((C d) (universel protocole)) (((C des) invocations) (distantes fonctions)))"
"((d (((C des) invocations) (distantes fonctions))) (universel protocole))"
"((d ((des (distantes fonctions)) invocations)) (universel protocole))"

Le troisième terme analysé est du groupe nominal, il est donc préservé par notre filtre. La réduction des combinateurs nous donne la forme normale :

"((d ((des (distantes fonctions)) invocations)) (universel protocole))"

Analyse du terme 4

[N : "position"] - [N\N : "relative"] - [N\N/N : " de"] - [N/T : "la"] - [T : "zone"] -[T\T/T : "de"] - [T/T : "l "] - [T : "écran"]
[N : " (relative position) "] - [N\N/N : "de"] - [N/T : "la"] - [T : " zone"] - [T\T/T : "de"] - [T/T : "l"] - [T : "écran"]
[N : " (relative position) "] - [N/N\N : " (C de) "] - [N/T : "la"] - [T : "zone"] - [T\T/T : "de"] - [T/T : "l"] - [T : "écran"]
[N/N: " ((C de) (relative position)) "] - [N/T : "la"] - [T : "zone"] - [T\T/T : "de"] - [T/T : "l"] - [T : "écran"]
[N/T: " (B ((C de) (relative position)) la) "] - [T : "zone"] - [T\T/T : "de"] - [T/T : "l "] - [T : "écran"]
[N: " (B ((C de) (relative position)) la) zone) "] - [T\T/T : "de"] - [T/T : "l"] - [T : "écran"]
[N/T: " (B ((C de) (relative position)) la) "] - [T : "zone"] - [T\T/T : "de"] - [T/T : "l"] - [T : "écran"]
[N/T: " (B ((C de) (relative position)) la) "] -[T : "zone"] - [T/T\T : " (C de) "] - [T/T : " l"] - [T : "écran"]
[N/T: " (B ((C de) (relative position)) la) "] - [T/T : " ((C de)zone) "] - [T/T : "l"] - [T : " écran"]
[N/T : " (B (B ((C de) (relative position)) la) ((C de)zone)) "] - [T/T : "l"] - [T : "écran"]
[N/T : " (B (B (B ((C de) (relative position)) la) ((C de)zone)) l) "] - [T : "écran"]
[N : " ((B (B (B ((C de) (relative position)) la) ((C de)zone)) l) écran) "]

Réduction des combinateurs

"((B (B (B ((C de) (relative position)) la) ((C de) zone)) l) écran)"
"((B (B ((C de) (relative position)) la) ((C de) zone)) (l écran))"
"((B ((C de) (relative position)) la) (((C de) zone) (l écran)))"
"(((C de) (relative position)) (la (((C de) zone) (l écran))))"
"((de (la (((C de) zone) (l écran)))) (relative position))"
"((de (la ((de (l écran)) zone))) (relative position))"

Ce terme candidat formé de huit termes est aussi validé par notre filtre car l'analyse syntaxique nous donne une expression applicative de type N. Après réduction des combinateurs, nous aboutissons à cette forme normale :

"((de (la ((de (l écran)) zone))) (relative position))"

Analyse du terme 5

[N : "langage"] - [N\N/N : "de"] - [N : "définition"] - [N\N/T : "de"] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] -[T : "affaire"]
[N : "langage"] - [N/N\N : " (C de) "] - [N : "définition"] - [N\N/T : "de"] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N/N : " ((C de) langage) "] - [N : "définition"] - [N\N/T : "de"] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N: " (((C de) langage) définition) "] - [N\N/T : "de"] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N: " (((C de) langage) définition) "] - [N/T\N : " (C de) "] - [T : "modèles"] - [T\T/T : "de"] – [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N/T: " ((C de) (((C de) langage) définition)) "] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N: " ((C de) (((C de) langage) définition)) modèles) "] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N/T: " ((C de) (((C de) langage) définition)) "] - [T : "modèles"] - [T\T/T : "de"] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N/T: " ((C de) (((C de) langage) définition)) "] - [T : " modèles"] - [T/T\T : " (C de) "] - [T : "processus"] - [T\T/T : "d"] - [T : " affaire"]
[N/T: " ((C de) (((C de) langage) définition)) "] - [T/T: " ((C de)modèles) "] - [T : "processus"] - [T\T/T : "d "] - [T : "affaire"]
[N/T: " (B ((C de) (((C de) langage) définition)) ((C de)modèles)) "] - [T : "processus"] - [T\T/T : "d"] - [T : "affaire"]
[N: " ((B ((C de) (((C de) langage) définition)) ((C de)modèles)) processus) "] - [T\T/T : "d"] - [T : "affaire"]
[N/T: " (B ((C de) (((C de) langage) définition)) ((C de)modèles)) "] - [T : "processus"] - [T/T\T : " (C d) "] - [T : "affaire"]
[N/T: " (B ((C de) (((C de) langage) définition)) ((C de)modèles)) "] - [T/T : " ((C d) processus) "] - [T : "affaire"]
[N/T: " (B (B ((C de) (((C de) langage) définition)) ((C de)modèles)) ((C d) processus)) "] - [T : "affaire"]
[N: " ((B (B ((C de) (((C de) langage) définition)) ((C de)modèles)) ((C d) processus)) affaire) "]

Réduction des combinateurs

"((B (B ((C de) (((C de) langage) définition)) ((C de) modèles)) ((C d) processus)) affaire)"
"((B ((C de) (((C de) langage) définition)) ((C de) modèles)) (((C d) processus) affaire))"
"(((C de) (((C de) langage) définition)) (((C de) modèles) (((C d) processus) affaire)))"
"((de (((C de) modèles) (((C d) processus) affaire))) (((C de) langage) définition))"
"((de ((de (((C d) processus) affaire)) modèles)) (((C de) langage) définition))"
"((de ((de ((d affaire) processus)) modèles)) (((C de) langage) définition))"
"((de ((de ((d affaire) processus)) modèles)) ((de définition) langage))"

Les plus longs termes candidats que nous avons traités tout au long de notre travail étaient composés de neuf termes. Ce terme traité ci-dessus en est un. Il est validé par notre filtre. Nous arrivons à la fin de l'analyse syntaxique à l'expression applicative :
[N: " ((B (B ((C de) (((C de) langage) définition)) ((C de)modèles)) ((C d) processus)) affaire) "].

Après la réduction des combinateur nous obtenons la structure :
"((de ((de ((d affaire) processus)) modèles)) ((de définition) langage))".

7.3 Conclusion

Nous venons de présenter à travers ce chapitre les résultats d'analyse de termes candidats de notre corpus. Nous avons effectué une évaluation sur 100 termes candidats. 96 % des termes candidats ont été analysés avec succès ; ils ont été validés par notre filtre linguistique. 4 % des termes ont été rejetés. Les termes rejetés sont ceux ayant l'agencement de type "Nom – Nom". Notre analyseur syntaxique considère ces termes comme syntaxiquement incorrects et les rejette. Pour remédier à ce genre de problème, nous avons considéré le deuxième terme comme modifieur du premier. Ce qui nous a donné l'agencement "Nom - adjectif" qui est accepté par notre analyseur syntaxique.
Nous allons consacrer le chapitre suivant à la conclusion de ce mémoire.

CHAPITRE 8

CONCLUSION

Nous avons présenté un filtre linguistique dont le but est l'identification des termes complexes. Ce filtre linguistique est fondé sur un modèle de Grammaire Catégorielle Combinatoire Applicative. C'est grâce à ce modèle catégoriel que nous avons pu identifier les termes complexes à partir d'une liste des termes candidats. Une analyse syntaxique des formes phénotypiques qui est obtenue par un calcul sur les types syntaxiques, nous a permis de vérifier si un terme candidat est du groupe nominal. Ce sont les termes candidats ayant comme catégorie le groupe nominal qui sont préservés par notre filtre linguistique. L'expression applicative obtenue après l'analyse syntaxique nous a donné après réduction des combinateurs la forme normale du terme complexe. Les résultats du traitement des termes complexes sont stockés dans une base de données XML. XML offre des possibilités intéressantes pour des manipulations ultérieures de ces résultats par des requêtes XQuery (XML Query).

Cette approche théorique a été implémentée en C++ pour un corpus important du français. Notre but est théorique. Nous voulons seulement prouver que toutes les règles que nous proposons sont efficaces et peuvent être implémentées dans un langage de programmation.

Nous avons effectué une évaluation sur 100 termes candidats. 96% des termes candidats ont été analysés avec succès ; ils ont été validés par notre filtre linguistique. 4% des termes ont été rejetés par notre filtre linguistique. La langue française permet l'agencement des termes : "Nom - Nom". Notre analyseur syntaxique considère ces termes comme syntaxiquement incorrects et les rejette. Nous avons résolu ce problème en considérant le deuxième terme comme modifieur du premier. Une autre possibilité pour résoudre ce genre de problème est de coupler la Grammaire Catégorielle Combinatoire Applicative avec une approche statistique. Cette dernière serait capable de reconnaître les structures de type "Nom - Nom".

Le plus significatif dans nos résultats est incorporé, non seulement dans la validation des termes complexes mais particulièrement dans leur structure fonctionnelle. Celle-ci permet de construire l'environnement de l'utilisation d'un concept de base représenté par un mot. Dans les exemples suivants analysés :

- Théorie complexe.
- Théorie des nombres.
- Théorie des nombres complexes.

Le mot *Théorie* est vue modifié par plusieurs expressions qui agissent en tant qu'adjectif et qui déterminent le contexte de son utilisation. Nous pouvons, de cette façon, systématiser, par exemple, l'extraction des relations qui pourraient généraliser le concept d'hyperonymie ou d'hyponymie. Avec cette intention, la gestion complète d'un graphique sémantique devient nécessaire. Dans un tel graphique les nœuds peuvent représenter des mots et les bords peuvent contenir des dispositifs pour classer les relations entre les nœuds.

Notre filtre linguistique est différent de la plupart des autres filtres linguistiques d'identification des termes complexes, parce que :

- Il tend à être multilingue. Avec la croissance du Web et des bases de données textuelles multilingues, cet aspect est significatif. Tout ce que nous avons besoin pour adapter l'approche à une nouvelle langue est un dictionnaire des types catégoriels avec les entrées lexicologiques de cette langue.
- Une théorie logique et linguistique solide soutient l'approche. Cette théorie est particulièrement flexible. Dans certains cas, les termes complexes peuvent être ceux ayant la catégorie grammaticale de l'expression verbale. Il serait assez alors de considérer que les termes complexes à valider ont les types catégoriels spécifiques aux expressions verbales

Avant de terminer nous tenons à informer le lecteur que notre travail a été présenté à la conférence internationale de Melbourne Beach, Floride, et a été apprécié par les participants. Il a aussi été publié en 2006: "**BISKRI, I.**, MUNYANA, N., &

HAMROUNI, B. 2006, "Annotation of the complex terms in multilingual corpora", in Proc. of FLAIRS 2006. Melbourne Beach. Florida. AAAI Press".

ANNEXE 1

Le corpus

1. base relationnelle
2. base de données
3. base de données relationnelles
4. nombre complexe
5. théorie de nombres complexes
6. base de la théorie des nombres
7. validation de termes complexes
8. calcul déductif des termes candidats
9. unités linguistiques
10. unités linguistiques concaténées
11. traitement de langue naturel
12. forme normale
13. règle de permutation
14. utilisation de la règle de permutation
15. règles combinatoires applicatives
16. règles d'application fonctionnelle
17. structure applicative
18. grammaire catégorielle
19. modèle de grammaire catégorielle combinatoire applicative
20. type catégoriel spécifique
21. langage de programmation
22. programmation procédurale
23. Déclaration d'une notation
24. compilation d'un programme
25. type de données

26. fonctions virtuelles
27. fonction membre virtuelle
28. fonction de tri
29. fonction de recherche
30. fonction membre d'une classe
31. nombreuses versions d'une fonction
32. mécanisme des méthodes virtuelles
33. méthodes publiques
34. déclaration du destructeur
35. déclaration de la classe
36. déclaration du destructeur de la classe de base
37. gestion des exceptions
38. valeurs d'attributs légales
39. valeur d'une variable
40. augmentation de la valeur d'une variable
41. surcharge des opérations
42. opérateur d'incrémentation
43. utilisation des nombreuses bibliothèques existantes
44. bibliothèque standard
45. commentaire de fin de ligne
46. espaces de noms
47. pointeur d'instance
48. pointeur d'instance de la classe dérivée
49. environnements de développement
50. chaînes de caractère
51. intégration de données
52. urbanisation des systèmes d'information modernes
53. architecture multiniveaux
54. architecture de système
55. revue de technologies
56. technologies de génération de pages dynamiques

57. modélisation multidimensionnelle des données
58. spécifications d'échanges
59. connexion des applications
60. téléchargement d'application portable
61. interface utilisateur
62. exécution du code applicatif
63. techniques de base
64. scénario typique de fonctionnement
65. position relative de la zone de l'écran
66. zone d'affichage
67. affichage de l'objet
68. largeur de la zone d'affichage
69. langage de description de données
70. langage d'échange de données
71. langage d'écriture de feuille de style
72. langage de transformation des documents
73. langage d'orchestration de services
74. langage de définition de modèles de processus d'affaire
75. nombre de caractéristiques optionnelles
76. variété de types d'éléments significatifs
77. liste de principaux types de nœuds
78. principales classes d'un parseur
79. évaluation des expressions de chemin
80. ensemble de déclaration d'espace de noms
81. protocole universel d'invocations des fonctions distantes
82. liens hypertextes multivalués
83. processus unique
84. processus d'affaire
85. nouvelles instances du processus
86. séquences de processus d'affaire
87. coordination de processus d'affaires collaboratifs

88. transaction d'affaires
89. création des programmes spécialisés
90. support de modèles de transaction
91. ensemble d'action atomique
92. problème de fiabilité
93. format de données riche
94. recherche d'information
95. arbre binaire
96. activité de production de message
97. assignation de variables globales
98. définition de flux de contrôle
99. structure de l'élément
100. schéma de collaboration

BIBLIOGRAPHIE

AJDUKIEWICZ, K., (1935). "Die syntaktishe Konnexität", Studia philosophica, vol. 1, 1-27.

ADES, A., STEEDMAN, M., (1982). "On the order of words" Linguistics and Philosophy , 4, 517-558.

BAR-HILLEL, Y., (1953). "A quasi-arithmetical notation for syntactic description", Language 29, 47-58.

BISKRI, I., (1995). La Grammaire Catégorielle Combinatoire Applicative dans le cadre de la Grammaire Applicative et Cognitive, Thèse de Doctorat, EHESS, Paris.

BISKRI, I., DESCLES, J. P., (1997). "Applicative and Combinatory Categorial Grammar (from syntax to functional semantics)", in Recent Advances in Natural Language Processing (selected Papers of RANLP 95) Ed. Ruslan Mitkov & Nicolas Nicolov. John Benjamins Publishing Company,
Numéro 136, pages 71-84.

BISKRI, I., MEUNIER, J. G., JOYAL, S., (2004). "L'extraction des termes complexes : une approche modulaire semi-automatique" Dans Le Poids des mots (Actes des 7èmes Journées Internationales d'Analyse Statistique des Données Textuelles, Louvain-La-Neuve, Belgique), Gérard Purnelle, Cédrick Fairon & Anne Dister (eds). Presses Universitaires de Louvain, Volume 1, pp 192-
201, ISBN 2-930344-49-0.

BISKRI, I., MUNYANA, N., HAMROUNI, B. (2006). "Annotation of the complex terms in multilingual corpora", in Proc. of FLAIRS 2006. Melbourne Beach. Florida. AAAI Press".

CURRY, B. H., FEYS, R., (1958). Combinatory logic , Vol. I, North-Holland.

DAILLE, B. (1994). "Study and Implementation of Combined Techniques for Automatic Extraction of Terminology", Proceedings of the Combining Symbolic and Statistical Approaches to Language Workshop (the Balancing Act), Las Cruces (New Mexico), USA, 1 st July 1994, 29--36.

DESCLES, J. P., (1990). Langages applicatifs, langues naturelles et cognition, Hermes, Paris.

DESCLES, J. P., SEGOND, F., (1990). "Topicalization : categorial analysis and applicative grammar" dans Lecomte 1992, L'ordre des mots dans les grammaires catégorielles, 13-37.

DESCLÈS, J. P., (1991). "Grammaire Applicative et Cognitive et logique combinatoire", T.A.Informations, 5-12.

DESCLÉS, J. P., (1996). Cognitive and Applicative Grammar: an Overview. in C. Martin Vide, ed. Lenguajes Naturales y Lenguajes Formales, XII, Universitat Rovra i Virgili. , 29-60.

GARDARIN, G. (2002). XML : Des bases de données aux services Web, Dunod, Paris.

GEACH, P., (1971). "A Program for Syntax", Synthese, 22, 3-17.

HADDOCK, E. KLEIN, et G. MORILL, (1987). Working papers in cognitive science volume I : Categorial Grammar, Unification Grammar and Parsing, Edinburgh Univ..

HUSSERL, E., (1913). Logische Untersuchungen, Max Niemeyer, Halle.

LAMBEK, J., (1958). "The Mathematics of Sentence Structure", American Mathematical Monthly, 65, 154-165.

LAMBEK, J., (1961). "On the calculus syntactic types" Proceeding of symposia in Applied Mathematics, vol. XII, America Mathematical Society, Providence, Rhode Island, 166-178.

LESNIEWSKI, S.T., (1989). Sur les fondements de la mathématique. Fragments (Discusions préalables, méréologie, ontologie). Traduit du polonais par Georges Kalinowski, Hermès.

PARESCHI, R., STEEDMAN, M., (1987). "A lazy Way to chart parse with categorial grammars" Acte du colloque ACL, Stanford, 1987.

SHAUMYAN, S. K., (1977). Applicational Grammar as a Semantic Theory of Natural Language, Edimburgh Univ. Press.

SHAUMYAN, S. K., (1982). "The goals of linguistic theory and applicative grammar", Mathématiques et sciences humaines, Tome I , 7-42.

SHAUMYAN, S. K., (1987). A Semiotic Theory of Natural Language, Bloomington, Indiana Univ. Press.

SHAUMYAN, S. K., (1998). Two Paradigms Of Linguistics: The Semiotic Versus Non-Semiotic Paradigm. In Web Journal of Formal, Computational and Cognitive Linguistics.

SCHÖNFINKEL, M., (1967) "On the building blocks of mathematical logic" dans J. Van Heijenoort, From Frege to Gödel : A source book in mathematical logic 1879-1931, Harvard Uni. Press, 355-366.

STA, J.D. (1998). "Automatic acquisition of terminological relations from a corpus for query expansion". Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, p.371-372, August 24-28, 1998, Melbourne, Australia

STEEDMAN, M., (1988). "Combinators and Grammars", dans Oehrle et alii, 1988, 207-263.

STEEDMAN, M., (1989), "Constituency and coordination in a combinatory grammar" dans M. BALTIN and T. KROCH, Alternative conceptions of phrase structure, University of Chicago press, 201-231.

STEEDMAN, M., (1989). Work in progress: Combinators and grammars in natural language understanding, Summer institute of linguistic, Tucson University.

STEEDMAN, M. (2000). The Syntactic Process, MIT Press/Bradford Books.

SZABOLCSI, A., (1987). "On combinatory categorial grammar", Acte du Symposium on logic and languages, Debrecen, Akademiai Kiado, Budapest, 151-162.

TEMPLEMAN, J. , OLSEN, A. (2002). Visual C++. Net étape par étape, Microsoft Corporation, Washington

YOUNG, M. J., (2001). XML étape par étape, Microsoft Press, 337-344

Les sites Web consultés

DAILLY, N. Tri par arbre binaire de recherche, http://www.dailly.info/algorithmes-de-tri/abr.php
(consulté le 11 Mars 2007)

GARRETA, H. Le langage C++ http://www.dil.univ-mrs.fr/~garreta/Polys/PolyCpp.pdf
(consulté le 14 Juillet 2007)