Université Paris VII
Centre d'Etudes et de Recherches en Informatique Linguistique
Laboratoire d'Automatique Documentaire et Linguistique

Thèse de doctorat en Informatique Fondamentale

Transformations des noms composés.

Christian JACQUEMIN

Jury:
Andrée BORILLO
Alain COLMERAUER (rapporteur)
Maxime CROCHEMORE (rapporteur)
Maurice GROSS

Université Paris VII
Centre d'Etudes et de Recherches en Informatique Linguistique
Laboratoire d'Automatique Documentaire et Linguistique

Thèse de doctorat en Informatique Fondamentale

Transformations des noms composés.

Christian JACQUEMIN

Jury:
Andrée BORILLO
Alain COLMERAUER (rapporteur)
Maxime CROCHEMORE (rapporteur)
Maurice GROSS

Je tiens, tout d'abord, à remercier le Professeur Maurice Gross, qui a accepté de diriger ce travail de thèse. Je lui suis reconnaissant des conseils et remarques pertinentes qui m'ont permis de progresser.

Je remercie Max Silberztein et Maxime Crochemore pour leurs remarques après la lecture de ce mémoire.

Je remercie le Professeur Gaston Gross pour les discussions que nous avons eues sur les noms composés.

Je remercie le Professeur Alain Colmerauer pour m'avoir aidé à la mise en forme et pour m'avoir conseillé sur les parties informatiques et mathématiques de ce travail.

Liana Popesco m'a soutenu tout au long de ce travail, son regard extérieur a été bénéfique dans l'évolution de celui-ci. Je la remercie également de m'avoir présenté à l'IUT de Poitiers, ce qui m'a permis d'y obtenir un poste d'ATER par la suite.

Je remercie tous ceux de l'IUT de Poitiers, administratifs et enseignants, qui m'ont accompagné dans ce travail et qui ont été très compréhensifs: Bernard Bilhou, Dominique Brunet, Jean Louis Décherat, Lionel Grignard. Que ceux que j'oublie ne s'en formalisent pas.

Je remercie également Marie-Joëlle Blosseville et Nadine Penot de la Direction des Etudes et Recherches de l'EDF, pour la qualité du travail que j'ai pu faire dans leur laboratoire et pour m'avoir montré l'aspect professionnel de la recherche documentaire.

Je remercie Marie-Pierre pour sa lecture attentive et pour ses corrections.

TABLE DES MATIERES

INTRODUCTION

Objet de l'étude

L'élaboration de lexiques-grammaires est une nécessité pour améliorer la qualité d'un analyseur. Il faut décrire finement les comportements syntaxiques associés aux entrées lexicales.

L'objet de ce travail a été de réaliser un analyseur de noms composés de la langue française. Nous nous sommes limités aux noms composés de structure nominale: `N Adj`, `N de N`, `N N` et `N Prép N` essentiellement. Ils sont fréquents dans le corpus de textes techniques que nous avons traité (par exemple: *puissance électrique, courant d'air, bloc moteur...*).

La reconnaissance morphologique des noms composés ayant déjà été traitée par ailleurs, nous avons mis l'accent sur l'analyse de noms composés ayant subi des transformations Les principales transformations observées sont: les modifications adjectivale ou adverbiale, la coordination, la nominalisation et l'adjectivation (par exemple: *générateurs et échangeurs de chaleur*).

Cette étude a permis de créer des outils informatiques de reconnaissance des noms composés réalisant les tâches suivantes:

(1) analyser les mots simples et les noms composés fléchis dans un texte,

(2) analyser les noms composés ayant subi des transformations.

Le point (1) motive la première partie de notre travail qui est axée sur le lexique et la reconnaissance morphologique. Afin de prendre en compte les gros volumes de données de la langue, nous décrivons les modes d'accès aux données (B-Arbre et rattachement lexical). Afin de proposer un formalisme lisible de saisie, nous développons un sous-ensemble de Prolog avec une unification dédiée à la langue.

La deuxième partie, la plus importante, répond au point (2). Afin de décrire les transformations sur les noms composés et sur les imbrications de noms composés, nous élaborons un modèle de la langue à trois niveaux. Le parallèle entre les propriétés logico-mathématiques de la langue et les observations linguistiques montre toutes les difficultés que présente une entreprise de formalisation. Un mécanisme de métarègles permet de mettre en oeuvre l'analyse automatique des transformations de noms composés.

Cette étude est accompagnée de deux développements informatiques:

(A) un compilateur de règles et de métarègles de noms composés, un analyseur / générateur s'appuyant sur les règles et leurs transformations par les métarègles,

(B) un algorithme approché de lexicalisation d'un dictionnaire de mots composés qui les rattache, de façon homogène, à un des mots simples qui les composent.

Présentation détaillée

La première partie est la réalisation d'un lexique formé de mots simples et de noms composés, où les noms composés sont regroupés en grappes autour des mots simples.

Le lexique des mots simples

Après avoir proposé un formalisme d'énonciation, nous détaillons l'accès aux mots simples par un B-Arbre de taille paramétrable, adapté à la recherche d'un mot dans un dictionnaire par sa chaîne de caractères. Les mots étant stockés sous leur forme canonique (chaîne maximale commune aux différentes flexions), les mots lus par l'analyseur doivent être défléchis. Un arbre lexicographique généralisé formé des suffixes de la langue permet de réaliser cette tâche.

Le lexique des noms composés

Pour justifier la notion de nom composé, nous montrons, par exemple, que *quantité de* $\mathcal{N}$s'analyse comme un syntagme nominal dans le cas général, et comme un nom composé pour *quantité de mouvement*.

En reprenant les principes de la programmation en logique, nous proposons une grammaire de description des noms composés. L'unification est orientée vers le traitement du langage naturel, elle offre les trois possibilités suivantes:

- contrainte: égalité d'une variable et d'une valeur,
- substitution: remontée de l'unification dans l'arbre d'analyse,
- accord: unification de deux variables dans un même nom composé.

Le compilateur permet de stocker les structures arborescentes des règles sur fichier et l'analyseur de reconnaître les noms composés fléchis dans les textes. Ils constituent l'application informatique (A) qui sera enrichie dans la deuxième partie.

La lexicalisation des mots composés sur les mots simples

Les mots composés sont regroupés (lexicalisés) autour des mots simples qu'ils contiennent, pour limiter le chargement des règles en analyse. Nous proposons un algorithme approché permettant d'obtenir, en un temps faiblement polynômial, une répartition homogène des mots composés sur le lexique des mots simples. La qualité de cet algorithme est évaluée, lorsque le lexique satisfait à certaines conditions.

Il a donné lieu au développement informatique (B) afin de le tester sur des ensembles de données.

La deuxième partie est la mise en oeuvre d'un lexique-grammaire pour l'analyse des noms composés ayant subi des transformations.

L'étude des noms composés

La description des formes figées du langage naturel peut être structurée en trois niveaux:

- les règles qui décrivent les noms composés,
- les métarègles de base qui permettent de produire de nouvelles règles à partir des précédentes,
- les métatransformations qui servent à enrichir les métarègles à l'aide de métarègles déjà existantes.

Cette partie débute par une typologie des transformations acceptées par les noms composés, qui est reprise dans la modélisation au moyen de métarègles. Trois catégories de transformations apparaissent. Celles de la catégorie 1 et 3 sont décrites par des métarègles ordinaires, celles de la catégorie 2 par des métarègles paramétrables, pour indiquer les éléments nouveaux qu'elles introduisent.

Les noms composés peuvent être assemblés pour créer de nouvelles formes figées appelées noms composés d'ordre supérieur. Deux mécanismes de construction sont utilisés principalement: la juxtaposition ou le recouvrement. Afin de rendre compte des interactions des transformations sur ce type de structure, et pour réduire la gamme des transformations, nous proposons quatre métatransformations:

- deux métatransformations de composition `[antéposition]` et `[connexe]` qui enrichissent les transformations sur une structure,
- `[extension]` qui induit une transformation sur un nom composé d'ordre supérieur à partir de celles sur un nom composé de base,
- `[croisement]` qui permet de croiser les transformations sur un nom composé où sont imbriqués d'autres noms composés.

La réalisation informatique

Afin de mettre en oeuvre l'organisation observée ici, nous avons réalisé un compilateur de métarègles. Un générateur permet de produire de nouvelles règles de nouvelles règles au moyen de métarègles (ou d'élisions). Il enrichit l'application informatique (A), qui permet alors d'analyser les noms composés ayant subi des transformations.

I MORPHOLOGIE DES NOMS COMPOSÉS

Pour pouvoir traiter les transformations sur les noms composés (la deuxième partie de ce travail), il est nécessaire de partir d'un lexique qui décrive les noms composés sous leur forme de base. Ce lexique doit également contenir une description des noms simples qui entrent dans la formation de ces noms composés. La première partie de ce travail présente l'architecture de ce dictionnaire de mots simples et de noms composés, ainsi que les outils de compilation et de reconnaissance morphologique.

Les trois point importants de cette partie sont les suivants:

- Les données lexicales sont volumineuses, donc l'optimisation de l'accès au lexique nécessite un développement adapté. Nous avons choisi d'utiliser **un B-Arbre statique**, orienté vers le problème de l'accès lexical. (I.1.b)
- Nous souhaitons proposer un formalisme d'énonciation des noms composés qui soit simple à utiliser et qui rende compte de la diversité des cas rencontrés. Pour cela, nous avons adapté le mécanisme général de **l'unification** tel qu'il existe dans le langage Prolog, à la représentation de cette partie du langage naturel. (I.2.b et e)
- Pour limiter le nombre de formes figées candidates à être reconnues dans une phrase, les mots composés sont rattachés aux mots simples (**la lexicalisation**). Les contraintes physiques du système informatique nous ont obligé à les répartir de la façon la plus homogène possible. Le problème à résoudre est NP-Complet, nous en donnons un algorithme approché. (I.3)

1 MOTS SIMPLES

a Formalisme

Nous présentons, dans la figure 1, le formalisme qui permet de décrire les mots simples dans le système informatique.

La **forme canonique** d'un mot est la partie commune maximale aux différentes flexions qu'il accepte. Ainsi, la forme canonique associée à toutes les flexions de l'adjectif national (*nationa-l*, *nationa-le*, *nationa-ux* et *nationa-les*) est *nationa*.

Certains verbes, qui admettent une grande diversité de flexions, tels que *savoir*, ont plusieurs formes canoniques qui correspondent chacune à une liste de flexions.

Nous ne détaillons pas cette présentation qui est uniquement un support à la suite de l'exposé. Le début de la liste de mots simples est donnée en annexe III.1.

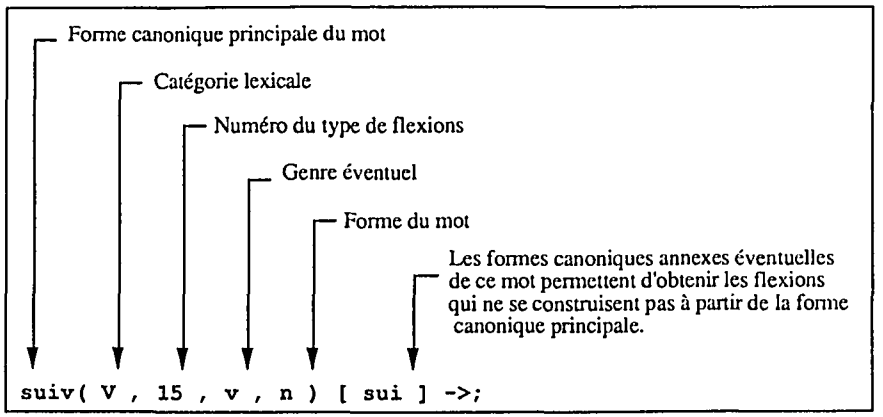


Figure 1: description des mots simples

b Accès aux mots simples par B-Arbre

Pour gérer l'accès rapide aux données, avec un minimum de place mémoire, les B-Arbres constituent des outils bien adapté. On a le choix entre un B-Arbre de taille variable, rééquilibrable en fonction des ajouts, mais dans lequel les pointeurs doivent être explicites; et un B-Arbre de taille fixe, mais sans pointeur, donc plus petit pour la même capacité.
La première solution est intéressante si on ajoute ou supprime fréquemment des données. Or ces opérations sont rares sur de gros lexiques, car l'insertion n'est pas interactive. Elle se fait par lots et permet de prendre le temps de recalculer entièrement le B-Arbre.
La deuxième solution économise la place mémoire, nous l'avons donc retenue. L'accès au lexique-grammaire se fait par un B-Arbre de taille fixe, paramétrable.[Aho 83]

Nous détaillons ici le mécanisme de construction puis de recherche dans le B-Arbre. Il est spécifiquement adapté à l'accès à un lexique par une chaîne de caractère. Nous précisons comment nous le parcourons en ne regardant qu'une partie de la chaîne, pour que les noeuds du B-Arbre ne stockent que des chaînes partielles. Nous utilisons pour cela la notion de **Rang de la Première Lettre Discriminatoire** entre deux chaînes données: l'indice de la première lettre distincte.

Les paramètres de l'arbre sont:
- la profondeur **p**,
- la taille **n** de chaque noeud: chaque noeud est composé de **n** "portes" permettant d'accéder aux noeuds de la profondeur supérieure,
- le nombre **d** de lettres discriminatoires retenues.

Le pointage entre les noeuds se fait de la façon suivante:
- la $i^{ème}$ porte de la racine pointe vers la première porte du $(i + 1)^{ème}$ noeud de profondeur 2

(comme l'indiquent les traits fins de la figure 2),
- la $i^{ème}$ porte du $j^{ème}$ noeud de profondeur 2 pointe vers la première porte du noeud $(j - 1) \times (n + 1) + i$ de profondeur 3... etc.

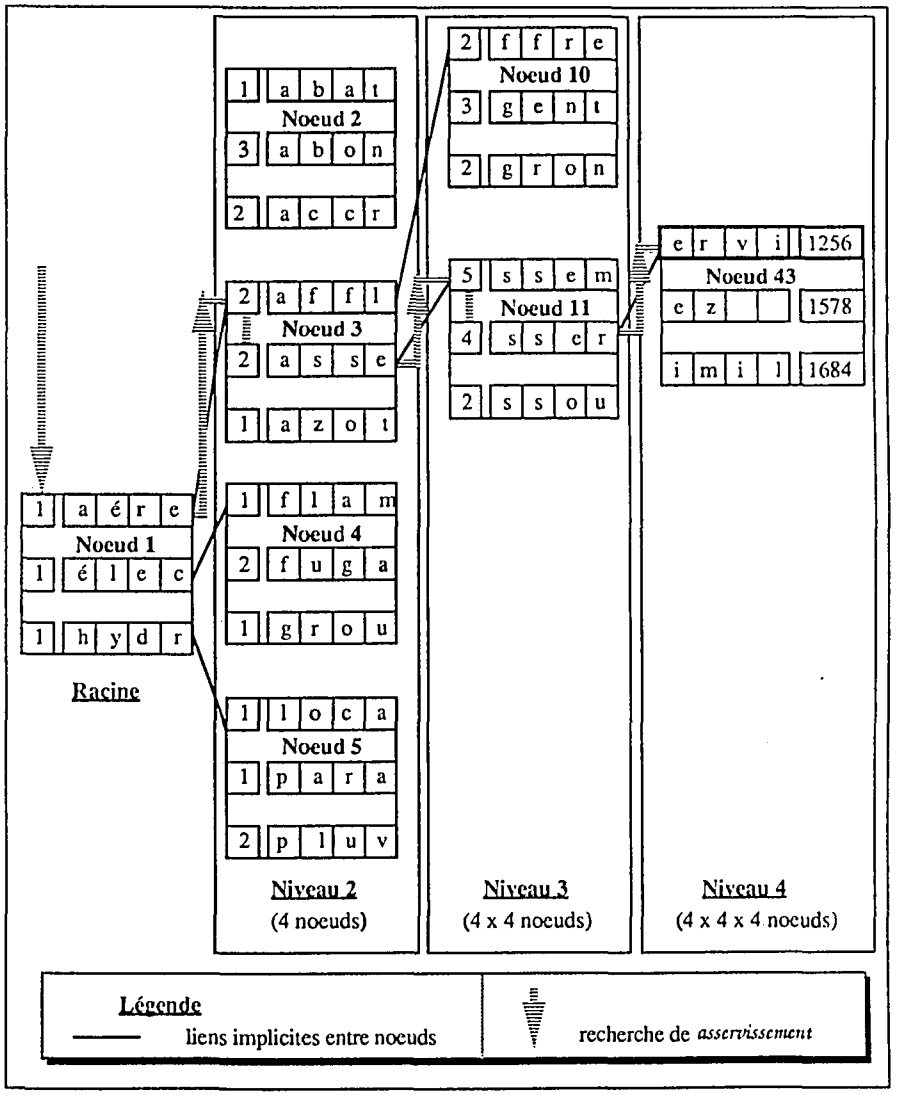


Figure 2: partie de B-Arbre de profondeur 4, avec des noeuds de 3 portes et 4 lettres discriminatoires.

Composition du B-Arbre

Deux mots formés de caractères ASCII étant donnés, on appelle rang de la première lettre discriminatoire (**RPLD**) pour ces deux mots, l'indice du premier caractère distinct: par exemple le **RPLD** de *assemblée* et de *assembler* est **8**.

Les **p - 1** premiers niveaux sont formés de noeuds intermédiaires de **n** portes (voir les noeuds 1 à 5 et 10 et 11 de la figure 2). Chaque porte est composée:

- de **d** caractères ASCII servant à aiguiller la recherche par comparaison entre le mot cherché et ces **d** caractères,
- d'un indice qui sera le **RPLD** des mots du noeud pointé au niveau supérieur.

Le niveau **p** est formé de noeuds terminaux de **n** portes, où le **RPLD** est remplacé par un pointeur dans le fichier du lexique (voir le noeud 43 de la figure 2).

Economie d'une porte par noeud de l'arbre

Les portes d'un noeud sont les bornes d'un intervalle fermé à gauche et ouvert à droite. Par exemple la racine (noeud 1 de la figure 2) détermine quatre segments:

['', *'aére'* [; [*'aére'*, *'élec'* [; [*'élec'*, *'hydr'* [; [*'hydr'*, *'zzzz'* [.

Si l'on parcourt le B-Arbre, on va pointer :

- vers la première porte du premier noeud du niveau **2**, avec un mot inférieur strict à *'aére'*,
- vers la première porte du deuxième, avec un mot supérieur ou égal à *'aére'* et inférieur strict à *'élec'*...
- Si c'est un mot supérieur ou égal à *'hydr'* on pointera vers la première porte du quatrième noeud. On s'aperçoit donc que, à l'aide de trois portes, on définit quatre segments. On pointe donc vers quatre noeuds de niveau **2**.

De même, le deuxième noeud du niveau **2** du B-Arbre (noeud 3 de la figure 2) est composé de trois portes qui déterminent les quatre segments suivants:

[*' aére'*, *'affl'* [; [*'affl'*, *'asse'* [; [*'asse'*, *'azot'* [; [*'azot'*, *' élec'* [; les deux extrémités: *'aére'* et *'élec'* étant héritées des portes du noeud père.

En réitérant ce raisonnement sur tout noeud de trois portes qui n'est pas la racine, on peut considérer que les quatre intervalles qu'il définit sont bornés à gauche, par la valeur de la porte qui pointe vers ce noeud. Ils sont bornés à droite par la valeur de la porte immédiatement supérieure. Cette définition est récursive descendante et s'arrête au niveau **0** avec deux portes fictives " " et *'zzzzzz'*

On en déduit que la taille du niveau de profondeur **i** du B-Arbre est de $(n + 1)^{i-1}$; ce qui donne une taille totale pour le B-Arbre de:

$(n + 1)^0 + (n + 1)^1 + ... + (n + 1)^{p-1} = ((n + 1)^p - 1) / n$

Du point de vue informatique, les portes des noeuds de l'arbre sont représentées dans un tableau unique. La dernière porte du noeud numéro **i** de niveau **j** précède la première porte du noeud numéro **(i + 1)** du niveau **j**, s'il existe, ou la première porte du noeud numéro **1** du niveau **(j + 1)** sinon.

Le rôle du rang de la première lettre discriminatoire

Afin de ne pas surcharger le B-Arbre en stockant la totalité des mots dans les portes, on a choisi de ne voir qu'une partie des mots: les **d** premières lettres discriminatoires. Or, à partir d'une certaine profondeur dans l'arbre, les mots des portes d'un noeud ont un certain nombre de lettres initiales en commun, il est donc inutile de faire porter la discrimination sur ces lettres communes.

Dans le noeud 3 de la figure 2, les portes **1** et **2** correspondent aux mots *affliger* et *assez*. Ces mots ont leur première lettre commune. On est sûr que tous les mots du lexique correspondant au noeud fils du segment entre les portes **1** et **2** auront un *a* en première lettre. Afin de rendre la discrimination plus pertinente, pour toutes les portes du noeud fils, on ne verra les mots qu'à partir de la lettre numéro **2** (le **RPLD**). Le noeud fils est représenté par le noeud 10 de la figure 2. Le **RPLD** de la deuxième porte est **3** puisque *agent* et *agronomie* ont les deux premières lettres communes.

Construction du B-Arbre

Le B-Arbre est construit à l'aide d'une procédure récursive.

Au préalable, on borne le lexique à l'aide d'un mot minimal " " et d'un mot maximal *'zzzzzzzz'*, afin que l'on puisse utiliser ces valeurs comme limites du lexique à l'appel de niveau **1**.

Au niveau **1**, on ordonne le lexique selon un ordre alphabétique choisi et on le partage en **n + 1** parties égales. On rattache virtuellement la $i^{ème}$ porte au premier mot de la $(i + 1)^{ème}$ partie. On ne garde que les **d** premières lettres de chaque mot., si les mots ne sont pas assez longs, on complète éventuellement par des blancs.

Supposons que la division du lexique donne:

à; aérer; électrique; hydraulique;...

- la première porte de la racine du B-Arbre a les lettres *'aére'*,
- la deuxième porte: *'élec'*; la troisième *'hydr'*... etc.

Cette affectation nécessite éventuellement de reculer le début de certaines tranches pour que tous les mots qui ont les mêmes caractères visibles soient dans la même tranche. Ainsi, si *électricité* précède *électrique* dans le lexique, il doit être incorporé à la troisième tranche.

On affecte ensuite à chaque porte, le **RPLD** calculé entre le mot du début et le mot de la fin de la tranche précédente. Par exemple, pour la porte numéro **2**, le **RPLD** entre *aérer* et *électricité* est **1**; la première lettre est discriminatoire. La racine est représentée par le noeud 1 de la figure 2.

On rappelle récursivement cette procédure de construction du B-Arbre sur chacune des **n + 1** tranches du lexique. On passe, comme paramètres, le **RPLD** des mots qui la composent, ainsi que le mot de début et le mot de fin de la tranche. A chaque niveau, on recommence le même travail que pour la racine, mais au lieu de considérer les mots à partir de leur première lettre, on les voit à partir de la première lettre discriminatoire.

Au niveau supérieur **p**, on divise la tranche de lexique fournie en **n + 1** parties. On garnit la porte numéro **i** avec le pointeur dans le fichier du lexique, vers le premier mot de la $(i+1)^{ème}$ tranche, et avec les **d** premières lettres suivant la première lettre discriminatoire.

Recherche dans le B-Arbre

Pour rechercher la position d'un mot dans le lexique, on compare ses **d** premières lettres aux d lettres de chacune des portes de la racine, jusqu'à trouver une porte (numéro i) dont la chaîne est supérieure à celle du mot, ou à atteindre la $n^{ième}$ porte. On note alors le **RPLD** correspondant et on va pointer vers le $i^{ème}$ noeud du niveau **2** dans le premier cas, ou vers le $(n+1)^{ème}$ noeud dans le deuxième.

On poursuit la recherche en s'arrêtant, à chaque noeud, sur la porte numéro **i** dont la chaîne est supérieure à celle du mot cherché, ou sur la dernière. La comparaison des mots se fait, sur les d lettres à partir du **RPLD** pour le mot recherché, et sur les **d** lettres de chaque porte.

La figure 2 montre la partie d'un B-Arbre explorée lors de la recherche du mot *asservissement*. Les trajets dans le B-Arbre figurent en traits gris épais, les pointages implicites entre portes et noeuds sont en traits fins.

Profondeur	N° du noeud dans ce niveau	RPLD	N° de la première porte de chaîne supérieure	Chaîne correspondante
1	1	1	2	*'élec'*
2	2	1	3	*'azot'*
3	(2 - 1) x 4 + 2	2	3	*'assou'*
4	(6 - 1) x 4 + 2	4	2	*'assez'*

Figure 3: recherche de *asservissement* dans le B-Arbre : les 4 étapes

Le tableau de la figure 3 rappelle, pas à pas, la recherche de *asservissement*. On remarque que l'indice du noeud, auquel on accède à l'étape **i**, est calculé selon la formule suivante:

$(j - 1) \times (n + 1) + k$

avec **j**, l'indice du noeud de l'étape précédente, **n** le nombre de portes et **k** le numéro de la porte d'arrêt à l'étape précédente.

Au niveau supérieur, on s'arrête sur la $i^{ème}$ porte dont la chaîne est supérieure au mot, ou sur la dernière porte. Dans le premier cas, si **i** est supérieur à **1**, on prend le pointeur dans le fichier associé à la porte **i - 1**; si **i** est nul, on prend l'indice de la dernière porte du noeud précédent. Dans le cas où on s'est arrêté sur la dernière porte, on prend l'indice de la dernière porte.

Ici on obtient **1256** comme pointeur dans le fichier. Il se peut que ce pointeur soit inférieur à celui du mot cherché; dans ce cas, un parcours séquentiel et limité du lexique doit permettre de le retrouver.

Conclusion pour ce B-Arbre

Implanté sur le système dès le début du développement, ce mode d'accès aux données s'est révélé efficace. Le programme nécessaire pour la création de l'arbre ou pour la recherche est compact; la place mémoire occupée par l'arbre est optimisée à cause de la notion de lettre discriminatoire et du pointage implicite.

Pour pouvoir accéder à un lexique important, nous avons choisi des noeuds de **9** portes et un arbre de profondeur **4**. Nous avons donc **1000** noeuds terminaux de **9** portes, ce qui signifie approximativement que, pour tout lexique de taille inférieure à **9000** mots simples, on pourra accéder directement au pointeur de chaque mot. Au-delà, un petit parcours séquentiel du lexique est nécessaire. La taille du B-Arbre ainsi constitué est de **5** kOctets pour les noeuds intermédiaires et de **77** kOctets pour les noeuds terminaux. Ces volumes de mémoire sont adaptés au matériel dont nous disposons.

Le temps de consultation du B-Arbre est de **p x n** comparaisons de chaînes de taille **d**. Le temps de recherche en mémoire est négligeable par rapport au temps d'accès au disque. Nous conseillons de choisir un B-Arbre large, avec beaucoup de noeuds terminaux et peu d'intermédiaires afin d'accéder au lexique le plus précisément possible.

c Analyse lexicale des mots simples : défléchissement

Dans cette partie, nous présentons la première étape de l'analyse des noms composés qui relève une phrase et la découpe en mots. Puis elle recherche, pour chaque mot fléchi, la liste des formes canoniques qui l'admettent pour flexion (**défléchissement**). Cette partie n'est pas approfondie. Nous l'avons fait figurer car elle sert de support à la reconnaissance des noms composés.
Nous précisons la façon dont nous effectuons le défléchissement à l'aide d'un **Arbre Lexicographique Généralisé** des suffixes inversés de la langue.

Découpage

La séparation des phrases est sommaire puisque le **découpage** est fait à chaque rencontre de point.

Les séparateurs des mots retenus sont tous les signes qui ne sont pas des caractères alphabétiques dans le code utilisé par la machine. Ces découpages sont en marge du sujet de cette étude, et ils ont déjà fait l'objet de travaux importants, voir [Silberztein 89]. On fragmente les mots à chaque fois que l'on rencontre un séparateur, sauf cas des fins de lignes où il peut y avoir une césure. A chaque mot est associée une caractéristique appelée sa **forme** qui indique de quel type de séparateur il est suivi.

Par exemple le mot composé *aujourd'hui* sera coupé en deux parties:

aujourd'hui ——> *aujourd* + **forme** apostrophe
——> *hui* + **forme** normale

aux signes de ponctuation on attribue la **forme** ponctuation:

) ——>) + **forme** ponctuation

Défléchissement

Les mots de la phrase courante sont **défléchis** pour trouver, dans le lexique, les formes canoniques des mot simples qui admettent le mot courant parmi leurs formes fléchies. Pour chaque suffixe possible d'un mot lu, on recherche la forme canonique correspondante dans le lexique. Si on la trouve, on la relève avec ses règles associées, en utilisant le mécanisme d'allocation, et en transformant des pointeurs numériques sur fichier en pointeurs réels en mémoire. Si au moins une de ses flexions est égale au mot cherché, on conserve les règles associées en mémoire et on crée autant d'homographes que de flexions trouvées en utilisant l'allocation des termes. Dans le cas contraire, on désalloue toutes les règles associées à la forme canonique potentielle trouvée dans le lexique.

Si nous prenons une phrase telle que: *Nous avions des systèmes experts.* La première étape de l'analyse constituera, à l'aide du dictionnaire des mots usuels ou du lexique-grammaire, à donner pour chaque mot, la liste des homographes qui le représentent.

Mot	Nombre d'homographes	Origine	Catégorie lexicale
Nous	1	Dictionnaire	Pronom
avions	2	Dictionnaire	Flexion du verbe *avoir*
		Dictionnaire	Pluriel du nom *avion*
des	1	Dictionnaire	Déterminant
systèmes	1	Lexique-grammaire	Pluriel du nom *système*
experts	2	Lexique-grammaire	Pluriel du nom *expert*
		Lexique-grammaire	Masc. pl. de l'adjectif *expert*

Cette étape laisse 1 x 2 x 1 x 1 x 2 = 4 phrases possibles candidates à une analyse syntaxique.

	nous	*avions*	*des*	*systèmes*	*experts*
1:	(Pronom)	(Verbe)	(Déterminant)	(Nom)	(Nom)
2:	(Pronom)	(Verbe)	(Déterminant)	(Nom)	(Adjectif)
3:	(Pronom)	(Nom)	(Déterminant)	(Nom)	(Nom)
4:	(Pronom)	(Nom)	(Déterminant)	(Nom)	(Adjectif)

Les mots usuels: prépositions, flexions des verbes auxiliaires... et leurs homographes sont reconnus directement dans le dictionnaire des mots fléchis.

Pour les mots peu fréquents, à partir d'un mot simple de la phrase fournie en entrée, le programme cherche à reconnaître un suffixe parmi les dernières lettres. Tous les suffixes connus de la langue considérée sont groupés, au préalable, dans un graphe étiqueté, déterministe, appelé **Arbre Lexicographique Généralisé (ALG)**, tel qu'il est décrit dans [Aho 83]. L'ALG reconnaît les suffixes à partir de la fin. Le mot inversé est présenté à cet automate. Pour tous les suffixes reconnus, on note la longueur de la sous-chaîne du mot qui représente sa forme canonique potentielle.
Par exemple, à partir du mot *entreprises* sont créées les formes canoniques possibles: *entrepr, entrepri, entrepris, entreprise, entreprises.* On recherche dans le lexique-grammaire toutes celles qui sont formes canoniques d'un mot simple et dont, une flexion au moins, fournit le mot cherché. Chaque forme fléchie trouvée constitue un homographe du mot.

Explicitons les étapes de création et d'utilisation de l'ALG.

1: recensement des suffixes de la langue

On relève tous les suffixes de la langue qu'utilise le système pour calculer les flexions à partir des formes canoniques. On les inverse pour les inclure dans un arbre lexicographique généralisé. Considérons comme exemple la sous-liste suivante: *e, se, tne, ze, snoe, tiae, tneiae*, correspondant à la liste des suffixes: *e, es, ent, ez, eons, eait, eaint* (figure 4).

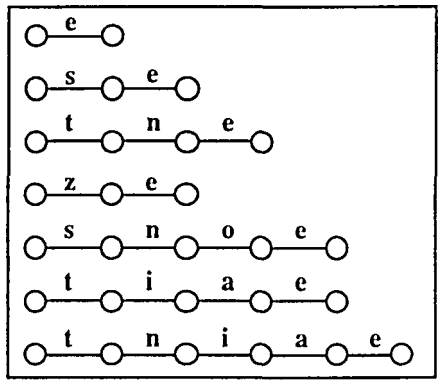


Figure 4: étape 1, liste des suffixes inversés proposés

2: construction de l'ALG à partir de la liste des suffixes inverses

Cet arbre est construit en y insérant successivement les suffixes de la liste précédente. La forme finale est celle de la figure 5.

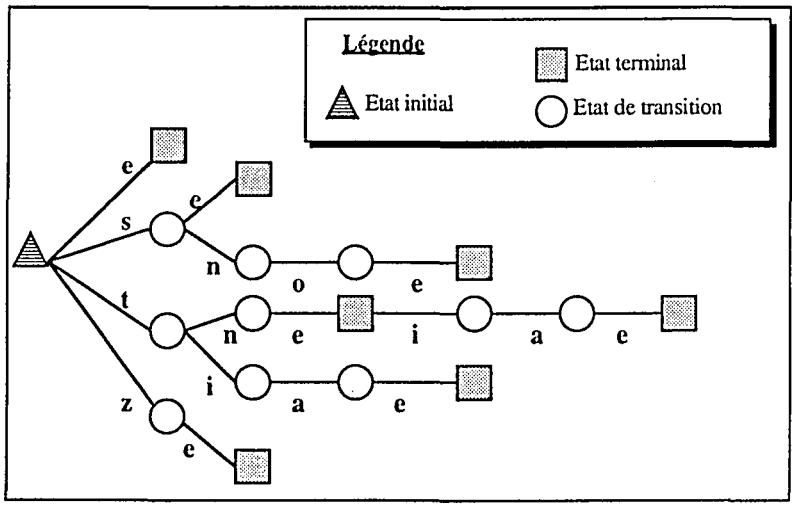


Figure 5: étape 2, construction de l'ALG

3: application de l'ALG au désuffixage du mot *mangeaient* (figure 65).

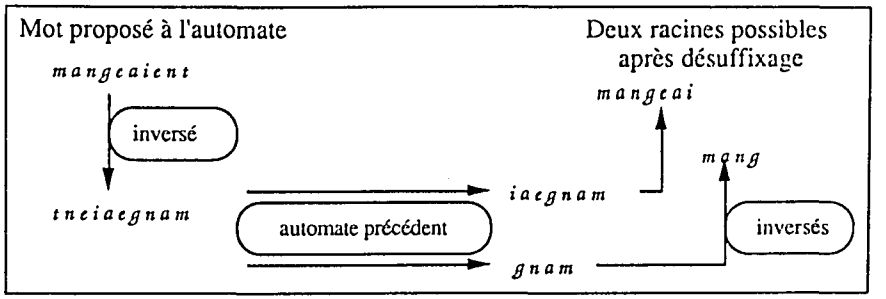


Figure 6: étape 3, application de l'ALG au désuffixage

Lorsqu'une flexion d'une forme canonique est reconnue, on relève dans le lexique-grammaire ses caractéristiques, ainsi que toutes les règles qui lui sont associées. Ces règles serviront à l'analyseur des noms composés qui est décrit au paragraphe suivant.

2 ANALYSE MORPHOLOGIQUE DES NOMS COMPOSÉS

Dans cette partie, nous décrivons la mise en oeuvre de la reconnaissance morphologique des noms composés. Nous avons choisi un système de représentation des noms composés à l'aide de règles Context Free, et un mécanisme d'unification repris du langage Prolog. Nous avons voulu que ce formalisme soit lisible, adapté aux contraintes de la langue, et qu'il puisse être prolongé pour décrire les transformations sur les noms composés dans la deuxième partie.

Le premier paragraphe (I.2.a) justifie cette notion de nom composé pour des formes de la langues qui peuvent être, à tort, considérées comme libres. Puis, le reste de cette partie I.2 est essentiellement informatique. Après avoir donné le formalisme d'énonciation des règles de noms composés, nous y décrivons les structures de données et les algorithmes qui réalisent les tâches de compilation et d'analyse.

a Notion de nom composé

Après avoir rappelé quelques références sur les noms composés, nous décrivons plus particulièrement le travail effectué au L.A.D.L. sur les noms composés.

Cette notion est un concept ancien de la langue: [Darmesteter 94] recense les diverses formes de noms composés, en s'attachant essentiellement aux formes très figées. Il propose un procédé de classification qui tient compte à la fois de la structure syntaxique du nom composé, et de la

structure phrastique dont il est issu par réduction. Il distingue, ainsi, la juxtaposition pour les noms composés comme *chef-d'oeuvre* qui sont de forme grammaticale, de la composition elliptique observée pour *timbre-poste* qui est la réduction de *timbre pour la poste*. Il justifie l'existence d'un nom composé sur des critères sémantiques qui lui font perdre son sens compositionnel, et le font apparaître comme une nouvelle entrée lexicale. Nous remarquons ici, comme dans [G. Gross 88A] et [G. Gross 90] que certains cas de réduction sont douteux ou inexistants pour des mots comme *homme de paille* ou *main courante*.

Plus récemment, [Robins 73] donne une notion morphologique de noms composés: il s'agit de mots formés de plusieurs morphèmes et ne pouvant pas subir de transformation. Cette présentation n'inclut pas des mots tels que *permis de chasse* qui peut se trouver sous la forme *permis de pêche ou de chasse*, où la connexité de la chaîne de base est brisée.

[Grevisse 86] propose une classification plus formelle qui repose sur des critères syntaxiques. Le nom composé se distingue du syntagme: l'un est une unité lexicale permanente, l'autre est une forme libre occasionnelle. La forme du composé (soudure des mots *agroalimentaire*, traits d'union *hydro-réglable* ou téléscopages *franglais*) n'influence pas sa classification puisqu'on trouve dans chaque catégorie, des composés de formes différentes. Cependant, une distinction est faite entre les locutions telles que *chemin de fer* où les mots sont séparés par des blancs et les "vrais" mots composés tels que *oiseau-mouche* ou *hydroélectrique* où les morphèmes sont soudés ou reliés par des tirets. Les différentes classes proposées sont :

- $\underline{V\ N_1}$: *lave-vaisselle*,
- $\underline{V\ N_0}$: *pense-bête*,
- Prép N et Prép V : *entrecôte, sous-estimer*,
- formes en N N où le deuxième nom est un complément du premier : *wagon-lit*,
- coordinations en N N, V V, Adj Adj : *porte-fenêtre, aigre-doux, tournevirer...*,
- nominalisations de syntagmes : *rendez-vous, va-et-vient...*,
- figements de syntagmes : *pot-de-vin, rond-point...*,
- adjonctions d'éléments étrangers : *granivore, dactylographie...*

Cette distinction entre nom composé et locution n'est pas un critère de discrimination, car les locutions peuvent appartenir à toutes le catégories précédentes. En diachronie, certains mots peuvent passer du statut de locution à celui de nom composé, les récentes propositions du Conseil Supérieur de la Langue Française [Le Monde du 21/6/90] dont cinq points ont été retenus par l'Académie Française, étendent l'usage du trait d'union et de la soudure. De nombreuses formes vont ainsi passer du statut de locution à celui de nom composé au sens de [Grevisse 86]. De plus, les classifications sont délicates comme le montre [Vives 90] pour les juxtapositions de deux noms. Ainsi, si le deuxième nom est un dérivé du verbe, il peut avoir un statut de nom ou d'adjectif. Dans *arbre moteur, moteur* est un adjectif, alors que dans *huile moteur*, il s'agit d'un nom.

Distinction entre nom composé et groupe nominal

Nous exposons, ici, la notion de nom composé telle qu'elle est retenue au L.A.D.L. L'acception prise pour cette notion est plus large que celles proposées précédemment.

Pour distinguer les formes libres des noms composés, nous reprenons les arguments développés dans [G. Gross 88A] et [G. Gross 90] où le critère de discrimination est le degré de figement. Il se mesure en appliquant à la structure les transformations normalement admises par un syntagme libre de même nature et en notant l'acceptabilité des formes obtenues. Par exemple, l'insertion d'un adverbe dans *système expert* donne la forme inacceptable **système très expert*, alors que *homme expert* devient *homme très expert*. L'étude des noms composés, qui correspondent à une structure syntaxique correcte, mais qui n'en ont pas toute la liberté, est donc particulièrement intéressante. Ces mots sont fréquents dans dans des corpus techniques tels que les textes de [D.E.R. 87] que nous avons choisi d'observer. Ils représentent des entrées lexicales dont la reconnaissance est pertinente en compréhension automatique.

Dans notre étude, nous nous limitons aux mots composés dont la structure est celle du groupe nominal, c'est-à-dire des séquences de mots dont la structure syntaxique, indépendamment du contexte où ils se trouvent, est du type N de (GN + N). Pour éviter qu'un automate ne les identifie comme des groupes nominaux, mais bien comme une entrée lexicale, il est nécessaire de les recenser de façon exhaustive. Nous les appelons donc **noms composés**.

Pour faire le lien avec la notion de groupe nominal, nous rappelons simplement la définition générique du groupe nominal donnée par [M. Gross 86N]. Il s'agit de la structure:

GN = Dét N de (GN + N)

où le premier nom représente le substantif de base du groupe nominal. Ce nom peut être complété par un modifieur GN (récursif) ou par de N (non récursif). Les parties annexes du groupe nominal formées par le déterminant et les modifieurs sont issues de dérivation de phrases de niveau supérieur.

de (GN + N) est un modifieur du nom. Dans le nom composé, il a un statut particulier. Ainsi, [Giry-Schneider 87] fait la différence entre des modifieurs lexicaux qui permettent de décrire les noms composés en N Adj, N Prép N ou N N et les modifieurs ayant un rôle syntaxique tels que *sans pareille* dans *Paul est d'une audace sans pareille*.

Etude d'un exemple: quantité de N

Sur un exemple, nous montrons que la connaissance d'un nom composé permet de donner une analyse syntaxique de la phrase où il se trouve qui est différente de celle qui aurait été faite avec une forme libre. Cette distinction confirme la nécessité de recenser de telles entrées lexicales.

Nous considérons la structure quantité de N et nous analysons les deux phrases suivantes:

(1) *Max calcule une quantité de mouvement*

(2) *Max calcule une quantité de (liquide + lits).*

- Dans la première phrase, *quantité de mouvement* est un nom composé, qui est l'objet direct du verbe *calculer* précédé d'un déterminant indéfini (figure 7).

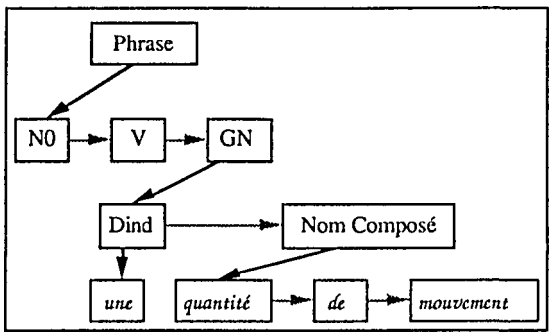


Figure 7: analyse de *quantité de mouvement* : une forme figée

- Dans la deuxième phrase, *liquide* est un nom de masse au singulier et *lits* un nom énumérable au pluriel, le nom déterminatif *quantité* forme le déterminant nominal *une quantité de*, appelé aussi déterminant indéfini dans [Grevisse 86] (figure 8).

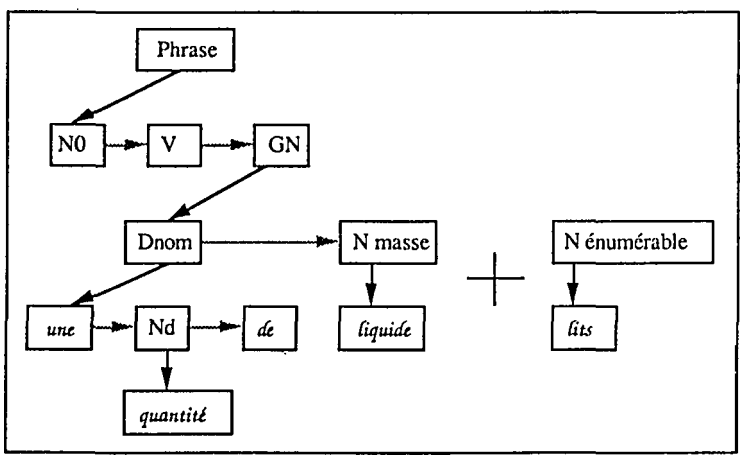


Figure 8: analyse de *quantité de (lits + liquide)* : une forme libre

Dans ces deux analyses, le verbe a deux objets directs différents:

- phrase 1, l'objet du verbe est N_1 de N_2 : *quantité de mouvement,*
- phrase 2, l'objet est N_2 : *liquide* ou *lits*..

L'acceptabilité de la deuxième structure est acquise grâce aux tableaux développés au L.A.D.L. tels que ceux de la page 67 de [M. Gross 86N]. En effet, *quantité* peut entrer dans un déterminant nominal si le nom déterminé est, soit un nom énumérable au pluriel, soit un nom de masse au singulier.

Le problème de distinction entre la phrase (1) et la phrase (2) ne peut être résolu que par la connaissance du nom composé *quantité de mouvement*, puisque les deux phrases ont exactement la même structure. Afin de ne pas analyser *quantité de mouvement* comme un groupe nominal libre, il convient de le recenser comme entrée lexicale unique de type nom composé.

Dans la forme libre quantité de N, certains noms tels que *mouvement* ou *chaleur* se distinguent des autres pour former une structure plus figée que le groupe nominal libre. Par exemple l'insertion d'un adjectif donne:

**Max calcule la quantité réelle de mouvement*

Max calcule la quantité réelle de liquide.

Le signe * mis devant la première phrase indique son inacceptabilité comme extension de la séquence *quantité de mouvement* pris comme une notion de Sciences Physiques. On pourrait, bien sûr, la trouver dans une phrase telle que:

Le départ de Max a déclenché une quantité énorme de mouvement.

Tous ces indicateurs de figement, recensés dans [G. Gross 88A] et [G. Gross 90], seront revus dans la deuxième partie, portant sur les transformations des noms composés.

Les noms composés observés sont des groupes nominaux présentant un caractère figé, c'est-à-dire des structures qui

- syntaxiquement, s'analysent comme un groupe nominal sans déterminant,
- ont un comportement syntaxique moins général que le groupe nominal libre correspondant,
- sémantiquement, correspondent à une entrée lexicale distincte dont le sens ne peut se déduire des différents constituants.

Nous n'avons pas observé les noms composés de structures diverses qui ne pourraient pas s'analyser comme des groupes nominaux libres tels que les structures en V N, Prép N comme *chauffe-eau, rendez-vous, en-cas* ou *à-côté*. Pour une typologie des noms composés et une prise en compte des variantes graphiques, on peut consulter [Mathieu-Colas 88] et [Mathieu-Colas 90].

b Formalisme: contrainte, substitution et accord

Nous ne détaillons pas cet exposé, puisque le problème de la reconnaissance morphologique des noms composés à l'aide de dictionnaires électroniques a été traité dans [Silberztein 89], [Silberztein 90] et [Maurel 89]. Nous présentons rapidement le formalisme afin de pouvoir l'utiliser dans les exemples que nous donnerons lors de l'étude des transformations des noms composés, et faire le lien avec le système informatique.

Représentation morphologique des noms composés: concaténation et liste

Dans cette partie, les noms composés sont représentés en fonction de leurs constituants, sans se soucier des règles d'accord et de variations en genre et nombre. Les terminaux sont notés par une chaîne de caractères et les non terminaux entre crochets <>.

Soient les mots *ministère de l'Education Nationale* et *ministère de la Recherche,* une seule règle suffit à représenter chacun d'eux sous forme d'une **concaténation**:

```
(1) <ministère de l'Education Nationale>
        -> ministère de l Education Nationale;
(2) <ministère de la Recherche> -> ministère de la Recherche;
```

On peut choisir de faire figurer ces deux ministères dans la même règle, il s'agit alors d'une **liste** de deux concaténations:

```
(3) <LISTE ministères> -> ministère de l Education Nationale
                        + ministère de la Recherche;
```

ou factoriser la dernière règle pour la rendre plus compacte:

```
(4) <LISTE ministères> -> ministère de (l Education Nationale
                                      + la Recherche);
```

ou introduire un non terminal pour reporter la liste des ministères dans une autre règle afin de pouvoir l'utiliser pour décrire les noms de ministres:

```
(5) <LISTE ministères> -> ministère de <LISTE noms de ministères>;
(6) <LISTE ministres> -> ministre de <LISTE noms de ministères>;
(7) <LISTE noms de ministères> -> l Education Nationale + la Recherche;
```

La factorisation peut se faire sur tous les types de termes, y compris sur des termes qui sont déjà eux-mêmes des factorisations:

```
(8) <LISTE ministres ou ministères> -> (ministère + ministre) de
                                      (l Education Nationale + la Recherche);
```

On voit donc que les deux systèmes de règles formés, d'une part par {(5), (6), (7)}, et d'autre part par {(8)}, reconnaissent tous les deux les quatre chaînes: *ministère de l'Education Nationale, ministère de la Recherche, ministre de l'Education Nationale, ministre de la Recherche* et celles-ci seulement.

La représentation des noms composés avec leurs caractéristiques lexico-syntaxiques

Nous avons ajouté à la grammaire des noms composés un système de contraintes sur des variables qui doivent être satisfaites lors de l'analyse. Au I.2.e, nous détaillons la mise en oeuvre informatique de la vérification de ces contraintes (unification).

Dans les exemples ci-dessous, nous nous plaçons dans le cadre de la grammaire accompagnée de variables sur les termes. Cette possibilité permet de représenter les caractéristiques lexico-syntaxiques des morphèmes ou des termes syntaxiques.

Exemple 1: introduction à la description des caractéristiques, notion de **contrainte**

Prenons d'abord le cas de la chaîne simple *éducation nationale:*

```
<Education Nationale> ->
    {N, f, s} éducation{N, f, s, n} nationa{A, f, s, n};
```

La description que nous avons adoptée pour les noms composés est une description des formes de bases, à partir de laquelle nous pourrons analyser ou produire toutes les flexions. C'est pourquoi le mot fléchi *nationale* a été remplacé par sa forme canonique *nationa,* ainsi que *éducation* qui est égal à sa forme canonique.

Chaque forme canonique est suivie de quatre champs: la catégorie lexicale, le genre, le nombre et la forme du mot (n pour normale). Ici les deux genres et les deux nombres sont féminins et singuliers puisque le nom composé ne peut être utilisé au pluriel.

`<Education Nationale>` est une nouvelle entrée lexicale dont les catégorie lexicale, genre et nombre sont décrits par la première accolade: `{N, f, s}`. La forme du nom composé n'est pas précisée, puisqu'elle est reportée sur les constituants.

Lorsque les valeurs sont données littéralement, on dit qu'elles vérifient des **contraintes**. Il s'agit de l'égalité entre une variable libre et une valeur, en programmation logique [Colmerauer 78]. L'absence de contrainte pour une variable est notée **v** (vide). Dans ce cas, toute valeur est acceptée.

Exemple 2: notion de **substitution**

La description précédente pourrait être enrichie des possibilités d'écrire chacun des deux mots avec une majuscule, avec les deux possibilités *éducation nationale* et *Education Nationale.*

```
<Education Nationale> ->
{N, f, s}   ({N, f, s} éducation{N, f, s, n} nationa{A, f, s, n}
          + {N, f, s} éducation{N, f, s, m} nationa{A, f, s, m});
```

On reconnaît dans cette expression, le choix entre les deux formes proposées, où la graphie des mots *éducation* et *nationale* est notée dans le quatrième champ: **n** pour normale et **m** pour majuscule. Ce choix est précédé de la description **{N, f, s}** qui donne les contraintes sur les valeurs des caractéristiques de l'élément qui sera substitué.

Plutôt que de d'indiquer littéralement la valeur des caractéristiques, on pourrait préférer demander à ce que soient automatiquement substituées les valeurs de l'élément de la liste qui sera retenu. Dans ce cas, chaque valeur qui doit être remplacée est marquée d'un point d'exclamation, comme dans la règle ci-dessous est équivalente à la précédente:

```
<Education Nationale> ->
{!, !, !} ({N, f, s} éducation{N, f, s, n} nationa{A, f, s, n}
          + {N, f, s} éducation{N, f, s, m} nationa{A, f, s, m});
```

Cette substitution peut également être utilisée dans une concaténation. Par exemple, le genre et le nombre du nom composé *éducation nationale* sont donnés par ceux du mot fléchi *éducation* qui est le premier mot de la chaîne. On notera chaque valeur qui doit être remplacée par un point d'exclamation, suivi de l'indice du mot qui fournira la valeur. La règle ci-dessous qui est équivalente à la précédente:

```
<Education Nationale> ->
{!, !, !} ({!1, !1, !1} éducation{N, f, s, n} nationa{A, f, s, n}
          + {!1, !1, !1} éducation{N, f, s, m} nationa{A, f, s, m});
```

La substitution peut être accompagnée d'une contrainte si l'on souhaite ne sélectionner que certaines valeurs parmi celles qui sont remontées dans l'analyse. Si la contrainte est vide, on peut l'omettre comme dans l'exemple précédent. Plutôt que de forcer directement le mot *éducation* à être au singulier, on pourrait reporter cette contrainte sur la représentation du nom composé:

```
{!1, !1 s, !1} éducation{N, f, y, n} nationa{A, f, s, n}
```

Celle-ci fournit exactement les mêmes résultats que la représentation précédente, mais la lisibilité serait moins bonne.

En résumé, il y a deux sortes de substitutions: une sur les têtes des chaînes où l'on précise l'indice du mot dont il faut prendre la valeur, et une sur les choix, qui permet de retenir la valeur de l'élément pris dans la liste. Dans les deux cas la substitution peut être accompagnée d'une contrainte. Il s'agit d'une remontée des valeurs des variables dans l'arbre d'analyse.

Exemple 3: notion d'accord

Dans la règle précédente, le genre et nombre de l'adjectif *national* sont précisés par des contraintes qui sont indépendantes des valeurs de *éducation*.

Il serait préférable de noter un accord entre le nom et l'adjectif. Nous utilisons la lettre a suivie de l'indice du mot, dans la concaténation courante, avec lequel le champ s'accorde. La règle ci-dessous est équivalente à la précédente:

```
<Education Nationale> ->
{!, !, !} ({!1, !1, !1} éducation{N, f, s, n} nationa{A, a1, a1, n}
          + {!1, !1, !1} éducation{N, f, s, m} nationa{A, a1, a1, m});
```

L'accord ne peut se faire qu'à l'intérieur d'une même concaténation. Le principe de l'accord est le suivant: si une des deux valeurs est vide, l'accord est réalisé, sinon les deux champs doivent avoir la même valeur. Il s'agit d'une égalité entre deux variables, en programmation logique. L'accord peut être accompagné d'une substitution et/ou d'une contrainte, sachant que l'ordre des priorités est le suivant:

- 1: la substitution est effectuée si elle existe,
- 2: la satisfaction de la contrainte est vérifiée si elle est présente,
- 3: l'accord est vérifié.

Lorsque la contrainte est vide, on peut l'omettre comme dans l'exemple précédent.

Accord rétroactif

En français, les accords entre les divers mots d'une concaténation peuvent toujours être énoncés en précisant pour chaque mot, à partir du deuxième, son éventuel accord avec un mot précédent. Par exemple, pour *faible sécurité technologique*, nous pouvons définir la règle suivante:

```
<faible sécurité technologique> ->
    {!2, !2, !2} faible{A, v, v, n} sécurité{N, f a1, a1, n}
                 technologique{A, a1, a1, n};
```

- *faible* est variable en nombre et en genre,
- [A1] *sécurité* est féminin et accorde genre et nombre à ceux de *faible*,
- [A2] *technologique* accorde genre et nombre indifféremment à *faible* ou à *sécurité*; ici nous avons choisi de l'accorder à *faible*.

Supposons que nous souhaitions analyser la chaîne d'entrée *faible sécurité technologique* à l'aide de la règle précédente. Les deux adjectifs admettent deux homographes que nous indiçons par leur genre et nombre: $faible_{(m,s)}\ faible_{(f,s)}\ technologique_{(m,s)}\ technologique_{(f,s)}$

Nous avons donc quatre séquences candidates à l'analyse dont seule la quatrième est correcte: $faible_{(m,s)}\ sécurité_{(f,s)}\ technologique_{(m,s)}$

$faible_{(f,s)}\ sécurité_{(f,s)}\ technologique_{(m,s)}$

$faible_{(m,s)}\ sécurité_{(f,s)}\ technologique_{(f,s)}$

$faible_{(f,s)}\ sécurité_{(f,s)}\ technologique_{(f,s)}$.

Nous présentons, dans la figure 9, le tableau de vérification des accords [A1] et [A2] lors de l'unification en fin d'analyse de chacune de ces quatre chaînes.

Nous constatons que seule la quatrième séquence vérifie les deux accords [A1] et [A2] contenus dans l'énoncé de la règle. D'autre part, le mécanisme de substitution donné par `{!2,!2,!2}`, donne après unification `{N,f,s}` (nom féminin singulier) pour la flexion du nom composé *faible sécurité technologique*. Ceci est cohérent avec les règles de l'analyse générale, puisqu'il s'agit alors d'une forme figée au féminin singulier.

Il serait plus lisible de donner une contrainte de genre féminin à *faible* dans l'énoncé de la règle, puisque les homographes masculins de cet adjectif ne vont pas produire d'analyse correcte lors de l'unification:

```
<faible sécurité technologique> ->
    {!2, !2, !2} faible{A, f, v, n} sécurité{N, f, a1, n}
                 technologique{A, a1, a1, n};
```

Le temps d'analyse de cette deuxième proposition n'est pas plus court, mais l'énoncé est plus clair.

Chaîne dont l'unification est vérifiée	Validité de l'accord A1	Validité de l'accord A2
faible(m,s) sécurité(f,s) technologique(m,s)	faux	vrai
faible(f,s) sécurité(f,s) technologique(m,s)	vrai	faux
faible(m,s) sécurité(f,s) technologique(f,s)	faux	faux
faible(f,s) sécurité(f,s) technologique(f,s)	vrai	vrai

Figure 9: vérification des accords de *faible sécurité technologique*

Une liste de noms composés est donnée en annexe III.1, avec le formalisme présenté dans cette section.

Synthèse, un exemple de règle de nom composé : <durée de vie>

Figure 10, nous présentons un exemple de règle complète. Certains points, tels que la liste des transformations acceptées par ce nom composé, ne seront traités que dans la partie II. Nous voulons donner un exemple complet pour pouvoir traiter, dans cette partie I, la compilation et l'analyse des règles.

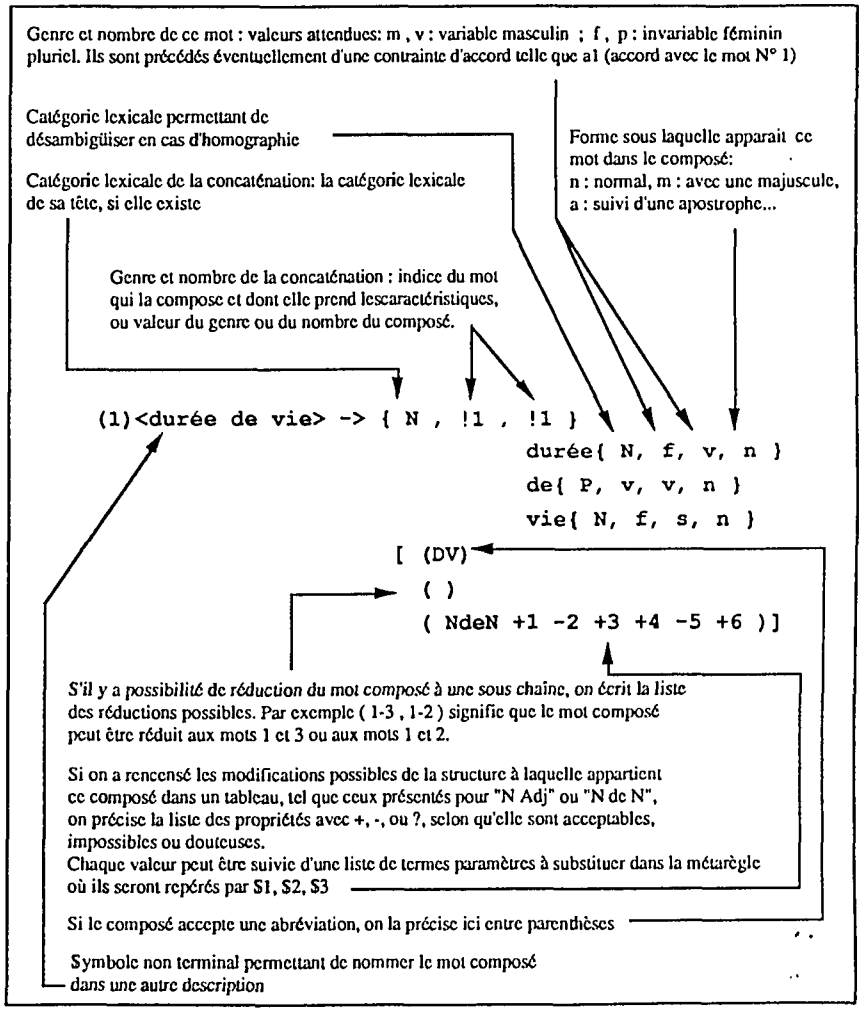


Figure 10: un exemple de règle complète: <durée de vie>

c Compilation et stockage des règles

Cette partie décrit la réalisation du **compilateur** des règles de noms composés. A partir d'une description de la forme extérieure des règles (donnée en annexe III.1), il produit un arbre qui représente la structure profonde de la règle. Cet arbre sert de support à la reconnaissance morphologique des noms composés qui est décrite au I.2.e.
Cet arbre est stocké sur fichier, il est rattaché à un des noms simples qui forme le nom composé. Il est chargé en mémoire à chaque fois qu'une flexion du nom simple est rencontrée dans une phrase, et l'analyse de cette règle est alors tentée. Le rattachement homogène des règles aux mots simples est décrit en I.3.

Compilateur des règles de noms composés

Compilation et édition de liens

La **compilation** est l'analyse syntaxique des règles de noms composés et la construction de l'arbre d'analyse syntaxique de chaque règle. La figure 11 montre celui obtenu pour `<durée de vie>`, donnée à la figure 10 du paragraphe précédent. Pour chaque noeud, nous avons indiqué le symbole de la forme extérieure des règles, donnée au III.1, qui a permis de le produire.

L'arbre de racine N_1 est celui sur lequel l'analyse de base va s'appuyer. Lors de l'analyse d'un texte, on cherche à reconnaître cette structure syntaxique par réécriture, puis on vérifie les contraintes sur les caractéristiques lexico-syntaxiques par unification.
Les parties de l'arbre concernant les métarègles et les élisions se trouvent directement reliées à la racine, et servent à produire des arbres d'analyse annexes à l'aide de l'arbre principal qui est ici l'arbre de racine N_1. Ce mécanisme sera décrit en II.2.b et II.2.c lors de la présentation des métarègles et du mécanisme d'élision.

Lorsque le compilation des règles est achevée, on réalise l'**édition de liens**. Les chaînes des mots simples sont remplacées par des références numériques uniques afin d'accélérer la reconnaissance et d'éviter les comparaisons et les stockages de chaînes. Celles des symboles non terminaux sont remplacées par un couple formé des deux éléments suivants:
- un pointeur dans le fichier vers le mot auquel est rattachée la règle qu'il représente,
- le rang de cette règle parmi celles du mot de rattachement.

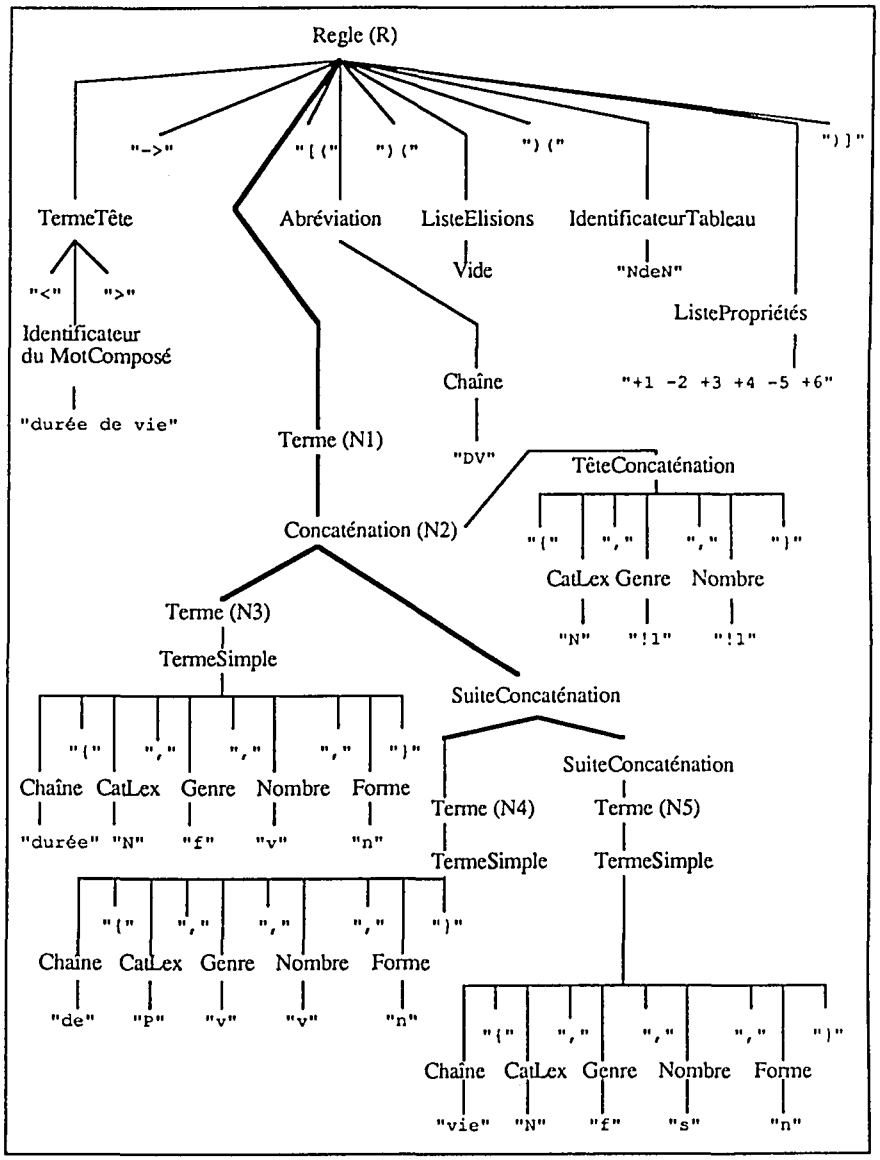


Figure 11: arbre d'analyse de la règle du nom composé `<durée de vie>`

Stockage

Lorsque la compilation et l'édition de liens sont achevées, on transfère les règles sur fichier. Nous allons montrer sur un exemple, la façon dont les règles sont stockées.

Figure 12, nous indiquons la représentation en mémoire de l'arbre syntaxique de la règle `<LISTE équipement informatique>`, à l'issue de la compilation. Avant que la règle ne soit transférée sur fichier, les termes sont reliés entre eux par des pointeurs.

```
<LISTE équipement informatique>
-> {N, !1, !1}
    ({N, !, !}   <LISTE micro-ordinateurs>{N, v, v}
               + <terminal de saisie>{N, v, v})
    ({A, a1 !, a1 !} portable{A, a1, a1, n}
                   + autonome{A, a1, a1, n}
                   +{A, v, v} grand{A, v, v, n} écran{N, a3 m, a3 s, n});
```

Nous ne considérons que les identificateurs des termes, sans les caractéristiques lexico-syntaxiques. La règle précédente devient alors:

```
<LISTE équipement informatique>
-> (<LISTE micro-ordinateurs> + <terminal de saisie>)
   (portable + autonome + grand écran);
```

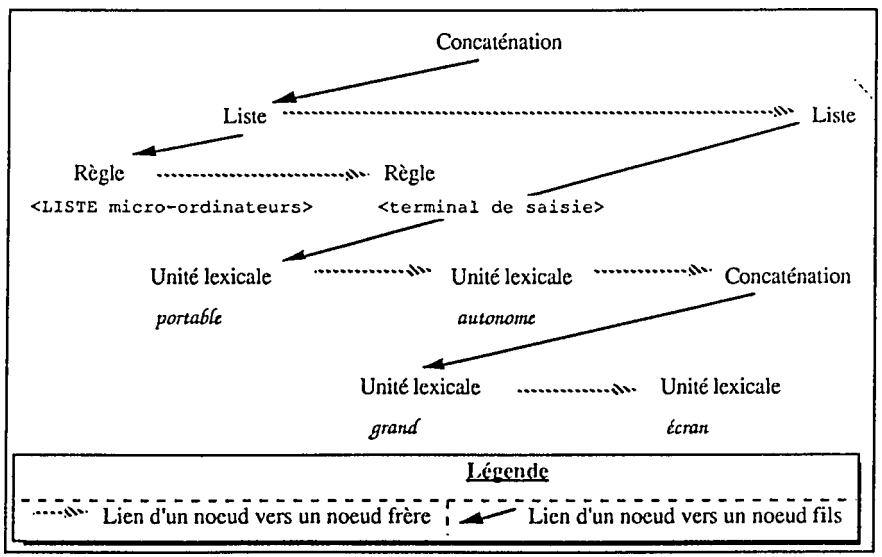


Figure 12: arbre syntaxique de: `<LISTE équipement informatique>`

Par un parcours en préordre de cet arbre syntaxique, on déduit une représentation séquentielle de la règle. Celle-ci est stockée dans le lexique-grammaire, et servira de support à la reconnaissance morphologique des noms composés. La figure 13 est une représentation linéaire de l'arbre d'analyse de `<LISTE équipement informatique>`. Nous n'y avons pas indiqué les caractéristiques lexico-syntaxiques de chaque terme et qui servent à contrôler l'analyse au moyen de l'unification.

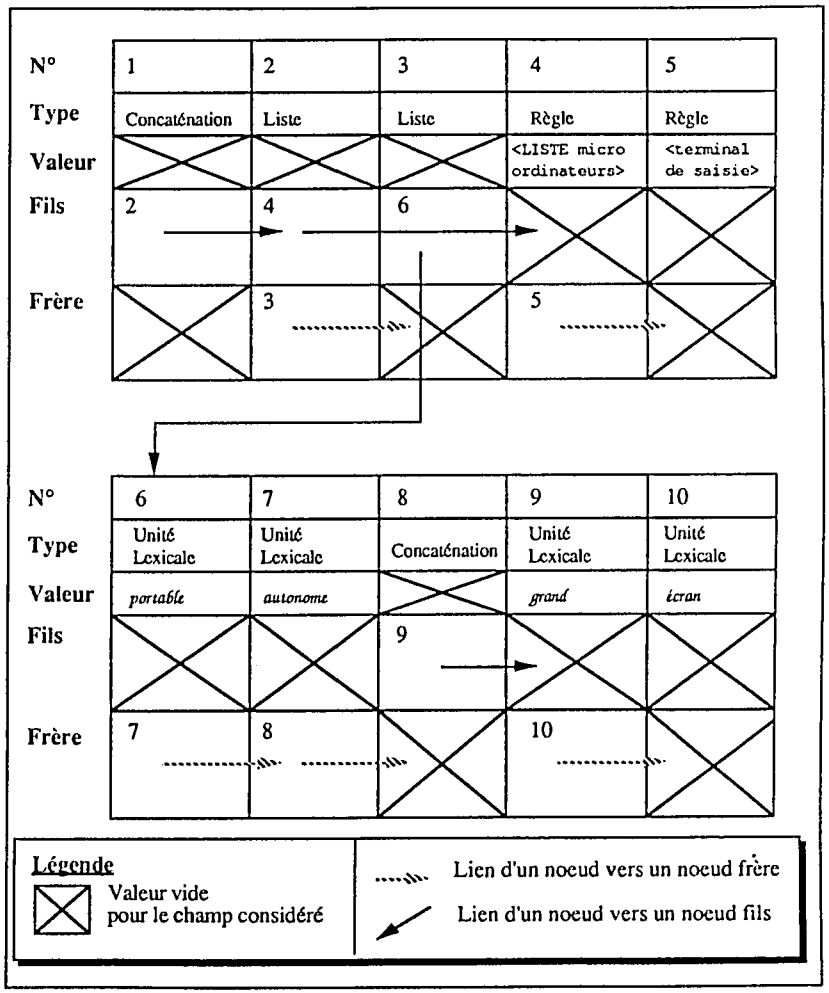


Figure 13: représentation plane en mémoire de l'arbre de: `<LISTE équipement informatique>`

Structures informatiques de stockage des règles sur fichier

La base lexicale est composée de deux dictionnaires.

- Le **dictionnaire des mots usuels** est formé des mots fréquents, il est chargé en mémoire à chaque début de session et sa taille est limitée. On y accède par Hash Code.

- Le **lexique-grammaire** renferme les mots simples et composés qui ne figurent pas dans le dictionnaire précédent. Il est résident sur fichier, on y accède par B-Arbre (voir I.1.b).

Afin d'optimiser l'efficacité de l'analyseur, les noms composés sont regroupés par grappes à un des mots qui les forment au moyen de la lexicalisation (voir I1.3). Ceux dont la reconnaissance ne présente pas d'intérêt ne sont pas rattachés.

Structure informatique du lexique-grammaire

Dans ce paragraphe, nous indiquons les structures informatiques mises en oeuvre pour stocker les règles compilées sur fichier. Elles sont représentées dans les figures 14 et 15. La liaison, entre règles de noms composés et mots simples, est réelle puisque le fichier est hétérogène et contient séquentiellement la chaîne du mot simple, ses caractéristiques, et les noms composés qui lui sont rattachés.

Nous décrivons ci-dessous la structure du lexique grammaire regroupée autour d'un mot simple.

1: En début de structure, se trouve la **chaîne** du mot, terminée par un caractère nul et suivie d'un terme spécial qui est la description du mot simple.

2: Les rubriques que comporte le **mot simple** sont:

- la référence (un nombre unique qui le désigne dans les noms composés où il apparaît),

- la description des caractéristiques lexicales, subdivisée en quatre rubriques: catégorie lexicale, genre s'il s'agit d'un nom ou d'un nom-adjectif, numéro de flexion indiquant les suffixes devant être ajoutés à la forme canonique pour avoir les différentes flexions et forme. La forme permet d'indiquer, par exemple, si le mot a obligatoirement une première lettre en majuscule (*Bretagne*) ou s'il est toujours suivi d'une apostrophe *(aujourd)*.

Après ce terme, se trouve la liste des têtes des règles.

3: Chaque **tête de règle** est composée essentiellement d'un terme (qui contient la référence de l'identificateur de règle et un pointeur vers la queue de règle), de la liste des élisions admises, et de la ligne d'un tableau de description des métarègles qui peuvent être appliquées au nom composé. Dans le cas où certaines métarègles nécessitent des paramètres, la tête de règle contient un pointeur vers la liste des termes qui servent à les paramétrer, dans l'ordre de leur apparition dans la ligne du tableau. La tête de règle devient, en analyse, la tête d'une liste de règles: la règle de base et toutes celles qui pourront être produites à partir de celle-ci, au moyen de métarègles ou d'élisions. Cette liste n'apparaît pas *in extenso* dans le lexique-grammaire, mais elle est construite au cours de l'analyse.

Les champs d'une tête de règle sont:

- la référence,

- la liste des élisions et la liste des métarègles applicables,

- un pointeur vers la queue de règle et un vers la liste des paramètres des métarègles,

4: La liste des têtes est suivie par la liste des **termes des règles**, chaque terme portant les caractéristiques lexicales décrites dans l'énoncé la règle, et deux pointeurs relatifs permettant de passer aux termes suivants (fils et frère). Il s'agit de pointeurs relatifs, pour qu'ils puissent garder la même valeur en mémoire et sur fichier.

Les champs d'un terme de règle sont:

- la référence s'il s'agit d'un mot. Pour un non terminal, qui représente l'appel d'une autre règle, on note la valeur du pointeur dans le fichier vers le mot simple porteur, et le rang parmi les règles de ce mot,

- le type du noeud: liste, concaténation, mot simple ou identificateur de règle,

- un pointeur relatif vers le fils du terme dans l'arborescence et un pointeur vers le frère,

- la description des caractéristiques lexicales, subdivisée en quatre rubriques: catégorie lexicale, genre, nombre et forme; Elles sont décrites selon trois critères que nous préciserons, au 11.6., lors de la présentation de l'unification: une contrainte sur la valeur, une substitution et un numéro d'accord éventuel.

5: Cette séquence est terminée par une suite de six zéros qui ferme l'unité lexicale avant de passer à la suivante. Avant toute sauvegarde d'une unité lexicale, on s'assure qu'elle ne comporte pas de groupe de six zéros à l'intérieur, pour que ce groupe joue son rôle de séparateur, et que l'on puisse se repérer dans le fichier en cas de destruction partielle des données.

Figure 14, nous avons pris le mot simple *vie*, auquel sont associées les deux règles suivantes:

```
<assurance vie> -> {N, !1, !1} assurance{N, f, v, n} vie{N, f, s, n};
<espérance de vie>
    -> {N, !1, !1} espérance{N, f, v, n} DE{P, v, v, n} vie{N, f, s, n};
```

Une métarègle, qui nécessite un paramètre, opère sur la première règle. Il s'agit de celle qui, au nom composé *assurance vie*, associe la forme verbale *assurer sur la vie*. Elle comporte le verbe *assurer*, comme paramètre. Ces notions sont précisées en II.2.

Cas des formes canoniques multiples

Certains mots simples nécessitent plusieurs formes canoniques pour décrire toutes leurs flexions. Par exemple, le verbe *percevoir* apparaît deux fois dans le lexique des mots simples, sous sa forme canonique principale: *perc* et sous sa forme canonique annexe: *perç*. Tous les noms composés sont rattachés à la forme canonique principale.

En analyse, lorsque l'on rencontrera une forme fléchie issue d'une forme canonique annexe, il faudra retrouver la forme canonique principale pour avoir accès aux noms composés qui sont attachés. Pour que cette démarche soit possible, on fait suivre le terme attaché à une forme canonique annexe de la forme canonique principale, comme sur la figure 15.

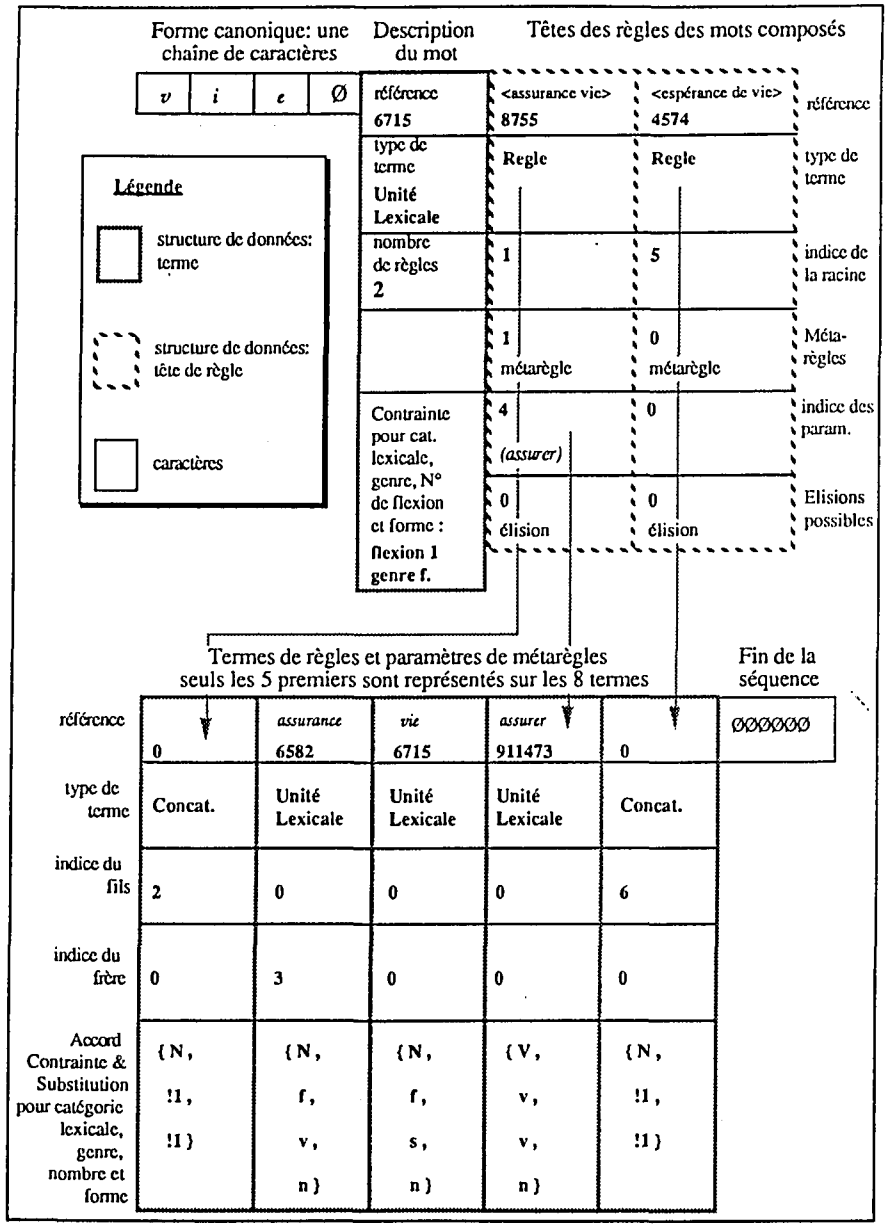


Figure 14: mots simples et noms composés rattachés

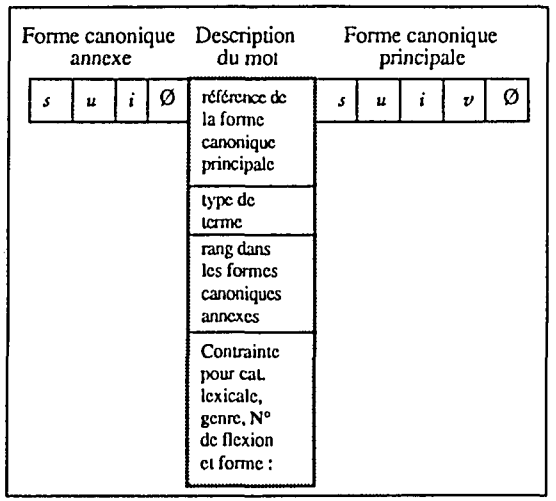


Figure 15: forme canonique annexe et forme canonique principale

d Chargement des règles en analyse

Nous rappelons les étapes du début de l'analyse pour préciser la façon dont les règles décrivant les noms composés sont chargées en mémoire.

Les mots de la phrase courante sont défléchis pour trouver, dans le lexique, les formes canoniques qui admettent le mot courant parmi leurs formes fléchies (voir I.1.c). Pour chaque forme canonique candidate, on la relève avec ses règles associées, en utilisant un mécanisme d'allocation, et en transformant des pointeurs numériques sur fichier en pointeurs réels en mémoire. Si au moins une de ses flexions est égale au mot cherché, on conserve les règles associées en mémoire. Dans le cas contraire, on désalloue toutes les règles associées à la forme canonique potentielle.

Comme le présente la figure 16; après lecture d'une phrase, la mémoire est ainsi organisée:

- Chaque mot fléchi pointe vers le premier homographe qui lui correspond s'il en existe un. Les homographes sont chaînés entre eux par les frères.
- Chaque homographe pointe vers la première règle qui lui est associée, et les règles sont chaînées entre elles. A chaque phrase, on conserve un pointeur sur la première règle chargée. C'est par ce pointeur que l'on accédera à la liste des règles pour tenter de les reconnaître successivement.
- Il n'y a pas d'accès direct, ni aux mots fléchis du lexique, ni aux termes des règles, car cela n'est pas nécessaire. L'accès aux mots fléchis se fait via le pointeur qui est dans les mots lus de la phrase. L'accès aux termes des règles se fait via les têtes des règles.

Exemple

Pour illustrer le chargement des règles de noms composés en mémoire, nous donnons un exemple: l'analyse de la phrase *durée de vie*. Nous supposons que le mot simple *durée* a une seule règle rattachée <durée de vie>, et que *vie* a <assurance vie> et <espérance de vie>.

Mot N°1: Pour le mot *durée*, on trouve trois formes canoniques potentielles après désuffixage: *durée, duré, dur*.

- La première est trouvée dans le lexique: le nom féminin *durée*. Il est relevé avec la règle qui est rattachée:

```
<durée de vie> -> {N, !1, !1} durée{N, f, v, n} DE{P, v, v, n}
                              vie{N, f, s, n};
```

La flexion singulier étant égale au mot cherché, la règle est conservée, et on trouve le premier homographe de *durée*.

- La deuxième forme canonique potentielle, *duré*, n'est pas rencontrée dans le lexique.

- La troisième, *dur*, a deux occurrences: le verbe *durer* de forme canonique *dur* et l'adjectif *dur*. La flexion participe passé, féminin singulier du verbe est égale au mot, le mot fléchi correspondant est chaîné au premier homographe déjà obtenu. Il n'y a pas de règle rattachée au verbe. L'adjectif *dur* n'a pas de flexion égale au mot cherché. Les règles, qui lui sont rattachées et qui avaient été chargées, sont désallouées.

Mot N°2: Le mot *de* est trouvé dans le dictionnaire en mémoire des mots usuels, avec un seul homographe, chaîné à la flexion du verbe *durer*.

Mot N°3: A *vie* sont associés un homographe (le nom féminin singulier) et les deux règles suivantes:

```
<assurance vie> -> {N, !1, !1} assurance{N, f, v, n} vie{N, f, s, n};
<espérance de vie> ->
        {N, !1, !1} espérance{N, f, v, n} DE{P, v, v, n} vie{N, f, s, n};
```

Elles sont chaînées à la première règle qui avait été chargée sur le mot *durée*.

La figure 16 représente la mémoire après la recherche des flexions des mots de la phrase.

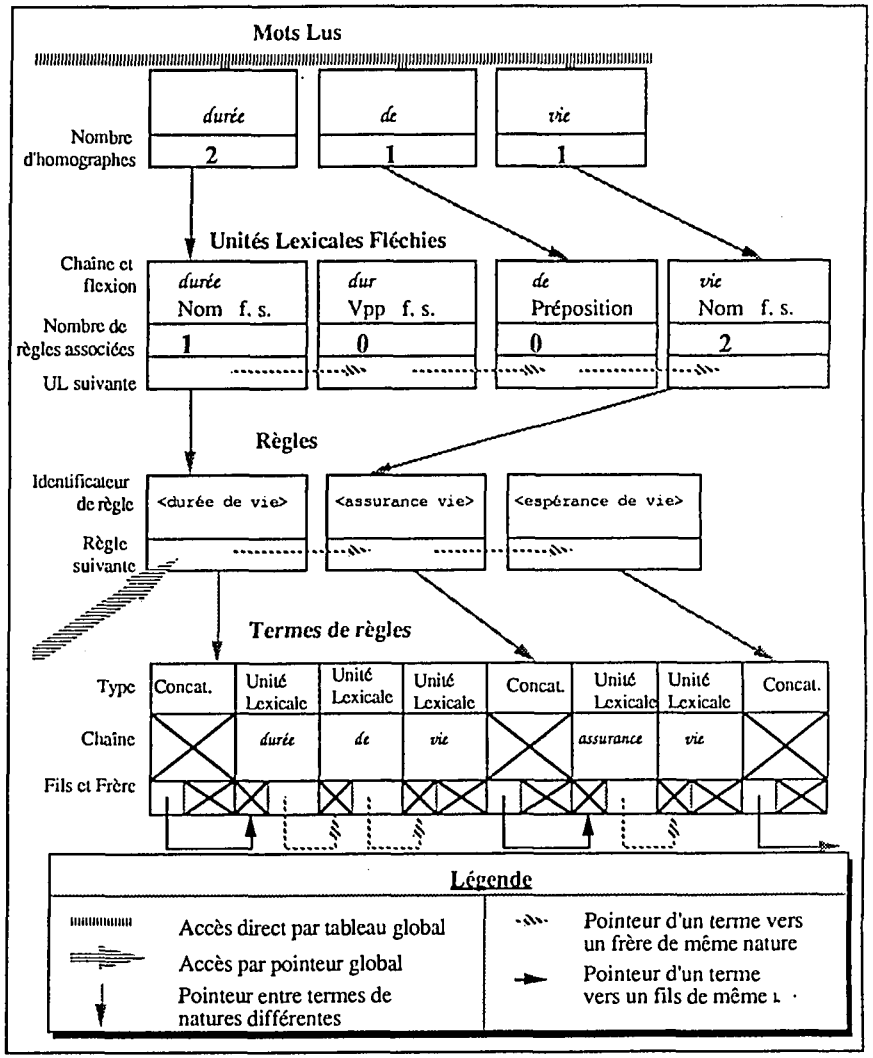


Figure 16: chargement en mémoire des règles associées aux mots fléchis lus dans la phrase
seuls 8 termes sur 11 sont représentés en dernière ligne.

e Utilisation des règles en analyse : réécriture et unification

On suppose, dans cette partie, que la phase lexicale de l'analyseur est effectuée. Les mots d'une phrase ont été défléchis et tous les homographes possibles ont été reconnus, les règles qui les accompagnaient ont été chargées. Grâce au procédé de lexicalisation (voir I.3), ce sont les seules règles pertinentes à reconnaître. Il s'agit alors d'analyser toutes les règles chargées, à l'aide des deux mécanismes de **réécriture** et **d'unification** qui sont repris de la programmation en logique et du langage Prolog.

La syntaxe d'énonciation des règles que nous avons définie en annexe III.1, permet de représenter chaque règle comme une arborescence de termes dont les noeuds sont des choix ou des concaténations et dont les feuilles sont des éléments lexicaux ou des identificateurs représentant la tête d'une autre règle. La **réécriture** est le mécanisme général de réécriture des règles en programmation logique qui consiste, à partir d'un but donné, à chercher à réécrire récursivement tous les termes de son arborescence, en substituant à tout identificateur, l'arborescence de la règle qu'il désigne.

- Une concaténation est vraie si et seulement si tous les termes, qui la composent, sont vrais successivement, alors qu'une liste est vraie si et seulement si un des éléments, qui la composent, est vrai.

-Un mot simple est vrai si et seulement si le mot suivant de la phrase correspond à une flexion de ce mot simple.

Le parcours de l'arbre se fait en préordre pour connaître le type d'un noeud avant de parcourir les branches qui en sont issues. En effet, la stratégie d'analyse est différente pour une concaténation, où l'on souhaite reconnaître séquentiellement toutes les branches, de celle d'une liste, où l'on cherche à reconnaître au moins une des branches issues du noeud considéré.
Afin d'obtenir toutes les solutions possibles à l'analyse de la phrase, en fonction des règles proposées, nous produisons, pour chaque liste, tous les termes qui peuvent être réécrits et non pas seulement le premier.
Nous assimilons une tête de règle à un choix entre la règle de base et toutes les règles pouvant être produites à partir de celle-ci au moyen des métarègles acceptées et des élisions possibles. La tête de règle ne porte pas d'informations lexico-syntaxiques pour l'unification, elle joue le rôle de tête de liste pour pouvoir revenir sur les différents choix que constituent les règles modifiées.
De même, les mots simples dans les termes de règles peuvent être considérées comme des listes: celles d'un choix entre tous les homographes associés à un mot donné dans le texte, et admettant la même référence que le terme courant.

Lors de la compilation, nous avons rattaché à chaque terme, liste ou concaténation un ensemble de contraintes sur les variables des caractéristiques lexico-syntaxiques. Lorsque l'analyse d'une règle est achevée, on vérifie l'**unification** de ces variables. On s'assure que les contraintes sont respectées, en remontant, dans l'arbre d'analyse, les valeurs trouvées sur les mots fléchis.

Cette **unification** n'est pas coroutinée comme dans le langage Prolog (vérifiée au fur et à mesure de l'analyse), mais elle est effectuée uniquement en cas de succès de la réécriture. Ce report en fin d'analyse présente des aspects positifs et négatifs:

- un gain de temps dans le cas où la règle à réécrire est bonne, puisqu'il n'y a pas de temps perdu en vérification inutile,

- la possibilité de détecter en fin d'analyse des erreurs telles que des fautes d'accord. Pour un vérificateur orthographique, il est intéressant de mener une telle analyse au bout en donnant les erreurs qui empêchent l'unification finale.

- Une perte de temps si on est sur une séquence de mots, égale à celle de la règle en cours, aux différences de contraintes près. Dans ce cas, les contraintes non vérifiées, n'empêchent pas la poursuite de la réécriture; et, on aboutira plus tardivement sur un cas d'erreur que si l'on avait tenté l'unification au fur et à mesure.

Nous avons choisi cette option, car nous pensons que ce dernier inconvénient est mineur. Dans le langage naturel, il est rare de trouver des règles différentes ayant une longue séquence en commun, avec des différences de contraintes sur les valeurs des caractéristiques lexico-syntaxiques. En production, le dernier inconvénient peut être rédhibitoire, et, le coroutinage est alors souhaitable.

Réécriture

Afin de suivre l'analyse d'une règle, nous allons prendre l'exemple de la règle R, faisant appel à la règle S, telles qu'elles sont décrites ci-dessous:

```
R:  <panneau de comptage>
        -> {N, !1, !1} panneau{N, m, v, n} DE{P, v, v, n}
                       <comptage électrique>{N, m, s};
S:  <comptage électrique>
        -> {N, !1, !1} comptage{N, m, s, n}
             ({A, a1 !, a1 !, !} électrique{A, v, v, n}
                                 + électronique{A, v, v, n});
```

La règle R ne peut être transformée au moyen d'une métarègle ou d'une élision.

La règle S peut être transformée au moyen d'une métarègle, on obtient alors:

```
S1: <comptage électrique>1
        -> {N, !1, !1} comptage{N, m, s, n}
                DE{P, v, v, n} LE{P, v, v, a} électricité{A, a3, a3, n};
```

et elle admet une élision qui donne:

```
S2: <comptage électrique>2 -> {N, !1, !1} comptage{N, m, s, n};
```

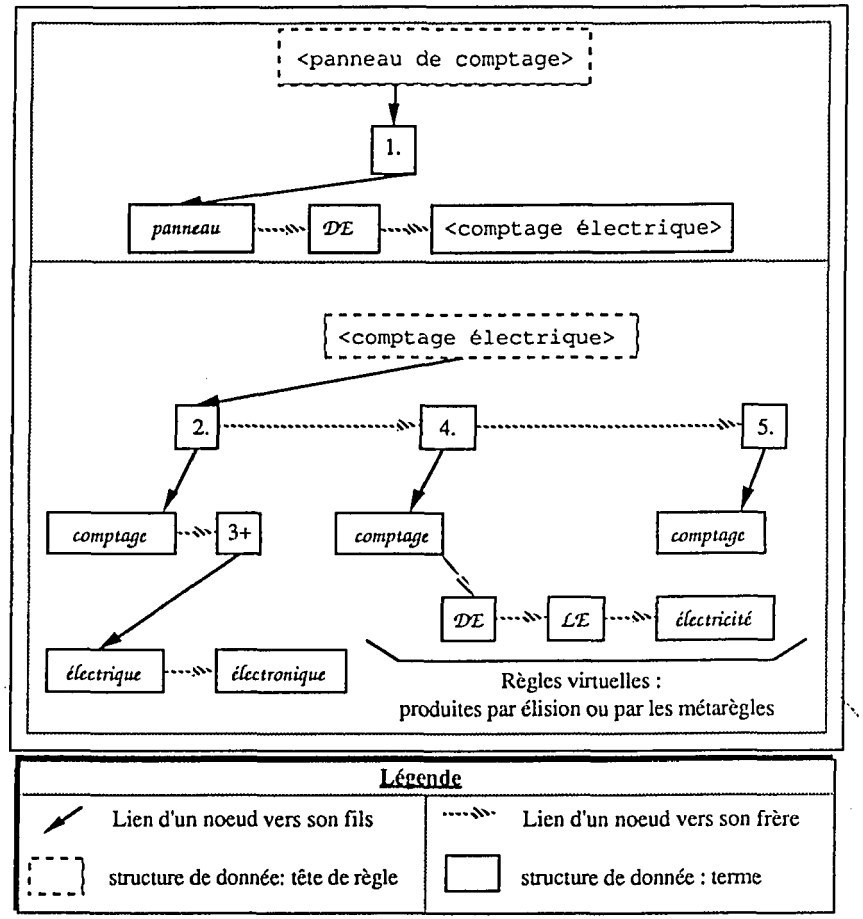


Figure 17: arbres des règles <panneau de comptage> et <comptage électrique>

Sur la figure 17, sont représentés les arbres obtenus après compilation des règles **R** et **S**. Les noeuds sont essentiellement de deux types:

- les noeuds de choix tels que **3+**,
- les noeuds de concaténation tels que **1.**, **2.**, **4.** ou **5.**

Les noeuds de **concaténation** servent à faire avancer l'analyse: un noeud de concaténation est réécrit si et seulement si tous ses fils le sont.

Les noeuds de **choix** servent à revenir sur des choix pour les essayer successivement (backtracker). Ils sont utilisés en cas d'échec pour revenir sur des choix antérieurs, ou en cas de succès pour chercher une autre solution.

Un **non terminal** (un terme d'appel de règle) tel que <comptage électrique> peut être considéré comme un choix entre trois règles: la règle de base **S**, la règle S_1, obtenue à partir de S au moyen de la métarègle autorisée et la règle S_2, obtenue par élision du deuxième terme de la queue de règle. Afin de représenter cette possibilité de choix, on crée les trois fils du noeud au terme <comptage électrique> de la règle **R**, comme nous l'avons montré sur la figure 17. On aurait donc pu écrire la règle S ainsi:

```
S: <comptage électrique>
    -> ([!, !, !]
       {N, !1, !1} comptage{N, m, s, n}
             {[A, a1 !, a1 !, !] électrique{A, v, v, n}
                                 + électronique{A, v, v, n}}
          + {N, !1, !1} comptage{N, m, s, n}  DE{P, v, v, n}
                        LE{P, v, v, a} électricité{A, a3, a3, n}
          + {N, !1, !1} comptage{N, m, s, n}];
```

En réécrivant chaque noeud non terminal (ici <comptage électrique>), par l'arbre qui lui correspond, on obtient l'arbre unique de la figure 18 pour la règle <panneau de comptage>.

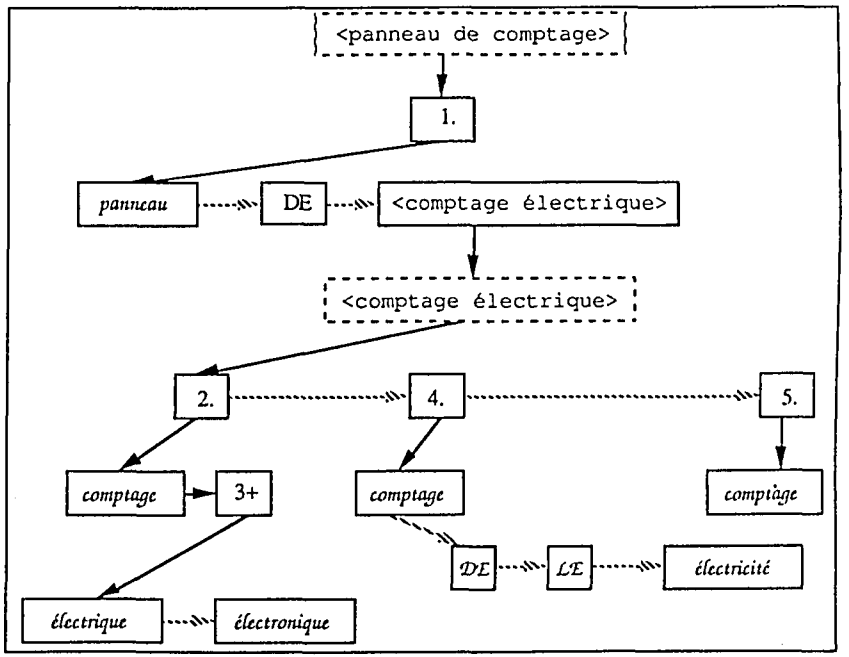


Figure 18: arbre de la règle <panneau de comptage> après remplacement des non terminaux par les arbres correspondants

Un terme de règle, représentant un **mot simple** tel que `électrique` dans la règle S, peut être, lors de l'exécution, considéré comme un choix entre les différents homographes du texte qui ont pu être relevés avec la même référence que lui.
Par exemple, si le mot *électrique* (respectivement *électriques*) se trouve dans le texte, on associe deux homographes au terme `électrique` de S: l'adjectif féminin singulier (respectivement féminin pluriel) et l'adjectif masculin singulier (respectivement masculin pluriel).
Au contraire, si le mot cherché ne figure pas dans le texte analysé, le terme `électrique` est la tête d'une liste de choix de zéro terme, et l'analyseur revient en arrière sur un échec.

Mécanisme de retour en arrière

On tente la réécriture d'une règle, en réécrivant, chaque noeud non terminal, par l'arbre qui lui correspond. Elle réussit si on effectue un parcours complet de l'arborescence, en ne parcourant qu'un fils de chaque liste ou règle, et en parcourant tous les fils de chaque concaténation.

- En cas d'échec sur une feuille, on revient au premier choix incomplet trouvé en dépilant les termes qui ont été réécrits.
- En cas de succès, c'est à dire lorsque tous les termes de toutes les concaténations ont été réécrits, on affiche le résultat en indiquant si l'unification est vérifiée. Puis, on cherche, comme dans le cas de l'échec un choix incomplet pour revenir en arrière, afin de donner toutes les solutions possibles.

Description des tableaux des figures 19 et 20, détaillant le mécanisme de réécriture, pour la règle `<panneau de comptage>`, avec le texte: *panneau de comptage électrique.*

Nous désignons les termes de règle par des lettres minuscules:

`panneau -> a`	`DE (dans R) -> b`	`<comptage électrique> -> s`
`comptage -> c`	`électrique -> d`	`électronique -> e`
`DE (dans S) -> f`		

les têtes de règles par des majuscules:

`<panneau de comptage> -> S`	`<comptage électrique> -> R`

les mots du texte par des majuscules:

panneau -> A	*de* -> B	*comptage* -> C
électrique -> D	(la flexion du masculin singulier est notée D1, et, celle du féminin singulier, D2).	

Le tableau de la figure 19 est composé, dans sa partie droite, d'une description de la pile d'exécution lors de la réécriture. Nous n'utilisons pas de fonction récursive, mais nous gérons explicitement la pile des termes qui sont réécrits. Les colonnes 2, 3, 4 et 5 décrivent les types et les valeurs des termes empilés à chaque étape.

A chaque terme ajouté, on associe une liste de pointeurs vers ses fils, qui ne sont pas encore dans la pile. Lorsque ceux-ci sont recopiés dans la pile d'exécution, les anciens pointeurs sont sauvegardés et remplacés par l'adresse du terme dans la pile. Si le terme empilé est un mot du texte, il n'a pas de fils. L'analyseur doit alors revenir en arrière pour rechercher la première concaténation incomplète.

- S'il n'y en a pas, et si l'unification est vérifiée, l'analyseur produit comme résultat d'analyse la chaîne courante. C'est ce qui se passe aux étapes 14 et 25. Ensuite, l'analyseur revient en arrière à la recherche d'un choix incomplet, s'il en trouve un, il recommence à ce niveau, sinon l'analyse s'arrête.
- En 15 où l'unification échoue, l'analyseur ne produit pas de résultat, et il poursuit comme dans le cas précédent.

Lorsque le mot suivant dans le texte n'a pas la même référence que le mot simple courant, comme en 16 ou 21, alors l'analyseur revient en arrière à la recherche d'un choix incomplet comme précédemment.

On remarquera que la réécriture d'un non terminal se fait en empilant successivement:

- le terme identificateur de la règle,
- un noeud de type Règle qui est composé d'une tête de règle dont le pointeur fils pointe vers la queue de règle et qui représente le choix entre la règle de base et toutes les règles qui pourront éventuellement être crées à partir de celle-ci. Le terme de la tête de règle a pour type Règle Normale s'il pointe vers la règle d'origine, ou Métarègle ou Règle Elision selon qu'il pointe vers une des règles qui a été produite par une métarègle ou par une élision.
- Le terme de queue de règle est, dans la majorité des cas, une concaténation.

La règle `<comptage électrique>` admet une transformation par une métarègle et par une élision. Aux deux retours en arrière, sur la tête de règle, correspondants aux choix d'indices 2 et 3, on produit, au choix 2, la transformation de S par la métarègle et, au 3, la transformation de S par l'élision. Voir au chapitre 12. pour les mécanismes de construction.

Le tableau de la figure 20 décrit, dans sa partie droite, pour chaque bloc de la pile, l'indice du fils parcouru. Le bloc écrit en caractères gras est le bloc dont la valeur de l'indice a été modifié. Il n'y en a qu'un seul par étape. Il correspond au terme père du terme empilé.
Si un terme est un mot du texte, il est noté F(euille). Les termes soulignés sont des termes de **choix** tels que les listes ou les têtes de règles. Les autres sont des **concaténations**, ou considérés comme tels s'ils n'ont qu'un seul fils (dans ce cas, il y a équivalence entre choix ou concaténation).

Dans la partie gauche de ce tableau la colonne 2 indique la chaîne de mots qui a été reconnue à chaque étape. Le symbole $ indique la chaîne vide. Les deux chaînes encadrées correspondent à un succès de l'analyse. L'analyseur reconnaît le nom composé complet *panneau de comptage électrique* (étape 14), et celui obtenu par élision de `<comptage électrique>`: *panneau de comptage*. Les colonnes 3, 4 et 5 commentent le mécanisme de retour en arrière:

Numéro d'étape	Noeuds		Types de noeuds		Pile des blocs créés en exécution														
	T1	Termes suivants (pteurs)	T1	Termes suivants (pointeurs)	Hauteur	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	R	1.	Règle Normale	Concat	1	R													
2	1.	a b s	Concat	Unit. Lex. / Unit. Lex. / Terme Règle	2	R	1.												
3	a	A	Unité Lexicale	Mot Lu	3	R	1.	a											
4	A		Mot Lu		4	R	1.	a	A										
5	b	B	Unité Lexicale	Mot Lu	5	R	1.	a	A	b									
6	B		Mot Lu		6	R	1.	a	A	b	B								
7	s	S1 S2 S3	Terme Règle	Règle Norm. / Règle Méta / Règle Elis.	7	R	1.	a	A	b	B	s							
8	S1	2.	Règle Normale	Concat	8	R	1.	a	A	b	B	s	S1						
9	2.	c 3+	Concat	Unité Lexicale / Liste	9	R	1.	a	A	b	B	s	S1	2.					
10	c	C	Unité Lexicale	Mot Lu	10	R	1.	a	A	b	B	s	S1	2.	c				
11	C		Mot Lu		11	R	1.	a	A	b	B	s	S1	2.	c	C			
12	3+	d e	Liste	Unité Lexicale / Unité Lexicale	12	R	1.	a	A	b	B	s	S1	2.	c	C	3+		
13	d	D1 D2	Unité Lexicale	Mot Lu / Mot Lu	13	R	1.	a	A	b	B	s	S1	2.	c	C	3+	d	
14	D1		Mot Lu		14	R	1.	a	A	b	B	s	S1	2.	c	C	3+	d	D1
15	D2		Mot Lu		14	R	1.	a	A	b	B	s	S1	2.	c	C	3+	d	D2
16	e		Unité Lexicale		13	R	1.	a	A	b	B	s	S1	2.	c	C	3+	e	
17	S2	4.	Règle Méta	Concat	8	R	1.	a	A	b	B	s	S2						
18	4.	c f g h	Concat	Un. Lex / Un. Lex / Un. Lex / Un. Lex	9	R	1.	a	A	b	B	s	S2	4.					
19	c	C	Unité Lexicale	Mot Lu	10	R	1.	a	A	b	B	s	S2	4.	c				
20	C		Mot Lu		11	R	1.	a	A	b	B	s	S2	4.	c	C			
21	f		Mot Lu		12	R	1.	a	A	b	B	s	S2	4.	c	C	f		
22	S3	5.	Règle Elision	Concat	8	R	1.	a	A	b	B	s	S3						
23	5.	c	Concat	Unité Lexicale	9	R	1.	a	A	b	B	s	S3	5.					
24	c	C	Unité Lexicale	Mot Lu	10	R	1.	a	A	b	B	s	S3	5.	c				
25	C		Mot Lu		11	R	1.	a	A	b	B	s	S3	5.	c	C			

← Retour en arrière sur succès ⟿ Retour en arrière sur échec

Figure 19: description de la pile d'analyse: parcours de l'arbre des règles R et S.

Numéro d'étape	Chaîne reconnue	Niveau de backtrack possible	Mot dont l'absence provoque le backtrack	Etapes avec résultat d'analyse. Nb de blocs dépilés en backtrack	Rang du choix courant (non souligné) ou rang du terme dans la concaténation (souligné) ou feuille (F). Le noeud, dont le rang est modifié, est en gras.														
					Hauteur	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	$	0			1	0/1													
2	$	0			2	**1/1**	0/3												
3	$	0			3	1/1	**1/3**	0/1											
4	A	0	A		4	1/1	1/3	**1/1**	F										
5	A	0			5	1/1	**2/3**	1/1	F	0/1									
6	AB	0	B		6	1/1	2/3	1/1	F	**1/1**	F								
7	AB	7			7	1/1	**3/3**	1/1	F	1/1	F	0/3							
8	AB	7			8	1/1	3/3	1/1	F	1/1	F	1/3	0/1						
9	AB	7			9	1/1	3/3	1/1	F	1/1	F	1/3	**1/1**	0/2					
10	AB	7			10	1/1	3/3	1/1	F	1/1	F	1/3	1/1	**1/2**	0/1				
11	ABC	7	C		11	1/1	3/3	1/1	F	1/1	F	1/3	1/1	1/2	**1/1**	F			
12	ABC	12			12	1/1	3/3	1/1	F	1/1	F	1/3	1/1	**2/2**	1/1	F	0/2		
13	ABC	13			13	1/1	3/3	1/1	F	1/1	F	1/3	1/1	2/2	1/1	F	1/2	0/2	
14	AB CD	13	D	1 bloc dépilé	14	1/1	3/3	1/1	F	1/1	F	1/3	1/1	2/2	1/1	F	1/2	1/2	F
15	AB CD	12	D	2 blocs dépilés	14	1/1	3/3	1/1	F	1/1	F	1/3	1/1	2/2	1/1	F	1/2	2/2	F
16	ABC	7	E	6 blocs dépilés	13	1/1	3/3	1/1	F	1/1	F	1/3	1/1	2/2	1/1	F	2/2	0/0	
17	AB	7			8	1/1	3/3	1/1	F	1/1	F	2/3	0/1						
18	AB	7			9	1/1	3/3	1/1	F	1/1	F	2/3	**1/1**	0/4					
19	AB	10			10	1/1	3/3	1/1	F	1/1	F	2/3	1/1	1/4	0/1				
20	ABC	7	C		11	1/1	3/3	1/1	F	1/1	F	2/3	1/1	1/4	1/1	F			
21	ABC	7	F	5 blocs dépilés	12	1/1	3/3	1/1	F	1/1	F	2/3	1/1	2/4	1/1	F	0/0		
22	AB	0			8	1/1	3/3	1/1	F	1/1	F	3/3	0/1						
23	AB	0			9	1/1	3/3	1/1	F	1/1	F	3/3	**1/1**	0/1					
24	AB	0			10	1/1	3/3	1/1	F	1/1	F	3/3	1/1	1/1	0/1				
25	ABC	0	C	11 blocs dépilés	11	1/1	3/3	1/1	F	1/1	F	3/3	1/1	1/1	1/1	F			

Figure 20: description des choix ou des rangs dans les concaténations

Si on fait fonctionner l'analyseur en **production**, le schéma diffère à partir des mots simples, puisque le problème de la présence du mot dans le texte ne se pose plus. Dans ce cas, on associe à chaque mot simple autant de flexions que l'on peut en produire pour la catégorie lexicale considérée, laissant le soin à l'unification de rejeter les flexions qui ne respectent pas les accords et les contraintes. Les tableau diffèrent à partir de l'étape 16, puisque au terme `électrique` qui est un adjectif sont associées les 4 flexions possibles. Seule la flexion masculin singulier vérifie l'unification, mais la flexion masculin pluriel, convient lorsqu'un retour en arrière a remplacé *comptage* par *comptages.*

La pile d'exécution

Pour faire le lien entre réécriture et unification, nous devons décrire plus en détail la pile d'exécution qui sert à stocker les termes au fur et à mesure de leur réécriture. Elle permet de savoir où en est le parcours des concaténations. Au retour en arrière, elle sert à restaurer l'environnement dans son état initial et à poursuivre les choix incomplets.

Les termes sont stockés dans des blocs, de taille de base fixe, suivis d'un nombre de pointeurs égal au nombre de fils du terme représenté. Ces blocs sont décrits par la figure 21. Ils sont composés d'un terme principal T_1 et d'une liste de pointeurs T_2, T_3... Si T_1 est un mot simple d'une règle, T_2, T_3... sont les adresses des homographes du terme courant relevé dans la phrase. Si T_1 est un non terminal (un terme d'appel de règle), T_2 pointe vers la tête de règle. Si T_1 est une tête de règle, il n'y a pas de terme T_2 puisque T_1 contient déjà lui-même un pointeur vers la queue de règle. Dans les autres cas (si T_1 est une liste ou une concaténation) T_2, T_3, T_4,... sont des pointeurs vers les fils de T_1 dans l'arbre d'analyse: son fils et les frères de celui-ci.

Le tableau de la figure 22 indique la composition des blocs, dans un cas particulier. Il décrit l'analyse de la règle `<panneau de comptage>`, énoncée précédemment.

Les colonnes 2 et 3 rappellent la valeur de chaque terme T_1 empilé, ainsi que celle des pointeurs vers les fils de T_1 qui sont nécessaires. Dans les colonnes 4 et 5 nous indiquons la mise en oeuvre informatique. En langage C, tous les pointeurs ont la même taille et qu'un même pointeur peut adresser des structures différentes par des conversions de type. C'est pourquoi, dans la colonne 5, nous ne précisons plus la nature des objets adressés.

La colonne 6 indique le type donné au bloc, qui est généralement celui du terme T_1. La colonne 7 donne la nature du bloc: choix, concaténation ou feuille. Les colonnes 8 à 12 permettent de situer un terme par rapport à ses frères, ce qui est utile pour les accords dans le mécanisme d'unification.

En dernière colonne, nous avons tracé les **liens** que nous remontons pour réaliser l'**unification**. La simultanéité des liens dans un choix est seulement apparente, car, à une étape donnée de l'unification, un seul des multiples liens qui peuvent remonter vers un noeud de type choix est effectivement présent. Vers une concaténation remontent autant de liens que de termes fils, ils sont mis bout à bout pour clarifier la présentation. Au paragraphe 11.6., nous détaillons le mécanisme de l'unification.

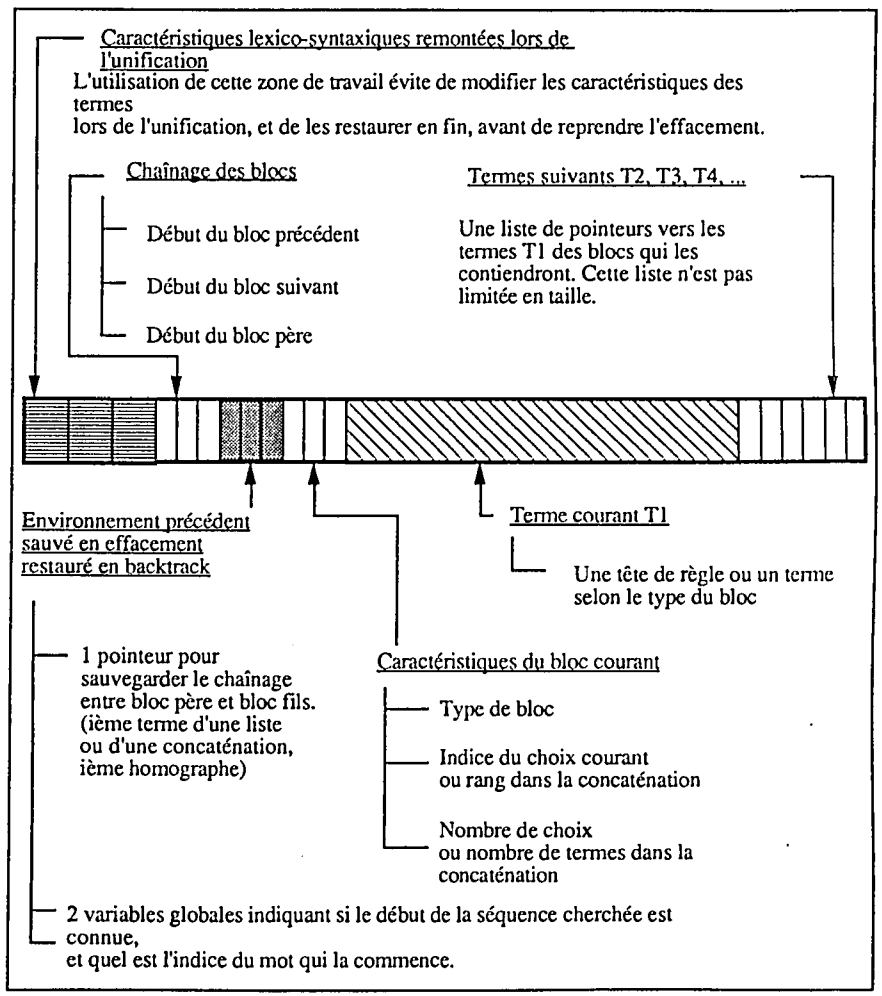


Figure 21: structure d'un bloc dans la pile d'exécution

Numéro d'étape	Noeuds T1	Noeuds Termes suivants (pteurs)	Types de noeuds T1	Types de noeuds Termes suivants (pointeurs)	Type du bloc courant Type utilisé en C	Liste, Concat. ou Feuille	N° bloc courant	N° bloc père	Rang du bloc dans sa fraternité	N° de l'étape introduisant le frère précédent	N° de l'étape introduisant le frère suivant	Chaines d'unification
1	R	1.	Tête de Règle	(R contient un pointeur)	_Regle	L	1	0	1/1			
2	1.	a b s	Terme	3 pointeurs	_Concat	C	2	1	1/1			
3	a	A	Terme	1 pointeur	_UniteLex	L	3	2	1/3		5	
4	A		Terme		_MotLu	F	4	3	1/1			
5	b	B	Terme	1 pointeur	_UniteLex	L	5	2	2/3	3	7	
6	B		Terme		_MotLu	F	6	3	1/1			
7	s	S1 S2 S3	Terme	3 pointeurs	_Terme Regle	L	7	2	3/3			
8	S1	2.	Tête de Règle	(S1 contient un pointeur)	_Regle Normale	C	8	7	1/3		17	
9	2.	c 3+	Terme	2 pointeurs	_Concat	C	9	8	1/1			
10	c	C	Terme	1 pointeur	_UniteLex	L	10	9	1/2		12	
11	C		Terme		_MotLu	F	11	10	1/1			
12	3+	d e	Terme	2 pointeurs	_Liste	L	12	9	2/2	10		
13	d	D1 D2	Terme	2 pointeurs	_UniteLex	L	13	12	1/2		16	
14	D1		Terme		_MotLu	F	14	13	1/2		15	
15	D2		Terme		_MotLu	F	14	13	2/2	14		
16	e		Terme		_UniteLex	L	13	12	2/2	13		
17	S2	4.	Tête de Règle	(S2 contient un pointeur)	_Regle Meta	C	8	7	2/3	8	22	
18	4.	c f g h	Terme	4 pointeurs	_Concat	C	9	8	1/1			
19	c	C	Terme	1 pointeur	_UniteLex	L	10	9	1/4	21		
20	C		Terme		_MotLu	F	11	10	1/1			
21	f		Terme		_UniteLex	L	12	9	2/4	19	*	
22	S3	5.	Tête de Règle	(S3 contient un pointeur)	_Regle Elision	C	8	7	3/3	17		
23	5.	c	Terme	1 pointeur	_Concat	C	9	8	1/1			
24	c	C	Terme	1 pointeur	_UniteLex	L	10	9	1/1			
25	C		Terme		_MotLu	F	11	10	1/1			

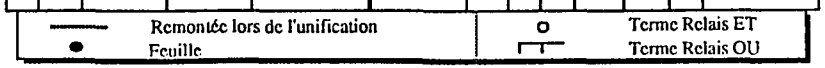


Figure 22: description des chaines d'unification

Unification

L'unification est la vérification, en cas de succès de réécriture d'une règle, des contraintes sur les caractéristiques lexico-syntaxiques des termes. Il s'agit d'une restriction de l'unification générale de la programmation en logique à un modèle restreint pour l'adapter aux besoins de cet étude.

La dernière colonne du tableau de la figure 22 indique les liens qui existent entre les différents blocs empilés au cours de l'analyse, lorsque l'on fait "remonter" les valeurs des caractéristiques lexico-syntaxiques (catégorie lexicale, genre, nombre, forme) des feuilles (les mots de la phrase) vers la racine (la règle initiale). L'unification correspond à un parcours en post-ordre de l'arbre d'analyse avec propagation des valeurs des noeuds fils vers le noeud père ayant trois aspects: la contrainte, la substitution et l'accord.

Exemple

Afin de mieux préciser ce mécanisme, nous allons suivre le cheminement de l'unification dans un cas particulier. Nous supposerons que la règle `<LISTE durée de vie des composants>` suivante a été réécrite sur la séquence de mots *durée de vie des [de les] diodes.*

```
<LISTE durée de vie des composants> ->
    {N, !1, !1} <durée de vie>{N, f, v} de{P, v, v, n}
                <LISTE déterminants>{v, v, v, v}
                <LISTE composants>{N, a3, a3 p, v};
<durée de vie> ->
    {N, !1, !1 durée{N, f, v, n} de{P, v, v, n} vie{N, f, s, n};
<LISTE déterminants> ->
    ({!, !, !, !} <>{Dd, v, v, v} + <>{Di, v, v, v} + <>{Dc, v, v, v});
<LISTE composants> ->
    ({N, !, !, !} transistor{N, m, v, v} + diode{N, m, v, v}
                + résistance{N, m, v, v});
```

Au I.2.b, nous avons présenté la description des caractéristiques lexico-syntaxiques des noms composés, selon trois rubriques: contrainte, substitution et accord. Nous les reprenons, pour le transférer au système informatique.

La contrainte est l'attente d'une valeur précise pour un terme.
S'il s'agit d'un élément terminal comme `vie{N, f, s, n}` dans `<durée de vie>`, le mot *vie* est attendu comme étant un nom féminin, singulier, de forme normale. De même, dans `<LISTE durée de vie des composants>`, on force le premier terme qui est un nom composé à être un nom féminin.

La substitution remonte la valeur d'un des fils vers le père. On vérifie que la valeur satisfait bien aux contraintes s'il y en a. Il existe 5 types de substitutions qui dépendent des types des noeuds, ces substitutions sont notées sur la figure 24.

(1): remontée automatique des valeurs d'un mot de la phrase vers l'unité lexicale de la règle avec vérification des contraintes éventuelles. L'absence de contrainte est notée par v.
(1'): mécanisme semblable substituant automatiquement les valeurs d'une tête de règle à celles du terme ayant appelé la règle et vérifiant également les contraintes.
(2): substitution éventuelle des valeurs du choix courant à celles de la tête si elles correspondent à une même valeur de la variable en !, sinon affectation de la nouvelle valeur proposée. La substitution est accompagnée d'une contrainte lorsque la variable ! est précédée d'une valeur.
(2'): mécanisme semblable au précédent, substitution éventuelle de la valeur d'un des termes de la concaténation. ce terme est précisé par son indice par (exemple !1). La variable peut également être précédée d'une contrainte sur la valeur.
(3): substitution automatique, sans possibilité de contrainte, entre la racine d'une queue de règle et la tête de règle.

L'accord: avant de remonter une valeur dans une concaténation, on doit s'assurer qu'il y a accord entre certains fils (entre certains syntagmes inclus dans un même syntagme). Cet accord est indiqué par l'indice du terme de la concaténation avec lequel la valeur doit être égale, par exemple **a1** pour l'accord avec le premier terme.

Syntaxe de description des caractéristiques lexico-syntaxiques

Afin d'énoncer précisément toutes les possibilités de description des caractéristiques lexico-syntaxiques d'un terme, le tableau de la figure 23 récapitule les écritures acceptables pour celui-ci. Il utilise les deux informations suivantes, sur le terme:

- son type: unité lexicale, identificateur d'appel d'une autre règle, tête de liste ou tête de concaténation,
- sa filiation: s'il est le fils d'une concaténation ayant des frères de rang inférieur, on peut envisager des contraintes d'accord.

La description est composée de trois colonnes:

- un mode de **contrainte** qui de permet bloquer l'unification, si la caractéristique donnée n'a pas la valeur attendue,
- un mode de **substitution** qui permet de calculer les caractéristiques du terme en fonction de celles de son fils,
- un mode d'**accord** qui permet de préciser les accords entre les termes d'une concaténation.

Afin d'alléger l'écriture, la substitution automatique, décrite par la lettre v, peut être omise si elle est accompagnée de contraintes. De même, lorsque le mode de substitution est précisé, l'absence de contrainte peut être représentée par v ou rien.

0 ou 1 valeur	Ecriture	0 ou 1 accord	Ecriture	1 mode de substitution	Ecriture
- absence de contrainte de valeur	v ou rien	- absence de contrainte d'accord	rien	- automatique pour les unités lexicales ou les termes d'appels de règle.	rien
- contrainte d'égalité sur une valeur	valeur	- si le terme est élément d'une concat.: contrainte d'accord avec un autre élément d'indice inférieur	ai i : indice d'accord	- substitution avec la valeur du terme retenu dans la liste - substitution avec la valeur du terme de rang i de la concaténation	! pour la subst., la valeur sinon !i i : indice du terme retenu dans la concat.

Les trois possibilités peuvent être simultanées

Figure 23: description des caractéristiques lexico-syntaxiques d'un terme

Description d'un exemple d'unification

Les figures 24 et 25 présentent l'unification.

Dans la figure 24, nous illustrons graphiquement le lien entre l'arbre d'analyse d'une structure lexicale donnée par une règle, et le parcours effectué pour la remontée des valeurs. Nous avons fait figurer l'arbre de l'analyse de la phrase *durées de vie de les diodes* à l'aide de la règle `<LISTE durée de vie des composants>` donnée au début de cette section.

Les chemins de **remontée** de l'unification sont représentés par des traits triples, accompagnés du numéro d'identification du type de substitution (**1, 1', 2, 2'** ou **3**) tels qu'ils ont été définis ci-dessus.

L'**accord** entre entre deux termes d'une concaténation, entre `<LISTE composants>` et `<LISTE déterminants>` ici, figure en trait plein horizontal identifié par la lettre A.

La figure 25 décrit le parcours en post-ordre de l'arbre d'analyse, afin de bien suivre la remontée des valeurs des caractéristiques (catégorie lexicale, genre, nombre et éventuellement forme). Dans les deux colonnes de droite, sont précisées les valeurs des caractéristiques attendues et celles remontées. On s'aperçoit que, pour l'exemple proposé, il n'y a jamais contradiction entre les contraintes et les valeurs trouvées, l'unification réussit donc. S'il y avait incompatibilité, l'unification ne serait pas vérifiée, et, en conséquence, l'analyse correspondante échouerait.

Dans le cas où l'on aurait proposé la séquence incorrecte *durées de vies de les diodes*, le parcours d'unification échouerait à l'étape 6 puisque les caractéristiques remontées seraient: **N, f, p, n ;** et celles attendues seraient: **N, f, s, n** avec une valeur du nombre différente. C'est le mécanisme de vérification des contraintes qui bloque l'unification, car la contrainte **s** (singulier) sur le nombre du terme *vie* n'est pas respectée.

La séquence: *durées de vie de la diodes* provoquerait une erreur à l'étape 23 de vérification de l'accord. L'accord entre les fils d'une concaténation ne se fait que lorsque l'unification a été réalisée avec succès sur tous les fils, et que la valeur de leurs caractéristiques est connue. De telles erreurs d'accord peuvent être indiquées à l'utilisateur du système informatique s'il le souhaite.

Notations

Dans la figure 24, nous notons **1.** le terme de type **concaténation** qui est la racine de la queue de règle de `<LISTE durée de vie des composants>`. De même, nous notons **2.** la racine de `<durée de vie>`, **3+** et **4+** les racines de type **liste** des règles `<LISTE déterminants>` et `<LISTE composants>`.

Pour les unité lexicales, nous avons:

```
durée -> f      de (dans <durée de vie>) -> c
vie -> h        de (dans <LISTE durée de vie des composants>)-> m
diode -> j
```

Pour les têtes de règles:

```
<durée de vie> -> R'
<LISTE durée de vie des composants> -> R
<LISTE déterminants> -> R"          <LISTE composants> -> R'"
```

Pour les termes d'appels des règles:

```
<durée de vie> -> B                 <LISTE déterminants> -> D
<LISTE composants> -> E
```

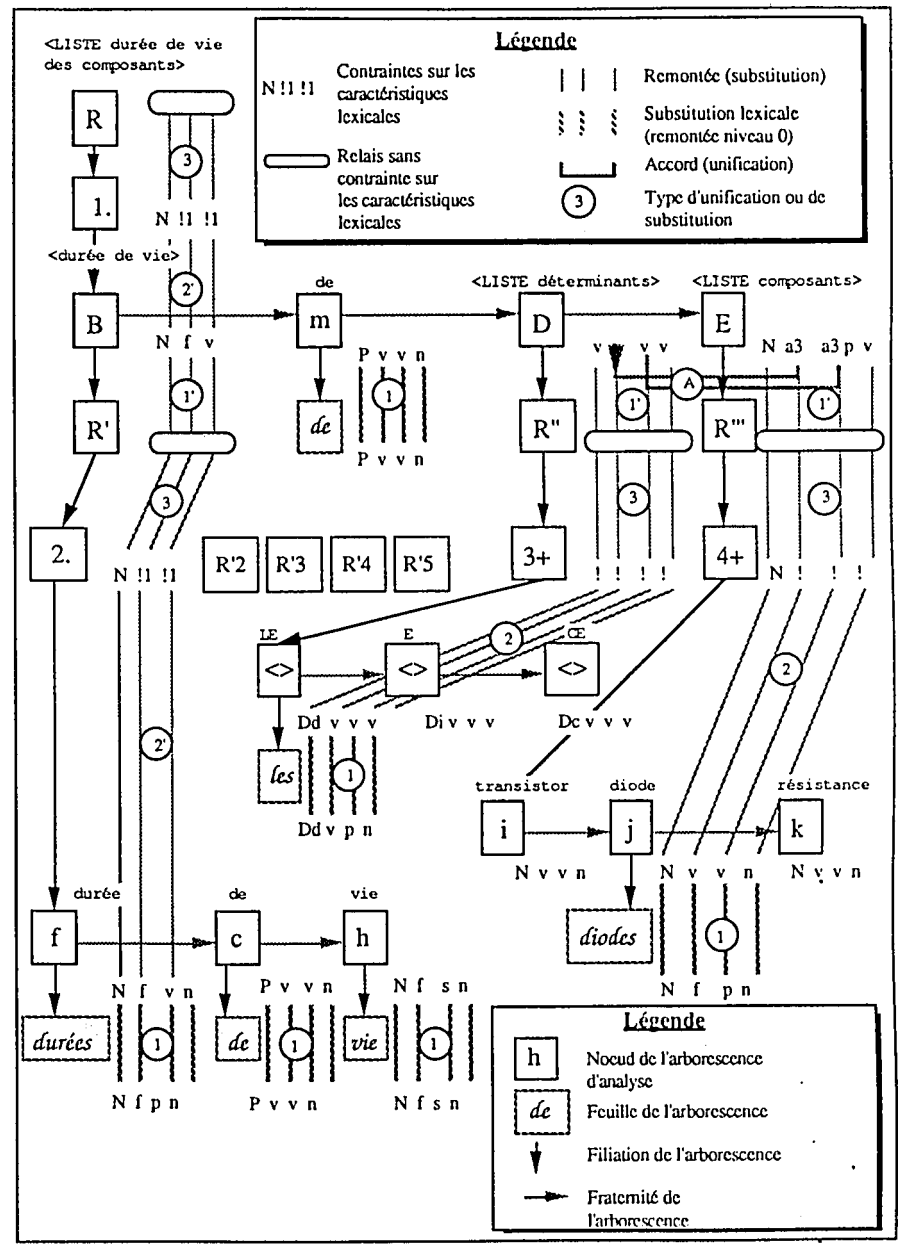



Figure 24: unification, parcours de remontée des valeurs dans l'analyse de *durées de vie de les diodes* à l'aide de la règle `<LISTE durée de vie des composants>`

Numéro d'étape	Noeuds de l'arbre		Liens de descendance	Liens de fraternité	Unification Vérifiée	Origine de l'unification	Origine de l'accord	Caractéristiques lexico-syntaxiques fournies par les règles				remontées en unification			
	Nom	Type						CL	G	N	Fo	CL	G	N	Fo
1	*durées*	Mot Lu										N	f	p	n
2	f	Unité Lex			1			N	f	v	n	N	f	p	n
3	*de*	Mot Lu										P	v	v	n
4	c	Unité Lex			1			P	v	v	n	P	v	v	n
5	*vie*	Mot Lu										N	f	s	n
6	h	Unité Lex			1 + accord éventuel			N	f	s	n	N	f	s	n
7	2.	Concat			2'			N	1	1		N	f	p	
8	R'	Règle Normale			3							N	f	p	
9	B	Terme Règle			1'			N	f	v		N	f	p	
10	*de*	Mot Lu										P	v	v	n
11	m	Unité Lex			1			P	v	v	n	P	v	v	n
12	*les*	Mot Lu										Dd	v	p	n
13	<>	Unité Lex			1			Dd	v	v	v	Dd	v	p	n
14	3+	Liste			2			*	*	*	*	Dd	v	p	n
15	R"	Règle Normale			3							Dd	v	p	n
16	D	Terme Règle			1'			v	v	v		Dd	v	p	n
17	*diodes*	Mot Lu										N	f	p	n
18	j	Unité Lex			1			N	v	v	n	N	f	p	n
19	4+	Liste			2			N	*	*	*	N	f	p	n
20	R'"	Règle Normale			3							N	f	p	n
22	E	Terme Règle			1' + accord éventuel			N	3	3		N	f	p	n
23	1.	Concat			2'			N	1	1		N	f	p	
24	R	Règle Normale			3							N	f	p	

Figure 25: unification, parcours en post-ordre de l'arbre d'analyse

f Utilisation des règles en synthèse

L'analyseur peut se transformer simplement en un générateur. Il permet d'obtenir, pour une règle donnée, tous les noms composés qu'elle reconnaît. Pour passer de l'analyseur au générateur, il suffit de modifier l'analyse des feuilles lexicales de l'arbre. Au lieu de rechercher le mot correspondant du texte et de provoquer un retour en arrière sur échec s'il ne le trouve pas, le programme produit toutes les flexions possibles. L'unification ne change pas, elle permet de sélectionner, parmi les flexions produites, celles qui respectent les contraintes et les accords.

La complexité de la production est importante. Une concaténation de **n** adjectifs admettant **4** flexions, **p** noms ayant **2** flexions et **q** verbes ayant **36** flexions, produit $4^n \times 2^p \times 36^q$ combinaisons sur lesquelles il faut tenter l'unification. Le coroutinage de l'unification n'est pas souhaitable en analyse. C'est une nécessité en production, afin de ne pas poursuivre sur des chaînes dont on peut prévoir, à l'avance, l'échec de l'unification.

Exemple

```
R:  <panneau de comptage> -> (N,!1,!1) panneau(N, m, v, n) DE(P, v, v, n)
                                        <comptage électrique>(N, m, s);
S:  <comptage électrique> -> (N, !1, !1) comptage(N, m, s, n)
        ((A,a1 !,a1 !,!) électrique(A,v,v,n) + électronique(A,v,v,n));
```

Si nous appliquons le mécanisme d'analyse, sans provoquer de retour en arrière sur échec, mais en produisant, pour chaque mot, toutes les flexions possibles, nous créons les **16** chaînes suivantes:

[1] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(m,s)}$
[2] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(f,s)}$
[3] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électriques*$_{(m,p)}$
[4] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électriques*$_{(f,p)}$
[5] *panneau*$_{(m,s)}$ *de comptages*$_{(m,p)}$ *électrique*$_{(m,s)}$
[6] *panneau*$_{(m,s)}$ *de comptages*$_{(m,p)}$ *électrique*$_{(f,s)}$
[7] *panneau*$_{(m,s)}$ *de comptages*$_{(m,p)}$ *électriques*$_{(m,p)}$
[8] *panneau*$_{(m,s)}$ *de comptages*$_{(m,p)}$ *électriques*$_{(f,p)}$
[9] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(m,s)}$
[10] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(f,s)}$
[11] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électriques*$_{(m,p)}$
[12] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électriques*$_{(f,p)}$
[13] *panneaux*$_{(m,p)}$ *de comptages*$_{(m,p)}$ *électrique*$_{(m,s)}$
[14] *panneaux*$_{(m,p)}$ *de comptages*$_{(m,p)}$ *électrique*$_{(f,s)}$
[15] *panneaux*$_{(m,p)}$ *de comptages*$_{(m,p)}$ *électriques*$_{(m,p)}$
[16] *panneaux*$_{(m,p)}$ *de comptages*$_{(m,p)}$ *électriques*$_{(f,p)}$

et les **16** autres chaînes notées de **17** à **32** obtenues en replaçant *électrique* par *électronique*.

Notons les chaînes dont l'unification réussira:

- **contraintes:** *comptage* doit être au singulier, les séquences **1** à **4**, et **9** à **12**, peuvent convenir (ainsi que **17** à **20**, et **25** à **28**),
- **accords:** *électrique* ou *électronique* doit s'accorder en genre et en nombre avec *comptage*, seules les séquences **1**, **7**, **9** et **15** vérifient cet accord (ainsi que **17**, **23**, **25** et **31**).

En conclusion, seules les formes suivantes du nom composé sont produites:

[1] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(m,s)}$
[9] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électrique*$_{(m,s)}$
[17] *panneau*$_{(m,s)}$ *de comptage*$_{(m,s)}$ *électronique*$_{(m,s)}$
[25] *panneaux*$_{(m,p)}$ *de comptage*$_{(m,s)}$ *électronique*$_{(m,s)}$

I.3 RATTACHEMENT D'UN MOT COMPOSE A UN MOT SIMPLE

Pour restreindre la taille mémoire de l'analyseur et augmenter ses performances, nous souhaitons limiter le nombre de règles à analyser. Seules résident en mémoire, au moment de l'analyse lexico-syntaxique d'une phrase, les règles qui ont été chargées après le défléchissement des mots de la phrase.

Afin de ne prendre que les règles pertinentes, nous relions chaque règle de mot composé à un des mots simples qui la constitue. Lorsque ce mot simple est rencontré dans la phrase, les règles qui lui sont reliées sont mises en mémoire et leur analyse est tentée. Par exemple, supposons que les règles décrivant *système expert* et *expert comptable* soient rattachées à *expert*. La lecture de la phrase *Nous utilisons des systèmes experts* où se trouve la flexion pluriel de *expert* provoquera le chargement de ces deux règles. Seule la première fournira une analyse correcte et permettra de reconnaître la flexion de *système expert*.

Les contraintes physiques du système informatique obligent à répartir les règles de mots composés sur les mots simples de façon homogène.

- La taille des mémoires tampon d'entrée sortie doit être dimensionnée pour permettre de lire toutes les règles rattachées à un mot simple en une seule fois (notion de **seuil de charge** des mots simples).
- L'analyse est descendante, la réécriture de toutes les règles chargées sera tentée. Donc, pour ne pas ralentir l'analyseur, il ne faut pas que les mots simples soient reliés à un trop grand nombre de règles.Ainsi, il est préférable de rattacher *eau potable* à *potable* plutôt que à *eau*.

De plus, l'analyse de règles inutiles rend difficile l'observation d'une trace.

Ce procédé de rattachement homogène des règles lexico-syntaxiques aux mots simples est désigné sous le nom de **lexicalisation**. Dans ce paragraphe, nous formalisons ce problème, nous montrons qu'il est NP-Complet. Nous mettons en oeuvre un algorithme approché polynômial permettant d'obtenir une solution approchée. Sous certaines conditions sur le lexique nous donnons une évaluation de la qualité. Les résultats expérimentaux permettent d'obtenir des résultats meilleurs que ce que laisse prévoir cette mesure de qualité.

Un exemple détaillé, présenté ci-dessous, permet d'appliquer cet algorithme sur un lexique de mots composés tirés du français. Sur ce cas particulier, l'algorithme approché permet d'obtenir une des solutions optimales.

Exemple N°1

Afin d'illustrer ce problème, nous allons nous donner l'ensemble des 41 mots composés suivants, choisis pour leur forte "concentration" sur un nombre réduit de mots simples:

art de la guerre	*être en froid*	*grand homme*
à froid	*être en guerre*	*grand place*
à la main	*être en place*	*grand temps*
à temps	*être son homme*	*Grande Guerre*
d'état à état	*faire de la place*	*Guerre Froide*
d'homme à homme	*faire de le sur place*	*homme à tout faire*
de place en place	*faire état*	*homme d'état*
de temps en temps	*faire froid*	*homme de l'art*
en état	*faire la guerre*	*homme de main*
état de guerre	*faire place*	*main dans la main*
état de l'art	*faire son temps*	*place d'art*
être dans les temps	*faire une place*	*se faire la main*
être dans tous ses états	*grand art*	*temps froid*
être en état	*grand froid*	

Ces mots composés n'utilisent que 11 mots simples non fréquents, nous excluons les prépositions, déterminants... Certains mots sont repris de [Nivat 88],qui a étudié plus particulièrement les mots composés de la forme homme de N et leurs liens avec les équivalents anglais. Afin de pouvoir se rapprocher d'une méthode formelle, nous faisons subir au lexique des mots composés, les trois transformations ci-dessous:

a Le problème aux nombres entiers

Prétraitement du lexique

Prétraitement de l'exemple N°1

Afin de passer d'un lexique de mots composés à un modèle formel, nous lui faisons subir trois opérations à chaque mot composé. Elles permettent d'en éliminer les mots fréquents et de le transformer en un couple formé d'un entier (sa taille) et d'un ensemble de mots simples (les mots non fréquents qui le constituent et auxquels il pourra être rattaché).

1: suppression des mots fréquents

Les mots fréquents (prépositions, déterminants, ...) ne peuvent servir pour le rattachement. Nous obtenons un problème équivalent en prenant, comme donnée, les mots composés dans lesquels tout mot fréquent a été remplacé par le symbole **VIDE**. Le début de la liste précédente devient donc:

art VIDE VIDE guerre *VIDE froid* *VIDE VIDE main...*

2: représentation des mots simples par leur forme canonique

Notre lexique ne contient pas les mots fléchis, mais uniquement les formes canoniques des mots simples avec des indications sur les flexions. Nous avons un problème équivalent en remplaçant les mots simples par leurs formes canoniques:

art VIDE VIDE guerre *VIDE froid* *VIDE VIDE main...*

3: regroupement des mots identiques et suppression du symbole VIDE

Pour le rattachement d'un mot composé, n'interviennent que deux aspects de ce mot: sa taille et l'ensemble des mots simples non fréquents qui y sont présents. Les deux informations qui doivent être transmises pour la modélisation mathématique sont :

- la taille initiale du mot composé qui est obtenue en comptant le nombre de mots simples qu'il contient,
- l'ensemble des mots simples auxquels on peut le rattacher qui est obtenu en regroupant les racines identiques de mots simples et en supprimant le symbole **VIDE**.

On obtient ainsi:

(4, {*art, guerre*}) (2, {*froid*}) (3, {*main*})...

Si l'ensemble des mots simples auxquels on peut rattacher un mot composé est vide, il doit être écarté des problèmes de rattachement. Dans l'exemple N°1, nous avons volontairement écarté, *a priori*, de tels mots composés.

Complexité en temps de opérations 1, 2 et 3

Il n'est pas nécessaire d'écrire les programmes de ces trois transformations pour se rendre compte qu'elles sont d'une complexité en temps qui est proportionnelle à la taille du lexique. Elles ne modifieront pas la complexité des opérations ultérieures de rattachement.

Modélisation

On modélise le problème du rattachement des mots composés aux mots simples, en se donnant un alphabet **A** de **n** lettres et un ensemble **C** de **c couples**. Chaque couple est constitué d'un entier positif strict appelé sa **taille** et d'un ensemble non vide de lettres de l'alphabet **A** appelé son **ensemble.**

$A = \{ a_1, a_2, \ldots a_n \}$ $C = \{ c_1, c_2, \ldots, c_i, \ldots, c_c \}$

avec $c_i = (t_i, E_i)$ et t_i entier positif et E_i sous-ensemble de A

Rappels sur la reconnaissance morphologique des noms composés

Le mécanisme d'analyse lexico-syntaxique, présenté au I.1 et 2, se résume ainsi. L'analyseur fait un premier passage pour défléchir les mots, et pour relever les règles lexicales qui leur sont

rattachées. Dans une deuxième étape, il fait l'analyse syntaxique avec les règles relevées. La position du mot, auquel est rattachée une règle, est (presque) indifférente pour la complexité ultérieure de l'analyse. Il vaut mieux minimiser les accès disques qui sont longs et coûteux, en répartissant bien la grammaire sur le lexique, et en rattachant certaines règles à un autre mot que le premier qui les compose, plutôt que d'accélérer légèrement l'analyse faisant figurer toutes les règles sur le premier mot. L'ordre du terme de rattachement dans la règle étant indifférent, nous avons choisi une modélisation par des couples comportant un ensemble non ordonné.

On appelle **rattachement** des couples de C sur les lettres de A toute fonction **r** de C dans A qui, à chacun de ces couples fait correspondre une des lettres qu'il contient et qui est celle à laquelle il est rattaché. On recense ainsi, pour chaque lettre de l'alphabet, le nombre de couples de C qui lui sont rattachés.

$r: C \longrightarrow A$

$(t_i, E_i) \longrightarrow r((t_i, E_i)) = a_j$ et a_j appartient à E_i

Exemple N°2

grand temps *Grande Guerre* *de temps en temps* *grands froids*

Après transformation, nous obtenons:

A = { *froid, grand, guerre, temps* }

C = { (2, {*grand, temps*}), (2, {*grand, guerre*}), (4, {*temps*}), (2, {*froid, grand*}) }

Un rattachement possible est le suivant:

r((2, {*grand, temps*})) = *temps* r((2, {*grand, guerre*})) = *guerre*

r((4, {*temps*})) = *temps* r((2, {*froid, grand*})) = *froid*

Définition d'une pondération X_r associée à chaque rattachement r

Pour chaque rattachement, on définit une **pondération** sur l'ensemble A. Le problème à résoudre est de choisir la fonction **r** telle que le système pondéré A soit le moins dispersé possible. Parmi les pondérations possibles, on en retient celle qui est la somme des tailles des couples rattachés. Elle est proche de la notion physique d'encombrement des mots composés rattachés à un mot simple. En effet, chaque règle de mot composé est formée d'une tête et de la liste des termes. Le nombre de termes est approximativement égal au nombre de mots simples dans le composé

$X_r: A \longrightarrow N$: entiers naturels

$a_j \longrightarrow X_r(a_j) = \sum_{(t_i, E_i) \text{ appartient à } r^{-1}(\{a_j\})} t_i$

Où t_i désigne la taille du couple (t_i, E_i). Nous notons cette mesure sous le nom de pondération X_r.

Deux rattachements définis sur l'exemple N°2

Les deux rattachements r_1 et r_2, définis ci-dessous, fournissent deux pondérations X_{r1} et X_{r2}:

r_1 : C	——> A	r_2 : C	——> A
(2, {*grand, temps*})	——> *grand*	(2, {*grand, temps*})	——> *grand*
(2, {*grand, guerre*})	——> *guerre*	(2, {*grand, guerre*})	——> *grand*
(4, {*temps*})	——> *temps*	(4, {*temps*})	——> *temps*
(2, {*froid, grand*})	——> *froid*	(2, {*froid, grand*})	——> *grand*

X_{r1} : A	——> N	X_{r2} : A	——> N
froid	——> 2	*froid*	——> 0
grand	——> 2	*grand*	——> 6
guerre	——> 2	*guerre*	——> 0
temps	——> 4	*temps*	——> 4

Définition d'une dispersion $D_{r,Smax}$ associée à chaque rattachement

On choisit ici un critère de dispersion proche de la réalité, il s'agit du cardinal de l'ensemble des lettres dont la pondération est supérieure à un seuil, défini en fonction des caractéristiques du système informatique. Une valeur faible, voire nulle, de cette dispersion assure qu'il n'y a pas de mots du lexique auxquels soient rattachés un trop grand nombre de mots composés.

Appelons s_{max} le seuil critique et $D_{r,Smax}$, le nouveau critère de dispersion ainsi défini:
$D_{r,Smax}$ = card({ a_i appartient à A; $X_r(a_i) > s_{max}$ })

Résultats obtenus en prenant $s_{max} = 4$: $D_{r1,Smax} = 0$ et $D_{r2,Smax} = 1$
Donc, pour un même seuil de 4, le résultat obtenu est meilleur en utilisant r_1 plutôt que r_2.

Calcul matriciel pour la pondération X_r

A tout ensemble C, de cardinal **c**, formé de couples, on peut associer de façon biunivoque:
- une matrice T_o, de dimension **(1, c)**, à coefficients entiers positifs stricts, qui décrit les tailles des couples de C,
- une matrice E_o, de dimension **(c, n)**, à coefficients **0** ou **1** qui décrit les ensembles des couples de C.

Le coefficient $e_{i,j}$ de E_o vaut **1** si la lettre a_j appartient à l'ensemble E_i du couple (t_i, E_i) et zéro sinon. Le coefficient t_i de T_o est égal à la taille t_i du couple (t_i, E_i).

Par exemple les matrices T_o et E_o associée à l'ensemble C précédent sont les suivantes:

$$T_o = \begin{vmatrix} 2 & 2 & 4 & 2 \end{vmatrix} \qquad E_o = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{vmatrix}$$

A chaque fonction de rattachement **r**, définie de C dans A, on associe une matrice unique R de dimension **(c, n)** à coefficients dans **{0, 1}**. Cette matrice a un seul élément non nul par ligne. Si $r_{i,j}$ est égal à **1**, alors le couple (t_i, E_i) est rattaché à la lettre a_j. Donc $r_{i,j}$ est tel que le coefficient correspondant de E_o, $e_{i,j}$, soit égal à **1** (un mot composé ne peut être rattaché qu'à un des mots simples qui le composent).

Par exemple les matrices R_1 et R_2 associées aux fonctions r_1 et r_2 sont les suivantes:

$$R1 = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{vmatrix} \qquad R2 = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{vmatrix}$$

Nous donnons ci-dessous la formules de calcul de $X_r(a_j)$, la valeur de la pondération X_r, associée à un rattachement **r**, pour la lettre a_j. $X_r(a_j)$ est la somme des termes t_i de T_o, en ne prenant que les indices **i** correspondant aux couples (t_i, E_i) rattachés à la lettre a_j, c'est à dire tels que $r_{i,j}$ soit égal à **1**.

$X_r(a_j)$ est le terme d'indice **j** de la matrice ligne $(T_o \times R)$, R étant la matrice de rattachement associée à **r**.

Complexité de calcul de la pondération X_r et de la dispersion $D_{r,Smax}$

Le calcul matriciel précédent permet de formaliser le calcul de la valeur de la pondération X, et donc celui de la dispersion associée. On peut mesurer le coût, en temps et en espace, de l'algorithme de calcul de la dispersion associée à une fonction **r**.
On comptera, comme un pas de calcul, une addition ou une multiplication élémentaire. Si on souhaite mesurer très précisément le temps de calcul pour une machine donnée, il faut revoir les approximations faites, car certains produits sont des tests, puisqu'il s'agit de produits par **0** ou **1**; ils sont donc plus rapides à effectuer qu'une multiplication entre deux entiers. Mais ces considérations ne changent pas l'ordre de grandeur que nous obtenons ici.

Calculons tout d'abord le temps de calcul pour le produit d'une matrice de dimensions **(m,n)** par une matrice **(n,p)**:

t = m x p x (n produits et la somme de ces n résultats)
= m x p x 2 n = 2 m n p

Le temps de X_r est donc de (1 x n x 2 c), soit un ordre de grandeur de (c n).
Le calcul de $D_{r,Smax}$ à partir de X_r t(X) + n
calcul de X comparaisons à s_{max} et somme des valeurs supérieures
soit un ordre de grandeur de (c n).

Aspect NP-Complet du problème

Problème de décision RMC (Rattachement des Mots Composés)

Nous appelons **RMC** le **problème de décision** suivant: on se donne
un alphabet $A = \{a_1, a_2, \ldots a_n\}$;
un ensemble **C** de couples formés d'un entier positif strict et d'un sous-ensemble non vide de A: $C = \{c_1, c_2, \ldots c_c\}$;
et un seuil s_{max}.
Il s'agit de trouver un rattachement **r** dont la dispersion $D_{r,Smax}$ est nulle.

Seuil minimal

Dans le cas d'un problème donné, nous allons déterminer la valeur minimale s_{min} à prendre pour le seuil s_{max}, afin qu'il puisse exister un rattachement pour lequel $D_{r,Smax}$ soit nul. Ceci équivaut à déterminer la valeur maximale du seuil, pour laquelle tout rattachement a une dispersion $D_{r,Smax}$ non nulle.
Appelons **EntSup()**, la fonction qui, à tout nombre réel non entier, associe sa partie entière plus un, et, à tout entier, associe lui-même.

$$s_{min} = \text{EntSup}\left(\left(\sum_{(ti,\,E_i)\ \text{appartient à C}} t_i\right) / n\right)$$

Appelons T la somme des tailles des couples de C.

$$s_{min} = \text{EntSup}(T / n)$$

Justification

La justification se fait en appliquant la définition de la partie entière supérieure. Pour un seuil de $s_{min} - 1$, on calcule la pondération maximale **P** que pourra avoir chaque lettre:

$$P \leq n\,(s_{min} - 1) < n\,s_{min} = n\,\text{EntSup}\left(\left(\sum_{(ti,\,E_i)\ \text{appartient à C}} t_i\right) / n\right)$$

- si T / n est entier

$P < n\,(T/n)$ (par définition de la partie entière supérieure)
donc $P < T$

- si T / n est non entier

$P < n \times (T/n + 1)$ (par définition de la partie entière supérieure)

donc $P < T$

Dans les deux cas, la somme des pondérations des lettres de l'alphabet est inférieure stricte à la somme des tailles des couples de C. Il existe donc des couples qui ne sont pas rattachés, si on se limite au seuil proposé. Afin de pouvoir rattacher tous les couples, la charge de certaines lettres doit nécessairement dépasser la valeur du seuil. La valeur de $D_{r,Smax}$ sera non nulle.

Exemple N°3

d'état à état — *faire de la place* — *faire place*
d'homme à homme — *faire de le sur place* — *homme d'état*
de place en place — *faire état*

Après transformation, nous obtenons:
- l'alphabet A = { *état, faire, homme, place* }
- et l'ensemble C = { (4, {*état*}), (4, {*homme*}), (4, {*place*}), (4, {*faire, place*}), (5, {*faire, place*}), (2, {*état, faire*}), (2, {*faire, place*}), (3, {*état, homme*}) }

Le seuil minimal de rattachement est ici de: $s_{min} = 28/4 = 7$. Un rattachement à un seuil de 7 ne peut être réalisé sur cet exemple. En effet, les trois premiers couples ont un rattachement impératif sur le seul élément qu'ils contiennent:

r((4, {*état*})) = *état* r((4, {*homme*})) = *homme* r((4, {*place*})) = *place*

Pour donner à *état* la pondération 7, il faut lui rattacher un couple de taille 3 où il apparaît; la seule possibilité est: (3, {*état, homme*}). La pondération de *homme* reste alors à 4, puisqu'il n'existe plus de couples le contenant. Or la valeur de 7 étant un minimum exact, on en déduit que tout rattachement de tous les mots oblige à dépasser ce seuil.

Ne pouvant réaliser le rattachement à 7, on tente de le faire avec un seuil de 8, et on obtient la possibilité suivante qui réalise cette valeur, puisque la somme des tailles des couples rattachés à chaque lettre est inférieure ou égale à 8:

r: C ——> A

(4, {*état*}) ——> *état*
(4, {*homme*}) ——> *homme*
(4, {*place*}) ——> *place*
(4, {*faire, place*}) ——> *place*
(5, {*faire, place*}) ——> *faire*
(2, {*état, faire*}) ——> *état*
(2, {*faire, place*}) ——> *faire*
(3, {*état, homme*}) ——> *homme*

Les pondérations sont les suivantes: *état* : 6 *homme* : 7
faire : 7 *place* : 8

RMC est NP-Complet

Pour montrer que le problème **RMC** est NP-Complet, nous allons vérifier qu'il existe un problème NP-Complet qui se réduit polynômialement à **RMC**. Nous prenons le problème du BinPacking qui est assez proche de **RMC** pour que la réduction polynômiale se déduise facilement.

Rappel: Problème du BinPacking

On se donne **n** articles $\{ u_1, u_2, ..., u_n \}$ de tailles $s(u_1), s(u_2), ..., s(u_n)$ avec $s(u_i)$ (pour tout **i** entre **1** et **n**) qui est compris entre **0** et **1**.
On cherche à mettre les **n** articles dans un nombre minimum de boîtes de capacité **1**.

Propriété

Le problème du BinPacking à coefficients rationnels (BP-rationnel) est NP-Complet, donc le problème de décision **RMC** est NP-Complet.

1: RMC appartient à NP

Une solution Δ de rattachement étant donnée, on peut vérifier polynômialement que cette solution respecte les contraintes, c'est à dire que la pondération de chacune des lettres est inférieure au seuil. Les temps de calcul ont été évalués précédemment.

2 : BP-rationnel se réduit polynômialement à RMC

Soit $D_1 = \{ s(u_1), s(u_2), ..., s(u_n) \}$ une donnée de BP-rationnel, soit **n** nombres rationnels entre **0** et **1**. Nous lui associons la donnée D_2 de **RMC** définie, pour **n** couples sur un alphabet **A** de cardinal **k**. Les matrice T_2 et E_2 associées sont les suivantes:

$$T_2 = | \; t_1 \; t_2 \; ... \; t_n \; | \qquad E_2 = \begin{vmatrix} 1 & 1 \; ... \; 1 \\ 1 & 1 \; ... \; 1 \\ .... & \\ 1 & 1 \; ... \; 1 \end{vmatrix}$$

et le seuil de rattachement est **t**, défini ci-dessous.

- **k** est le nombre minimal de boîtes utilisées par BP-rationnel
- les coefficients t_i et le seuil **t** sont définis en réduisant les $s(u_i)$ au même dénominateur:
  $s(u_i) = t_i / t$ (pour tout **i** entre **1** et **n**).

Le calcul de D_2 par rapport à D_1 est polynômial par rapport à la taille de D_1.

Les $s(u_i)$ étant rationnels,

notons $s(u_i) = r_i / q_i$; r_i et q_i étant deux entiers.

soit $t = \prod q_j$ (temps de calcul **n**)

alors $s(u_i) = (r_i \times \prod_{j \neq i} q_j) / (q_i \times \prod_{j \neq i} q_j) = (r_i \times \prod_{j \neq i} q_j) / t = t_i / t$

(**n** fois un temps de calcul égal à **n**).

Prouvons que la réponse à D_1 est oui si et seulement si la réponse à D_2 est oui.

- Si la réponse à D_1 est oui, alors il existe une partition de $I = \{ 1, 2, ..., n \}$ en **k** ensembles $J_1, J_2, ... J_k$ tels que:

$$\sum_{j \text{ appartient } J_i} s(u_j) \leq 1 \qquad \text{(pour tout } i \text{ entre } 1 \text{ et } k)$$

Pour tout **i** entre **1** et **k** et pour tout **j** appartenant J_i, rattachons chaque couple (t_j, E_j) à la lettre a_i. La pondération de ce rattachement est:

$$X(a_i) = \sum_{j \text{ appartient } J_i} t_j = t \left(\sum_{j \text{ appartient } J_i} t_j / t \right) \leq t$$

donc la réponse à D_2 est oui.

- Si la réponse à D_2 est oui

alors il existe un rattachement des **n** couples c_j aux lettres a_i, donc une partition de l'ensemble **J** en **k** ensembles $J_1, J_2, ... J_k$ tels que

Pour tout **i** entre **1** et **k**

$$\sum_{j \text{ appartient } J_i} t_j \leq t$$

donc

$$\sum_{j \text{ appartient } J_i} s(u_j) = \sum_{j \text{ appartient } J_i} t_j / t$$

$$= (1/t) \sum_{j \text{ appartient } J_i} t_j \leq (1/t)\, t = 1$$

donc la réponse à D_1 est oui.

Lien avec la programmation linéaire

Pour résoudre RMC, on peut utiliser la programmation linéaire en nombres entiers.

On prend comme variables r_{ij} les termes de la matrice de rattachement R.
$r_{ij} = 1$ <=> le couple **i** est rattaché à la lettre **j** (et zéro sinon)
On note e_{ij} les termes de la matrice E représentant les ensembles des couples.
$e_{ij} = 1$ <=> le couple **i** contient la lettre **j** (et zéro sinon)
Et t_i désigne la taille du couple **i**.

Le système en r_{ij}, à résoudre par programmation linéaire est:

Pour tout **i** entre **1** et **c** (1) $\sum_j e_{ij} r_{ij} \geq 1$ (tout couple est rattaché au moins une fois)

Pour tout **j** entre **1** et **n** (2) $\sum_i t_i r_{ij} \leq s_{max}$ (la charge de toute lettre doit être inférieure au seuil)

Pour tout **j** entre **1** et **n** et
pour tout **i** entre **1** et **c** (3) r_{ij} appartient à { 0, 1 }

b Un algorithme approché polynômial

Notion de charge globale

On appelle **charge globale d'une lettre a de A**, la valeur maximale que peut prendre la pondération X_r pour cette lettre, pour toutes les fonctions de répartition **r** possibles. Cette valeur correspond à la somme des tailles des couples auxquels appartient **a**. Nous présentons, sur un exemple, le calcul des charges globales en fonction des matrice T_o et E_o associée à l'ensemble C.

Charges globales de l'exemple N°3

$T_o = |\ 4\ 4\ 4\ 4\ 5\ 2\ 2\ 3\ |$ $E_o = \begin{vmatrix} 1&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&1&0&1 \\ 0&1&0&1 \\ 1&1&0&0 \\ 0&1&0&1 \\ 1&0&1&0 \end{vmatrix}$

$G_o = T_o \times E_o$
G_o est la matrice ligne, dont chaque coordonnée g_i représente la somme des tailles des couples contenant la lettre a_i au moins une fois.
Pour la matrice E_o donnée, on obtient $G_o = |\ 9\ 13\ 7\ 15\ |$.

Etape 1: rattachement aux lettres non globalement surchargées

Préliminaire à l'étape 1

On effectue la suppression des lettres n'appartenant à aucun des couples, car elles ne peuvent servir à la lexicalisation. On supprime toutes les colonnes d'indice **i** de la matrice E_o correspondant à un terme d'indice **i** nul dans la matrice G_o.

Lettres non globalement surchargées

On appelle **lettre globalement surchargée** par rapport au seuil s_{max}, toute lettre, telle qu'il existe une fonction de répartition **r**, telle que la valeur de la pondération X_r associée soit supérieure au seuil s_{max}. D'après les calculs sur les charges globales:
la lettre n_i est non globalement surchargée pour X_r par rapport à s_{max}
<=> l'élément d'indice **i** de G_o est inférieur à s_{max}.

Algorithme de l'étape 1

Cette étape du calcul consiste à rattacher les couples aux lettres non globalement surchargées, chaque fois que cela est possible. Ce qui revient, dans le cas de la langue naturelle, à rattacher tous les mots composés qui contiennent un mot "rare" à ce mot. Il ne reste alors que les mots composés, dont tous les mots simples sont globalement surchargés.

On crée à partir de la matrice G_o deux matrices extraites, de même dimension, $G_{o,s}$ et $G_{o,ns}$ qui sont les matrices caractéristiques des éléments dont la valeur est supérieure (respectivement inférieure) au seuil. Ces deux matrices représentent les lettres qui sont (respectivement ne sont pas) globalement surchargées.
Pour $G_o = |\ 9\ 13\ 7\ 15\ |$ et un seuil de **8**, les matrices obtenues sont:

$G_{o,s} = |\ 1\ 1\ 0\ 1\ |$ et $G_{o,ns} = |\ 0\ 0\ 1\ 0\ |$.

Le produit $E_o \times G_{o,ns}$ donne une matrice colonne, dont les indices des éléments non nuls sont les indices des couples à rattacher à une de leurs lettres non globalement surchargées. On peut donc extraire des matrices T_o et E_o les lignes correspondant aux indices ainsi trouvés. On extrait de E_o les colonnes des lettres non globalement surchargées, puisque tous les couples, qui en contenaient au moins une, y ont été rattachés.

On réitère ce procédé tant que le nombre de vecteurs lignes ou colonnes de E_o n'est pas nul et qu'il existe des lettres qui ne sont pas globalement surchargées. On effectue au plus **min(c, n)** opérations.
A l'arrêt des itérations, si la matrice n'est pas vide, tous les couples ne contiennent que des lettres globalement surchargées, et, c'est par une répartition des rattachements des couples entre ces lettres, que l'on pourra obtenir une bonne dispersion. C'est l'objet de l'étape 2 suivante

Etape 1 sur l'exemple N°3

Nous reprenons l'exemple N°3. Pour un seuil de 8: $G_{o,ns} = | 0\ 0\ 1\ 0 |$, *homme* est la seule lettre non surchargée de A.

$E_o\ x^t G_{o,ns} = \begin{matrix} |\ 0\ | \\ |\ 1\ | \\ |\ 0\ | \\ |\ 0\ | \\ |\ 0\ | \\ |\ 0\ | \\ |\ 0\ | \\ |\ 1\ | \end{matrix}$

Ce produit donne les indices des lignes des couples qui contiennent les lettres non globalement surchargées

Nous extrayons, de E_o la colonne 3 correspondant à *homme*, et les lignes 2 et 8 correspondant à (4, {*homme*}) et à (3, {*état, homme*}). Nous appelons E_1, la matrice ainsi obtenue.La première itération nous fait rattacher: (4, {*homme*}) et (3, {*état, homme*}) à *homme*

$$E_1 = \begin{matrix} |\ 1\ 0\ 0\ | \\ |\ 0\ 0\ 1\ | \\ |\ 0\ 1\ 1\ | \\ |\ 0\ 1\ 1\ | \\ |\ 1\ 1\ 0\ | \\ |\ 0\ 1\ 1\ | \end{matrix}$$

En considérant la nouvelle matrice E_1, on obtient: $G_1 = | 6\ 13\ 15 |$ et $G_{1,ns} = | 1\ 0\ 0 |$. *état* n'est plus globalement surchargée. On y rattache les deux couples qui la contiennent: (4, {*état*}) et (2, {*état, faire*}) sont rattachés à *état*. On obtient une nouvelle matrice E_2.

$$E_2 = \begin{matrix} |\ 0\ 1\ | \\ |\ 1\ 1\ | \\ |\ 1\ 1\ | \\ |\ 1\ 1\ | \end{matrix}$$

En considérant la nouvelle matrice E_2, on obtient $G_2 = | 11\ 15 |$ et $G_{2,ns} = | 0\ 0 |$. Les deux lettres restantes de A: *faire* et *place* sont globalement surchargées, on doit donc arrêter les itérations de la méthode employée.

Complexité de l'étape 1

A chaque itération, on supprime au moins une colonne et une rangée de la matrice E_o, qui est donc notée E_i au rang **i**. A la ième itération, soient n_i et c_i les nombres de lettres et de couples restants. D'après ce qui précède:

pour tout **i**, $n_{i+1} \leq n_i - 1$ et $c_{i+1} \leq c_i - 1$

donc pour tout **i**, $n_{i+1} < n - i$ et $c_{i+1} < c - i$

On peut donc en déduire une majoration du temps T mis pour effectuer les itérations jusqu'à l'arrêt de cette étape:

$T \leq \sum_{i \text{ appartient à } [\,1,\ \min(c,\,n)\,]} T_i$ (T_i temps mis à l'itération de rang **i**)

- temps de calcul de G_o: $2\, n_i\, c_i$ (produit d'une matrice $(1,c_i)$ par une (c_i,n_i))
- temps de calcul de $G_{o,ns}$: n_i
- extraction de E_{i+1}: $2\, n_i\, c_i$ (produit d'une matrice (c_i,n_i) par une $(n_i,1)$)
  et n_i comparaisons
  et $c_{i+1}\,(\,n_{i+1} - 1\,)$ recopies

$$T \leq \sum_{i \text{ appartient à } [\,1,\ \min(c,\,n)\,]} 4\, n_i\, c_i + n_i + (\,n_{i+1} - 1\,)\, c_{i+1}$$

$$\text{donc } T \leq \sum 4\, n_i\, c_i + n_i + (\,n_i - 1\,)\, c_i = \sum 5\, n_i\, c_i + n_i - c_i \leq 5\, n_i\, c_i + n_i$$

$$\text{donc } T \leq \sum_{i \text{ appartient à } [\,1,\ \min(c,\,n)\,]} (\,n - i\,)(\,5c - 5i + 1\,) \quad (\text{car } n_i \leq n - i \text{ et } c_i \leq c - i)$$

Premier cas $c \leq n$

$$T \leq \sum_{i \text{ appartient à } [\,1,\ c\,]} (\,n - i\,)(\,5n - 5i + 1\,)$$

$$\text{donc } T \leq 5 \sum_{i \text{ appartient à } [\,1,\ c\,]} (\,n - i\,)^2 + \sum_{i \text{ appartient à } [\,1,\ c\,]} (\,n - i\,)$$

$$= 5 \sum_{i \text{ appartient à } [\,1,\ n-1\,]} n^2 + \sum_{i \text{ appartient à } [\,1,\ n-1\,]} n$$

$= O(n^3)$ On a donc T qui est de l'ordre de n^3 pour **n** grand.

Deuxième cas $c > n$

$T = O(c^3)$ On a donc **T** qui est de l'ordre de c^3 pour c grand.

Etape 2: toutes les lettres sont globalement surchargées

Cette étape consiste à répartir **c** couples sur **n** lettres dont la configuration est donnée par deux matrice T_o et E_o, sachant que toutes les lettres de l'alphabet sont globalement surchargées. Le seuil s_{max} étant donné, n recherche un rattachement des couples aux lettres, c'est-à-dire une fonction **r** de **C** dans **A** telle que $D_{r,Smax} = 0$.

Algorithme de l'étape 2

Les priorités que l'on va affecter aux lettres et aux couples sont calculées en fonction des charges globales de chacune des lettres de l'alphabet.

- **Lettres de priorité maximale**: les lettres les plus prioritaires sont celles dont la charge globale est la moins forte. Moins la charge d'une lettre est forte, plus les choix de rattachement de couples à cette lettre sont limités. Or si les choix sont restreints, les risques d'erreurs sont plus faibles.

- **Couples de priorité maximale**: pour une lettre donnée, les couples de priorité maximale sont ceux dont les lettres sont les plus chargées. Plus la charge globale des lettres d'un couple est élevée, plus on aura intérêt à le rattacher à la lettre courante. Il est souhaitable de ne pas reporter le problème du rattachement de ce couple sur ses autres lettres qui sont critiques.

Cette deuxième notion est à préciser. On calcule pour chacun des couples le minimum et le maximum de la charge globale de chacune des lettres (non comprise la lettre de rattachement courante). Un couple composé uniquement de la lettre courante a une charge maximale et minimale infinies. A partir de ces extrema de charge, nous allons définir la priorité ainsi:

- les couples ayant la charge globale minimum la plus forte sont les plus prioritaires,
- à priorité égale, les couples ayant la charge globale maximum la plus forte sont les plus prioritaires,
- les lettres qui sont déjà rattachées ont une priorité infinie. En particulier, un couple composé uniquement de lettres déjà rattachées aura une priorité infinie, puisque, si on ne le rattache pas à la lettre courante, il ne pourra pas l'être dans la suite de l'algorithme.

- **Remplissage optimal des lettres**: nous accepterons de permuter un couple avec un couple moins prioritaire, qui permet d'obtenir une charge pour la lettre considérée qui est plus proche du seuil. La permutation ne peut évidemment pas se faire avec un couple de priorité infinie, qui ne pourrait plus être rattaché ultérieurement.

On donne au seuil la valeur minimale. On cherche à rattacher les couples aux lettres, afin que les charges de toutes les lettres soient inférieures à s_{min}. Si le rattachement échoue, on recommence avec un seuil $s_{min} + 1$ etc... Il n'y a pas de stratégie de retour en arrière, afin de s'assurer d'obtenir une solution en un temps polynômial. En contrepartie, nous n'obtenons pas nécessairement la meilleure solution, mais une solution approchée dont nous déterminerons la qualité sous certaines conditions.

Algorithme approché complet sur l'exemple N°4

d'état à état	*faire de le sur place*	*homme à tout faire*
d'homme à homme	*faire état*	*homme d'état*
de place en place	*faire place*	*en état*
faire de la place	*faire une place*	

A = { *état, faire, homme, place* }

C = { (4, {*état*}), (4, {*homme*}), (4, {*place*}), (4, {*faire, place*}),
(5, {*faire, place*}), (2, {*état, faire*}), (2, {*faire, place*}), (3, {*faire, place*}),
(4, {*faire, homme*}), (3, {*état, homme*}), (2, {*état*}) }

E_o =

c1	1 0 0 0
c2	0 0 1 0
c3	0 0 0 1
c4	0 1 0 1
c5	0 1 0 1
c6	1 1 0 0
c7	0 1 0 1
c8	0 1 0 1
c9	0 1 1 0
c10	1 0 1 0
c11	1 0 0 0

T_o = | 4 4 4 4 5 2 2 3 4 3 2 |

Nous avons noté les couples ci.

Dans un premier temps, nous faisons fonctionner cet algorithme sur l'exemple ci-dessus, puis nous l'appliquerons sur le lexique que nous avions donné au début de ce paragraphe qui comporte un volume de données plus important.

L'algorithme se déroule en deux étapes. La première phase est le rattachement des mots aux lettres non surchargées pour le seuil choisi. La deuxième étape prend, tour à tour, la lettre la plus prioritaire parmi les lettres restantes, et y rattache des mots qui la contiennent afin de lui donner une charge la plus proche possible du seuil, en prenant en premier les mots les plus prioritaires.

Charges globales des lettres: *état:* 11, *faire:* 20, *homme:* 11, *place:* 18.

La somme des tailles des couples est de **37**; or il y a **4** lettres. Le seuil minimal que l'on peut retenir est un seuil s_{min} de **10**. L'algorithme va dérouler les deux étapes avec un seuil de **10**.

Etape 1

Dans ce cas particulier, toutes les lettres sont globalement surchargées, il n'y a pas de rattachement possible à l'étape 1.

Etape 2:

Les lettre les plus prioritaires sont *état* et *homme* car leurs charges globales sont de **11**. Nous prenons la première dans l'ordre alphabétique: *état*. Pour la lettre *état*, les couples qui peuvent être rattachés sont c_1, c_6, c_{10} et c_{11}. On calcule, pour ces quatre couples, la charge minimale et la charge maximale des lettres qui les composent.

Charge minimale: ∞, 20, 11, ∞

Charge maximale: ∞, 20, 11, ∞

On classe ces couples par priorité décroissante: par charge minimale décroissante et pour une même charge maximale, par charge maximale décroissante.

Classement: c_1, c_{11}, c_6, c_{10}.

Les trois premiers couples fournissent une charge de **8**. Pour obtenir un meilleur remplissage, nous permutons c_6 avec c_{10} afin d'obtenir une charge de **9**.

Les mots composés: *d'état à état*, *homme d'état* et *en état* sont rattachés à *état*.

Réitération de l'étape 2 sur les lettres successives de priorité maximale, jusqu'à rattacher tous les couples.

L'algorithme permet de rattacher tous les couples à des lettres. Nous montrons ci-dessous que le rattachement obtenu est le rattachement optimal que nous trouvons par déduction. Ceci ne sous-entend pas que notre algorithme pourra toujours trouver la meilleure solution. Nous montrerons les deux points suivants:

- expérimentalement, nous obtenons de très bonnes performances par simulation sur des ensembles de données fictifs. Nous n'avons pas pris le lexique des mots composés relevés dans les textes de la D.E.R. pour cette simulation, car il est trop petit et trop dispersé pour pouvoir constituer un exemple délicat.
- Nous pouvons donner une garantie de qualité de la solution obtenue, à certaines conditions sur le lexique.

Bilan des rattachements dans l'ordre où ils ont été faits. La notion de capital restant est précisée plus loin.

Lettre	Charge	Couples rattachés	Capital restant
état	9	c_1, c_{11}, c_{10}.	2
homme	8	c_2, c_9	0
faire	10	c_5, c_6, c_8	0
place	10	c_3, c_4, c_7	0

Les mots composés:

- *d'état à état*, *homme d'état* et *en état* sont rattachés à *état*.
- *d'homme à homme* et *homme à tout faire* sont rattachés à *homme*
- *faire de le sur place*, *faire état* et *faire une place* sont rattachés à *faire*
- *de place en place*, *faire de la place* et *faire place* sont rattachés à *place*.

Rattachement optimal sur l'exemple N°4

Nous calculons "à la main" le meilleur rattachement possible à un seuil de **10**.

La charge globale est de **37**, et on dispose de **4** lettres pouvant accepter une charge de **10**, on peut donc perdre au maximum **3** unités sur toutes les lettres. Pour *état*, on ne peut pas faire mieux qu'une charge de **9**, et, pour *homme*, on peut obtenir une charge de **8** maximum. Les contraintes obligent donc à une charge de **8** pour *homme*, **9** pour *état* et **10** pour les deux autres lettres.

Pour obtenir la charge de **8** sur *homme*, il faut rattacher *d'homme à homme* et *homme à tout faire* à *homme*, ce qui oblige à rattacher *d'état à état*, *homme d'état* et *en état* à *état*, puisque ces mots n'ont plus d'autres lettres auxquelles les relier. De même, il faut rattacher *de place en place* à *place*. Ayant épuisé le capital de **3** lettres sur *état* et *homme*, il nous faut donner une charge de **10** à *place* et à *faire*. Il reste donc une charge de **6** à fournir à *place* qui ne peut se faire qu'au moyen de *faire de la place* et *faire place*. Les **3** mots restants contiennent tous *faire* et sont donc rattachés à ce mot.

D'après cette démonstration, nous avons prouvé que, en fonction des contraintes rencontrées, la répartition réalisant une charge maximale de **10** sur toutes les lettres est unique pour cet exemple.

Qualités

- Cet algorithme est **déterministe**: son ordre de complexité est en c^3 ou en n^3 comme nous le verrons. En cas d'échec, il n'y a que deux solutions: essayer un autre algorithme avec la même valeur du seuil, ou recommencer celui-ci avec un seuil supérieur d'une unité.
- Cet algorithme est **auto-correcteur**: si on accumule des choix de rattachement défavorables à une lettre, la charge globale de cette lettre va se maintenir par rapport aux charges des autres lettres qui diminueront. Dans les calculs de priorité des couples qui la contiendront, cette lettre fera augmenter la valeur de la charge maximale, donc la priorité de ces couples.
- Cet algorithme tend à **aplanir la fonction de répartition** des lettres sur les couples puisqu'il prend en premier les couples qui contiennent des lettres chargées, et diminue ainsi la pondération de ces lettres. Il est intuitif, qu'avec des lettres uniformément réparties sur les couples, on peut le plus facilement rattacher les couples de façon homogène. L'étape **1** de l'algorithme règle

le cas de toutes les lettres ayant une charge globale inférieure au seuil; il élimine donc toutes les parties basses de la fonction de répartition.

- Cet algorithme **détecte les situations critiques**: lorsqu'un couple est composé uniquement de lettres déjà prises, il acquiert une priorité infinie qui lui assure d'être retenu, sauf s'il y a plus de couples, avec une priorité infinie, que la lettre ne peut en contenir. Il ne détecte pas le cas où le seuil minimal ne pourra plus être respecté parce que le capital de lettres utilisables est épuisé. Ce point est revu plus loin.

Défauts

La méthode des priorités, utilisée ici, est une forme de méthode sérielle. Si dans la majorité des cas elle peut permettre d'obtenir directement la solution optimale, dans certaines situations, elle peut présenter des inconvénients.

- Dans certains cas il serait préférable de **ne pas prendre un couple de priorité maximale** pour en prendre un de priorité légèrement plus faible, mais dont le rattachement ne réussira pas ultérieurement.

- La méthode **n'est pas stable**. Si on a obtenu un seuil S_0 pour un ensemble de couples C, il est possible que la méthode donne un seuil S_1 supérieur à S_0 pour un ensemble de couples C' inclus dans C. Pour C', le calcul des priorités est différent de celui fait pour C et donne un classement des couples prioritaires différent. Il se peut que l'algorithme, alors appliqué à C', prenne trop tôt des couples qui bloquent des rattachements ultérieurs, et que la performance de l'algorithme sur C' soit moins bonne.

Les figures 26 et 27 montrent l'évolution de la charge des lettres après quelques itérations de l'algorithme. D'une part, tous les couples contenant au moins une lettre non globalement surchargée sont rattachés à cette lettre. Ces lettres disparaissent du graphique de la figure 26 puisqu'elles sont rattachées dès les premières étapes. D'autre part, les charges des lettres ont tendance à se niveler en raison des choix faits dans l'algorithme, c'est ce que montre la figure 26.

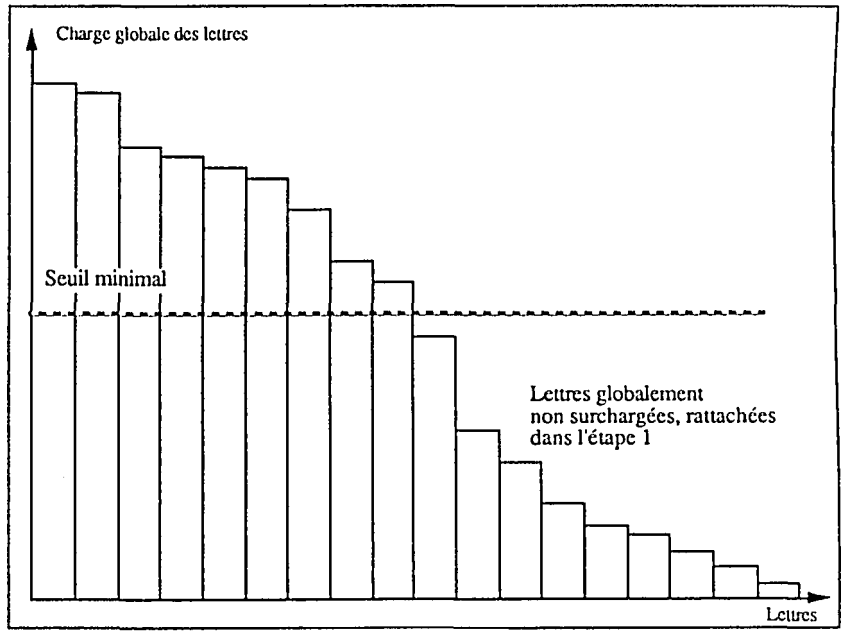


Figure 26: représentation de la charge initiale des lettres

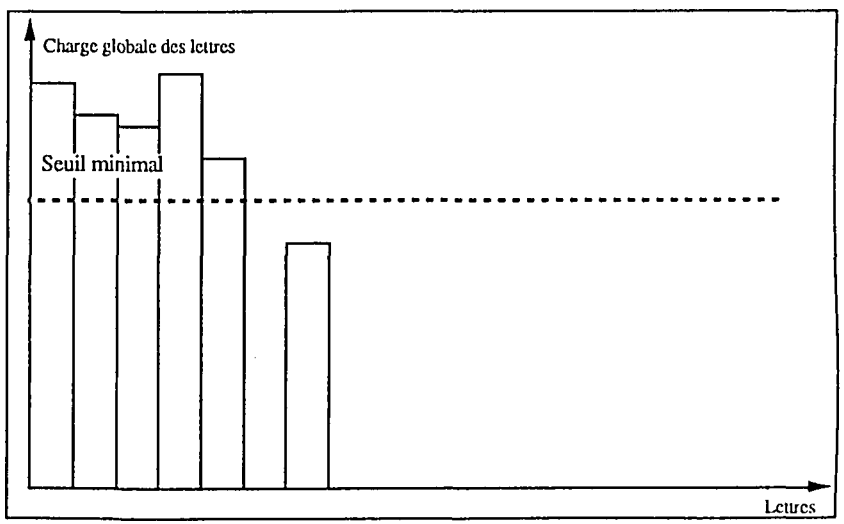


Figure 27: charge des lettres après l'étape 1 et quelques rattachements de l'étape 2

Variante

La variante proposée commence par remplir les lettres les plus chargées, afin de laisser plus de liberté pour rattacher les couples et mieux remplir les lettres.

Application de cette variante sur l'exemple N°4

Expérimentalement, cette solution donne des résultats légèrement inférieurs à ceux de l'algorithme initial. Nous décrivons ci-dessous les résultats obtenus par cette méthode sur l'exemple N°4, en prenant le seuil de **10**. Nous constatons que le rattachement de tous les couples ne peut être fait en respectant ce seuil.

Lettre	Charge	Couples rattachés	Capital restant
faire	9	c_4, c_5	2
état	8	c_1, s_6, c_{11}	0
homme	8	c_2, c_9	-2

Après le rattachement sur la lettre *homme*, deux raisons nous font arrêter l'algorithme:

1) le couple c_{10} ne pourra plus être rattaché à aucune des lettres restantes, puisqu'il ne contient que les lettres *état* et *homme* dont les rattachements sont déjà effectués.

2) La charge globale initiale est de **37**, or le seuil retenu est de **10**, nous avons donc un capital de charge de **4 x 10 - 37 = 3** qui peut être dépensé lors des rattachements inférieurs au seuil. A chaque rattachement donnant une charge inférieure au seuil retenu, nous diminuons le capital de la différence entre le seuil **10** et la charge de la lettre correspondante. Lorsque ce capital devient négatif, le rattachement des couples restants ne pourra plus être effectué en respectant le seuil, puisque la charge globale de ces couples est supérieure au produit du seuil par le nombre de lettres restantes.

Après le rattachement de *homme*, le capital restant est de **-2**, la charge globale des couples restants est de **12**, et il ne reste plus qu'une lettre non rattachée: *place* avec un seuil maximal de **10**. Ce qui prouve que, pour rattacher tous les couples restants, il faudrait que la charge de *place* dépasse de 2 unités le seuil minimal.
Nous donnons ci-dessous une définition plus formelle de cette notion de capital.

Nous recommençons donc le rattachement en choisissant un seuil de **11**. Le capital à dépenser est de **4 x 11 - 37 = 7**. Les deux lettres *état* et *homme* ne sont pas globalement surchargées, on peut donc leur rattacher tous les couples qui les contiennent.

Lettre	Charge	Couples rattachés	Capital restant
état	11	c_1, c_6, c_{10}, c_{11}	7
homme	8	c_2, c_9	4
place	11	c_3, c_4, c_8	4
faire	7	c_5, c_7	0

Les mots composés:

- *d'état à état*, *faire état*, *homme d'état* et *en état* sont rattachés à *état*.
- *d'homme à homme* et *homme à tout faire* sont rattachés à *homme*
- *de place en place*, *faire de la place* et *faire une place* sont rattachés à *place*.
- *faire de le sur place* et *faire place* sont rattachés à *faire*

Sur cet exemple, la variante est moins efficace que l'algorithme initial utilisant en premier les lettres les moins chargées. En effet, n'ayant pas connaissance des contraintes qui pourront peser sur les lettres non encore rattachées, elle risque de privilégier des couples qui auraient dû être rattachés plus tard, car contenant des lettres de charge globale faible.

Les trois variantes testées

Dans le tableau donné en annexe III.4, nous établissons les résultats obtenus avec divers algorithmes sur des données différentes. Nous présentons rapidement les caractéristiques de chacun des algorithmes.

Algorithme approché variante 1: algorithme approché initial. Le tri des couples, en fonction de leur priorité, est un tri standard par insertion.

Algorithme approché variante 2: algorithme approché initial. Le tri des couples par priorités décroissantes est un **tri rapide** [Aho 83]. Cet algorithme n'est pas plus performant que le précédent pour les données de type **Alpha**, puisque la taille des listes à trier est assez courte, et que la mise en oeuvre du tri rapide nécessite des appels récursifs et des recopies qui peuvent être pénalisant en temps.

Algorithme approché variante 3: variante de l'algorithme initial, définie ci-dessus, qui consiste à remplir en premier les lettres les plus chargées.

Gestion du capital de charge à dépenser

Une lettre ayant été choisie, les couples étant rattachés à une lettre donnée, nous proposons de permuter un couple avec un autre de priorité inférieure, si ce deuxième remplit "mieux" la lettre considérée. Pour observer le déroulement de l'algorithme, et prendre des décisions sur des heuristiques à choisir en cours de déroulement, on peut mesurer le **capital de charge** pouvant être perdu en ne remplissant pas les lettres jusqu'au seuil.

Rappelons la notion de seuil minimal s_{min}, défini par:

$s_{min} = \text{EntSup}(T / n)$ T la somme des tailles des couples de C.

Le capital de charge, disponible à un moment donné de l'algorithme, est la différence entre le produit du nombre de lettres non rattachées par le seuil s_{min}, et la somme des tailles des couples non rattachés.

Si on veut obtenir un remplissage des lettres inférieur ou égal à ce seuil, on a donc un capital de $(s_{min} \times n - T)$ unités pouvant être perdues sur les lettres incomplètement remplies. On peut tenir le compte de ce capital au cours du déroulement de l'algorithme, et bloquer la recherche d'une solution au cas où le capital est épuisé et devient négatif. En effet, si ce cas se produit, il y aura au moins une lettre dont la charge devra être supérieure au seuil pour pouvoir rattacher tous les couples restants.

Vers une construction d'heuristiques

Afin de ne pas dépenser trop vite ce capital d'unités, il convient donc de bien remplir les lettres et de trouver un compromis entre un remplissage maximal et un remplissage des couples les plus prioritaires. Il est nécessaire de définir une moyenne de capital pouvant être dépensé au cours de l'algorithme, en supposant que cette utilisation soit régulière. Dans le cas où cette moyenne viendrait à être dépassée au cours du déroulement, il conviendrait de donner un poids plus fort à l'algorithme de remplissage qu'à l'algorithme de priorité.

De même que l'on a donné une priorité infinie aux couples composés uniquement de lettres déjà rattachées, on peut envisager de donner une priorité infinie à l'algorithme du sac à dos dans le cas où le capital est épuisé ou près de l'être. Ce qui n'empêchera pas, dans le cas où l'algorithme du sac à dos donne plusieurs solutions équivalentes de choisir la plus prioritaire au sens des priorités définies précédemment.

Ecriture et exemple

Algorithme en pseudo-langage

L'algorithme global effectue successivement les étapes **1** et **2** en choisissant pour l'étape **2** une des méthodes proposées.

```
(* calculer le seuil minimal *)
Seuil := EntSup( ChargeGlobale / NombreDeLettresNonRattachées )

Répéter

    (* ETAPE 1 *)

    Tant Que
        (Il existe une lettre de charge globale inférieure à Seuil
          Et NombreDeCouplesRattachés < NombreDeCouples)
    Faire

            Rattacher Tous les couples qui contiennent cette lettre à
                cette lettre

            (* calculer le seuil minimal *)
            SeuilCourant := EntSup( ChargeGlobale
                                    / NombreDeLettresNonRattachées )

            Si ( SeuilCourant > Seuil )
            Alors
                Seuil := SeuilCourant;
            Fin Si

    Fin Tant Que


    (* ETAPE 2 *)

    (* calculer le capital de charge à dépenser *)
    Capital :=    Seuil * NombreDeLettresNonRattachées - ChargeGlobale


    Tant que (Il existe une lettre non rattachée Et Capital >= 0
              Et NombreDeCouplesRattachés < NombreDeCouples)
    Faire

        Prendre la lettre Lettre la plus prioritaire parmi les non rattachées

        Si la charge globale de Lettre est inférieure à Seuil
        Alors

            Rattacher tous les couples qui contiennent Lettre à Lettre

        Sinon

            Classer les couples qui contiennent Lettre par priorités
```

```
            décroissantes

        Tant Que Seuil n'est pas atteint
        Faire
            Rattacher les couples les plus prioritaires
        Fin Tant Que

        Modifier éventuellement le rattachement des couples pour
            que la charge de Lettre soit plus proche du seuil

        Capital := Capital  - ( Seuil - Charge( Lettre ) )

        Si il existe un couple de priorité infinie qui n'a pu être
            rattaché à Lettre
        Alors
            Capital := nombre négatif
            (* permet de sortir de la boucle Tant que *)
        Fin Si

    Fin Si
Fin Tant que

Seuil := Seuil + 1

Jusqu'à NombreDeCouplesRattachés = NombreDeCouples
```

Remarques

1 On peut ensuite réitérer l'étape 2 sur des variantes de l'algorithme et choisir celle qui aura donné les meilleurs résultats.

2 Une détection précoce des situations de blocage est mise en place au moyen du décompte du capital et de la détection de couples de priorité infinie ne pouvant être rattachés. Cette détection permet d'abandonner l'algorithme à un seuil donné, en ayant connaissance de son échec avant échéance. Bien que cette détection soit faite généralement vers la fin de l'algorithme, elle permet d'éviter de poursuivre une recherche inutile.

L'intérêt d'une telle surveillance, est de permettre à l'expérimentateur de suivre l'évolution de l'algorithme, comme on suit le déroulement d'une expérience de Sciences Physiques, et d'en déduire, en conséquence, les heuristiques les mieux adaptées au cas qui le concerne.

Algorithme approché appliqué à l'exemple N°1

Après application des trois transformations sur l'exemple N°1, donné en début du chapitre I.3, nous obtenons la liste suivante:

0 : (4, {*art, guerre*})	1 : (3, {*être, froid*})	2 : (2, {*grand, homme*})
3 : (2, {*froid*})	4 : (3, {*être, guerre*})	5 : (2, {*grand, place*})
6 : (3, {*main*})	7 : (3, {*être, place*})	8 : (2, {*grand, temps*})
9 : (2, {*temps*})	10 : (3, {*être, homme*})	11 : (2, {*grand, guerre*})
12 : (4, {*état*})	13 : (4, {*faire, place*})	14 : (2, {*froid, guerre*})
15 : (4, {*homme*})	16 : (5, {*faire, place*})	17 : (4, {*faire, homme*})
18 : (4, {*place*})	19 : (2, {*état, faire*})	20 : (3, {*état, homme*})
21 : (4, {*temps*})	22 : (2, {*faire, froid*})	23 : (4, {*art, homme*})
24 : (2, {*état*})	25 : (3, {*faire, guerre*})	26 : (3, {*homme, main*})
27 : (3, {*état, guerre*})	28 : (2, {*faire, place*})	29 : (4, {*main*})
30 : (4, {*art, état*})	31 : (3, {*faire, temps*})	32 : (3, {*art, place*})
33 : (4, {*être, temps*})	34 : (3, {*faire, place*})	35 : (4, {*faire, main*})
36 : (5, {*état, être*})	37 : (2, {*art, grand*})	38 : (2, {*froid, temps*})
39 : (3, {*état, être*})	40 : (2, {*froid, grand*})	

Nous récapitulons les étapes du rattachement, sachant que la charge globale est de **125**, le nombre de lettres de **41**, le seuil minimal est donc de **12** avec un capital disponible de **7**.

Seuil de **12**, étape **1**: rattachement des couples (mots composés) aux lettres (mots simples) non globalement surchargées

grand: rattachement de *grand homme, grand place, grand temps, Grande Guerre, grand art, grand froid.*

Charge de **12**, capital restant de **7**

froid: rattachement de *être en froid, à froid, Guerre Froide, faire froid, temps froid*

Charge de **11**, capital restant de **6**

Seuil de **12**, étape **2**: toutes les lettres sont globalement surchargées

guerre: rattachement de *faire la guerre, état de guerre, être en guerre,*

Charge de **9**, capital restant de **3**

temps: rattachement de *à temps, de temps en temps, faire son temps*

Charge de **9**, capital restant de **0**

main: rattachement de *à la main, main dans la main, se faire la main*

Charge de **11**, capital restant de **-1**

On arrête l'algorithme, car le capital à dépenser est épuisé. Le rattachement, par cette méthode, ne peut se faire à un seuil de **12**. On recommence toutes les étapes, avec un seuil supérieur d'une unité, soit un seuil de **13**.

Seuil de 13, étape 1:

Dans notre cas particulier, on retrouve les mêmes rattachements que pour le seuil 12 sur les lettres *grand* et *froid* à l'étape 1. Après *froid*, le capital restant est de 15.

guerre: rattachement de *art de la guerre, être en guerre, faire la guerre, état de guerre*
Charge de 13, capital restant de 15

art: rattachement de *homme de l'art, état de l'art, place d'art*
Charge de 11, capital restant de 13

temps: rattachement de *à temps, de temps en temps, faire son temps, être dans les temps*
Charge de 13, capital restant de 13

Seuil de 13, étape 2: toutes les lettres sont globalement surchargées

être: rattachement de *être en place, être dans tous ses états, être en état*
Charge de 11, capital restant de 11

état: rattachement de *d'état à état, faire état, homme d'état, en état*
Charge de 11, capital restant de 9

homme: rattachement de *être son homme, d'homme à homme, homme à tout faire*
Charge de 11, capital restant de 7

main: rattachement de *à la main, homme de main, main dans la main*
Charge de 10, capital restant de 4

faire: rattachement de *se faire la main, faire de la place, faire de la sur place*
Charge de 10, capital restant de 4

place: rattachement de *de place en place, faire place, faire une place*
Charge de 9, capital restant de 0.

L'algorithme proposé permet de rattacher le lexique donné, avec un seuil supérieur d'une unité au seuil minimum, dans un temps restreint, dont nous verrons qu'il est proportionnel au cube de la taille du lexique dans le pire des cas.

Commentaires sur la qualité du résultat obtenu sur l'exemple N°1

Nous avons cherché à vérifier, si la solution (seuil de 13) obtenue par l'algorithme approché, est la meilleure possible, dans ce cas particulier. Nous avons exploré tous les rattachements possibles pour cet ensemble de couples. Un programme testant toutes les solutions doit explorer 2^{32} possibilités, soit environ 4×10^9, ce qui sur micro-ordinateur donne un temps de calcul de l'ordre de 25 jours.

Cet algorithme peut être amélioré, de la même façon que l'on élague des branches dans l'arbre de recherche d'un coup à jouer dans le jeu des échecs, sans lui retirer son exhaustivité. On teste, pour chaque couple, tous les rattachements possibles à chacune des lettres qui le compose. Au lieu de

revenir sur tous les couples, successivement, pour parcourir la branche suivante de l'arbre de choix, l'algorithme modifié revient en arrière jusqu'au couple dont le rattachement a donné à la lettre correspondante une charge égale au seuil trouvé. On élague ainsi la partie restante de la branche sur laquelle on se trouve, et qui ne donnera pas un meilleur résultat que celui obtenu.
Un contrôle est également effectué à la montée des choix, pour ne pas prendre un rattachement qui donnera à une lettre, une charge supérieure au seuil minimal trouvé. Ceci permet de ne pas parcourir une branche dont on sait, à l'avance, qu'elle ne produira pas de meilleur résultat.

L'algorithme, ainsi amélioré, permet de connaître la valeur optimale du seuil et d'obtenir un rattachement correspondant. Il est beaucoup plus rapide que l'algorithme exhaustif, puisque l'obtention du résultat se fait en une cinquantaine de secondes. Mais ce résultat ne doit pas tromper sur les qualités de ce programme dont la complexité reste exponentielle et qu'il serait déraisonnable de vouloir appliquer sur un lexique de plus de cent mots composés (le temps d'exécution serait au mieux de l'ordre de $50 \times \alpha^n$ secondes, avec $\alpha > 1$, pour un lexique de $n \times 40$ mots composés).

Le résultat obtenu montre que le seuil optimal est bien de 13, celui que nous avions trouvé par l'algorithme approché.

Pour terminer avec cet exemple, nous avons également voulu connaître le nombre de solutions au seuil de 13 parmi tous les rattachements possibles. Pour cela, nous avons réécrit l'algorithme de parcours exhaustif, optimisé lui aussi, pour ne pas parcourir des branches conduisant à un résultat de seuil supérieur à 13. Nous obtenons 35336 solutions optimales, ce qui donne une proportion de l'ordre de 8 millionièmes pour les bonnes solutions.
Ceci confirme, sur un exemple, la qualité de l'algorithme approché, qui nous a permis d'atteindre avec une complexité faiblement polynômiale la solution optimale.

Les variantes 2 et 3 de l'algorithme approché permettent également de rattacher le lexique proposé avec un seuil de 13. Nous verrons que, dans le cas général, l'algorithme approché variante 3 donne de moins bons résultats pour les rattachements. Dans la plupart des cas, il oblige à accepter un seuil légèrement plus élevé.

Sur le lexique proposé, l'algorithme approché variante 2 diffère de la variante 1 en ce qu'il effectue les rattachements suivants:

faire de la place est rattaché à *place* au lieu de *faire*
faire place est rattaché à *faire* au lieu de *place*

et l'algorithme approché variante 3 diffère de la variante 1 par les rattachements suivants:

être son homme est rattaché à *être* au lieu de *homme*
faire état est rattaché à *état* au lieu de *faire*
homme de main est rattaché à *homme* au lieu de *main*
faire place est rattaché à *faire* au lieu de *place*
se faire la main est rattaché à *main* au lieu de *faire*
être en état est rattaché à *état* au lieu de *être*

Complexité en temps de l'algorithme approché

Temps de calcul de l'étape 1 (n: nombre de lettres ; c nombre de couples)
Reprenons la majoration de la complexité temps de l'étape 1, obtenue précédemment.

$c \leq n$: $T_1 \leq n^3 + n^2 + n = O(n^3)$

$c \geq n$: $T_1 \leq c^3 + c^2 + c = O(c^3)$

Temps de calcul de l'étape 2

Calculons une majoration de T_2, le temps de calcul de l'étape 2.

$T_2 \leq \sum_{i \text{ appartient à } [1, \min(c, n)]} T_i$ (T_i temps mis à l'itération de rang i)

En abordant l'étape 2, il reste, au pire, n lettres et c couples. Le nombre maximal de couples contenants chaque lettre est, au pire, c. Soit V, la variété maximale des couples. La variété d'un couple est le cardinal de son ensemble. Dans le cas général, V est majoré par n, la taille du vocabulaire. Dans le cas de la langue française, V est une constante de l'ordre de 5, négligeable devant la taille du vocabulaire.

L'étape 2 s'effectue en, au plus, n itérations (autant que de lettres). Chaque lettre est présente dans, au pire, c couples de variété maximale V.

- A chaque itération, le recalcul de la matrice G_0 n'est pas effectué. Chacune de ses composantes est mise à jour après les rattachement, en diminuant la charge de chaque lettre présente dans un couple rattaché de la charge de ce couple. Ce qui donne une complexité de mise à jour de $V \times c = O(c)$ (V la variété maximale des couples, et c le nombre maximal de couples contenants une lettre donnée).

- Le temps de calcul des charges maximales et minimales des lettres sur chaque couple est au pire de V-1. Soit, pour tous les couples, au pire: $c \times V = O(c)$.

- Le tri par priorités décroissantes des couples rattachés à une lettre. Un tri est d'un ordre de complexité, au pire en $O(c^2)$ pour une taille c, majorant du nombre de couples contenant une lettre. Pour le tri rapide, l'ordre de complexité est en $O(c \log(c))$.

- A chaque lettre, on rattache au plus c couples (pouvant être égal à 2 c dans le cas de la recherche d'un couple pouvant être permuté avec un couple déjà rattaché pour optimiser le remplissage). L'opération de rattachement est donc en $O(c)$.

Soit un temps total pour T_1 en: $O(c) + O(c) + O(c^2) + O(c) = O(c^2)$ de l'ordre du carré du nombre de couples.

$c \leq n$

$T_2 \leq \sum_{i \text{ appartient à } [1, n]} T_i$ chaque T_i étant en $O(n^2)$

donc $T_2 = O(n^3)$ donc $T_1 + T_2 = O(n^3)$

$c \geq n$

$T_2 = O(c^3)$ donc $T_1 + T_2 = O(c^3)$

Conclusion

Les résultats, au pire, sont donc du même ordre de grandeur, pour les deux étapes: un polynôme de degré 3 en c si la taille du lexique est supérieure à celle du vocabulaire, et de degré 3 en n dans le cas contraire.
Il s'agit là de valeurs maximales qui ne sont jamais atteintes, en particulier, parce que le nombre de couples contenant une lettre donnée est très inférieur à la taille totale c du lexique. On peut donc estimer que, dans la pratique, l'ordre de grandeur observé sera de degré 2.

L'algorithme approché est de complexité polynômiale de puissance 2 ou 3. Il est tout à fait réaliste de l'utiliser dans le cas de données volumineuses telles que celles que nous rencontrons dans la langue naturelle.

Qualité de l'algorithme approché

[Baase 78] présente des algorithmes approchés pour le problème du BinPacking et celui du sac à dos.Dans un cas d'optimisation par minimisation, la qualité de ces algorithmes est mesurée par la valeur maximale du quotient suivant: la valuation du pire résultat divisée par la valuation du résultat optimal. Dans le cas contraire d'une optimisation par maximisation, le quotient inverse sert de mesure de la qualité de l'algorithme. Ce quotient est donné en fonction de la taille du problème (S) ou de la valuation de la meilleure solution (R).

Pour le problème du sac à dos, l'algorithme proposé, part de sous-ensembles de taille k de l'ensemble initial, et tente de les étendre en piochant dans les objets restants classés par ordre décroissant. La valuation à maximiser, est la somme des tailles des objets retenus. Les pires quotients R et S sont de (1 + 1/k), quelle que soit la taille du problème. L'algorithme remplit , au pire, seulement (1 / (1 + 1/k)) du volume occupé par la meilleure solution. Si n est la taille du problème, il est de complexité $O(k \times n^{k+1})$, alors que l'algorithme exhaustif est en $O(2^n)$ (essai de tous les sous-ensembles).

Pour le BinPacking, l'algorithme retenu, choisit le premier objet convenable parmi les objets restants, classés par ordre décroissant. La valuation à minimiser est le nombre de boîtes utilisées. Le quotient **R** est, au plus, de (4/3 + 1/3 m), où **m** est la valuation de la meilleure solution. Le quotient **S** est au pire de 3/2, quelle que soit la taille du problème. Par cet algorithme, on utilise, au pire, une fois et demi plus de boîtes que la meilleure solution. Celui-ci est polynômial en $O(n^2)$, alors que l'algorithme exhaustif est en $O((n/2)^{(n/2)})$.

Dans le cas du rattachement des mots composés, nous prenons comme valuation du rattachement la charge maximale des lettres de l'alphabet par la pondération **X**. Afin de fournir aux utilisateurs de l'algorithme approché, une borne supérieure pour la charge des lettres, nous allons mesurer la qualité de l'algorithme en fonction de la valuation minimale théorique. Cette valuation a déjà été définie précédemment, nous l'appelons s_{min}, il s'agit de la valeur minimale à prendre pour le seuil maximal s_{max} de rattachement.

$$s_{min} = \text{EntSup}\Big(\sum_{(t_i, E_i) \text{ appartient à } C} t_i / n \Big)$$

Les mesures de qualité **R** et **S**, données dans [Baase 78], sont relatives au résultat de l'algorithme optimal. Celle que nous proposons est liée à la donnée du problème, indépendamment du meilleur rattachement. Il s'agit plus d'une mesure de la qualité du résultat que de la qualité de l'algorithme.

S'il n'y a aucune condition sur la taille des couples

Il est impossible de donner d'autre majoration de la charge par lettre, que la charge globale. Considérons le cas extrême d'un ensemble de couples composé d'un seul couple. Ce couple sera rattaché à l'une des lettres qui le compose, cette lettre aura pour charge la taille du couple qui est également la charge globale de chacune des lettres.

Exemple

A = { a, b } et C = { (56, { a, b}) }

Charges globales de **a** et **b**: **56**

Si (**56**, { **a**, **b**}) est rattaché à **a**, la charge de **a** est **56** et celle de **b** est **0**.

S'il n'y a aucune condition sur la répartition des couples sur les lettres

Il est impossible de donner d'autre majoration de la charge par lettre, que la charge totale de la lettre la plus chargée. Considérons le cas extrême d'un ensemble de couples composé d'une part de couples dont les ensembles sont uniformément répartis sur **99** lettres, et d'autre part de couples dont les ensembles sont formés avec une seule lettre **a**. Les couples formés avec **a** devront être rattachés à **a**, et obligeront à un seuil bien supérieur au seuil minimal.

Exemple

A = { a, a_1, a_2, ..., a_{99} }

C est composé de 110 couples (1, { a }) et de deux fois les 99 couples obtenus par permutation circulaire:

(5, {a_1, a_2, a_3, a_4, a_5}), (5, {a_2, a_3, a_4, a_5, a_6}), ... , (5, {a_{99}, a_1, a_2, a_3, a_4})

Charge globale de **a**: **110**; de a_1, a_2, ..., a_{99}: **50**; seuil minimal: **11**, toutes les lettres sont globalement surchargées.

Tous les couples ne contenant que **a** ne peuvent être rattachés qu'à **a**. La charge de **a** après rattachement sera de **110** soit **10** fois le seuil. Les couples restants peuvent être rattachés à leur première lettre, ils donneront à chaque lettre a_1, a_2, ..., a_{99} une charge uniforme de **10**.

Pour déterminer un seuil maximal pour l'algorithme, nous devons supposer que la répartition n'est pas trop mauvaise. Les cas défavorables sont ceux où la répartition, bien que globalement lourde, est plus concentrée sur certaines lettres.

Si la répartition est assez bonne et les tailles des couples assez petites:

Supposons que les ensembles des couples soient suffisamment bien répartis, c'est à dire qu'ils vérifient la propriété:

(A): à chaque étape de rattachement des couples sur une lettre, l'ensemble des couples qui ne contiennent plus que cette lettre comme lettre non rattachée, a une charge totale inférieure au seuil,

(B): la taille maximale de tous les couples est inférieure au seuil minimal retenu.

Appliquons sur un tel ensemble, un algorithme vérifiant les trois propriétés ci-dessous:

(1) lors du choix d'une lettre pour un rattachement, les lettres non surchargées sont prises en priorité,

(2) lors du rattachement d'une lettre globalement surchargée, les couples qui ne contiennent que cette lettre comme lettre non rattachée sont pris en priorité,

(3) lors du rattachement d'une lettre globalement surchargée, après avoir pris les couples qui ne contiennent que cette lettre comme lettre non rattachée, on complète avec des couples quelconques en s'arrêtant sur le premier couple permettant d'obtenir une charge strictement supérieure au seuil.

Propriété de seuil maximal

Soient **c** couples, de taille maximale **t**, formés sur un alphabet A de **n** lettres.

L'algorithme ayant les propriétés (1), (2) et (3) sur un lexique vérifiant les propriétés (A) et (B), permet d'obtenir un rattachement de tous les couples à une des lettres qui les composent avec une charge maximale de $s_{min} + t - 1$

s_{min} étant la valeur minimale du seuil : $s_{min} = \text{EntSup}\left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i / n \right)$

Démonstration par récurrence de la propriété de seuil maximal

Nous montrons qu'en rattachant les lettres avec une charge maximale de $s_{min} + t - 1$, sous les conditions précédentes, la valeur du seuil minimal ne peut que diminuer.

- La propriété est vraie au rang 1

$A = \{ a \}$ et $C = \{ (t_1, \{a\}), (t_2, \{a\}), \ldots, (t_c, \{a\}) \}$

$$s_{min} = \text{EntSup}\left(\sum_{j \text{ appartient à } [1, n]} t_j / 1 \right) = \sum_{j \text{ appartient à } [1, n]} t_j$$

Tout couple de **C** est rattaché à **a**; la charge obtenue pour a est s_{min}, elle est bien inférieure à $s_{min} + t - 1$.

- Supposons la propriété vraie au rang n - 1 et plaçons nous au rang n

Premier cas: si aucune lettre n'a une charge globale supérieure à $s_{min} + t - 1$, d'après la, propriété (1), on rattache indifféremment les couples à chacune de leurs lettres. La propriété est donc vraie au rang **n**.

Deuxième cas: si au moins une lettre de A a une charge globale supérieure à $s_{min} + t - 1$; on choisit une de ces lettres, soit **a**.

D'après la propriété (2) de l'algorithme, on rattache à **a** tous les couples qui ne contiennent que cette lettre comme lettre non rattachée. D'après la propriété (A) du lexique, la charge obtenue ne dépasse pas s_{min}. Ceci permet d'affirmer, que tous les couples, devant être rattachés impérativement, sont bien pris, et que l'algorithme ne risque pas de s'achever en laissant des couples non rattachés.

Puis, d'après la propriété (3), on ajoute des couples jusqu'à dépasser une charge de s_{min}. La plus grande charge inférieure à s_{min} possible pour a est $s_{min} - 1$; et tout couple de **C** permet d'obtenir une charge inférieure à s_{min} puisque d'après (B): $t < s_{min}$.

Donc, la valeur maximale de la première charge supérieure à s_{min} est $s_{min} + t - 1$.

On a donc rattaché à la lettre **a**, un ensemble de couples dont la charge est supérieure ou égale à s_{min}. Soit **c** cette charge et C_o l'ensemble des couples:

$$\text{(I)} \qquad s_{min} \leq c < s_{min} + t$$

Pour les **n-1** lettres restantes, calculons la valeur du seuil minimal:

$$\left(\sum_{(t_i, E_i) \text{ appartient à } C - C_o} t_i \right) / n - 1 = \left(\left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i \right) - c \right) / n - 1$$

$$= \left(\left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i \right) / n \right)(n / n - 1) - (c / n - 1)$$

Or, par définition du seuil minimal:

$$s_{min} - 1 < \left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i \right) / n \leq s_{min}$$

éq à $$\text{(II)} \quad (s_{min} - 1)(n / n - 1) < \left(\left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i \right) / n \right)(n / n - 1) \leq s_{min}(n / n - 1)$$

En soustrayant les deux inégalités (I) et (II); on obtient:

$$(s_{min} - 1)(n / n - 1) - (s_{min} + t) / (n-1) < \left(\sum_{(t_i, E_i) \text{ appartient à } C} t_i \right) / (n-1) - c / (n-1)$$

$$\leq s_{min}(n / n - 1) - s_{min} / (n-1)$$

éq à $$s_{min} - (n + t) / (n-1) < \left(\sum_{(t_i, E_i) \text{ appartient à } C - C_o} t_i \right) / (n-1) - c / (n-1) \leq s_{min}$$

Or $(n + t) / (n-1) > 1$; donc cette double inégalité prouve que le seuil minimal pour les **n-1** lettres restantes est au plus s_{min}.

La propriété de récurrence étant vraie au rang **n-1**, on peut rattacher tous les couples de $C - C_o$ à toutes les **n-1** lettres restantes avec une charge maximale par lettre de $s_{min} + t - 1$. La propriété est encore vraie au rang **n**.

Liens entre les conditions de la propriété de seuil maximal et les cas étudiés

Conditions sur le lexique

Les lexiques, que nous avons donné en exemple, vérifient la propriété (A). Les mots composés sont suffisamment bien répartis sur les mots simples pour ne pas se trouver, à une itération de l'algorithme, avec trop de mots composés devant impérativement être rattachés à cette étape. Mais nous ne pouvons pas affirmer que, tout lexique de la langue, vérifiera (A).
Dans le cas contraire, nous n'avons plus de garantie sur la valeur maximale du seuil. Il faut arrêter l'algorithme et le recommencer à un seuil plus élevé, pour ne pas laisser des mots non rattachés. Nous faisons alors passer le capital à un nombre négatif dans l'algorithme écrit en pseudo-langage.

La propriété (B) est vérifiée sur nos exemples, car la taille maximale des mots composés est inférieure à **10** alors que les seuils sont toujours supérieurs ou égaux à **15**. Elle est vérifiée pour le cas du français.

Conditions sur l'algorithme

Les variantes **1** et **2** de l'algorithme approché respectent les propriété (**1**) et (**2**). Elles attribuent une charge infinie aux couples qui ne contiennent plus que la lettre courante comme lettre non rattachée. Ces couples deviennent ainsi les plus prioritaires pour le rattachement. Pour la propriété (**3**), au lieu de dépasser le seuil aveuglément de la longueur d'un couple, nous produisons des tentatives successives en autorisant le dépassement du seuil d'une unité supplémentaire à chaque fois. L'obtention du résultat est plus lente, mais la qualité du rattachement est meilleure. Les exemples montrent qu'un dépassement d'une ou deux unités est suffisant dans tous les cas étudiés

La variante **3** de l'algorithme approché ne prend pas en priorité les lettres non globalement surchargées. Elle ne vérifie pas la propriété (**1**). La propriété de seuil maximal ne permet pas de donner une limite supérieure du seuil pouvant être atteint dans ce cas.

II LES TRANSFORMATIONS DES NOMS COMPOSÉS

La première partie de cette étude a présenté la reconnaissance morphologique des noms composés. Dans cette deuxième partie, nous mettons en oeuvre la reconnaissance des noms composés ayant subi des transformations. Par exemple, nous voulons pouvoir reconnaître *générateur industriel de système expert* comme une transformation de *générateur de système expert*.

En préalable à l'implantation informatique, nous détaillons l'étude des transformations des noms composés.
Au II.1.a, nous classons, en **trois catégories**, les transformations possibles sur trois structures de base: `N de N`, `N Adj` et `N à N`. Au II.1.b, nous proposons un formalisme de métarègle qui permet de les représenter et qui fait le lien avec l'expression des règles de noms composés, donnée en première partie.
Les derniers paragraphes du II.1 traitent des noms composés comportant, dans leur structure, une autre forme figée: les **noms composés imbriqués**. Pour décrire les transformations que peuvent subir de telles structures, nous faisons appel à des **métatransformations** qui permettent de reprendre les métarègles vues en II.1.a et b pour les noms composés de base. Nous en déduisons une représentation de la langue à trois niveaux.
La partie II.2 montre la réalisation informatique d'un générateur de règles au moyen de métarègles qui permet d'analyser les transformations subies par les noms composés décrites au II.1.

1 ÉTUDE LINGUISTIQUE DES TRANSFORMATIONS

Le problème des transformations de formes plus ou moins figées a déjà été envisagé dans [Salkoff 73]. Il propose une grammaire en chaîne du français qui tient compte des possibilités d'ajouts courts sur un nom dans le groupe nominal. Il montre que cette notion pose de nombreux problèmes dus à la nécessité de décrire en détail les diverses transformations que peuvent subir ces groupes nominaux. Ainsi, il constate que *un homme élégant grenouille* n'est pas correct alors que *machine américaine à laver la vaisselle* est acceptable.

De telles transformations, et il y en a encore de nombreuses, nous amènent à refuser un modèle génératif global pour représenter les noms composés. Ces structures plus ou moins figées nécessitent une description au coup par coup en précisant pour chacune les possibilités d'ajouts. On ne peut faire l'économie de l'énoncé exhaustif des transformations acceptables pour chaque nom composé car:

- ne pas accepter de transformation de la chaîne initiale, revient à ne pas recenser certains mots présents dans les textes et dont la compréhension est pertinente (silence dans l'indexation),
- accepter toutes les transformations, fait reconnaître des structures figées là où il y a des structures libres (bruit dans l'indexation ou indexation erronée).

Les noms composés ne se répartissent pas en deux classes: entre les formes figées d'un côté et les constructions "libres" de l'autre, on observe une palette étendue de possibilités. Dans des corpus de textes techniques ou scientifiques, ils sont plus nombreux que dans les textes généraux, et ils ont des structures moins figées que celles des mots de la langue courante. Dans de tels domaines, il est donc encore plus intéressant de connaître les transformations possibles. Le linguiste devra, soit bien connaître le domaine considéré, soit disposer d'un grand nombre de textes, soit se faire assister par un expert de ce domaine.

a Les catégories de transformations

Notre étude étant orientée vers le traitement automatique de la langue, nous ne nous sommes intéressés qu'aux transformations visibles subies par les noms composés. Ainsi, parmi les critères de figement recensés dans [G. Gross 88A] et [G. Gross 90], nous distinguons les propriétés directement utiles à une analyse automatique de celles qui relèvent de la description du mot. Ensuite, nous proposons une classification des transformations en trois catégories, définies sur des critères formels: conservation des mots ou remplacement de certains mots par d'autres.

Les critères de discrimination

Nous n'étudions pas ces critères, car ils ne peuvent être utilisés en analyse, mais ils permettent au linguiste de décider si un nom composé est plus ou moins figé. Ainsi, la rupture paradigmatique observée dans l'expression *intervention manuelle* qui ne s'étend pas en *intervention intellectuelle*, est un critère supplémentaire pour accepter ce nom composé comme étant une expression figée, mais ne peut servir en analyse automatique. Il résulte de l'observation d'un lexique complet de noms composés et des absences de certains mots dont on aurait pu conjecturer la présence en étendant une construction à une famille de mots.

Les critères syntaxiques

Ces critères nous intéressent, car ils vont permettre de ne reconnaître le nom composé lorsqu'il a subi des transformations acceptables. Le double but à atteindre est:

- d'éviter des reconnaissances trop larges en ne sélectionnant que les transformations convenant à cette structure,
- de ne pas manquer une reconnaissance du nom composé parce qu'une transformation acceptable a été oubliée..

Traduction de ces transformations dans l'énoncé d'un nom composé

Nous regroupons les transformations acceptées par les noms composés par structures (N de N, N Adj et N à N) et nous le numérotons. Puis, nous enrichissons la forme des règles de description des noms composés, vue au I.2.b, d'une liste de nombres précédés d'un signe +, -, ou ?, selon que la transformation associée est possible, impossible ou peu acceptable.

Par exemple:

```
<programmation à distance>
-> (N, !1, !1) programmation(N, f, v, n) à(P, v, v, n)
        distance(N, f, s, n)
        [() () (NàN +1 -2 -3 +4(programmer(V, v, v, n)) ?5 -6 -7)];
```

signifie que le nom composé considéré vérifie les transformations 1, 4 et 5 du tableau concernant les structures N à N avec la notion d'acceptabilité qui est liée aux caractères +, -, ou ?, comme nous l'indiquons dans le tableau de la figure 28. Le mot suivant +4 dans cet énoncé est un paramètre passé à la transformation 4 de ce tableau qui permet d'accepter la forme verbale *programmer à distance*. Le verbe souligné correspond à *programmation*, et n'est pas nécessairement déductible du nom d'origine, c'est pourquoi il doit être précisé avec l'énonciation de la règle.

	Modifieur du premier N	Modifieur du deuxième N	Coordination	Forme verbale V "à" N	Prédicativité : N "est à" N	Lien avec une forme en N Adj	Acceptation de N "à" Dét N
programmation à distance	+	-	-	+	?	-	-

Figure 28: tableau d'acceptabilité des transformations du nom composé *programmation à distance*

Trois catégories de transformations des noms composés

Dans cette partie, nous allons dégager trois sortes de transformations de la structure des noms composés, chacune présentant des difficultés d'analyse différentes:

Catégorie 1 Les transformations qui conservent tous les mots de la structure initiale, mais qui peuvent

- briser la connexité de la structure (insertions...),
- changer l'ordre des termes (permutations...),
- réaliser ces deux opérations simultanément (détachement...).

Catégorie 2 celles qui transforment certains mots de la séquence en des mots "pleins" qui ne sont ni des pronoms, ni des particule pré-verbales, ni le mot vide.

- Elles remplacent un adjectif ou un verbe par un nom (nominalisation...).
- Elles changent la flexion d'un mot (en particulier celle des verbes).

Catégorie 3 celles qui suppriment certains mots ou les transforment en des mots "vides" tels que pronom, coordination, mot vide... (coordination, référence elliptique, interrogation...).

Les transformations de la première catégorie sont les plus simples à observer car tous les mots sont présents lors de l'analyse. Celles de la deuxième catégorie sont également faciles à reconnaître, si les mots nouveaux introduits dans la chaîne ont été prévus explicitement dans l'énoncé de la règle. Celles de la troisième catégorie sont beaucoup plus délicates à observer car nous n'avons qu'une partie des mots initiaux présents et nous devons conjecturer d'après le reste de la phrase si nous sommes bien dans un cas où la transformation a eu lieu.

Nous allons donner quelques exemples de transformations classées selon cette typologie. Nous avons regroupé ces opérations en fonction de la structure du groupe nominal sur lequel elles s'appliquent: (N de N, N de Dét N, N à N et N N). Nous précisons à chaque fois son indice dans le tableau correspondant (voir figures 41,42 et 43). Cette présentation est homogène avec les tableaux utilisées au L.A.D.L.

Catégorie 1: tous les mots sont conservés

Transformation N° 3 de N Adj: adjonction d'un adverbe.

La structure N Adv Adj est possible pour certains noms composés:
production locale se trouve dans: *une production souvent locale,*
alors que *système expert,* n'acceptera pas de telles transformations.

Si un adverbe est accepté, le mot se trouvera également dans des comparatifs vrais ou intensifs:
un production plus locale que celle de l'acier
**un système plus expert que celui de nos concurrents*
un production plus locale que régionale
**un système plus expert qu'informatique.*

Cette adjonction d'un adverbe est l'extension à une forme figée de la modification adverbiale d'un adjectif qui se rencontre sur les formes libres:
Max lit un livre (très + peu + énormément +...) intéressant.
Dans le cas de formes libres, la validité de la transformation est liée à la compatibilité entre l'adverbe et l'adjectif, et dépend peu du nom. Dans le cas d'un nom composé, la validité ne pourra pas être déduite de la seule observation de l'adverbe et de l'adjectif. Ainsi, le nom composé *matière grise* n'accepte pas cette transformation, tandis que le nom composé *automate programmable* se rencontre sous la forme *automate entièrement programmable.*

Transformation N° 6 de N Adj: acceptabilité de la forme en N non Adj.

Cette acceptabilité semble plus liée à l'adjectif seul qu'à l'ensemble Nom Adjectif. Ainsi *état gazeux* et *état non gazeux* sont acceptables tous les deux, de même que, dans un contexte différent, et pour une entrée différente de l'adjectif, on accepte *boisson gazeuse* et *boisson non gazeuse.*
Réaction chimique n'a pas pour opposé *réaction non chimique.*

Transformation N° 1 de N de N: modifieur adjectival du premier N.

Modifieur postposé

L'acceptabilité de cette transformation est difficile à décider, car elle dépend du contexte d'utilisation du mot.
La boucle actuelle d'essai
**le courant fort d'air*
?la courbe croissante de charge

Dans ce cas, la modification du nom peut être différente de la forme qu'elle aurait sur une structure syntaxique libre. Une deuxième transformation se compose avec la modification pour réorganiser le composé et rejeter le modifieur hors du composé afin d'en préserver la connexité, c'est pourquoi nous la noterons **[connexe]**. Nous la reprendrons au II.1.d, en la formalisant comme une métatransformation, car elle permet d'enrichir l'ensemble des transformations définies sur une structure.
Ainsi la transformation:

[Modif]	N	-> N Adj
	perte	-> *perte réduite*

est suivie de [connexe] qui se compose avec la métarègle initiale, et permet de conserver le nom composé d'un seul tenant.

[Modif]	N de N	-> N Adj de N
	perte de chaleur	-> *perte réduite de chaleur*
[connexe]	N Adj de N	-> N de N Adj
	perte réduite de chaleur	-> *perte de chaleur réduite*

L'application de [connexe] n'est pas systématique et dépend du degré de figement du composé face à la modification adjectivale du nom. Ainsi, pour certains composés, on peut trouver les deux formes:
direction fixe de propagation et *direction de propagation fixe.*

Modifieur antéposé

Il n'y a pas de problème à accepter un modifieur avant le premier nom, et il est alors difficile de savoir s'il s'applique au premier nom seul ou au nom composé dans son ensemble:

le nouvel algorithme de commande ou *la basse température de fusion.*

Appelons **[antéposition]** la transformation qui place un adjectif devant le nom. Elle opère sur la forme N Adj et permet de garder l'adjectif à proximité du nom qu'il modifie, renforçant ainsi l'intelligibilité de la structure. Il s'agit également d'une métatransformation, elle sera revue au II.1.d.

Ainsi la transformation:

```
[Modif]          N          -> N Adj
                 champ      -> champ important
```

est suivie de `[antéposition]` qui se compose avec la métarègle initiale, et permet de conserver le nom composé d'un seul tenant.

```
[Modif]          N de N                  -> N Adj de N
                 champ d'application     -> champ important d'application
[antéposition]   N Adj de N                        -> Adj N de N
                 champ important d'application     -> important champ d'application
```

Nous verrons, au II.1.c, que l'emploi d'adjectifs antéposés sur les noms composés est plus acceptable que sur les mots simples seuls, puisqu'il a le double avantage de laisser l'adjectif proche du nom qu'il modifie, et de ne pas briser l'unité du nom composé.

Transformation N° 2 de N de N: modifieur adjectival du deuxième nom et s'appliquant uniquement à celui-ci:

système d'équations donnera *système d'équations indépendantes.*

Il convient, ici, de bien distinguer cette transformation de la précédente, même si la deuxième forme de la précédente (*perte de chaleur réduite*), est identique à celle-ci dans le cas d'un adjectif postposé: `N de N Adj`. Dans le cas où il n'y a pas de discrimination possible au moyen de l'accord entre un des deux noms et l'adjectif, c'est la compatibilité entre le premier ou le deuxième nom et l'adjectif qui peut parfois nous renseigner.

Dans le cadre d'une analyse syntaxique de formes libres, ces deux structures superficielles en `N de N Adj` correspondent à deux structures syntaxiques distinctes (figure 29 et 30).

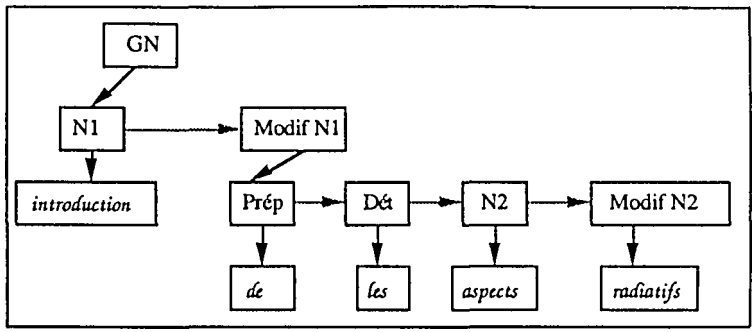


Figure 29: modification du deuxième nom: *introduction des aspects radiatifs*

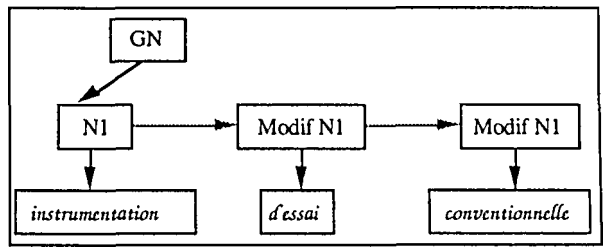


Figure 30: modification du premier nom: *instrumentation d'essai conventionnelle*

Intérêt en analyse automatique: la résolution du problème de rattachement des modifieurs.

La discrimination entre ces deux formes de modifications d'un nom composé en `N de N` (modification du premier ou du deuxième nom) permet de résoudre, dans ce cas particulier, le problème du rattachement des modifieurs, qui joue un rôle important dans une interprétation sémantique ultérieure de la phrase. Une analyse à l'aide de règles de grammaire générative donne un résultat ambigu, alors que la connaissance des transformations possibles des noms composés décrite dans le lexique permet de distinguer ces deux formes dans le cas où une seule des deux est autorisée. Cette transformation s'étend naturellement à des modifieurs autres qu'adjectivaux pouvant s'appliquer au deuxième nom. Ainsi *mémoire de thèse* deviendra *mémoire de thèse de physique.*

Pour la modification du deuxième nom, il n'y a pas insertion d'un nouvel élément dans la chaîne. Bien que ce modifieur soit extérieur au nom composé, il est intéressant de savoir s'il est inclus dans la structure ou s'il doit être traité à un niveau supérieur de l'analyse. Nous distinguons ci-dessous 4 cas de modifications du type `N de N Adj` faisant intervenir un nom composé.

Dans les trois premiers exemples (figures 31, 32 et 33), l'adjectif est rattaché à la structure:

- dans le premier cas `N de N` est un nom composé et l'adjectif modifie le premier nom (transformation N°1 de `N de N`),
- dans le deuxième cas, `N de N` est aussi un nom composé dont le deuxième nom est modifié par l'adjectif (transformation N°2 de `N de N`),
- dans le troisième cas, `N Adj` est un nom composé qui forme avec la préposition *de* un modifieur du premier nom.

Dans le quatrième cas (figure 34), aucune des trois structures précédentes n'est possible, l'adjectif doit donc être rattaché à un autre niveau de l'analyse.

Malgré une bonne connaissance des formes figées, il peut persister des ambiguïtés sur le rattachement des modifieurs, comme dans *courant d'air frais*. *Courant d'air* est un nom composé qui admet les modifications 1 et 2 de `N de N`, mais l'adjectif *frais* peut aussi bien modifier *courant* que *air*. Pour *courant d'air frais*, il n'est pas certain que la distinction entre modifieur du premier et du deuxième nom soit nécessaire, on peut alors considérer *frais* comme un modifieur de *courant d'air*.

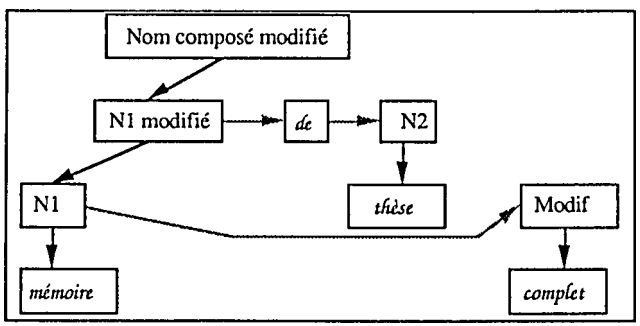


Figure 31: cas 1, modifieur postposé du premier nom, déplacé par [connexe] *mémoire de thèse complet*

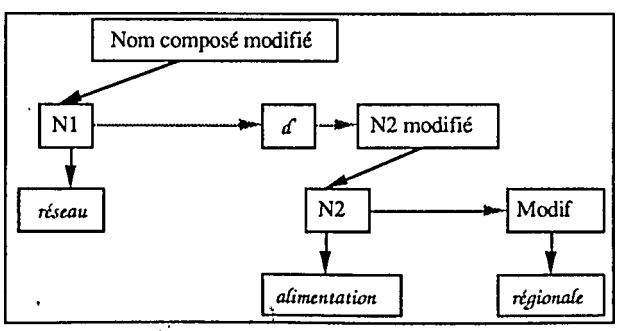


Figure 32: cas 2, modifieur postposé du deuxième nom *réseau d'alimentation régionale*

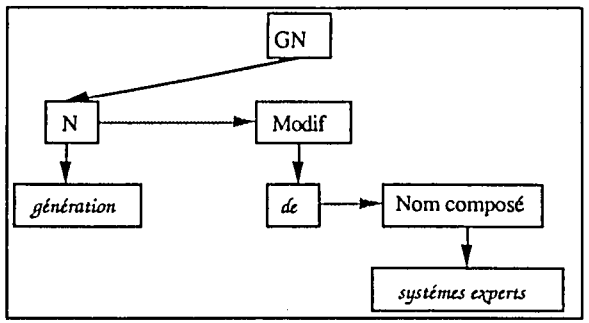


Figure 33: cas 3, modifieur formant un nom composé avec le deuxième nom
génération de systèmes experts

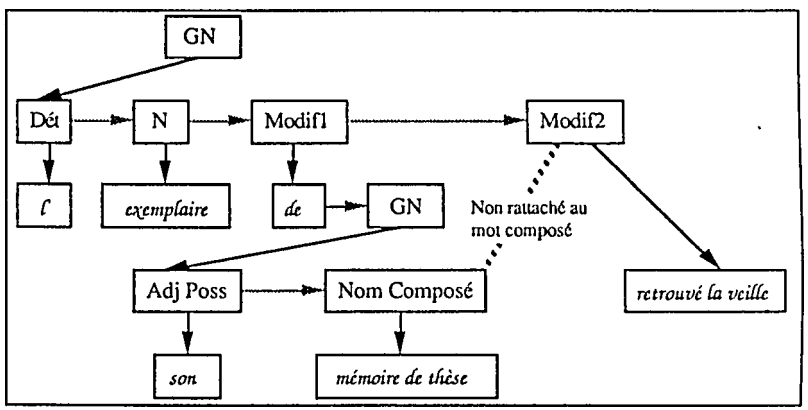


Figure 34: cas 4, modifieur non rattaché au nom composé
Max donne l'exemplaire de son mémoire de thèse retrouvé la veille

Transformation N° 1 de `N Adj`: prédicativité, ces mots peuvent entrer dans la structure `N est Adj`.

Une équation polynômiale se trouve dans *cette équation est polynômiale,*

un débit gazeux ne se trouve pas dans **ce débit est gazeux* en gardant son sens.

L'acceptabilité de cette structure conduit pas systématiquement à `N est Adv Adj`:

**cette équation est très polynômiale*

Transformation N° 7 de `N Adj`: modifieur adjectival du nom.: *air frais ambiant* s'obtient à partir de *air ambiant*.

Transformation N° 5 de N de N: acceptabilité de la forme en N de Dét N ou des formes similaires telles que N de Ddéf Nsing Modif par exemple.

Régulateur de tableau donne *régulateur du tableau électrique,*

courant d'air ne subit pas une telle transformation: **le courant de l'air chaud*

Cette acceptabilité permet à certains noms composés tels que *régulateur de tableau* de partager les transformations des structures N de N et N de Dét N. Les similitudes de comportement entre ces deux structures suggèrent de regrouper ces deux formes de noms composés distinctes pour plus de concision. Ce choix a été fait au L.A.D.L.

Catégorie 2 : certains mots pleins sont remplacés par d'autres

En présentant la notation des transformations au début de ce paragraphe, nous avons indiqué que certaines d'entre elles doivent connaître le ou les nouveaux mots qui sont introduits dans la structure. Ainsi, dans la transformation N°5 de N Adj présentée ci-dessous, *four cimentier* donne *four de cimenterie*. Il faut donc mettre *cimenterie* en paramètre. En plus de la forme canonique de ce mot, il faudra fournir des indications sur sa nature (nom féminin), ou sur son comportement dans la structure modifiée (nombre, accords éventuels...).
Dans les exemples de cette catégorie, nous soulignons le nouveau mot introduit.

Transformation N° 2 de N Adj: nominalisation, la forme N Adj peut se transformer en une forme $N\ V_{sup}$ Nominalisation.

Un écoulement turbulent est lié à *cet écoulement présente des turbulences.*

Cette transformation est souvent permise sur les noms composés:

la production nationale = la production de la nation = la production faite par la nation

équation énergétique = équation de l'énergie = équation vérifiée par l'énergie

érosion sédimentaire = érosion des sédiments = érosion subie par les sédiments

l'intelligence artificielle ne peut être reliée avec **l'artificialité de l'intelligence.*

Transformation N° 5 de N Adj: lien entre la forme N Adj et une forme à complément de nom N de N.
S'il s'agit d'un adjectif de relation, donc non prédicatif, le passage entre les deux formes est possible en général. Ainsi:

four cimentier est équivalent à *four de cimenterie,*

code numérique ne donne pas **code de nombre.*

Dans ce deuxième cas la valeur prédicative de *numérique* empêche le passage à une forme à complément de nom.

Les relations d'adjectivation sont symétriques et peuvent être un passage réciproque vers N de N ou N à N ou N en N avec une préférence pour la forme en N Adj ou en N Prép N.

Transformation N° 4 de N de N: lien entre une forme N de N et une forme N Adj.

Cette transformation est la réciproque de la transformation vue pour la transformation N°5 de N Adj. *Apport d'énergie* et *apport énergétique* sont deux structures différentes pour un même concept.

Ces transformations ne sont pas régulières: *gisement de pétrole* donne *gisement pétrolifère*, *coke de pétrole* ne donne pas **coke pétrolifère*. Le sens de l'adjectif *pétrolifère* lie la possibilité de passage à la forme N Adj au nom qui le précède. Pour automatiser l'adjectivation, il est possible d'attribuer deux traits distincts pour les deux premiers noms, tels que "renfermer" pour *gisement* et "corps minéral" pour *coke* et deux traits similaires pour les deux adjectifs *pétrolifère* et *pétrolier*. La compatibilité des traits entre le nom et l'adjectif permet de choisir le bon adjectif lors du passage de N de N à N Adj et empêche les associations incorrectes **coke pétrolifère* ou *gisement pétrolier*. Pour les adjectivations de noms, [Dubois 65] et [Dubois 70] propose d'attribuer au deuxième nom des traits qui permettent d'en déduire l'affixe pour la construction de l'adjectif associé. Cette solution ne résout pas le problème de la compatibilité avec le premier nom et soulève de nombreuses questions qui dépassent le cadre de cette étude. Nous avons choisi d'indiquer clairement, pour chaque composé, l'adjectif associé, en laissant de côté un système de traits plus compact, mais plus difficile à gérer.

Examinons l'intérêt d'une telle transformation selon que l'on envisage l'aspect opératoire ou l'aspect taxinomique:

- une telle opération permet de faire le lien "horizontal" entre des structures dérivées, et d'associer à une forme adjectivale, une forme nominale dans le cadre d'une forme composée,
- pour ce qui est de l'utilisation des métarègles dans les mécanismes d'analyse, l'énoncé et l'utilisation de telles métarègles peut présenter des dangers. Il est certes économique, de pouvoir décrire deux formes (adjectivales et nominales), à partir d'une même forme de base plutôt que de les donner en parallèle, mais celle des deux formes qui sera déduite de l'autre sera décrite de façon incomplète. On aura seulement son expression sans connaître les transformations qui peuvent s'y appliquer.

Autant il est utile qu'une base de données lexicales possède des informations de type nominalisation / adjectivation; autant leur intérêt opératoire est réduit puisque spécifier le lien entre les deux formes ne dispense pas de les décrire de façon exhaustive (en particulier pour les transformations qu'elles acceptent).

Catégorie 3: des mots pleins sont remplacés par des mots vides

Elisions dans les cas de référence

Nous nous intéressons, ici, aux cas de référence lexicale ou de coréférence, où un mécanisme d'ellipse permet de désigner une structure par une partie de celle-ci.

[M. Gross 86N] distingue deux types de références: la coréférence qui est la relation sémantique entre deux groupes nominaux du discours, et la référence lexicale qui est le lien entre deux groupes nominaux sans qu'il y ait référence identique au monde extérieur. Ces deux possibilités produisent des phénomènes syntaxiques proches.

Nous souscrivons au point de vue développé dans [M. Gross 90] qui indique que la référence, présentée ici, est, à la fois une référence lexicale (désignation d'un élément lexical identique) et une coréférence (désignation d'un objet du discours unique). En effet, la réduction de la forme initiale apparaît généralement après une présence de la forme étendue dans le texte, qui permet ensuite de la réduire pour représenter le même élément lexical ou le même objet. Pour des notions connues du lecteur et lorsque le contexte permet de désambiguïser, la référence elliptique peut se faire sans énoncé préalable de la forme complète comme pour *les douze (pays de la C.E.E.), le compteur (électrique + kilométrique)* ou *le (vin + feu) rouge*. Dans ce cas, c'est une référence au contexte, partagée et implicite, qui permet d'éviter l'énonciation initiale.

Nous étudions simultanément ces deux types de référence, en nous plaçant dans le cas où elles sont faites au moyen d'une partie de la chaîne du nom composé, et non pas à l'aide d'un pronom seul, puisque ce dernier cas relève de la syntaxe générale du groupe nominal. Par exemple, *logement type F1, logement F1, F1* et *type F1* servent à faire référence à *logement de type F1*.

On pourra aussi avoir *la carte* ou *cette carte* pour désigner *carte bancaire* (coréférence). *Compression mécanique de vapeur* a été rencontrée sous la forme abrégée: *compression de vapeur*, ou *compression* (référence lexicale)

Les réductions ne portent pas toujours sur les fins des chaînes; elles sont liées à la structure syntaxique du groupe nominal considéré. Sur la figure 35, les parties optionnelles ont été entourées.

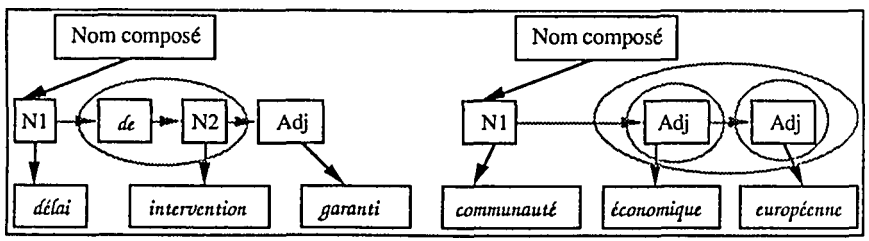


Figure 35: réductions possibles pour *délai d'intervention garanti* et *communauté économique européenne*

Les noms composés contenant des noms propres sont fréquemment référencés par une partie de la structure, généralement le nom propre y est inclus:

un micro-ordinateur IBM devient *un IBM*

un micro-ordinateur compatible avec un micro-ordinateur IBM devient *un compatible*, ou un *un compatible IBM*.

La **méthode systématique d'investigation** possible consiste à construire toutes les sous-structures d'une structure donnée, et à noter, pour chacune, si elle peut représenter le même terme que le terme d'origine dans un cas de référence. Par exemple, *compression mécanique de la vapeur* est formé de cinq mots, il y a donc 2^5 - 2 candidats, dont certains ne correspondent pas à des structures syntaxiquement acceptables. Nous donnons la liste des sous-chaînes ci-dessous, en barrant celles qui ne sont pas des structures nominales correctes et en soulignant celles qui sont retenues.

Taille 1: *compression, mécanique, ~~de~~, la, vapeur*

seul le premier est retenu, il correspond au nom principal du syntagme nominal.

Taille 2: *compression mécanique, ~~compression de~~, ~~compression la~~, compression vapeur, ~~mécanique de~~, ~~mécanique la~~, mécanique vapeur, ~~de la~~, ~~de vapeur~~, la vapeur*

Taille 3: *~~compression mécanique de~~, ~~compression mécanique la~~, compression mécanique vapeur, ~~compression de la~~, compression de vapeur, ~~compression la vapeur~~, ~~mécanique de la~~, mécanique de vapeur, ~~mécanique la vapeur~~, ~~de la vapeur~~*

Taille 4: *~~compression mécanique de la~~, compression mécanique de vapeur, ~~compression mécanique la vapeur~~, compression de la vapeur, mécanique de la vapeur*

Les chaînes, telles que **compression de la*, n'ont pas une structure syntaxique correcte. La plupart de celles qui sont analysables comme un groupe nominal libre, sont retenues. Pour que la référence soit correcte, il faut que les composants conservent leur catégorie lexicale: *la* ne peut être un pronom et *mécanique* ne peut être un nom. C'est pourquoi, *mécanique de vapeur*, bien que syntaxiquement analysable comme un `N de N` ne sera pas retenue puisque *mécanique* doit rester adjectif.

Certains cas de référence elliptique avec changement de catégorie lexicale peuvent être corrects, ils sont appelés **dérivations impropres ou implicites** par [Grevisse 86]. Cette notion de changement de catégorie lexicale doit être considérée avec précautions. Par exemple, *le supérieur* qui sert à désigner *l'enseignement supérieur*, ne pourra pas se trouver dans toutes les structures lexicales acceptables pour un nom libre. Ainsi, il sera difficile de lui associer un modifieur adjectival comme dans ?*le supérieur français* ou **le supérieur universitaire*, alors que les structures complètes sont acceptables: *l'enseignement supérieur français* ou *l'enseignement supérieur universitaire*.

Certaines références font appel à des termes plus spécifiques ou plus génériques dont la compréhension nécessite une connaissance du contexte: *ce moyen de paiement* ou *la carte bleue* servent à représenter *carte bancaire*. Ce type de référence est très utilisé dans le style journalistique: *M. Rocard, le Premier Ministre, Matignon* désignent la même personne, et *IBM, Big Blue, le N°1 mondial* la même entreprise. Ces liens extralinguistiques n'ont pas été recensés dans noure système qui s'attache aux transformations syntaxiques.

La coordination

Le mécanisme de la coordination est une difficulté importante de l'analyse automatique, car il oblige à faire des conjectures sur les possibilités d'associations de syntagmes. Si nous nous intéressons au terme *outil de recherche*, nous pouvons le trouver dans des coordinations où le nom composé est factorisé avec un autre mot de structure proche:

outil de recherche et de développement.

Nous reprenons la définition de la coordination donnée dans [Grevisse 86]. Il s'agit de la mise en relation explicite ou implicite d'éléments de même statut (des phrases ou des termes qui ont la même fonction par rapport au même mot). Nous ne distinguons pas ce que les grammaires traditionnelles appellent **juxtaposition** [Béchade 86] ou **coordination implicite** [Grevisse 86], qui est une coordination sans présence de conjonction, de la "vraie" coordination avec conjonction. Les mécanismes syntaxiques qui les sous-tendent sont similaires. Pour justifier cela, [Touratier 90] fait remarquer que le couple formé par une conjonction de coordination et un constituant est de même nature que le constituant seul. Ainsi écrit-on indifféremment:

Lus achète, répare, vend des meubles ou *Lus et achète et répare et vend des meubles.*

Pour introduire une définition transformationnelle, nous rappelons que [Chomsky 57] se sert de la coordination pour introduire la notion de **transformation facultative**, en donnant une définition formelle sous la forme:

`X Y Z` *et* `X W Z -> X Y` *et* `W Z`

à condition que `X` et `Y` soient des constituants de la phrase. Lorsque `X` et `Y` sont de natures différentes, certains cas de coordination sont encore possibles comme pour *une carte bancaire ou de crédit* ou douteux comme: *?il est arrivé en avion et en colère.*

[Ruwet 67] précise que cette définition est trop large, car elle permet d'accepter des formes asyntaxiques telles que *?Pourquoi partez-vous et fermez la porte.* Elle nécessite une analyse plus fine qui tienne compte des constituants des structures de base. Cette description doit être très précise car, comme le remarque [Hobaek Haff 90], s'il est impossible de coordonner une interrogative et une impérative comme le montre l'exemple précédent, il est possible de coordonner une déclarative

et une impérative comme dans *vous pouvez venir, mais soyez prudent.* Afin de décrire de telles possibilités, elle observe deux variables: le type de proposition et le type d'acte illocutoire. Elle note que le type de conjonction de coordination joue également un rôle important dans l'acceptabilité: *ou* et *mais* fournissent des structures plus acceptables que *et* dans le cas de coordinations de constituants de natures différentes, ainsi pour le cas de déterminants de nombres différents: *le ou les responsables seront reconnus* alors que *?les et les responsables seront reconnus.*

[Gadzar 85] propose un mécanisme universel de coordination qui s'appuie d'une part sur des règles syntagmatiques et des règles de précédence linéaire, et d'autre part sur des schémas de règles. Les coordinations sont séparées en deux types: les binaires et les n-aires avec n supérieur à deux. Des catégories différentes peuvent être conjointes, à condition que l'intersection de leurs traits de tête produise une nouvelle tête acceptable par la structure dominante. Bien que faisant appel aux principes des Grammaires Syntagmatiques Généralisées (exposées dans [Miller Torris 90]), cette description a un fondement linguistique: on peut coordonner deux structures qui ont suffisamment de points communs pour constituer un constituant coordonné acceptable en remplacement des constituants initiaux.

Mais il reste des cas où la coordination ne peut être vue comme la réduction de deux structures complètes comme le fait remarquer [Pinkster 90] pour *le bleu et le rouge alternent.* Il présente alors la coordination comme un mécanisme d'expansion d'un élément avec un élément semblable.

La coordination des noms composés

Dans le cadre des noms composés, nous présentons un mécanisme de coordination par réduction de deux structures, il est illustré par la figure 36.

Soient S_1 et S_2 deux structures syntaxiques de même catégorie, la transformation C de coordination leur associe une structure coordonnée $S_{12} = C(S_1, S_2)$ avec les contraintes suivantes sur les arbres syntaxiques de S_1 et S_2:

- les feuilles de S_1 forment une chaîne Ch_1 qui est la concaténation de trois chaînes C_1, C_2 et C_3; C_2 correspond à un sous-arbre de S_1,
- les feuilles de S_2 forment une chaîne Ch_2 qui est la concaténation de trois chaînes C_1, C'_2 et C_3; C'_2 correspond à un sous-arbre de S_2 de même nature que celui correspondant à C_2.

Ch_{12}, la chaîne de la structure coordonnée S_{12} est la concaténation des cinq chaînes C_1, C_2, `ConjCoord`, C'_2 et C_3 où `ConjCoord` est une conjonction de coordination. La coordination peut être réalisée à condition que C_2 et C'_2 soient des structures syntaxiques complètes et compatibles. C_1 et C_3 peuvent éventuellement être la chaîne vide (figure 36).

S_1 et S_2 -> $S_{12} = C(S_1, S_2)$

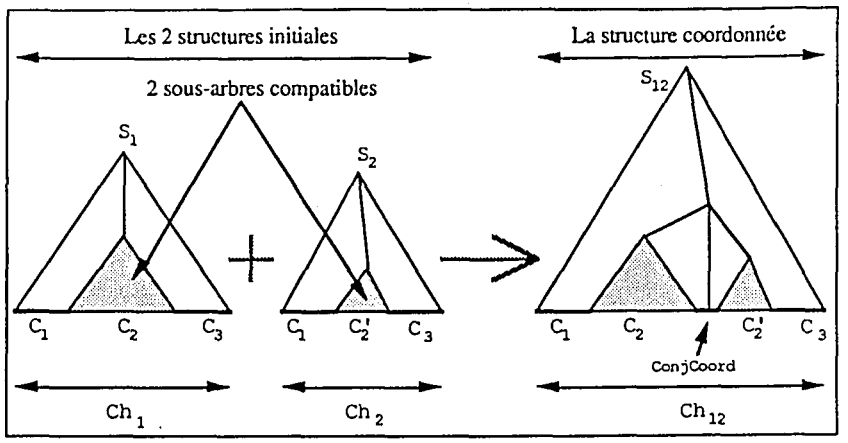


Figure 36: coordination dans le cas général

Cette description n'est pas complète puisque la coordination peut se faire sur plus de 2 chaînes, elle peut porter sur des chaînes ayant déjà subi une coordination et il existe d'autres formes de coordinations plus dépendantes de la structure.

Exemple

Pour *outil de recherche*, nous pouvons donner de façon exhaustive la liste des sous-structures qui sont candidates à une coordination, en prenant tous les sous-arbres de la structure considérée. *Outil de recherche* est analysé avec la grammaire suivante du groupe nominal:

```
<GroupeNominal> ->  Nom <ModifieurDuNom>;
<GroupeNominal> ->  Nom;
<ModifieurDuNom> -> Prép <GroupeNominal>;
<ModifieurDuNom> -> Nom;
```

L'arbre de cette analyse est reporté dans la figure 38.

Nous notons entre parenthèses les chaînes pouvant être coordonnées: *((outil) ((de) (recherche)))*, elles sont entourées sur la figure 37.

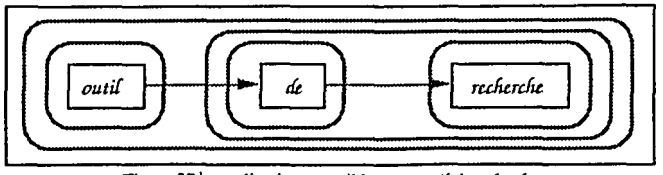


Figure 37: coordinations possibles sur *outil de recherche*

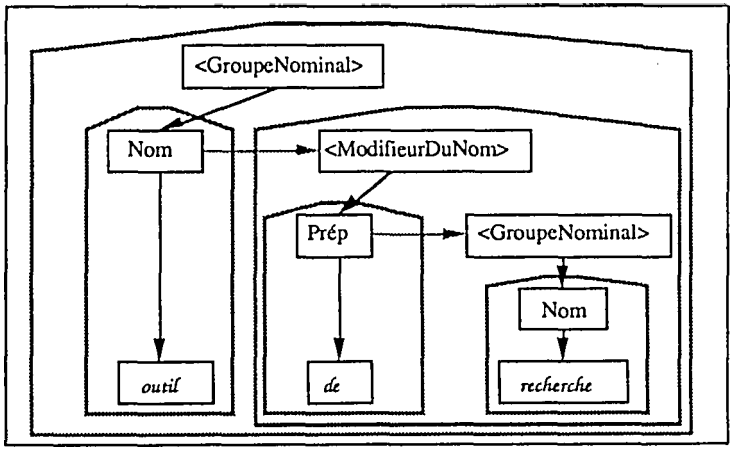


Figure 38: coordinations dans le cas d'un nom composé en N de N
sous-arbres candidats à une coordination

Les cinq possibilités de coordination sont donc:

- *recherche* dans *outil de recherche et développement* (1),
- *de recherche* dans *outil de recherche et de développement* (2),
- *outil de recherche* dans *outil de recherche et procédé de développement* (3),
- *outil* dans *outil ou procédé de recherche* (4), la figure 39 donne l'arbre d'analyse de cette coordination,
- la coordination sur la seule préposition est plus difficile à obtenir dans le cas général. Certaines prépositions qui admettent un antonyme telles que *avec* et *sans* ou *pour* et *contre*, se prêtent bien à ce type de coordination. Avec *de*, on peut trouver *outil de ou pour la recherche* (5), qui nécessite l'ajout d'un déterminant. Parmi les noms composés figés relevés dans notre corpus, la coordination *intervention sur ou hors site* est beaucoup plus plausible.

La structure (1) est ambiguë car *recherche et développement* est un nom composé. On peut donc donner deux analyses de (1): N_{11} de (N_{12} et N_2) ou N_{11} de N_{comp}. En fonction du contexte, l'une ou l'autre des deux analyses sera privilégiée.

Comme coordination, (1) est moins acceptable que (2). En particulier, si les formes de la préposition *de* sont différentes dans les deux noms composés coordonnés, ou si *de* est sous la forme *d'*, la forme (1) est peu acceptable: *?injection d'air ou oxygène* ou *?injection d'air ou gaz carbonique.*

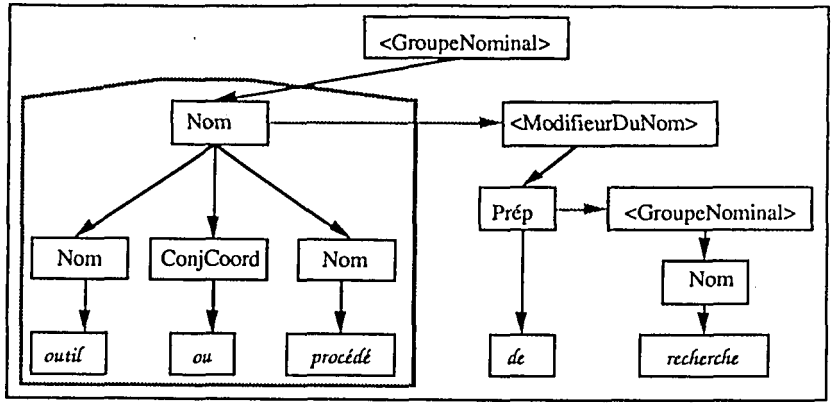


Figure 39: analyse syntaxique de *outil ou procédé de recherche*

L'acceptabilité de la coordination, tout comme l'insertion des modifieurs est liée, d'une part au degré de figement des deux formes coordonnées, d'autre part à un compromis entre l'intelligibilité de la structure produite (plus les termes sont répétés plus la compréhension est aisée) et la légèreté du style (plus les termes sont factorisés, plus la forme obtenue est économique).
La chaîne *outil de* ne peut être coordonnée, car ce n'est pas une structure syntaxique complète au sens de la grammaire précédente: **outil de ou procédé de recherche* (6). Dans la figure 40, nous montrons pourquoi *outil de* ne peut être analysé comme un syntagme.

Nous avons étudié la coordination au niveau du seul groupe nominal, d'autres coordinations contenant *outil de recherche* peuvent être construites en y incluant une partie de la phrase dans laquelle il est inséré. La structure coordonnée ainsi constituée n'est plus nécessairement de type nominal:

- inclusion d'un déterminant:
  un outil ou un procédé de recherche,
- inclusion d'une préposition et d'un déterminant:
  à un outil ou à un procédé de recherche,
- inclusion d'un verbe, d'une préposition et d'un déterminant:
  ajouter à un outil ou appliquer à un procédé de recherche.

Plus la séquence coordonnée est longue, moins la forme figée *outil de recherche* est visible.

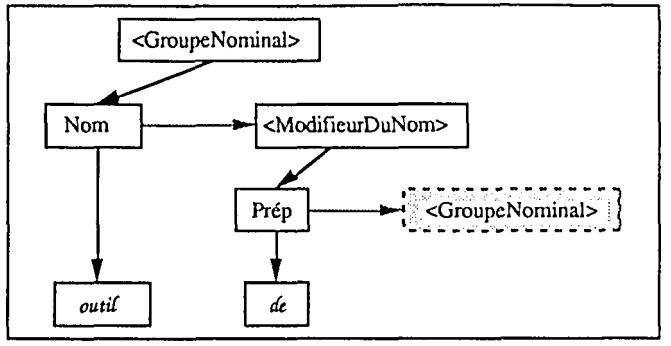


Figure 40: échec de l'analyse de *outil de* comme syntagme

Cas de constituants coordonnés de catégories différentes

Lorsque les parties communes sont compatibles, les arbres conjoints peuvent correspondre à des structures syntaxiques différentes:

carte bancaire ou de téléphone.

Bancaire et *de téléphone* sont deux modifieurs du nom, l'un est adjectival, l'autre est prépositionnel.

Autres formes de coordination

Nous n'avons pas la prétention, dans cet exposé, de présenter toutes les formes de coordination, mais uniquement les cas les plus fréquents. Il s'agit de mettre en évidence, sur ces exemples, comment l'utilisation de métarègles permet de décrire les transformations subies par les formes figées dans des coordinations.
Parmi les autres formes de coordination citons celles-ci:

- les compléments absolus [Grevisse 86] tels que: *les centrales électriques ont une durée de vie de 15 ans, sauf les hydroélectriques,*
- les coordinations implicites: les formes sans conjonction de coordination telles que *techniques de décomposition-coordination* (tiret), *postes HT/MT* (barre oblique), *maintenance développement des logiciels* (juxtaposition),
- les coordinations sur des noms composés soudés tels que: *fours mono et multizones* ou *fours mono- et multizones,*
- les coordinations avec pronominalisation telles que: *les conditions météorologiques et leur évolution prévue.*

Nous retrouvons, dans les cas des noms composés, toutes les formes possibles de coordinations pouvant être définies sur toute sous-structure syntaxique de la phrase. Nous sommes ici à la limite entre une étude sur les coordinations de noms composés et celles sur les groupes nominaux libres.

Coordination entre une forme libre et une forme figée

Il semble que les seuls candidats à être coordonnés avec un nom composé soient d'autres noms composés. Cette propriété limite donc la recherche des coordinations possibles à l'intérieur du lexique des noms composés, elle constitue un bon critère de figement.

**Max prend la carte routière et la verte.*

**Max boit du vin rouge ou frais.*

Coordination et compatibilité sémantique

[Pinkster 90] montre que l'égalité de catégorie syntaxique entre éléments coordonnés n'est ni une condition nécessaire, ni une condition suffisante pour la coordination. Il ne s'agit pas d'une condition suffisante puisque **l'homme et la clé ouvrent la porte*. Cette condition n'est pas nécessaire, puisque, comme nous l'avons vu ci-dessus, il est possible de coordonner des constituants de catégories différentes. Il y a donc combinaison d'une similitude syntaxique et d'une compatibilité sémantique. Ainsi, pour réaliser une coordination de noms composés, il est nécessaire de trouver deux noms composés dont les constituants communs correspondent aux mêmes entrées. *Carte routière* et *carte bancaire* ne pourront pas être coordonnés, car les entrées de *carte* dans ces deux mots sont différentes.

Inversement, la possibilité de coordination entre deux noms composés ayant des éléments communs est un critère pour que les constituants factorisés correspondent aux mêmes entrées. Les exemples ci-dessous permettent de distinguer trois entrées du mot *carte:*

carte routière ou d'état major

carte bancaire ou de téléphone

carte de tarot ou de bridge

Si nous nous plaçons dans l'hypothèse suivante: l'acceptabilité de la coordination entre noms composés est associative. Nous avons la règle suivante:

si a b c et a b d donnent a b c *et* d
et si a b d et a b e donnent a b d *et* e,
alors a b c et a b e donneront a b c *et* e.

Nous supposons également qu'elle est indépendante des conjonctions de coordination employées. Nous en déduisons que l'acceptabilité de la coordination se vérifie à l'aide d'une classification des constituants par entrées compatibles. Ainsi *carte routière* et *carte d'état major* correspondent à la même entrée de *carte*, et il peut donc y avoir factorisation de cette partie commune.

Certains noms composés, tels que *pomme de terre,* peuvent difficilement entrer dans une coordination, parce qu'ils correspondent à des structures très figées pour lesquelles il est difficile de trouver une structure voisine avec une partie commune dont la signification est la même dans les deux noms composés. Nous associons à *pomme* un numéro d'entrée qui ne se retrouve que dans peu de noms composés tels que *pomme frite* et qui autorise la construction de *pomme de terre ou frite.* De nombreux autres noms composés comportant le mot *pomme* en tête, ne peuvent être coordonnés avec *pomme de terre: pomme d'Adam, pomme de discorde, pomme d'arrosoir...*
Nous n'avons pas mis en place une telle classification, et nous nous sommes contenté, pour certaines structures, de savoir si le nom composé pouvait entrer dans une coordination avec des mots de même structure.

Transformation N° 4 de N Adj

Capacité nominale et *charge nominale* donnent *charge ou capacité nominale.*

Industrie chimique et *industrie d'équipement* peuvent se combiner pour donner *industrie chimique ou d'équipement.*

Gaz carbonique est très figé; il est difficile de l'insérer dans une coordination.

Transformation N° 3 de N de N

Unité de recherche se combine avec d'autres noms composés en N de N pour donner *unité de recherche et de production,* la préposition commune *de* n'étant pas nécessairement répétée comme pour *outil de modélisation et simulation.*
Comme nous l'avons vu ci-dessus, la coordination d'un N de N peut se faire avec un N Adj, avec un N de Dét N ou tout autre nom composé ayant une partie commune: *réseau informatique ou de domotique.*

Tableaux d'acceptabilité des transformations

Nous reprenons, dans les tableaux des figures 19, 20 et 21, les transformations que nous avons dégagées cette section, pour les réunir par structures communes de noms composés.

	Modifieur du premier N	Modifieur du deuxième N	Coordination	Lien avec une forme en N Adj	Acceptabilité de N "de" Dét N
courant d'air	-	+	+	-	-
station de travail	-	-	-	-	-
four de cuisson	+	+	+	-	+
vapeur d'eau	+	-	+	-	+

Figure 41: transformations de la structure N de N

	Modifieur du premier N	Modifieur du deuxième N	Coordination	Forme verbale V "à" N	Prédicativité : N "est à" N	Lien avec une forme en N Adj	Acceptabilité de N "à" Dét N
pompe à chaleur	+	-	-	-	-	-	-
torche à plasma	+	+	-	-	-	-	-
relevé à distance	+	-	-	+	-	-	-

Figure 42: transformations de la structure N à N

	N "est" Adj	N Vsup Nominalisation	N Adv Adj	N Adj1 Conjc Adj2	Lien avec N "de" N	N "non" Adj	Modifieur du N
air ambiant	-	-	-	-	-	-	+
écoulement turbulent	+	+	+	+	-	+	+
pouvoirs publics	-	-	-	+	-	-	-
état gazeux	-	-	+	+	-	+	-

Figure 43: transformations de la structure N Adj

Le tableau concernant les formes N Adj (figure 21) convient également pour les formes où l'adjectif est remplacé par une forme verbale comme: N V_{pp} ou N V_{ant}. Celui sur les formes N de N (figure 19) convient également pour les formes en N de Dét N à condition de remplacer la transformation (5): "acceptabilité de la forme N de Dét N" par la transformation (5'): "acceptabilité de N de N". Nous nous limitons au cas de la coordination avec des noms composés de structure identique.

Pour les modifications du premier ou du deuxième nom, nous ne considérons que les deux cas simples suivants:

- modification du premier nom: acceptabilité d'une forme en N_1 Adj de N_2, avec Adj modifieur postposé de N_1,
- modification du deuxième nom: acceptabilité d'une forme en N_1 de N_2 Adj, avec Adj modifieur postposé de N_2.

L'acceptabilité de la modification du premier nom augmente si on autorise la composition par [connexe] produisant la structure N_1 de N_2 Adj, avec Adj modifieur de N_1 ou les modifications du type *courant d'air frais* où *frais* peut aussi bien se rapporter à *courant* que *air*.

Pour les coordination, nous ne mesurons que l'acceptabilité de la coordination avec un nom composé de même structure. Pour la structure N Adj, nous ne considérons que les coordinations N_1 ConjCoord N_2 Adj ou N Adj_1 ConjCoord Adj_2, comme pour *écoulement fluide ou gazeux*.

b Expression des transformations au moyen de métarègles

Dans ce paragraphe, nous proposons des outils formels pour rendre compte des transformations subies par les formes figées. Nous reprenons les règles définies dans la première partie, pour l'enrichir d'un mécanisme de métarègles. Elles permettent de transformer une règle en une nouvelle règle qui représente le nom composé sous sa forme modifié. Nous examinons les possibilités de composer entre elles les métarègles, en tenant compte des contraintes linguistiques.

La notion de métarègle

La transformation d'une structure de nom composé revient à créer une nouvelle règle à partir de la règle initiale. Il y a une certaine régularité dans la création de ces règles, et, pour l'illustrer, nous allons prendre quelques exemples de transformations sur des structures en les regroupant sous les trois catégories que nous avons définies au II.1.a. Afin d'observer la productivité de ces mécanismes, nous devons également conserver la classification des noms composés par grandes familles: N de N, N à N, N Adj...

Nous appelons **métarègle**, toute fonction de l'ensemble des règles admissibles pour la grammaire fournie sur lui-même. Ces métarègles ressemblent aux transformations de la grammaire transformationnelle telles qu'elles sont définies dans [Chomsky 57]. Nous conservons le terme de métarègle car il s'agit de mécanismes algébriques parallèles aux transformations qui sont des concepts linguistiques. [Sabah 88] rappelle la différence entre des métarègles qui permettent de produire de nouvelles règles à partir de règles existantes, et les schémas de règles qui sont des règles où certains termes sont des variables pouvant être remplacées par des symboles de la grammaire.

Soit $\mathbf{R}$ l'ensemble des règles admissibles pour la syntaxe d'énonciation des règles de noms composés. Soit $\mathbf{R}_{\emptyset}$ un sous-ensemble de $\mathbf{R}$, une métarègle M est une relation de $\mathbf{R}_{\emptyset}$ dans $\mathbf{R}$, ainsi notée:

M: $\mathbf{R}_{\emptyset}$ ------> $\mathbf{R}$
r ------> M(r)

Il n'y a pas correspondance biunivoque entre transformations et métarègles, pour les deux raisons suivantes.

- Il est possible de construire une métarègle ne correspondant à aucun mécanisme linguistique connu ou à une particularité si exceptionnelle qu'il est préférable de la recenser comme une nouvelle forme figée plutôt que comme transformation d'une autre structure. Ainsi, le lien entre une forme N_1 de N_2 et une forme en N_2 de N_1 (exemple *consigne de température* et *température de consigne*) peut être facilement représenté par une métarègle, mais n'a pas d'intérêt linguistique.

- Inversement, toute transformation peut être considérée comme une application de l'ensemble des règles de la grammaire dans lui-même, donc, à toute transformation, on peut associer une métarègle, en définissant pour chaque règle son image. Mais, une métarègle ne présente d'intérêt que si on peut la formaliser comme une transformation régulière qui, à toutes les règles d'une famille de structures, associe, par un mécanisme stable ou paramétrable, la règle transformée. Notre modèle a été réalisé pour répondre à cette exigence de régularité et pour permettre, le cas échéant, de paramétrer les métarègles.

Forme d'une métarègle

D'après l'étude linguistique qui précède, nous avons classé les possibilités de transformations sur les noms composés par structures syntaxiques semblables. Donc, une métarègle donnée opère sur des règles ayant le même arbre syntaxique, à quelques différences près sur les valeurs des variables. Plus précisément, ces règles doivent toutes avoir en commun une partie d'arbre syntaxique à partir de leur racine.

La représentation d'une métarègle est donc formée de deux arbres syntaxiques:

- **l'arbre de gauche** qui devra s'unifier avec l'arbre syntaxique de la règle initiale,
- **l'arbre de droite** qui sera celui de la nouvelle règle produite par la métarègle après unification de ses variables avec l'arbre de gauche.

Nous notons donc la métarègles de la façon suivante: a_g -> a_d; avec a_g l'arbre de gauche et a_d l'arbre de droite.

La représentation des métarègles n'est pas symétrique. Il est possible de définir un mécanisme d'inversion des métarègles, à condition que la métarègle considérée permette de le faire (en particulier, si deux règles distinctes correspondent à des images de règles distinctes par cette métarègle). La plupart des métarègles sont aisément réversibles. Le seul point délicat est constitué par les **métarègles paramétrables**. Elles nécessitent, pour être inversées, que les règles transformées aient connaissance du paramètre permettant de revenir à la règle d'origine. Par exemple la nominalisation de *équation énergétique* en *équation de l'énergie* nécessite que le paramètre `(de l'énergie)` soit donné dans la règle `<équation énergétique>`. Pour la transformation inverse, il faut fournir à la règle `<équation de l'énergie>` le paramètre `(énergétique)`. Un automatisme est envisageable pour enrichir le lexique des noms composés avec les paramètres des métarègles réciproques.

Comme le montre [M. Gross 75], un mécanisme transformationnel ainsi défini est trop puissant. C'est pour cette raison que nous le relions aux connaissances linguistiques qui apparaissent dans les tableaux donnés à la fin du paragraphe précédent. L'application des métarègles sur une structure n'est pas systématique. Il faut faire un inventaire, pour chaque nom composé, des transformations acceptées. Seul un linguiste peut le faire. Cette opération n'est pas automatisable.

Bien que, formellement, ces métarègles opèrent sur des arbres syntaxiques, elles sont décrites sous une forme plane comme nous le verrons dans la suite de ce paragraphe. En dehors des cas où l'utilisation des arbres permettra de mieux comprendre les mécanismes, nous présentons les termes des métarègles sous forme de chaînes, pour rester près de l'observation linguistique. Les métarègles sont un outil au service de la description de la langue. Cet outil ne doit, en aucun cas, précéder l'observation.

Les compositions de métarègle

Les métarègles sont ainsi définies, qu'il est possible d'appliquer une métarègle sur une règle qui est elle-même le résultat d'une transformation. Par exemple la nominalisation de l'adjectif dans une règle `N Adj` produit une séquence `N de N` sur laquelle nous pouvons ensuite appliquer toutes les transformations relatives à cette structure. Ceci nous permet évidemment de retrouver la structure initiale en `N Adj` par nominalisation de l'adjectif, si la grammaire est correcte.
Mais une telle récursivité doit être utilisée prudemment, car,

- si la nouvelle structure produite est un nom composé de la langue, les transformations qui le concernent seront déjà recensées. L'intérêt de la métarègle est de représenter la synonymie, elle n'a pas à être composée par les métarègles de la nouvelle structure.
- Si la nouvelle structure comporte des paradigmes (coordination, pronominalisation...) et / ou des élisions (référence...), lui appliquer des métarègles demande des précautions car on opère sur une nouvelle structure de base.

A la fin de ce paragraphe, nous examinons la combinaison des transformations sur une même structure, car elle permet, à partir d'un nombre restreint de transformations de base, de produire une palette plus riche de transformations hybrides obtenues par combinaison. Aux paragraphes II.1.c et d, nous étudions les croisements des transformations sur une structure supérieure avec celles définies sur les formes de base.

Traitement informatique

Le traitement informatique des métarègles et des élisions est présenté dans la partie II.2.c. L'annexe III.2 donne un exemple de métarègles, dans le formalisme présenté ci-dessous.

Formalisme des métarègles (catégorie 1 et 3)

Un cas particulier de métarègle de catégorie 1: l'ajout d'un adverbe

eau potable -> eau peu potable

Prenons la transformation N° 1 de `N Adj` qui, à un nom composé en `N Adj`, associe la structure `N Adv Adj`:

eau potable -> eau peu potable

```
{N, !1, !1} eau{N, f, v, n} potable{A, a1, a1, n}
-> {N, !1, !1} eau{N, f, v, n} peu{Av, v, v, n} potable{A, a1, a1, n};
```

Le mécanisme de création de la nouvelle règle se fait en deux temps:

- insertion d'un adverbe dont le genre et le nombre sont vides et dont la forme est normale,
- rétablissement éventuel des indices d'accord dans le nom composé. Sauf spécification contraire, on suppose que les accords sont conservés dans le nom composé et qu'il faut rétablir leurs indices si les places des constituants ont changé lors de la transformation. Dans le cas où cet automatisme n'est approprié, les nouveaux accords doivent être décrits explicitement comme nous le verrons plus loin.

Enoncé de la métarègle dans ce cas particulier

La description que nous venons de donner pour la transformation de *eau potable* s'applique à tout nom composé ayant cette structure et acceptant cette transformation. Nous pouvons la noter:

```
N Adj(3) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
-> {N, !1, !1} !x{N, !a, !b, !c} <>{Av, v, v, v} !y{A, !d, !e, !f};
```

Avec les précisions suivantes:

- en tête de la métarègle se trouvent le nom du tableau où elle est décrite et le numéro de la colonne correspondante,
- `!x, !y, !a, !b...` sont des variables de la métarègle, elles permettent de transmettre des valeurs de l'arbre de gauche vers l'arbre de droite. Après unification de l'arbre de gauche avec la règle de *eau potable* nous aurons le système suivant:

  `{!x = eau, !y = potable, !a = f, !b = v...},`
- la notation `<>` pour l'adverbe inséré, signifie que tout adverbe pourra être accepté (paradigme étendu à toute une catégorie lexicale).

Extension à des structures comportant des listes

Cette métarègle peut s'étendre à des structures qui ne sont pas des N Adj, mais dont une partie de l'arbre est commune. La formule *(point + temps) (fort + faible)*, représentant les quatre noms composés *point fort, point faible, temps fort* et *temps faible*, correspond à la structure de la figure 22. Elle admet la transformation: *(point + temps)* Adv *(fort + faible)*.
Eau potable qui admet la même transformation est associée à l'arbre de la figure 23.

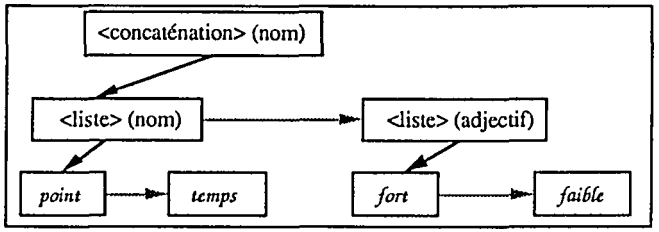


Figure 22: arbre d'analyse de la règle décrivant l'entrée lexicale multiple *(point + temps) (fort + faible)*

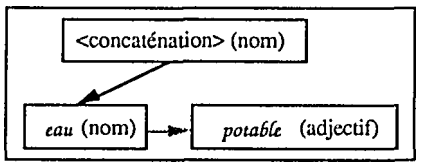


Figure 23: arbre d'analyse de la règle décrivant l'entrée lexicale simple *eau potable*

Jusqu'au niveau 2, ces deux arbres sont superposables et les noeuds ont les mêmes catégories lexicales. Il est possible de représenter le mécanisme de la création de la nouvelle règle par une seule transformation, comme le montre la figure 24. Arbre 1 et Arbre 2 représentent des arbres syntaxiques quelconques, éventuellement réduits à une seule feuille pour *eau potable*.

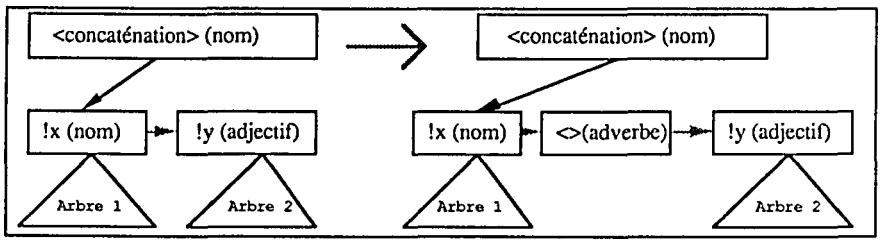


Figure 24: mécanisme de la métarègle d'insertion d'un adverbe dans une structure en N Adj

Nous pouvons étendre une métarègle définie sur la structure plane d'une entrée unique en N Adj, à une structure contenant des listes et définissant plusieurs entrées, telle que $(N_1 + N_2)$ $(Adj_1 + Adj_2)$. Cette facilité offre un confort d'utilisation, car elle permet de noter, de façon condensée, des transformations opérant sur une famille d'entrées lexicales de structures voisines. Mais elle présente des dangers:

- la simultanéité de plusieurs entrées va compliquer l'application de règles d'interprétation puisqu'il faudra désigner la règle associée à chaque entrée,
- les transformations de la catégorie 1 opèrent indépendamment des entrées et s'appliquent de façons identiques quels que soient les mots choisis dans les listes, alors que les transformations de la catégorie 2 qui remplacent certains mots pleins par d'autres mots pleins seront différentes selon les choix effectués dans les listes d'une même règle.

Bien que séduisante, cette possibilité de désigner plusieurs entrées de façon condensée présente plus d'inconvénients que de facilités, et il convient de la réserver à quelques cas techniques particuliers correspondant aux variations d'une même entrée comme *appartement de type (T1 + T2 + T3 +...)* ou *(éducation + Education) (nationale + Nationale)*, plutôt qu'au regroupement d'entrées distinctes comme pour l'exemple précédent.

Modification implicite des indices des variables d'accord ou de substitution dans les règles.

Dans les cas d'accord ou de substitution, les variables de l'arbre de droite peuvent être différentes de celles de gauche si les indices des termes auxquels elles font référence ont changé de rang de place dans la chaîne. Considérons par exemple la transformation qui permet d'ajouter un adverbe en tête d'une forme en Adj N:

haute tension -> très haute tension.

La métarègle s'écrit ainsi:

```
Adj N (3) {!a, !b, !c} !x{A, !d, !e, !f} !y{N, !g, !h, !i}
-> {!a, !b, !c} <>{Av, v, v, v} !x{A, !d, !e, !f} !y{N, !g, !h, !i};
```

Si on l'applique sur la règle de *haute tension:*

```
{N, !2, !2} haut{A, v, v, n} tension{N, a1 f, a1, n}
```

l'unification des variables, sans prise en compte du rétablissement des indices, fournit le résultat erroné suivant:

```
{N, !2, !2} <>{Av, v, v, v} haut{A, v, v, n} tension{N, a1 f, a1, n}
```

Le système informatique gère automatiquement cette correction, et incrémente d'une unité l'indice de tous les mots de la phrase. Le résultat correct, que nous obtenons après transformation, est:

```
{N, !3, !3} <>{Av, v, v, v} haut{A, v, v, n} tension{N, a2 f, a2, n}
```

Un cas particulier de métarègle de catégorie 3: la coordination

Nous considérons le cas d'une coordination sur le nom composé *flux thermique* par factorisation du nom initial. En représentation simplifiée, les deux formes coordonnées se notent:

(1) *flux thermique* -> *flux thermique* `ConjCoord Modif`

(2) *flux thermique* -> *flux* `Modif ConjCoord` *thermique*

où `ConjCoord` désigne toute conjonction de coordination, et `Modif` un modifieur du nom (adjectival, prépositionnel...).

Comme nous l'avons vu au II.1.a les formes (1) et (2) sont incomplètes car elles ne désignent qu'une partie des coordinations possibles sur ce nom composé, en particulier parce qu'elles ne décrivent pas les coordinations par factorisation de l'adjectif terminal comme *flux ou courant thermique*. Elles sont également trop larges, puisque n'importe quel modifieur ne pourra pas être coordonné avec l'adjectif, les coordinations se faisant essentiellement entre formes figées correspondant à la même entrée du nom factorisé:

? *flux radiatif ou ferroviaire*

flux routier ou ferroviaire

Les cas les plus fréquents de coordination par factorisation du mot initial que nous rencontrons se limitent aux associations des noms composés suivants:

flux thermique, flux radiatif, flux électromagnétique, flux électrostatique...

ou tout autre nom composé du lexique ayant une structure du type `flux Adj` ou `flux Prép N` et correspondant à la même entrée du mot *flux*.

Les transformations (1) et (2) précédentes s'écrivent ainsi dans le formalisme des métarègles:

```
(1) N_Adj(4) (N, !1, !1) !x(N, !a, !b, !c) !y(A, !d, !e, !f)
    -> (N, !1, !1) !x(N, !a, !b, !c)
                   !y(A, !d, !e, !f) <>(Cc, v, v, v) <>(A, !d, !e, v);

(2) N_Adj(4) (N, !1, !1) !x(N, !a, !b, !c) !y(A, !d, !e, !f)
    -> (N, !1, !1) !x(N, !a, !b, !c)
                   <>(A, !d, !e, v) <>(Cc, v, v, v) !y(A, v, p, !f);
```

Dans un souci d'analyse ou d'indexation des textes, il peut être pénalisant d'accepter des structures trop larges. Une telle imprécision conduit à des erreurs d'interprétation, mais permet de reconnaître des structures non explicitement prévues. Dans le cas de la production, il convient de cerner au plus près, les seules structures admissibles. Le travail du linguiste sera d'affiner les métarègles et d'enrichir la description des transformations admises par les noms composés.

Métarègles paramétrables (catégorie 2)

Une catégorie particulière a été prévue pour ces métarègles, car elle doivent recevoir de la part de la règle sur laquelle elles s'appliquent des informations qui leur permettent de produire correctement la nouvelle règle. Nous avons vu, au II.1.a,. que le nom associé à un adjectif par une règle de nominalisation n'est pas nécessairement unique. Il faut, dans ce cas, le passer en paramètre à la métarègle de nominalisation.

Un cas particulier de métarègle de catégorie 2: la nominalisation

résidu pétrolier -> résidu de pétrole

```
(N, !1, !1) résidu(N, m, v, n) pétroli(A, al, al, n)
-> (N, !1, !1) résidu(N, m, v, n) DE(P, v, v, n) pétrole(N, m, s, n);
```

Les informations dont doit disposer la métarègle sont de deux sortes:

- les informations communes à toutes les transformées de toutes les règles de cette structure, telles que le terme: `DE(P, v, v, n)`, pour l'exemple ci-dessus. Elles sont de même nature que celles qui étaient fournies dans les métarègles de catégorie 1, et peuvent figurer directement dans la métarègle.

- Les informations spécifiques à l'exemple considéré. Elles sont en caractères gras soulignés dans la règle: **`pétrole(N, m, s, n)`**. Il s'agit ici du terme formé par le nom obtenu à partir de l'adjectif, sa catégorie lexicale, son genre, son nombre et sa forme: c'est un paramètre de la métarègle.

Lors de la compilation, les termes paramètres sont stockés, associés à chaque règle. Il suffit que la métarègle indique dans quel ordre elle souhaite les utiliser en les notant $1 pour le premier, $2 pour le deuxième...

Ainsi la transformation précédente est équivalente au système:

```
(N, !1, !1) résidu(N, m, v, n) pétroli(A, al, al, n)
-> (N, !1, !1) résidu(N, m, v, n) DE(P, v, v, n) $1;
```

avec:

```
$1 = pétrole(N, m, s, n)
```

les dernières informations sont à fournir dans la règle de *résidu pétrolier* qui s'écrit donc:

```
<résidu pétrolier>
-> (N, !1, !1) résidu(N, m, v, n) pétroli(A, al, al, n)
   [() () (N Adj -1 -2 -3 +4 +5(pétrole(N, m, s, n1) -6 +7)];
```

Nous devons inclure ce paramètre dans la métarègle qui devient donc:

```
N Adj (5) (N, !1, !1) !x(N, !a, !b, !c) !y(A, v, v, v)
-> (N, !1, !1) !x(N, !a, !b, !c) DE(P, v, v, v) $1;
```

Les élisions (catégorie 3)

Nous ne traitons pas les transformations qui suppriment des mots dans la chaîne au moyen de métarègles. Nous avons mis en place une notation simplifiée, pour les deux raisons suivantes.

- D'une part, toutes les structures admettent des élisions et les possibilités sont tellement multiples qu'il serait fastidieux et inefficace de vouloir toutes les donner *a priori*.

- D'autre part, l'élision est un mécanisme simple pour lequel il suffit de préciser les éléments supprimés (ou conservés), sachant que la partie informatique se charge de la modification des indices de substitution ou d'accord qui sont modifiés. L'écriture de métarègles serait un formalisme lourd pour cette application.

Nous faisons figurer en fin de règle la liste des élisions qui sont admissibles sur la structure considérée. Par exemple, le nom composé *logement de type (F1 + F2 + F3)* admet trois élisions:

logement type (F1 + F2 + F3): mots d'indices 1, 3 et 4 dans la chaîne,

logement (F1 + F2 + F3): mots d'indices 1 et 4,

type (F1 + F2 + F3): mots d'indices 3 et 4.

L'élision par référence doit être distingué de l'élision par coordination qui implique, d'une part la présence d'un autre nom composé de même structure, et d'autre part le respect de règles de construction de coordination précises. Lorsque l'analyse d'une coordination est possible, il convient de la préférer à l'analyse d'une référence pour éviter une ambiguïté artificielle.

On note les élisions en indiquant la liste des mots conservés:

```
<LISTE logements>
-> (N, !1, !1) logement(N, m, v, n) DE(P, v, v, n)
        type(N, m, s, n) <LISTE types de logements>(N, v, v, v)
        [() (1-3-4, 1-4, 3-4) ()];
```

`(1-3-4, 1-4, 3-4)` indique que les élisions suivantes sont possibles:

- conservation des mots d'indices 1, 3 et 4 : *logement type (F1 + F2 + F3)*
- conservation des mots d'indices 1 et 4 : *logement (F1 + F2 + F3)*
- conservation des mots d'indices 3 et 4 : *type (F1 + F2 + F3)*

Cette notation fait apparaître, au niveau de la règle, les mots dont la présence est facultative. Or, pour la lexicalisation (voir I.3), il est nécessaire de savoir quels sont les mots auxquels on peut rattacher la règle pour qu'elle soit chargée dans tous les cas de transformations possibles. La partie commune aux quatre élisions est formée par la liste *(F1 + F2 + F3)*. Il ne faut pas relier la règle à cette liste, qui est trop longue. Il faut choisir, dans les mots restants, un sous-ensemble minimal de mots auxquels rattacher la règle, en tenant compte des élisions. Ici, la seule possibilité est de lexicaliser la règle sur *logement* et *type*.

Compositions de transformations sur une même structure

Les transformations que nous avons définies sur les noms composés, peuvent, pour certaines d'entre elles, se cumuler sur une même entrée lexicale. Ainsi la présence d'un modifieur adjectival du nom initial peut être simultanée à celle d'un modifieur du nom terminal sur un nom composé en N de N, comme dans *vrai code de bonne conduite*. Cette structure a été obtenue à partir de *code de conduite* par composition des deux transformations `(M1)` et `(M2)` suivantes (dans ce cas, l'ordre des compositions n'a pas d'importance):

`(M1)` $(N_1$ de $N_2)$ -> $((N_1$ $Adj_1)$ de $N_2)$

`(M2)` $(N_1$ de $N_2)$ -> $(N_1$ de $(N_2$ $Adj_1))$.

Pour alléger l'écriture, nous adoptons une représentation des métarègles plus simple que celle proposée au début du II.1.b. Seules sont indiquées les catégories lexicales de chacun des termes. Cette perte d'information, par rapport au formalisme complet, n'a pas d'incidence sur les applications où nous utilisons cette notation réduite. Afin de laisser apparaître la structure d'arbre sous-jacente et les groupements de termes qui correspondent à des syntagmes, nous les mettons entre parenthèses.

Il est nécessaire d'étudier le mécanisme qui permet de définir une nouvelle transformation à partir de plusieurs transformations de base. Nous ne le traitons pas ici, mais dans [Jacquemin 91] nous étudions en détail les automatismes qui permettent de réaliser la composition des transformations.

Acceptabilité d'une composition

Il est évident qu'un nom composé n'acceptera la composition de deux transformations `M1` et `M2` prises dans cet ordre que s'il accepte déjà `M1` et `M2`. Cette condition n'est pas suffisante car il se peut que le composé modifié soit trop surchargé pour qu'on puisse encore identifier la forme figée. Ici, comme dans les exemples précédents, rien ne peut remplacer l'étude par un expert des transformations possibles et une évaluation de leurs acceptabilités. La charge est lourde, en raison du nombre de transformations potentielles produites par composition, et nous n'en avons pas entrepris une étude exhaustive. Ces transformations, bien que possibles, restent assez exceptionnelles. Il convient cependant de les étudier, pour avoir une meilleure couverture de la langue.

Etude d'un exemple

Afin d'étudier la mise en oeuvre de la composition des transformations sur une même structure, nous avons décidé de nous restreindre au cas de la structure N de N (qui est avec N Adj la structure la plus fréquente).

Nous donnons les 6 métarègles représentant les transformations de base:

(M1) (N_1 de N_2) -> ((N_1 Adj_1) de N_2)
(M2) (N_1 de N_2) -> (N_1 de (N_2 Adj_1))
(M3) (N_1 de N_2) -> (N_1 (de N_2 ConjCoord de GN))
(M3') (N_1 de N_2) -> (N_1 de (N_2 ConjCoord GN))
(M4) (N_1 de N_2) -> ((N_1 ConjCoord GN) de N_2))
(M5) (N_1 de N_2) [Adj_1] -> (N_1 Adj_1)
(M6) (N_1 de N_2) -> (N_1 de Dét N_2)

Pour construire toutes les transformations possibles, on utilise un arbre de profondeur **n**, dont chaque noeud a 6 fils (figure 25).

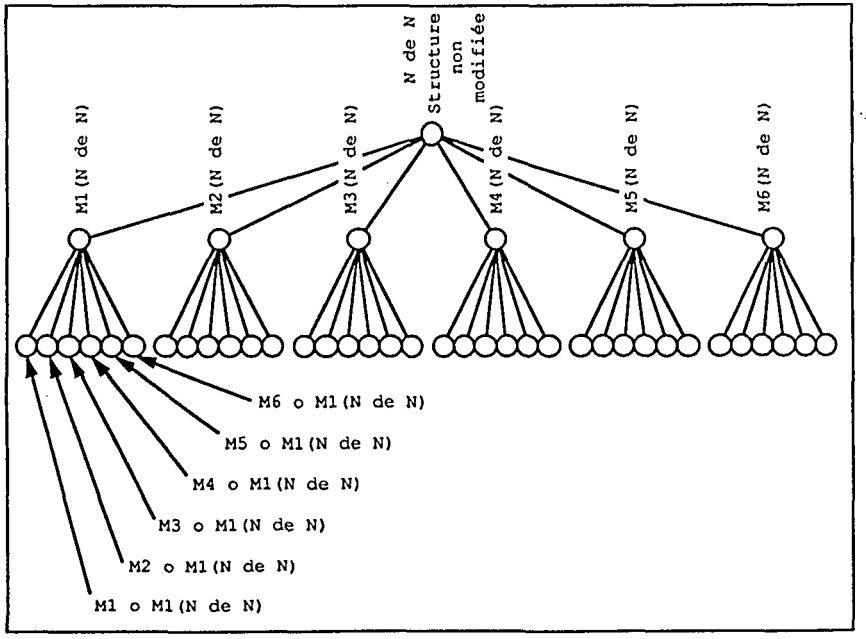


Figure 25: composition exhaustive des métarègles sur une structure en N de N

Les noeuds de profondeur 1 représentent les transformations de base, ceux de profondeur 2 les compositions de deux transformations... La profondeur de l'arbre représente le nombre maximal de transformations que l'on peut composer sans perdre l'intelligibilité de la structure obtenue. Pour un arbre de profondeur **n**, l'arbre a alors:

$1 + 6 + 6^2 + ... + 6^n = ((6^{n+1} - 1) / 5)$ noeuds.

Le deuxième noeud en partant de la gauche, à la profondeur 2, représente la transformation de la structure de base en N de N par M1 puis M2, soit, par exemple, la structure *vrai code de bonne conduite*.

A l'observation des transformations de base et de leurs compositions, on peut dégager trois propriétés:

- **(i)** la transformation M5 ne peut s'effectuer qu'une seule fois sur la structure de base et ne peut être composée avec aucune autre transformation car elle produit une structure en N Adj,
- **(ii)** la composition des transformations est commutative: pour toutes transformations M et M', les composées MoM' et M'oM, si elles existent, produisent les mêmes structures à partir de N de N,
- **(iii)** une transformation ne peut être appliquée qu'une fois sur une structure.

Nous ne pouvons pas donner de démonstration rigoureuse des propriétés (i), (ii) et (iii) puisque les compositions sont construites "à la main", nous indiquons seulement une justification sommaire des deux dernières propriétés (la première est évidente).

Si **(ii)** est vraie, alors l'ordre d'application des transformations sur une structure nominale est indifférent. [Bach 74] montre que les transformations de la phrase sont ordonnées selon un cycle transformationnel et qu'un non respect de cet ordre aboutit à la création de phrases asyntaxiques. Dans le cadre du groupe nominal, l'exigence de séquentialité des transformations peut être supprimée, car, seules les coordinations pourraient être considérées comme n'étant pas permutables avec d'autres transformations. Par exemple:

(M2oM3) (N_1 de N_2) -> (N_1 (de (N_2 Adj_1) ConjCoord de (GN Adj_2)))
(M3oM2) (N_1 de N_2) -> (N_1 (de (N_2 Adj_1) ConjCoord de GN))

L'application de la coordination après l'ajout d'un modifieur adjectival distribue le modifieur sur les deux termes coordonnés, alors que la composition symétrique ne met de modifieur adjectival que sur le nom du composé d'origine. Ce qui distingue ces deux structures, c'est la transformation du nom qui n'appartient pas au composé, et cela est du ressort des transformations acceptées par l'entrée lexicale coordonnée: N_1 de GN. Nous donnerons donc comme image unique de la composition de M2 et de M3, quel que soit l'ordre retenu (M2oM3 ou M3oM2), la structure: (N_1 (de (N_2 Adj_1) ConjCoord de GN)).

Pour **(iii)**, la composition étant commutative, appliquer plusieurs fois une même transformation sur une structure, revient à l'appliquer successivement plus d'une fois sur la structure de base. On

aboutit alors à une structure de coordination implicite. Or la coordination est déjà une transformation de base; la produire par des compositions de transformations avec elles-mêmes créerait des ambiguïtés inutiles.

En dehors de M5, il reste 5 transformations pouvant être composées entre elles, sachant que l'ordre des compositions est indifférent et que chaque transformation ne peut s'appliquer plus d'une fois sur une structure. Le nombre de compositions de transformations est donc égal au nombre de sous-ensembles d'un ensemble à 5 éléments, soit $2^5 = 32$, soit réellement 31 transformations si l'on excepte la **transformation identique** qui ne modifie pas la structure. En ajoutant M5 on obtient 32 transformations ainsi réparties:

- 6 transformations de base,
- 10 compositions de 2 transformations prises parmi M1, M2, M3, M4 et M6,
- 10 compositions de 3 transformations prises parmi M1, M2, M3, M4 et M6,
- 5 compositions de 4 transformations prises parmi M1, M2, M3, M4 et M6,
- 1 composition des 5 transformations M1, M2, M3, M4 et M6.

Le graphe de la figure 26 permet de les visualiser. Il est obtenu à, partir de l'arbre précédent en regroupant les noeuds qui correspondent à des permutations différentes (par exemple M1oM2oM3 avec M1oM3oM2, M2oM1oM3, M2oM3oM1, M3oM1oM2 et M3oM2oM1), en supprimant les noeuds où une même transformation est présente plus d'une fois (par exemple M1oM1oM3) et ceux qui contiennent la transformation M5 sans être égal à M5 (par exemple M5oM6). La non répétition d'une même transformation limite la profondeur de l'arbre à 5, et les regroupements laissent, comme prévu, 32 noeuds au graphe.

La description de la coordination ne se réduisant pas aux seules transformations M3 et M4, le nombre de compositions de transformations est plus élevé que ce que nous avons calculé. Nous nous sommes placés dans cette hypothèse simplificatrice pour clarifier la présentation.

Nous donnons ci-dessous quelques exemples de compositions de transformations à partir du nom composé de base *baisse de charge:*

(M1) ($(N_1$ $Adj_1)$ de $N_2)$
importante baisse de charge

(M1oM3) ($(N_1$ $Adj_1)$ (de N_2 ConjCoord de GN))
importante baisse de charge et de tension

(M1oM2oM3) ($(N_1$ $Adj_1)$ (de (N_2 $Adj_2)$ ConjCoord de GN))
importante baisse de charge électrique et de tension

(M1oM2oM3oM6) ($(N_1$ $Adj_1)$ (de (N_2 $Adj_2)$ ConjCoord de Dét GN))
importante baisse de la charge électrique et de la tension.

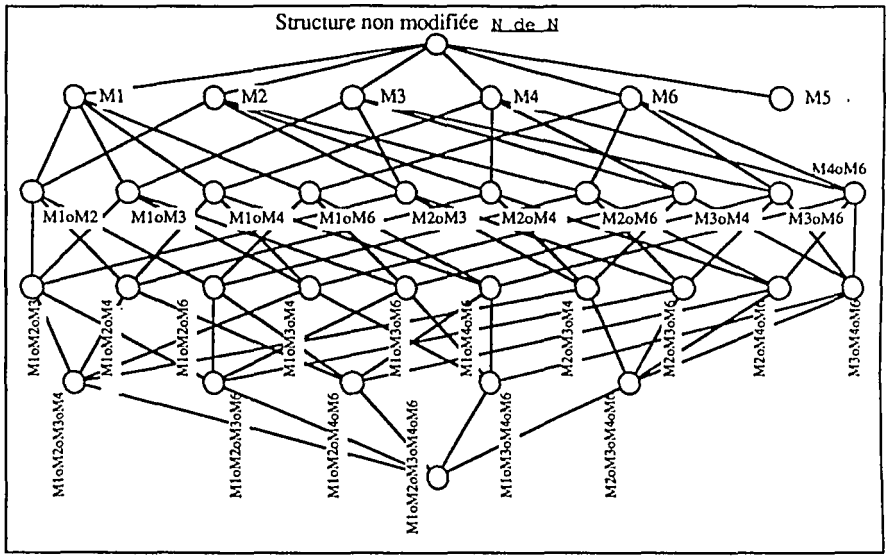


Figure 26: composition des métarègles sur une structure en N de N après réduction

Liste des transformations

Au II.1.a, nous avons donné des tableaux indiquant, pour une entrée donnée, l'acceptabilité des transformations définies pour cette structure. D'après ce que nous avons vu ci-dessus, il faudrait étendre cette description aux compositions de transformations. Donc, pour la structure N de N, le tableau décrivant les acceptabilités devra contenir, non pas 6, mais 32 colonnes, chacune correspondant à une composition différente des transformations de base.

Pour simplifier, nous supposons que la composition de **n** transformations de base sur une même structure est possible si et seulement si chacune de ces **n** transformations est acceptée seule. Ainsi la transformation composée M1oM2oM3 sur *baisse de charge* qui permet de produire ou d'analyser *importante baisse de charge électrique et de tension*, sera acceptable si et seulement si les trois transformations M1, M2 et M3 le sont. Elles permettent respectivement de produire *importante baisse de charge*, *baisse de charge électrique* et *baisse de charge et de tension*. Cette condition est nécessaire, mais non suffisante. La composition des 5 transformations M1, M2, M3, M4 et M6 sur le mot précédent donne *importante baisse et atténuation de la charge électrique et de la tension*, qui, bien que formellement acceptable, est trop éloignée de la forme figée *baisse de charge* pour pouvoir être acceptée comme transformation de celle-ci.

Dans le corpus étudié, nous n'avons pas rencontré plus de deux transformations composées sur une même structure, et, ce cas est exceptionnel. *Capacité de transport minimal spécifiée*, contient un modifieur du premier nom (composé par [connexe]) et un modifieur du deuxième nom.

c Noms composés imbriqués

Certains noms composés, tels que *générateur de (systèmes experts)*, contiennent, à l'intérieur de leur structure, un autre nom composé déjà recensé dans le lexique. Il est intéressant d'étudier spécialement de telles formes, car la connaissance des transformations acceptées par le nom composé de base (ici *système expert*), va pouvoir être reprise pour le nom composé d'ordre supérieur (ici *générateur de systèmes experts*).
Ce paragraphe se propose de classifier ces noms composés imbriqués, et d'observer la façon dont la connaissance des transformations de noms composés de base peut être reprise au niveau supérieur.

Nous appelons **nom composé d'ordre supérieur**, tout nom composé construit à partir d'au moins un autre nom composé déjà attesté dans le lexique. Cette construction peut éventuellement utiliser des mots simples. Nous séparons ces noms composés en deux catégories:

- la juxtaposition de noms composés, chaque constituant reste identifiable,
- l'élaboration d'une forme composite, dans laquelle au moins une des structures initiales est altérée: le recouvrement.

Par opposition, nous nommons **noms composés de base**, les formes figées nominales composées uniquement de noms simples telles que nous les avons étudiées au II.1.a et b.

Noms composés imbriqués par juxtaposition

La juxtaposition peut se faire entre deux noms composés, comme le montre la figure 27 pour *(conduite d'arrivée) d'(eau pressurisée)*. Cette juxtaposition, dans le cadre d'une analyse générative, correspondrait à la structure syntaxique de la figure 28.

La juxtaposition peut également avoir lieu entre un nom composé et un nom simple comme dans *brûleur à (assistance plasma)*. Cette deuxième possibilité doit être distinguée des cas de recouvrement de deux formes composées que nous verrons dans le point suivant tels que *direction de propagation du rayonnement* où les deux formes *direction de propagation* et *propagation du rayonnement* correspondent à des noms composés du lexique.

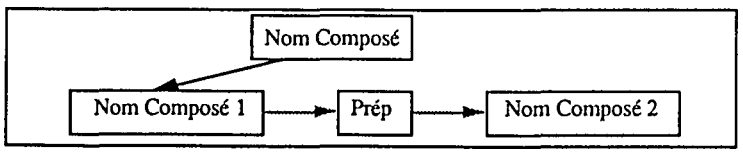


Figure 27: nom composé obtenu par juxtaposition de deux noms composés: N_{comp} Prép N_{comp}

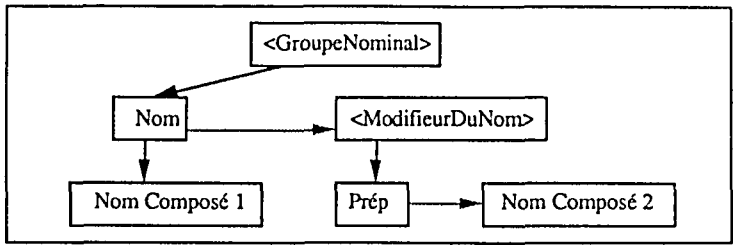


Figure 28: analyse d'un nom composé obtenu par juxtaposition en N_{comp} Prép N_{comp}

Différents modes de juxtaposition

Juxtaposition d'un nom simple et d'un nom composé:

Cette juxtaposition s'articule généralement autour d'une préposition. Dans le corpus étudié, *de* est la plus fréquente comme dans *impédance de (court-circuit)* ou *générateur de (système expert)*. La fréquence de la préposition *de* est à rapprocher du grand nombre de formes de base recensées en N de N. Cette structure se note N de N_{comp} où N_{comp} est le nom composé. Celui-ci appartient indifféremment à l'une des formes de base telle que N Prép N, N Adj, Adj N...

Nous trouvons également des structures formées avec la préposition *à* comme *réacteur à (eau pressurisée)* ou d'autres prépositions plus rares dans *poutre en (béton armé)* ou *calcul par (éléments finis)*. Certaines juxtapositions, de manière analogue à la forme de base en N N, se font sans préposition, comme pour *moniteur (temps réel)* ou *câble (haute tension)*. Ce dernier exemple peut aussi être considéré comme une forme réduite de *câble à (haute tension)*.

Certaines structures de ce type représentent des noms de parties de système, comme *tubes de générateur de vapeur*, d'autres sont des modifieurs qui ont acquis un statut figé puisqu'ils permettent de décrire un objet unique comme *régulateur à actions PID* ou *brûleur à assistance plasma*.

Juxtaposition d'un nom composé et d'un nom simple

Le mot simple ajouté est soit un adjectif seul *(champ électrique) statique* ou *(base de données) relationnelle*, soit une structure en Prép N *(plate forme) d'essai* ou *(modèle numérique) de simulation*. Cette juxtaposition est beaucoup plus rare que la précédente et prête à confusion sur son degré de figement. La structure du nom composé de base est également indifférente.

Il n'est pas simple de décider si de telles formes sont des formes figées ou s'il s'agit de formes libres puisqu'on rencontre toutes les nuances possibles. Dans le cas particulier de la juxtaposition d'un nom composé et d'un nom simple, la spécificité du modifieur est un bon critère pour évaluer le degré de figement. Cette spécificité est d'autant moins grande que le modifieur est plus facilement permutable avec l'autre modifieur qui compose le premier nom composé:

application graphique disponible -> **application disponible graphique*

application graphique interactive -> *application interactive graphique*

Juxtaposition de deux noms composés

Cette juxtaposition est assez rare et généralement peu figée. Toute la gamme des structures de base est candidate à cet assemblage et le lien entre les noms composés se fait au moyen de la préposition *de* dans *(pompe de circulation) de l'(eau de refroidissement)*, de la préposition *à* dans *(réseau de distribution) à (moyenne tension)* qui existe également sous la forme *(réseau de distribution) (moyenne tension)*.

Lorsque le premier nom composé est de la forme N de N ou N Prép N, la juxtaposition avec préposition comme dans (N de N) de (N Modif) est fréquemment ambiguë quant à sa structure, puisqu'on peut également la considérer comme juxtaposition de (N de (N de (N Modif)) *(pompe de (circulation d'(eau de refroidissement))*, les trois formes N de N, N Modif et N de (N Modif) étant attestées. (Voir les autres cas de noms composés imbriqués à la fin de ce paragraphe).

Représentation des noms composés formés par juxtaposition

L'idée la plus naturelle pour représenter de telles formes est de définir le nom composé supérieur à partir des formes de base. Si nous reprenons le nom composé *(conduite d'arrivée) d'(eau pressurisée)*, nous avons noté que *conduite d'arrivée* et *eau pressurisée* correspondent à des entrées lexicales pour le domaine considéré. Nous pouvons donc les faire figurer dans le lexique et représenter la forme de niveau supérieur à l'aide de celles-ci:

- les deux formes de base:

```
<conduite d'arrivée> ->     {N, !1, !1} conduite{N, f, v, n}
                            DE{P, v, v, v} arrivée{N, f, v, n};
<eau pressurisée> ->        {N, !1, !1} eau{N, f, v, n}
                            pressuris{V, a1, a1, n};
```

- le nom composé supérieur défini à partir de celles-ci:

```
<conduite d'arrivée d'eau pressurisée> -> {N, !1, !1}
        <conduite d'arrivée>{N, f, v} DE{P, v, v, v}
        <eau pressurisée>{N, f, v};
```


Cette solution s'étend aux noms composés formés d'une forme de base et d'un mot simple comme *poutre en (béton armé)*. Elle s'applique à toute représentation récursive d'entrées lexicales.

Puisque la forme supérieure fait appel, pour sa reconnaissance, à l'analyse des deux formes de base, la rencontre du nom composé *(conduite d'arrivée) d'(eau pressurisée)* dans un texte produira trois formes reconnues: les deux formes de base *conduite d'arrivée* et *eau pressurisée* et la chaîne complète.
On souhaiterait alors, ne fournir qu'un seul résultat d'analyse (le nom composé supérieur) et ne pas signaler les noms composés de base, mais il est difficile de supprimer cette ambiguïté des résultats d'analyse, car, la reconnaissance d'une forme figée ou d'une forme syntaxique n'implique pas que l'on soit en sa présence. Les exemples (1) et (2) ci-dessous, présentent deux phrases contenant la chaîne *générateur de vapeur*, mais seule la première contient le nom composé qu'elle représente:

(1) *utiliser un générateur de vapeur*

(2) *remplir un générateur de vapeur.*

Transformations des noms composés obtenus par juxtaposition

Au chapitre II.1.a, nous avons étudié les transformations qui opèrent sur les noms composés de base, nous les avons classées en trois catégories. Nous allons reprendre ces transformations pour étudier leur généralisation sur des noms composés formés par la juxtaposition de formes de base.

Catégorie 1 des transformations sur les noms composés obtenus par juxtaposition (tous les mots sont conservés, seules interviennent des insertions ou des substitutions).

Exemple: transformation sur une structure $(N_{11}\ de\ N_{12})\ de\ (N_{21}\ Adj_{22})$

Prenons une structure en $(N_{11}\ de\ N_{12})\ de\ (N_{21}\ Adj_{22})$ telle que *(conduite d'arrivée) d'(eau pressurisée)*, et examinons la modification du premier nom N_{11} par un adjectif. Nous avons introduit la transformation [connexe] au II.1.a pour rejeter un modifieur à l'extérieur d'un nom composé de base afin d'en préserver la connexité. Elle s'applique également aux modifieurs internes d'une juxtaposition de noms composés, et, dans ce cas aussi, elle dépend du degré de figement de la structure.

La modification du N_{11} est représentée par la transformation suivante:

$(N_{11}$ de $N_{12})$ de $(N_{21}$ $Adj_{22})$ -> $(N_{11}$ Adj de $N_{12})$ de $(N_{21}$ $Adj_{22})$
[Modif] *(conduite d'arrivée) d'(eau pressurisée)*
-> *(conduite rigide d'arrivée) d'(eau pressurisée).*

Comme nous l'avions vu au II.1.a pour la transformation N°1 de N de N la transformation [connexe] peut opérer sur la modification précédente en s'appuyant sur le nom composé de base *(conduite d'arrivée)* et donner le résultat suivant:

$(N_{11}$ Adj de $N_{12})$ de $(N_{21}$ $Adj_{22})$ -> $(N_{11}$ de $N_{12})$ Adj de $(N_{21}$ $Adj_{22})$
[connexe] *(conduite rigide d'arrivée) d'(eau pressurisée)*
-> *(conduite d'arrivée) rigide d'(eau pressurisée),*

puis opérer à nouveau en s'appuyant sur le nom composé complet:

$(N_{11}$ de $N_{12})$ Adj de $(N_{21}$ $Adj_{22})$ -> $(N_{11}$ de $N_{12})$ de $(N_{21}$ $Adj_{22})$ Adj
[connexe] *(conduite d'arrivée) rigide d'(eau pressurisée)*
-> *(conduite d'arrivée) d'(eau pressurisée) rigide .*

Nous déduisons que les trois seules positions d'un modifieur postposé à *conduite* dans *conduite d'arrivée d'eau pressurisée* sont données par les trois structures ci-dessous:

(A) $(N_{11}$ Adj de $N_{12})$ de $(N_{21}$ $Adj_{22})$
(B) $(N_{11}$ de $N_{12})$ Adj de $(N_{21}$ $Adj_{22})$
(C) $(N_{11}$ de $N_{12})$ de $(N_{21}$ $Adj_{22})$ Adj.

L'acceptabilité de ces trois formes est liée au degré de figement du nom composé de base *conduite d'arrivée* et à celui du nom composé de niveau supérieur *conduite d'arrivée d'eau pressurisée.*
Pour que **(A)** soit acceptable, *conduite d'arrivée* ne doit pas être trop figée. Pour que **(B)** le soit, *conduite d'arrivée* doit être plus figée pour rejeter le modifieur hors de la chaîne alors que *conduite d'arrivée d'eau pressurisée* doit être une structure assez libre pour accepter une insertion. Plus les formes *conduite d'arrivée* et *conduite d'arrivée d'eau pressurisée* sont figées, plus **(C)** est acceptable puisque le modifieur est rejeté hors du composé.
Le degré de figement de *eau pressurisée* n'intervient pas dans cette modification puisque le modifieur ne peut s'y insérer.

Exemple: modification sur une structure $(N_{11}$ $Adj_{12})$ de $(N_{21}$ Prép $N_{22})$

La forme **(C)** précédente est difficile à comprendre à cause de l'absence de ponctuation. Nous précisons ce point sur le cas d'une autre structure.

(résidu pétrolier) de (distillation sous vide), liquide,

La présence de virgules autour de l'adjectif *liquide* confirme la nécessité de séparer l'adjectif du nom composé *distillation sous vide* et simultanément, de rendre son rattachement à *résidu pétrolier* plus facile. Cette marque écrite, se traduirait par une pause dans l'énonciation telle que:

(résidu pétrolier) de (distillation sous vide) # liquide #

voire une périphrase qui permette de rapprocher le nom et son modifieur:

(résidu pétrolier) de (distillation sous vide) # ce résidu est liquide #

qui est semblable à la forme initiale des modifications adjectivales, qui sont ensuite réduites par la transformation [qui être Z] [M. Gross 86N]:

(résidu pétrolier) de (distillation sous vide), Qu ce résidu est liquide,

Cette forme est appelée apposition par les grammaires traditionnelles. Nous rejoignons, ici, la correspondance établie par [Grevisse 86] entre ce détachement et les relatives non déterminatives.

Une surcharge de modifieurs successifs sur un même mot (les modifieurs figés internes au nom composé et ceux ajoutés dans l'utilisation particulière) finissent par rendre la compréhension complexe. La difficulté de rattachement pertinent des modifieurs à un des mots de la structure explique que leur emploi soit assez rare dans les textes étudiés. On rencontre une proportion importante d'adjectifs placés devant le nom dont l'utilisation ne pose pas d'ambiguïté de rattachement.

grand (four tunnel) de (traitement thermique)

et certains adjectifs dont l'emploi en position postposée est plus naturel lorsque le nom est seul, se rencontreront avant le nom lorsque celui-ci est inséré dans un composé comme nous le verrons plus loin, dans ce paragraphe:

traditionnels (régulateurs à actions PID)

Certains adjectifs, tels que *optimisé*, peuvent difficilement être antéposés:

(brûleur à assistance plasma) optimisé.

Ici, l'accord aide à faire la discrimination du rattachement, mais c'est une chance.

Deux types de transformations sur un nom composé obtenu par juxtaposition (catégorie 1)

Nous reprenons le fonctionnement du mécanisme de métarègle, que nous avons mis en place dans le cas des transformations des noms composés de base (au II.1.a), pour qu'il s'applique également aux formes composites. Nous séparons les transformations sur une forme composite en deux types :

- les **automatismes** qui correspondent à une extension automatique de la transformation d'un constituant de base *haute tension* au nom composé supérieur *ligne (haute tension)* :

ligne (très haute tension), (Adj N -> Adv Adj N)

- les **nouvelles transformations** qui sont définies sur la structure composite *ligne haute tension*: et qui ne correspondent pas à l'extension de la transformation du constituant de base *haute tension* :

ligne aérienne (haute tension) (N N_{comp} -> N Adj N_{comp})

Les automatismes sur un nom composé obtenu par juxtaposition (catégorie 1)

Nous étudions ici les transformations d'un nom composé d'ordre supérieur, définies à partir des transformations des noms composés de base inclus dans sa structure. Pour étudier cette extension, nous nous appuyons sur les mécanismes de réécriture mis en place pour les règles de noms composés. Nous indiquons, comment, la production dynamique des règles au moyen de métarègles, permet d'étendre les modifications des noms composés de base à ceux de niveau supérieur.

Ajout de modifieurs

Les transformations définies sur les noms composés de base N de N s'appliquent à tout nom composé d'ordre supérieur ayant un N de N dans ses constituants tel que (N de N) de (N Adj). Lors de l'analyse de la règle <conduite d'arrivée d'eau pressurisée>, il est fait appel à la réécriture de <conduite d'arrivée> et <eau pressurisée>, et, donc, toutes les métarègles définies sur ces deux structures sont appliquées automatiquement.

La construction dynamique des règles au moyen de métarègles produit toutes les formes composées à partir des transformations de chacun des deux constituants.
D'une part, l'application des métarègles d'ajout de modifieur sur la règle de *conduite d'arrivée* produit six formes, la structure de base et cinq transformations:

deux modifications sur le premier nom

conduite Adj *d'arrivée*
Adj *conduite d'arrivée* (par application de [antéposition])
conduite d'arrivée Adj (par application de [connexe])

et trois modifications sur le deuxième nom

conduite d'arrivée Adj
conduite DE Adj *arrivée* (par application de [antéposition]).

D'autre part, les métarègles produisent cinq formes sur la règle de *eau pressurisée*, la structure de base et quatre transformations:

une insertion d'un adverbe

eau Adv *pressurisée*

et trois modifications sur le premier nom

eau Adj *pressurisée*
Adj *eau pressurisée* (par application de [antéposition])
eau pressurisée Adj (par application de [connexe])

L'application dynamique de ces transformations permet de produire 6 x 5 = 30 formes distinctes candidates. Dans ce cas particulier, il semble que toutes les formes candidates sont attestées.

Préservation de la connexité

Aux transformations ainsi effectuées, peut se composer celle, vue au II.1.a, qui préserve la connexité de l'ensemble et qui repousse hors du composé une modification appliquée sur le premier constituant. Une application de [connexe], sur l'ensemble ne peut se faire que si elle a déjà eu lieu une première fois pour sortir du premier composé un modifieur du premier nom:

[connexe] *conduite en aluminium d'arrivée*
-> *conduite d'arrivée en aluminium*

puis cette transformation s'applique à nouveau sur le nom composé de niveau supérieur pour donner:

[connexe] *(conduite d'arrivée) en aluminium d'(eau pressurisée)*
-> *(conduite d'arrivée) d'(eau pressurisée) en aluminium*

Dans ce cas particulier, cette nouvelle composition ne peut se produire que dans le cas d'un modifieur long comme nous l'avons vu précédemment. Théoriquement, elle peut opérer sur les sept composés de forme:

(conduite d'arrivée) Adj de N_{comp2}

où Adj est un modifieur adjectival et où N_{comp2} est une des sept structures possibles de *eau pressurisée*. Elle s'applique d'autant plus difficilement que N_{comp2} est lui-même surchargé par d'autres transformations. On retrouve, ici encore, les limites d'intelligibilité des expressions de constructions récursives trop complexes.

Antéposition des modifieurs

La deuxième transformation possible sur les modifications de noms composés supérieurs est celle vue au II.1.a qui permet de placer des adjectifs en position antéposée. Par exemple, pour l'adjectif *traditionnel*, on observe une augmentation de l'acceptabilité de l'antéposition lorsque la taille du composé augmente:

* *traditionnels régulateurs*

? *un traditionnel régulateur électronique*

un traditionnel régulateur à actions PID

Nous rappelons ci-dessous le mécanisme de `[antéposition]`:

`[Modif]` *régulateur à actions PID*
-> *régulateur traditionnel à actions PID*

`[antéposition]` *régulateur traditionnel à actions PID* (1)
-> *traditionnel régulateur à actions PID* (2)

Il y a une légère différence de sens entre la phrase (1) où *à actions PID* modifie *régulateur traditionnel* alors que dans (2) *traditionnel* modifie *régulateur à actions PID*. Cette possibilité d'antéposition est indiquée dans [Grevisse 86] pour les formes libres. Elle permet, par exemple, d'éviter de séparer un nom et son complément comme dans *un des plus originaux linguistes de son époque*. Elle sert également à équilibrer les syntagmes nominaux, en plaçant avant le nom, des modifieurs qui le suivraient, s'il était seul: *la peu évangélique loi du profit*.

[Wunderli 87] pose que tout modifieur peut apparaître aussi bien en antéposition qu'en postposition. Il s'agit alors d'attribuer à chaque adjectif une position normale, les infractions à cette norme correspondent à des modifications fines de sens dans *une nouvelle méthode* ou *une méthode nouvelle*, ou plus fortes dans *un homme pauvre* et *un pauvre homme*. Cette dernière possibilité est décrite également dans [Grevisse 86].

Considérations sur l'ordre des modifieurs

De nombreux travaux concernent le problème de la place des modifieurs dans le groupe nominal. L'étude de cet ordonnancement distingue d'une part la **position absolue** du modifieur (avant ou après le nom) et sa **position relative** par rapport à d'autres modifieurs dans la même position absolue. [De Sutter 88] montre que ces deux problèmes sont reliés. Il établit une classification sémantique des modifieurs qui permet de conjecturer leur position relative. Il ne s'agit pas de contraintes aussi strictes que celles sur l'ordre des particules pré-verbales données dans [M. Gross 86N]. [De Sutter 88] explique les cas d'inversions de l'ordre attendu des modifieurs,

comme *vin rouge français* par rapport à *voiture française rouge*, par une variation de sens de *rouge* entre les deux syntagmes. Nous pensons plutôt que la différence de comportement vient de ce que *vin rouge* est une forme figée alors que *voiture rouge* est libre.
[Grevisse 86] rappelle que de nombreux linguistes ont abordé le problème de l'ordre des modifieurs en essayant de dégager des critères universels, mais qu'il se trouve toujours des contre-exemples pour les mettre en défaut. L'exemple précédent montre que des critères sémantiques ne peuvent suffire et que les données lexicales jouent un rôle important.

La transformation `[connexe]` ne s'applique pas systématiquement sur les modifications des noms composés, ainsi le modifieur court *étendue* est inséré à l'intérieur de la structure dans *logique étendue des prédicats*. Il s'agit là d'une tendance à éloigner du nom les modifieurs les plus longs. Ce mécanisme, déjà décrit dans la phrase par Harris dans [Harris 76] p.148, s'observe sur les formes libres avec des relatives ou des participes à compléments comme dans:

les tubes (examinés) (contenant des défauts)
les informations (expérimentales) (acquises au cours des premières expériences)

Dans le cas de modifieurs de structures identiques, comme, par exemple, une liste d'adjectifs, il semble qu'ils soient ordonnés du plus spécifique (ou plus intensionnel) vers le plus général (ou le plus extensionnel) comme pour:

caméra électronique miniaturisée bleue

où seuls les trois premiers mots constituent une forme figée. Au sein de ce nom composé, il semble que le premier adjectif soit plus spécifique que le second. D'autres exemples confirment cette impression:

barrettes photosensibles tournantes
réseau téléphonique commuté.

Conflits entre ces trois contraintes

Les trois contraintes vues précédemment peuvent être en conflit ou agir en complémentarité.
- (1) la transformation `[connexe]`
- (2) la transformation `[antéposition]`
- (3) les considérations sur l'ordre des modifieurs

(1) et (3) peuvent être contradictoires: (1) est retenue dans *tableau de bord local* alors que (3) l'a emporté dans *logique étendue des prédicats*. Dans le cas de noms composés peu figés, il n'est pas possible de prédire la forme qui sera retenue. Dans le cas de formes très figées comme *tableau de bord* la préservation de la connexité (1) l'emporte.

L'application de (2) entraîne implicitement celle de (1) puisque l'antéposition préserve la connexité: *futur poste source*.

De même, (2) et (3) ne sont pas antinomiques puisque l'antéposition n'empêche pas de conserver les modifieurs les plus courts en tête: *nouveau tableau électrique (monobloc) (moyenne tension) (à trois interrupteurs)*.

En cas de conflit entre (1) et (3), il se peut que certains assemblages permettent de mauvaises interprétations du sens du composé: l'association locale de certains modifieurs peut produire une séquence licite mais erronée. Ainsi *capture (globale) (des produits de fission)* sera préférée à *capture (des produits de fission) (globale)*, puisque dans cette dernière forme l'adjectif semble se rapporter plus à *fission* que à *capture*. La considération sur l'ordre des modifieurs sera donc privilégiée pour mettre les modifieurs adjectivaux avant les prépositionnels.

Les nouvelles transformations sur un nom composé obtenu par juxtaposition (catégorie 1)

Nous définissons ici les transformations propres à une structure composite, non plus à partir de celles de ses constituants, mais par analogie avec la forme de base ayant la même structure syntaxique. Ainsi *(conduite d'arrivée) d'(eau pressurisée)* correspond à la structure syntaxique en N_{comp} de N_{comp} associée à la forme de base en N de N. Nous allons examiner les transformations qui s'appliquent sur cette structure de base, pour valider leur extension à une structure analogue dont les constituants ne sont pas des mots simples, mais des noms composés ayant éventuellement subi des transformations.

Nous reprenons la liste des transformations de N de N:

- La modification du premier nom ou du deuxième nom.

Ces transformations ne sont pas nouvelles puisqu'elles sont déjà décrites par les transformations de ces deux noms composés, avec composition éventuelle par la fonction [connexe]. A l'ajout d'un modifieur à l'un des deux noms dans la structure N de N, nous faisons correspondre l'ajout d'un modifieur à un des deux noms composés de la structure N_{comp} de N_{comp}.

- La coordination, au niveau supérieur, est différente des coordinations locales qui opèrent sur les sous-arbres et qui ne peuvent dupliquer les branches de l'arbre supérieur. Les trois métarègles représentant la coordination, et obtenues par extension de celles sur N de N, sont:

(M3) (N_{compl} de N_{comp2}) -> (N_{compl} (de N_{comp2} ConjCoord de GN))
(M3') (N_{compl} de N_{comp2}) -> (N_{compl} de (N_{comp2} ConjCoord GN))
(M4) (N_{compl} de N_{comp2}) -> ((N_{compl} ConjCoord GN) de N_{comp2})).

Les trois possibilités sont observées dans notre cas particulier:

les conduites d'arrivée (d'eau pressurisée ou de vapeur surchauffée)

? *les conduites d'arrivée (de vapeur surchauffée ou eau pressurisée)*

(les conduites d'arrivée et les vannes) d'(eau pressurisée).

Comme nous l'avons déjà précisé, ces trois métarègles ne constituent pas la totalité des coordinations pouvant être définies sur cette structure. Par exemple, ces transformations n'incluent pas la possibilité de répétition du déterminant dans la coordination comme dans le troisième exemple.
Nous ne donnons pas la liste exhaustive de toutes ces coordinations pour ne pas surcharger la présentation.

- L'acceptabilité de N de Dét N_{comp} doit être décrite ici, sachant qu'elle peut s'appliquer sur les formes modifiées des groupes nominaux comme nous le verrons plus loin lors des compositions de transformations de catégories différentes. Ainsi *cohérence de (base de règles)* accepte l'insertion d'un déterminant devant le deuxième nom composé pour donner *cohérence de la (base de règles)*.

Catégorie 2 des transformations sur les noms composés obtenus par juxtaposition (certains mots pleins sont transformés en d'autres mots pleins)

Automatismes sur un nom composé obtenu par juxtaposition (catégorie 2)

Certaines transformations définies sur une forme de base peuvent s'appliquer automatiquement à la structure supérieure. Parmi les transformations de catégorie 2, se trouvent les abréviations, elles s'étendent sans difficulté à une forme obtenue par juxtaposition de noms composés. Par exemple:

réacteur à eau pressurisée -> REP

s'applique au nom composé *(circuit du secondaire) des (réacteurs à eau pressurisée)* sans adaptation:

(circuit du secondaire) des (réacteurs à eau pressurisée) -> (circuit du secondaire) des (REP)

Nouvelles transformations sur un nom composé obtenu par juxtaposition (catégorie 2)

Il est exceptionnel qu'une nominalisation ou adjectivation soit associée à un nom composé d'ordre supérieur, du type:

N_{comp} de N_{comp} -> N_{comp} Adj.

Parmi les juxtapositions, seront donc candidates à la définition d'une transformation de catégorie 2, celles contenant un mot simple en position de modifieur. Ce sont donc les juxtapositions d'un nom

composé et d'un adjectif ou d'un syntagme prépositionnel qui peuvent subir une transformation de l'adjectif en syntagme prépositionnel ou l'inverse. Dans le corpus étudié, nous n'avons pas de noms composés suivant cette transformation, et nous ne pouvons fournir que des contre-exemples:

(base de données) relationnelle ne donne pas **(base de données) de relations*

Catégorie 3 des transformations sur les noms composés obtenus par juxtaposition (certains mots pleins sont supprimés ou transformés en mots vides)

Pour ces transformations, nous pouvons encore distinguer celles qui se déduisent des transformations des éléments de base, de celles qui doivent être définies spécifiquement sur le composé de niveau supérieur. Nous observons, dans un premier temps, le mécanisme des élisions, puis nous considérons celui des coordinations.

Elisions sur un nom composé obtenu par juxtaposition

Automatismes sur un nom composé obtenu par juxtaposition (catégorie 3: élisions)

Sur la structure N_{comp1} de N_{comp2} peuvent être définies des élisions obtenues par extension des élisions sur les formes de base N_{comp1} et N_{comp2}. Par exemple, le nom composé *(tambour filtrant) des (stations de pompage)* est la juxtaposition des deux entrées *tambour filtrant* et *stations de pompage* admettant respectivement comme élisions *tambour* et *stations*. Par extension au composé global, on en déduit les (2 x 2 - 1) élisions suivantes:

tambour des stations de pompage
tambour filtrant des stations
tambour des stations

Nouvelles transformations sur un nom composé obtenu par juxtaposition (catégorie 3: élisions)

La structure étant N_{comp1} de N_{comp2}, ces nouvelles élisions sont obtenues par extension de celles définies sur une structure en N_1 de N_2 en remplaçant N_1 par N_{comp1} et N_2 par N_{comp2}. Or, la seule élision possible sur N_1 de N_2, est celle qui ne garde que N_1, étendue à N_{comp1} de N_{comp2}, elle ne conserve que N_{comp1}.

Sur l'exemple précédent *(tambour filtrant) des (stations de pompage)* de structure N_{comp1} de N_{comp2}, on peut définir l'élision qui conserve N_{comp1} seul, celui-ci pouvant être lui-même une forme incomplète. On obtient donc deux nouvelles élisions:

tambour filtrant
tambour.

Ces formes réduites sont déjà répertoriées comme élisions de l'entrée lexicale N_{comp1}. Si on les fait figurer comme élisions de la forme N_{comp1} de N_{comp2}, on introduit une ambiguïté artificielle. Si une des deux formes réduites *tambour filtrant* ou *tambour* est précédée d'un pronom possessif indiquant une coréférence, alors elle indique bien le résultat d'une transformation d'une forme en N_{comp1} de N_{comp2} telle que:

la station de pompage .... Son tambour filtrant ...

Dans le cas d'une juxtaposition de deux noms composés ou d'un nom simple et d'un nom composé, il n'y a pas de nouvelle élision. Toutes les élisions de la structure composite peuvent être obtenues par automatisme à partir de celles des structures de base. Seule la juxtaposition de trois structures (qui est un cas rarissime) nécessitera une définition de l'élision d'un des trois termes qui soit définie au niveau supérieur.

Coordination sur un nom composé obtenu par juxtaposition

Automatismes sur un nom composé obtenu par juxtaposition (catégorie 3: coordination)

Nous montrons, sur un exemple, que les règles de construction de la coordination ne sont pas aussi simples que des règles de calcul algébrique.Nous confirmons ainsi, que la prudence est nécessaire lorsque l'on traite des propriétés logico-mathématiques du langage.

Sur quatre structures de base, ayant deux à deux des mots communs, on peut envisager des coordinations sur les composants de base qui s'étendent automatiquement à la structure supérieure. Considérons les quatre noms composés suivants:

(1) *constructeur de matériel informatique*
(2) *fournisseur de matériel informatique*
(3) *développeur de logiciel informatique*
(4) *fournisseur de logiciel informatique*

ils sont tous de structure N de N_{comp}, avec N_{comp} qui est de la forme N Adj tel que *matériel informatique* ou *logiciel informatique*. Ces structures de base peuvent être coordonnées en N_{21} ConjCoord N_{22} Adj_2 puisqu'elles sont dans un domaine sémantique proche, ce qui donne *matériel et logiciel informatiques.*

Lorsqu'elles constituent une structure en N_1 de N_2 Adj_2, pour que la coordination puisse s'étendre, il faudrait, logiquement, que les formes N_1 de N_{21} Adj_2 et N_1 de N_{22} Adj_2

soient attestées. Donc, d'après la liste exhaustive des quatre composés précédents, seule la coordination *fournisseur de matériel et (de) logiciel informatique* devrait être attestée.
Or, nous avons rencontré (5) *des constructeurs et des fournisseurs de matériels et de logiciels informatiques.*

Si nous traitons cette coordination comme une formule algébrique de factorisation, et si nous écrivons cette formule, nous avons deux parenthésages possibles:

(5') *(des constructeurs et des fournisseurs) (de matériels et de logiciels informatiques)*

(5'') *des constructeurs et (des fournisseurs (de matériels et de logiciels informatiques)).*

Nous ne considérons pas les deux parenthésages possibles de *de matériels et de logiciels informatiques* puisque ces deux possibilités ne changent pas notre analyse ultérieure et que les considérations de sens font penser qu'il s'agit de *(de matériels et de logiciels) informatiques* plutôt que de *(de matériels) et (de logiciels informatiques).*

(5') n'est pas un parenthésage possible puisque le développement par double distributivité donne lieu à une forme illicite *des constructeurs de logiciels informatiques* qui ne fait pas partie des formes initiales attestées.
(5'') est correct sur le plan formel, mais n'est pas satisfaisant pour le sens déduit, puisqu'il n'attribue pas de modifieur à *constructeur*, or le sens de la phrase fait penser qu'il s'agit de (1)*constructeurs de matériels informatiques.*

Afin de mettre en évidence les trois noms composés de base (1), (2) et (4) à partir de (5), ((3) ne peut être obtenu puisque *développeur* ne figure pas dans (5)), nous prenons le parenthésage (5') de la forme `(a + b) (c + d)` et nous nous écartons des règles opératoires algébriques formelles en posant que:

`(a + b) (c + d)` peut se développer `ac + bc + bd`

dans la mesure où la séquence `ad` n'est pas acceptable. Cette possibilité est compatible avec l'éloignement en distance de `a` et `d` dans la formule `(a + b) (c + d)`. Elle aurait été plus difficile à admettre sur une formule telle que `(b + a) (d + c)`: *(des fournisseurs et des constructeurs) (de logiciels et de matériels informatiques)* où `a` et `d` sont proches.

Sur cet exemple, nous voyons que les règles strictes de l'algèbre sont modulées par l'éloignement de certains termes qui permet à la compréhension humaine de ne pas associer des termes incohérents. L'application de principes mathématiques, tels que ceux de l'algèbre pour la syntaxe de la coordination, ou ceux de la logique formelle pour l'interprétation de la coréférence ou de la coordination, permettent de modéliser certains phénomènes de langue. Cependant, on trouve fréquemment des contre-exemples qui contredisent les modèles et obligent à les adapter pour qu'ils

respectent les mécanismes observés. Ainsi [Fauconnier 74] montre que l'utilisation de variables logiques pour interpréter la coréférence échoue dans les cas de références multiples, par exemple.Il propose une "théorie allégée" qui prend en compte le comportement spécifique de la langue, mais en gardant un aspect logique et systématique. Une telle démarche nécessite une observation rigoureuse de la langue, une création de modèles *ad hoc* et une validation des choix faits.

Les rapports entre Linguistique et Mathématiques ne se résument pas à l'application simple d'outils mathématiques à la langue, comme peut le laisser penser [McCawley 81]. En particulier, la coordination ne se résume ni à des formules algébriques sur le plan syntaxique, ni à des tables de vérité pour l'interprétation. [Lentin 90] analyse l'utilisation des mathématiques dans l'oeuvre de Harris. Il montre que Harris a cherché à reprendre des principes mathématiques tels que ceux énoncés par des algébristes comme Birkhoff et Mac Lane. Ces théories lui ont servi à définir les mécanismes des transformations et des opérations. Il n'a pas cherché à plaquer un modèle sur la langue, mais il a repris dans les mathématiques (algèbre, puis λ-calcul plus récemment) des principes qui lui ont servi à élaborer ses notions linguistiques.

Nouvelles transformations sur un nom composé obtenu par juxtaposition (catégorie 3: coordination)

Nous définissons, ici, les coordinations s'appliquant sur la structure de niveau supérieur. Elles produisent des formes qui ne sont pas déductibles des coordinations sur les noms composés de base, et, de plus, elles sont plus fréquentes que celles obtenues à partir des coordinations des formes de base.

Ainsi *station de travail scientifique* et *station de travail technique* sont deux noms composés en N_{comp} `—Adj`, dont les deux coordinations possibles au niveau supérieur sont N_{comp1} `—Adj`$_1$ `—ConjCoord Adj`$_2$ et N_{comp1} `—ConjCoord`N_{comp2} `—Adj`$_1$, qui produisent toutes les deux des formes attestées telles que *station de travail (scientifique et technique)* ou *(station de travail ou micro-ordinateur) scientifique.*

Analyse automatique de la coordination sur un nom composé formé par juxtaposition

Pour traiter la coordination en analyse automatique, on peut, soit produire systématiquement toutes les forme coordonnées possibles, soit ne déclencher cette production que dans le cas de la rencontre d'une conjonction de coordination. Dans notre application informatique, nous avons choisi la première méthode qui est la plus sure, mais également la plus lente puisque sa complexité est une fonction exponentielle de la taille des structures. Nous ne prétendons pas, pour autant couvrir tous les cas de coordination pour les raisons suivantes:

- nous ne produisons pas d'analyse globale de la phrase, or la coordination ne peut être traitée de façon exhaustive en se plaçant dans le cadre restreint des structures locales,

- pour traiter tous les cas de coordination, il est nécessaire de vérifier que la lexicalisation choisie est suffisante. Il faut s'assurer qu'il y aura suffisamment de mots présents dans le texte pour provoquer le chargement des règles correspondant aux formes de base qui composent la coordination,

- de plus, nous connaissons bien les règles rigoureuses de construction, or nous avons vu que certaines figures échappent à celles-ci, sans rendre le texte inintelligible. Nous n'avons pas recensé de façon exhaustive toutes ces constructions.

Dans le cas où l'on ne ne déclenche la construction de formes coordonnées que lorsqu'une conjonction est rencontrée, l'analyse est plus rapide que la nôtre, mais présente l'inconvénient de ne pas s'appliquer dans le cas de coordinations sans conjonction. Dans notre système, les règles sans conjonction sont explicitement décrites comme:

(M4') $(N_1$ de $N_2)$ -> $((N_1$ GN) de $N_2))$

qui est la version sans conjonction de coordination de:

(M4) $(N_1$ de $N_2)$ -> $((N_1$ ConjCoord GN) de $N_2))$

et qui permet d'analyser ou de produire une forme telles que:

les constructeurs fournisseurs de micro-ordinateurs.

Composition de transformations de catégories différentes sur un nom composé obtenu par juxtaposition

Dans ce paragraphe, nous étudions la composition des transformations entre constituants de base et noms composés de niveau supérieur. Ce problème peut se réduire à un tableau à deux entrées: d'une part les transformations sur la structure supérieure, d'autre part, les transformations sur la ou les structures de base. L'acceptabilité d'une transformation sur l'une est liée à la forme de l'autre par le degré d'acceptabilité qui se trouve au croisement des transformations.

La transformation en N de Dét N est acceptée pour *cohérence de (base de règles)*, elle donne *cohérence de la (base de règles)*. Cette transformation peut se croiser avec celles qui opèrent sur *base de règles*:

cohérence de la (base)	(élision sur *base de règles*)
cohérence de la (base de règles d'expertise)	(modifieur postposé de *règles*)
cohérence de la (grande base de règles)	(modifieur antéposé de *base*).

Rien n'indique que cette uniformité soit toujours vrai. Généralement, le nom composé de niveau supérieur est moins figé que le nom composé de base qui le constitue, donc les transformations de celui-ci seront acceptées au niveau du composé supérieur quelle que soit sa forme. Il se peut que

certaines incompatibilités sémantiques apparaissent comme dans la transformation de *bulles d'air* dans *rideau de grandes bulles d'air*. Le composé obtenu ne correspond pas à une réalité du monde extérieur.

Un exemple de croisement de transformations

Nous étudions le croisement exhaustif des transformations sur le nom composé obtenu par juxtaposition d'un nom simple et d'un nom composé: *câble à (haute tension)*.

Nous mettons en colonnes les transformations éventuelles sur Adj N, et en lignes celles sur N à (Adj N). Les transformations sur N à (Adj N) sont obtenues par extension des transformations de N à N. Ce mécanisme d'extension et celui du croisement des métarègles entre forme de base et structure de niveau supérieur sont décrits au II.1.d.

Donnons quelques exemples permettant de justifier les valeurs d'acceptabilité fournies (les numéros entre parenthèses font référence aux cases numérotées du tableau de la figure 29):

(1): *ce câble est à haute tension*

(2): *ce câble est à très haute tension*

(3): *ce câble est à haute ou moyenne tension*

(4): * *ce câble est à tension de hauteur*

(5): *ce câble est à haute tension continue*

(6): la forme verbale est inacceptable: * *câbler à haute tension.*

(7): *câble haute tension*

Quelques remarques peuvent être faites à l'observation de la matrice de la figure 29.

- Si une transformation n'est pas acceptée sur le composé de base lorsqu'il est seul, elle ne l'est pas plus lorsqu'il est inclus dans le composé global. De même, lorsqu'une transformation sur la forme supérieure n'est pas acceptable avec la forme de base non modifiée, il est peu probable qu'elle le soit plus avec la forme de base modifiée. Pour qu'une transformation croisée soit vraie, il est nécessaire, mais non suffisant que les deux transformations de base le soient. Dans l'ensemble, le croisement des transformations a tendance à faire baisser l'acceptabilité.

- La transformation du deuxième nom de la forme globale n'est pas étudiée, puisqu'elle est directement observée sur le nom composé de base (ligne "inutile" de la figure 29).

- L'acceptabilité de la transformation du nom du composé de base dans le cadre du composé supérieur est liée au type de modifieur par des contraintes essentiellement sémantiques (voir ci-dessus).

Sur cet exemple, nous constatons que les transformations sont **transversales**, c'est à dire que l'acceptabilité d'une transformation locale ne dépend pas de la forme de la structure globale et réciproquement. Le croisement d'une transformation sur la forme globale et d'une transformation sur la forme de base est acceptable si et seulement si ces deux transformations le sont. Il est donc légitime de procéder comme nous l'avions fait, c'est à dire d'étudier indépendamment les transformations sur les composants de base et sur la forme supérieure puisqu'il ne semble pas y avoir de corrélation entre ces phénomènes. Nous verrons au I.2.b que cette propriété est compatible avec une gestion dynamique des métarègles.

Dans le cas d'indépendance, la matrice s'obtient ainsi à partir des valeurs d'acceptabilité des transformations sur le nom composé de base et sur la structure supérieure (on suppose qu'il n'y a que + et -):

- tous les signes -, apparaissant sur une ligne ou une colonne, sont prolongés sur toute la longueur de cette ligne ou cette colonne,
- le reste est rempli par des +.

haute tension / *câble à haute tension*	Forme normale	N "est" Adj	N Vsup Nominalisation	Adv Adj N	Adj1 Conjc Adj2 N	Lien avec N "de" N	Modifieur du N
Mot composé "haute tension" seul	+	-	-	+	+	-	+
Forme normale	+	-	-	+	+	-	+
Modifieur du premier N	+	-	-	+	+	-	+
Modifieur du deuxième N	Inutile						
Coordination	+	-	-	+	+	-	+
Forme verbale V "à" N	$-_6$	-	-	-	-	-	-
Prédicativité : N "est à" N	$+_1$	-	-	$+_2$	$+_3$	$-_4$	$+_5$
Lien avec une forme en N Adj	-	-	-	-	-	-	-
Acceptation de N "à" Dét N	-	-	-	-	-	-	-
Acceptation de N N	$+_7$	-	-	+	+	-	+

Figure 29: composition des transformations sur un nom composé supérieur *câble à (haute tension)*

Conclusion sur les noms composés obtenus par juxtaposition

Le croisement des transformations sur la structure supérieure, avec celles qui opèrent sur les structures de base, composées éventuellement avec la fonction [connexe] permettent de générer toutes les transformations possibles sur une structure supérieure. De plus, nous verrons au II.1.d que les transformations sur une structure supérieure telle que N de (Adj N) peuvent se déduire de celles sur la structure de base N de N (car l'analyse syntaxique de N de N produit un arbre ayant une partie commune avec celui de l'analyse de N de (Adj N)).

Il y a donc homogénéité entre les transformations des noms composés construits par juxtapositions de formes de base, et transformations sur ces formes de base. Cette similitude incite à fragmenter les structures longues en structures élémentaires minimales, et à utiliser un vocabulaire de description formé essentiellement des formes suivantes: N de N, N Adj, N à N, N Prép N, N N. Nous reviendrons au II.12.d sur ces mécanismes d'extension et de croisement des transformations.

Noms composés imbriqués par recouvrement

Différents modes de recouvrement

Recouvrement par mise en commun d'un nom central

La structure supérieure est obtenue à partir de deux structures en N Prép N telles que le dernier mot de la première soit commun avec le premier mot de la deuxième. Il s'agit de la forme la plus fréquente de recouvrement dans notre corpus.

(1) *panneau de comptage* et *comptage d'électricité* -> *panneau de comptage d'électricité*

Ce recouvrement correspond à la structure suivante:

N_1 de N_2 et N_2 de N_3 -> (N_1 de (N_2 de N_3))

Il s'agit ici du remplacement du deuxième nom N_2 dans le premier composé par le nom composé (N_2 de N_3). Pour l'exemple (1), la représentation de la structure syntaxique est donnée dans la figure 30. Cette représentation nous renseigne sur la façon dont les transformations sur les formes de base vont pouvoir être étendues à cette structure composite.

Ce remplacement aurait pu se faire sur tout autre nom composé de structure nominale, dont la tête est *comptage*, comme la structure N_2 Adj ou une structure N_2 N_3:

(2) *panneau de comptage* et *comptage électrique* -> *panneau de comptage électrique*

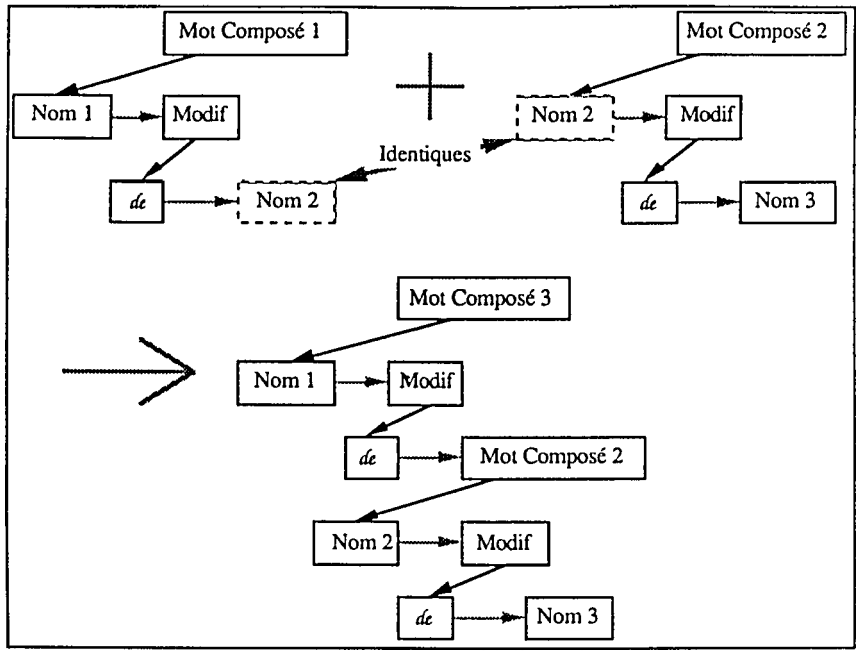


Figure 30: structure syntaxique d'un nom composé supérieur en $(N_1$ de (N_2) de $N_3)$:
panneau de comptage d'électricité

Recouvrement par mise en commun d'un nom initial

Certains noms composés sont obtenus par mise en commun d'un même nom initial comme pour *((application industrielle) de l'électricité)* obtenu à partir de *application industrielle* et *application de l'électricité*. Ils correspondent à une structure syntaxique de double modification du nom. Ces formes sont rares et leur degré de figement est faible, le recouvrement par mot commun central produit des formes beaucoup plus stables.

La représentation de la structure syntaxique dans le cas de l'exemple (2) est donnée dans la figure 31, elle correspond au recouvrement suivant:

N_1 Adj et N_1 de N_2 -> $((N_1$ Adj) de $N_2)$

Ces formations d'une nouvelle structure ne doivent pas être confondues avec d'éventuelles transformations de la forme de base, vues précédemment, comme *panneau de comptage scellé* où le nom composé *comptage scellé* ne correspond pas à une entrée lexicale de la langue, et où le V_{pp} modifie le nom *panneau*.

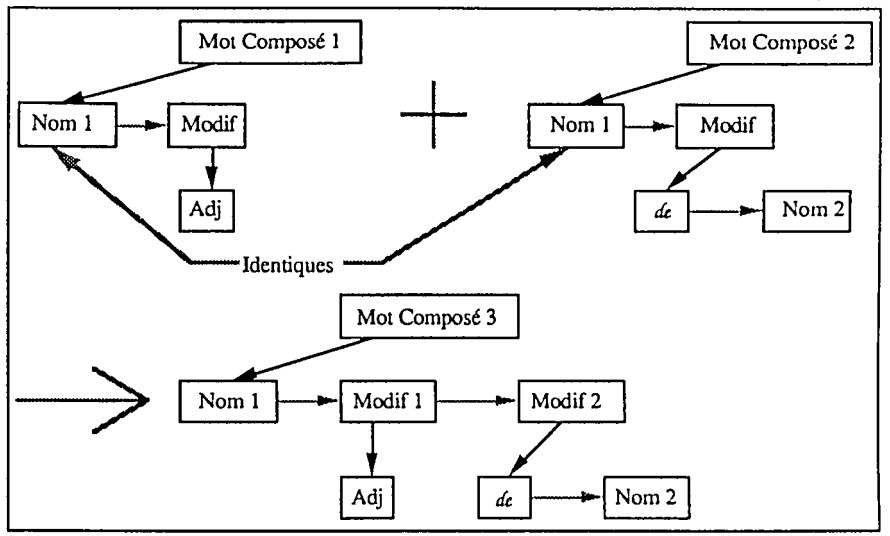


Figure 31: structure syntaxique d'un nom composé supérieur en $((N_1$ Adj) de $N_2)$:
application industrielle d'électricité

Représentation des noms composés obtenus par recouvrement

Recouvrement par mise en commun d'un nom central

Un seul des deux composés de base, dont le recouvrement constitue le nom composé d'ordre supérieur, se trouve dans la représentation de celui-ci. Si les deux composés de base sont:

```
<panneau de comptage> -> (N, !1, !1) panneau(N, m, v, n)
                         de(P, v, v, v) comptage(N, m, s, n);
<comptage de l'électricité> -> (N, !1, !1)
                         comptage(N, m, s, n) de(P, v, v, v)
                         l(Dd, v, v, v) électricité(N, f, s, n);
```

le composé global s'écrit:

```
<panneau de comptage de l'électricité> -> (N, !1, !1)
                         panneau(N, m, v, n) de(P, v, v, v)
                         <comptage de l'électricité> (N, m, s);
```

Une grammaire, contenant ces règles, appliquée à l'analyse de *panneau de comptage de l'électricité*, permet de reconnaître les formes non modifiées suivantes: *panneau de comptage*, *comptage de l'électricité* et *panneau de comptage de l'électricité*.

Cette représentation est bien adaptée à la structure syntaxique observée, puisque nous avons

conservé le deuxième nom composé au lieu du premier. Si nous considérons l'arbre syntaxique correspondant, nous remarquons que le sous-arbre syntaxique qui coïncide avec la représentation du recouvrement jusqu'à la profondeur 3 est égal à l'arbre associé à la règle ci-dessus. Il serait également possible de garder, dans la représentation du nom composé supérieur, le premier nom composé <panneau de comptage>.

Il serait moins efficace de considérer que les deux entrées de base *panneau de comptage* et *comptage d'électricité* sont les élisions d'une entrée globale *panneau de comptage d'électricité* définie uniquement à partir de noms simples. Dans ce cas, nous n'aurions pas de moyen de décrire finement leurs propriétés syntaxiques. Nous n'aurions pas, non plus, la possibilité de décrire le comportement du composé supérieur à partir des transformations de la forme de base *panneau de comptage* étendues automatiquement à la structure supérieure.

Recouvrement par mise en commun d'un nom initial

Si les deux composés de base sont:

```
<application industrielle> -> (N, !1, !1)
            application(N, f, v, n) industriel(NA, a1, a1, n);
<application de l'électricité> -> (N, !1, !1)
            application(N, f, v, n) DE(P, v, v, n)
            LE(Dd, f, s, a) électricité(N, f, s, n)
```

le composé global s'écrit:

```
<application industrielle de l'électricité> -> (N, !1, !1)
            <application industrielle>(N, f, v) DE(P, v, v, n)
            LE(Dd, f, s, a) électricité(N, f, s, n)
```

La forme du nom composé impose de ne prendre dans la règle globale que le premier nom composé de base. Le premier modifieur *industriel*, inclus dans <application industrielle> a un statut différent du deuxième modifieur *de l'électricité* qui est utilisé dans la description du nom composé global. A cause de cette dissymétrie, cette représentation est plus discutable que celle choisie pour les formes de recouvrement à mot commun central. Dans ce cas de mot initial commun, nous pourrions choisir une représentation n'utilisant que des mots simples.

Dans la suite de ce paragraphe sur le recouvrement, nous étudions les transformations des noms composés obtenus ainsi. Sur les deux exemples précédents, la représentation choisie est celle d'une juxtaposition d'un nom simple et d'un nom composé ou d'un nom composé et d'un nom simple. Il est donc possible de reprendre les transformations observées sur la juxtaposition.
Pour l'étude des transformations sur les noms composés obtenus par recouvrement, nous nous limitons donc à la description des différences avec la juxtaposition. Celles-ci proviennent de l'inclusion d'une deuxième forme figée dans le composé qui se superpose avec la première et qui n'est pas visible dans l'énoncé du mot composé supérieur.

Catégorie 1 des transformations sur les noms composés obtenus par recouvrement

Nous retrouvons les transformations obtenues par automatisme ou définies sur la structure supérieure comme dans le cas de la juxtaposition, mais avec une différence.
Nous prenons comme exemple de structure obtenue par recouvrement: *(niveau d'(émission) des messages)*. Pour la juxtaposition, seuls les noms composés de base visibles dans la règle sont dans le lexique. Pour le recouvrement, un seul des deux noms composés *niveau d'émission* ou *émission des messages* sera visible. Une seule des deux transformations suivantes pourra être obtenue par automatisme:

[Modif] *niveau d'émission des messages*
-> *niveau modéré d'émission des messages* (si *niveau d'émission* est visible)

[Modif] *niveau d'émission des messages*
-> *niveau d'émission sonore des messages* (si *émission des messages* est visible)

Si *niveau d'émission* apparaît dans la règle, une première application de [connexe] permet au modifieur se rapportant à *niveau* de sortir du premier nom composé *niveau d'émission*, la deuxième application le fait sortir du nom composé supérieur.

[connexe] *niveau modéré d'émission des messages*
-> *niveau d'émission modéré des messages*

[connexe] *niveau modéré d'émission des messages*
-> *niveau d'émission des messages modéré*.

Catégorie 2 des transformations sur les noms composés obtenus par recouvrement

Dans le cas de recouvrement, les transformations de catégorie 2 ne doivent pas altérer le nom intermédiaire qui sert de charnière entre les deux structures, ou le nom initial qui est commun. S'il y a transformation, elle ne peut se produire que sur les parties qui ne sont pas communes. Dans le corpus étudié, cette transformation n'a pas été rencontrée, elle pourrait cependant se trouver sur une forme telle que:

(direction de (propagation) du rayonnement) -> (direction de (propagation) radiative)

Lien entre les ATN et le formalisme des règles

Il est possible de faire le lien entre les transformations sur les noms composés et une représentation par un ATN (Augmented Transition Network) [Winograd 83].
Les réseaux de transition sont équivalents aux grammaires régulières. Si l'on souhaite construire

des réseaux équivalents aux grammaires Context Free, on utilise des RTN (Recursive Transition Network) dont les arcs peuvent porter comme étiquette le nom d'un réseau. Les RTN sont fortement équivalents aux grammaires Context Free: ils peuvent reconnaître le même langage et produire la même structure à l'analyse. Généralement, une équivalence faible (le même langage, mais pas les mêmes structures) est suffisante. On produit alors des arbres plus "plats" au moyen des réseaux qu'au moyen des règles équivalentes. Les règles contiennent plus de non terminaux.

Les ATN forment un sur-ensemble des RTN, il s'agit de réseaux auxquels sont ajoutées des conditions et des actions et qui utilisent des registres associés. Ces grammaires ATN s'inspirent du formalisme de Woods et permettent de définir des réseaux respectant les contraintes de la langue telles que les accords verbe sujet, l'utilisation du contexte non local, les dépendances de longue distance, gestion des cas... [Winograd 83] donne une définition du groupe nominal qui ne se limite pas au cas des formes figées et qui utilise le formalisme des ATN, prouvant ainsi leur adéquation aux descriptions syntaxiques de ces formes.

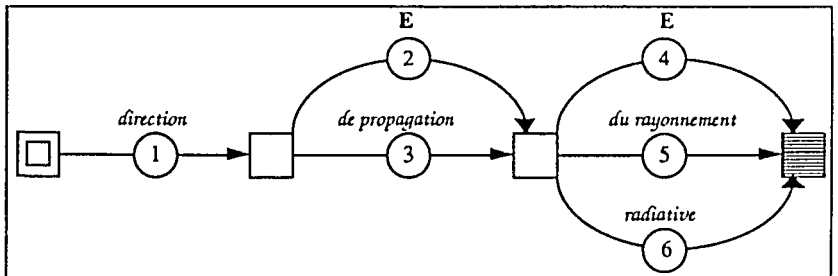
Dans notre étude, nous avons préféré utiliser un formalisme de règles, repris de la programmation en logique. Nous le trouvons plus proche d'une description lexicale que celui des graphes. Il s'agit en fait d'un formalisme *ad hoc*, dont toutes les fonctionnalités ont été réalisées dans le seul souci de décrire la langue. En règle générale, les formalismes de description des langues reposent sur des bases mathématiques claires telles que les réseaux de transition ou les règles de la logique formelle, et sont adaptés en fonction des contraintes et des cas particuliers rencontrés dans la langue. Tous les outils nécessitent une application constante aux phénomènes observés. Le choix d'un formalisme n'est pas très important, l'essentiel est de s'assurer qu'il permet de décrire précisément la langue.

Lien entre une règle de nom composé et un ATN sur un exemple

Dans le cas de la règle d'un nom composé, il est possible de créer un ATN faiblement équivalent qui reconnaisse le même langage que cette règle. Nous utilisons une représentation simplifiée telle que celle de la figure 32. Nous comparons les formes reconnues par cet automate, avec celles pouvant être analysées par les deux règles suivantes:

(a) `<direction de propagation> -> direction de propagation;`
admettant une élision qui produit: `direction`

(b) `<direction de propagation du rayonnement>`
`-> <direction de propagation> du rayonnement`
admettant une élision qui produit: `<direction de propagation>` et une adjectivation qui remplace `du rayonnement` par `radiative`.



Croisement	(b)	élision de (b)	adjectivation de (b)
(a)	1-3-5	1-3-4	1-3-6
élision de (a)	1-2-5	1-2-4	1-2-6

Figure 32: réseau de transition associé à *direction (E + de propagation) (E + du rayonnement + radiative)* et lien avec les règles correspondantes (a) et (b).

Nous notons 1-2-4, la forme reconnue en parcourant les arcs 1, 2 et 4, soit *direction* puisque le symbole **E** représente la chaîne vide. Nous donnons pour chaque parcours possible du graphe son équivalent avec les règles de noms composés:

1-3-5 *direction de propagation du rayonnement*	nom composé supérieur (b)
1-3-4 *direction de propagation*	élision de (b)
1-3-6 *direction de propagation radiative*	adjectivation de (b)
1-2-5 *direction du rayonnement*	élision de (a) étendue à (b)
1-2-4 *direction*	élision de (a) croisée avec celle de (b)
1-2-6 *direction radiative*	élision de (a) croisée avec l'adjectivation de (b)

Le lexique comporte le nom composé de base (a) dont la juxtaposition avec `du rayonhement` donne le nom composé d'ordre supérieur (b). Les seules transformations recensées sont l'élision et l'adjectivation de (b) et l'élision de (c). Le graphe proposé permet de produire ou d'analyser tous les croisements possibles entre les transformations de (a) et celles de (b) (y compris (a) et (b) sous leur forme initiale), comme le montre le tableau de la figure 32.

Il est ainsi possible de représenter de façon équivalente des transformations de catégories 2 et 3 par des règles ou par des ATN. Les transformations de catégorie 1 nécessitent l'appel de graphes permettant d'analyser les modifieurs insérés. Dans un formalisme de réseaux de transition, les métarègles sont des opérations sur les réseaux.

Catégorie 3 des transformations sur les noms composés obtenus par recouvrement

Elisions sur un nom composé obtenu par recouvrement

Nous considérons le nom composé d'ordre supérieur *système d'équations différentielles*, obtenu par recouvrement de *système d'équations* et *équations différentielles*. Nous représentons ce nom composé par:

(a') <équations différentielles> -> équations différentielles;

admettant une élision qui produit: équations

(b') <système d'équations différentielles>
-> système d' <équations différentielles>

admettant une élision qui produit: système.

Nous recensons les deux types d'élisions obtenues sur le nom composé supérieur: les automatismes, et les nouvelles élisions.

- L'extension automatique des élisions définies sur (a') : le composé supérieur (b') peut être référencé par *système d'équations*, car (a') peut se transformer, en contexte, en *équations*.

- La nouvelle élision définie sur le composé supérieur (b'), qui utilise *système* pour désigner *système d'équations différentielles*, n'apporte pas de nouveauté puisqu'elle correspond déjà à l'élision du deuxième terme dans <système d'équations>.

Coordination sur un nom composé obtenu par recouvrement

Automatismes sur un nom composé obtenu par recouvrement (catégorie 2: coordination)

Toutes les coordinations définies sur *terminaux informatiques* peuvent être étendues à *concentrateur de terminaux informatiques*, à condition que la coordination obtenue soit une forme nominale.

Nous donnons en exemple deux possibilités de coordination:

(1) (N_1 Adj_1) -> (N_1 (Adj_1 ConjCoord Modif))
(2) (N_1 Adj_1) -> ((N_1 ConjCoord GN) Adj_1)

Ces règles appliquées au nom composé donnent les formes suivantes:

(1') (terminaux (informatiques ConjCoord Modif))
(2') ((terminaux ConjCoord GN) informatiques)

Ces transformations peuvent être étendues automatiquement au composé supérieur *concentrateur de terminaux informatiques* pour donner les deux exemples ci-dessous:

(1") *concentrateur de terminaux informatiques ou graphiques*

(2") *concentrateur de terminaux ou consoles informatiques*

Nouvelles coordinations sur un nom composé obtenu par recouvrement (catégorie 3: coordination)

Ces nouvelles coordinations sont définies directement sur la structure du nom composé en N_1 de (N_2 Adj_1) par extension des métarègles de coordination de N_1 de N_2 à une structure en N_1 de N_{comp}. Nous détaillerons cette possibilité au II.1.d, nous indiquons ici la liste des nouvelles métarègles ainsi obtenues:

(a) (N_1 de (N_2 Adj_1)) -> (N_1 (de (N_2 Adj_1) ConjCoord de GN))
(b) (N_1 de (N_2 Adj_1)) -> (N_1 de ((N_2 Adj_1) ConjCoord GN))
(c) (N_1 de (N_2 Adj_1)) -> ((N_1 ConjCoord GN) de (N_2 Adj_1))

obtenues par extension de:

(a') (N_1 de N_2) -> (N_1 (de N_2 ConjCoord de GN))
(b') (N_1 de N_2) -> (N_1 de (N_2 ConjCoord GN))
(c') (N_1 de N_2) -> ((N_1 ConjCoord GN) de N_2)).

Elles permettent d'analyser les exemples suivants:

(a") *concentrateur de terminaux informatiques ou de modems*

(b") *concentrateur de terminaux informatiques ou modems*

(c") *concentrateur ou multiplexeur de terminaux informatiques.*

Autres cas de noms composés imbriqués

Présentation de la méthode de classification

Certaines formes figées n'appartiennent pas clairement à la catégorie des juxtapositions ou des recouvrements. Elles sont difficiles à classer pour plusieurs raisons:

- d'une part le domaine de langue ne nous était pas familier et nous ne pouvions pas décider du statut figé de certaines expressions ou, ce qui revient au même, de l'acceptabilité de certaines transformations,
- d'autre part certaines chaînes, qui sont d'une structure syntaxique normale de groupe nominal, sont ambiguës selon le rattachement choisi pour les modifieurs.

Afin de classer ces formes nous utilisons deux références:

- l'état du lexique: les formes figées qui y sont recensées,
- une grammaire du groupe nominal où les non terminaux sont entre crochets <> et où les terminaux sont des catégories lexicales:

```
(a) <GroupeNominal> ->          Nom <ModifieurDuNom>;
(b) <GroupeNominal> ->          Nom;
(c) <ModifieurDuNom> ->         Adjectif;
(d) <ModifieurDuNom> ->         Prép <GroupeNominal>;
(e) <ModifieurDuNom> ->         Nom;.
```

Les terminaux peuvent s'unifier sur des mots du lexique de même catégorie lexicale, qu'il s'agisse de mots simples ou de formes figées. Par exemple, si *code de calcul mécanique* est dans le lexique comme nom composé (s'il est présent et s'il a la catégorie lexicale `N` dans la description lexicale de la tête), il peut s'unifier avec `Nom` dans les règles `(a)`, `(b)` ou `(e)`.

Nous allons analyser *code de calcul mécanique* (une chaîne `N de N Adj`) à l'aide de cette grammaire. Au cours de l'analyse, les trois formes figées nominales recherchées dans le lexique sont *code de calcul, calcul mécanique* et *code de calcul mécanique*. En fonction de la présence ou l'absence de ces formes, la grammaire précédente nous permettra de décider de la structure de *code de calcul mécanique*.

Différentes analyses possibles en fonction du lexique

Une structure syntaxique libre

- (1) Une forme `N de N Adj`, analysable comme un groupe nominal à l'aide de la grammaire précédente en utilisant successivement les règles `(a)`, `(d)`, `(a)` et `(c)`. Cette analyse suppose qu'aucune des trois formes figées proposées n'est présente dans le lexique. Nous pouvons donc lui associer la représentation syntaxique de la figure 33.

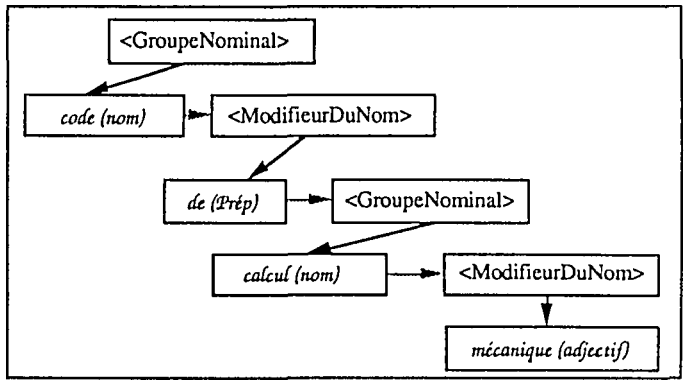


Figure 33: analyse de *code de calcul mécanique* lorsqu'aucune forme figée n'est présente dans le lexique

- (2) Une modification d'une forme figée `(N de N)` par un adjectif:
  (code de calcul) mécanique

obtenue en appliquant les règles `(a)` et `(c)`, avec, pour la règle `(a)`, `Nom` = *code de calcul*, ce qui suppose que *code de calcul* est une entrée du lexique.

- (3) Une modification d'un nom simple par un nom composé en `(N Adj)`:
  code de (calcul mécanique)

obtenue en appliquant les règles `(a)`, `(d)` et `(b)`, avec, pour la règle `(b)`, `Nom` = *calcul mécanique* (entrée du lexique).

- (4) Dans le cas où les deux noms composés *code de calcul* et *calcul mécanique* sont recensés dans le lexique, il s'agit une structure ambiguë analysable de deux façons différentes:
  - une modification d'un nom simple par un nom composé en `(N Adj)` (cas (3) avec *calcul mécanique* entrée du lexique),
  - une modification d'une forme figée en `(N de N)` par un adjectif (cas (2) avec *code de calcul* entrée du lexique).

Une forme figée obtenue par juxtaposition

- (5) Juxtaposition d'un nom simple et d'un nom composé en `(N Adj)`:
  (code de (calcul mécanique))

à condition que *code de calcul mécanique* et *calcul mécanique* soient des entrées du lexique.

- (6) Juxtaposition d'un nom composé en N de N et d'un nom simple:
((code de calcul) mécanique)

à condition que *code de calcul mécanique* et *code de calcul* soient des entrées du lexique.

Une forme figée obtenue par recouvrement

- (7) Recouvrement de deux formes figées en (N de N) et (N Adj)
((code de (calcul) mécanique))

à condition que le lexique contienne les trois noms composés *code de calcul, calcul mécanique* et *code de calcul mécanique.*

Une forme figée dont aucune sous structure figée n'est attestée

- (8) une forme figée de structure (N de N Adj)
(code de calcul mécanique)

code de calcul mécanique est seul présent dans le lexique.

Nous construisons le tableau des possibilités de représentation en fonction de l'appartenance des noms composés au lexique: *code de calcul, calcul mécanique* ou *code de calcul mécanique.* Pour simplifier, nous n'avons pas considéré qu'il pouvait y avoir recouvrement sur les mots initiaux, dans le cas où *code mécanique* serait dans le lexique. Dans ce cas, le travail réalisé ici devrait être enrichi de quelques formes supplémentaires.

Un signe + dans une colonne indique qu'une forme figée appartient au lexique. En fonction de ces appartenances, nous choisissons l'une des représentations classées de (1) à (8) selon le tableau de la figure 34.

Conclusion sur la cohérence du lexique

Quel que soit l'état de connaissance que l'on ait du domaine de langue à décrire, il faut que le lexique reste cohérent. Il peut être amené à évoluer en fonction de l'usage de la langue, certaines formes initialement considérées comme libres devenant suffisamment contraintes pour être considérées comme figées.

Inversement, une forme figée étant donnée, on peut produire son arbre syntaxique à l'aide de la grammaire précédente et en déduire une liste de noms composés à rechercher dans le lexique. Puis, en fonction des formes trouvées, il reste à choisir la représentation appropriée afin de préserver la cohérence.

Du point de vue informatique, la représentation de la juxtaposition (4) et celle du recouvrement (6) sont identiques, et c'est uniquement dans le cas de transformation de coordination et des transformations de catégorie 2 que les formes reconnues diffèrent.

Analyse obtenue *code de calcul mécanique*		**Formes figées éventuellement présentes dans le lexique** *code de calcul*	*calcul mécanique*	*code de calcul mécanique*
Forme libre	(1) *code de calcul mécanique*	-	-	-
	(2) *(code de calcul) mécanique*	+	-	-
	(3) *code de (calcul mécanique)*	-	+	-
	(4) *(code de (calcul) mécanique)*	+	+	-
Forme figée obtenue par juxtaposition	(5) *(code de (calcul mécanique))*	-	+	+
	(6) *((code de calcul) mécanique)*	+	-	+
Forme figée obtenue par recouvremen	(7) *((code de (calcul) mécanique))*	+	+	+
Forme figée simple	(8) *(code de calcul mécanique)*	-	-	+

Figure 34: représentation de *code de calcul mécanique* en fonction de la présence dans le lexique de *code de calcul, calcul mécanique* ou *code de calcul mécanique*

d Enrichissement des métarègles par les métatransformations

Au cours du paragraphe précédent, nous avons introduit deux transformations particulières: [antéposition] et [connexe] qui permettent de déduire de nouvelles transfonnations (à partir de transformations déjà existantes). Elles ont donc une fonction particulière, et nous nous les étudions plus précisément. Nous y ajoutons deux autres transformations qui sont à peu près du même type. L'une permet de croiser les transformations sur les noms composés imbriqués. L'autre étend les transformations définies sur des noms composés simples tels que N de N, à des structures telles que N de (N Adj).

Notion de métatransformation

Nous convenons d'appeler **métatransformation** les transformations opérant sur les métarègles et permettant d'en produire de nouvelles. Elles permettent de limiter le nombre de transformations à décrire.
Par exemple, la métatransformations [antéposition] permet de décrire l'ajout d'un modifieur adjectival antéposé à partir de la métarègle correspondant au modifieur postposé.

Pour donner quelques exemples, nous partons des métarègles définies sur N de N.

(1) $(N_1$ de $N_2)$ -> $((N_1$ $Adj_1)$ de $N_2)$
(2) $(N_1$ de $N_2)$ -> $(N_1$ de $(N_2$ $Adj_1))$
(3) $(N_1$ de $N_2)$ -> $(N_1$ (de N_2 ConjCoord de GN))
(3') $(N_1$ de $N_2)$ -> $(N_1$ de $(N_2$ ConjCoord GN))
(4) $(N_1$ de $N_2)$ -> $((N_1$ ConjCoord GN) de $N_2))$
(5) $(N_1$ de $N_2)$ $[Adj_1]$ -> $(N_1$ $Adj_1)$
(6) $(N_1$ de $N_2)$ -> $(N_1$ de Dét $N_2)$

Elles sont présentées sous une forme plus condensée que celle utilisée dans le système informatique. Nous n'avons fait figurer que la catégorie lexicale des mots.

- Nous avons noté par un symbole commençant par une majuscule tel que N_1 les variables de métarègle correspondant à un paradigme sur une catégorie lexicale (ici un nom). Elles sont notées sous une forme telle que !x{N, !a, !b, !c} dans le formalisme présenté au II.1.b.
- Les mots simples sont écrits en minuscules comme la préposition de.

Nous ne donnons pas le formalisme détaillée des métatransformations. Nous ne distinguons que les constantes et les variables.

- Les variables des métatransformations, dont la catégorie lexicale est connue, sont notées comme celles des métarègles, mais en les faisant précéder d'un point d'exclamation: $!N_1$ par exemple. Les variables dont la catégorie lexicale est inconnue, s'écrivent sous la forme !x.
- Les constantes sont des termes de métarègles qui peuvent être soit des variables de métarègle, soit des constantes de métarègle.

Métatransformation de composition: [antéposition] et [connexe]

Nous définissons une catégorie de métatransformations appelée **métatransformation de composition**. Elles servent à produire une nouvelle métarègle à partir d'une métarègle préexistante et s'appliquant déjà sur la structure considérée. Ainsi en est-il de [connexe] et de [antéposition], qui servent à déplacer un modifieur dans une forme figée.

Extension des métarègles à l'aide des deux métatransformations opérant sur les transformations de catégorie 1

La métatransformation **[connexe]** chasse une insertion hors d'une chaîne d'origine. Elle dépend de la structure sur laquelle elle s'applique. Nous la présentons pour N de N:

(Cxe) $(!N_1$ de $!N_2)$ -> $((!N_1$!Adj1) de $!N_2)$
=> $(!N_1$ de $!N_2)$ -> $((!N_1$ de $!N_2)$ $!Adj_1)$

Nous montrons, sur l'exemple de la métarègle (1) précédente, comment fonctionne (Cxe).

Les métatransformations sont des applications de l'ensemble des métarègles dans lui-même. A toute métarègle composée d'un membre de gauche et d'un membre de droite, elles associent une nouvelle métarègle définie de la même manière. Elles s'écrivent donc sous forme d'un couple de couples de termes:

T_1 -> T_2 => T_3 -> T_4

Le couple de gauche (T_1 -> T_2) doit s'unifier avec la métarègle d'origine et celui de droite (T_3 -> T_4) sert à produire la nouvelle métarègle. Le mécanisme opératoire est le même que celui des métarègles: il s'effectue en trois temps:

- unification du membre de gauche de la métatransformation avec la métarègle d'origine,
- propagation de l'unification entre les variables de métatransformation du membre de gauche vers celui de droite,
- détachement du membre de droite pour produire la nouvelle métarègle.

Examinons ces trois étapes dans le cas particulier de la métarègle

(1) $(N_1$ de $N_2)$ -> $((N_1$ $Adj_1)$ de $N_2)$

- l'unification du membre de gauche de (Cxe) avec (1) donne le système suivant:

$!N_1 = N_1$ $!N_2 = N_2$ $!Adj_1 = Adj_1$

- la propagation dans le membre de droite, puis le détachement de celui-ci, donnent une nouvelle métarègle de forme:

Cxe[(1)]=(7) $(N_1$ de $N_2)$ -> $((N_1$ de $N_2)$ $Adj_1)$

[connexe] est bien une métatransformation de composition car elle produit une nouvelle métarègle pour N de N à partir de la métarègle (1) déjà définie sur cette structure.

La métatransformation **[antéposition]** permute un nom et un adjectif.

(Antép) $!N_1$ -> ($!N_1$!Adj1)
=> $!N_1$ -> ($!Adj_1$ $!N_1$)

L'application de (Antép) échoue sur toutes les métarègles définies initialement sur N de N (de 1 à 6), mais elle réussit sur la nouvelle métarègle (7) que nous venons de produire en appliquant (Cxe) sur (1). L'unification est la suivante:

$!N_1 = (N_1$ de $N_2)$ $!Adj_1 = Adj_1$

Une variable de métatransformation telle que $!N_1$ ne s'unifie pas seulement avec les termes simples des métarègles mais aussi un syntagme nominal tel que (N_1 de N_2).
(Antép) et (Cxe) permettent d'ajouter la métarègle (8) ainsi définie:

Antép[Cxe[(1)]] = (8) (N_1 de N_2) -> (Adj_1 (N_1 de N_2)).

[antéposition] est également une métatransformation de composition car elle opère sur (7) qui vient d'être produite pour N de N.

Acceptabilité des nouvelles métarègles obtenues

Ces métatransformations de composition permettent de produire de nouvelles métarègles servant à décrire des transformations de noms composés. Ainsi (7) permet d'analyser *banque de données internationales* et (8), *principaux produits de fission*.

Il convient donc d'ajouter à la liste des transformations sur les noms composés, celles qui sont ainsi obtenues. L'acceptabilité de ces nouvelles métarègles (7) et (8) n'est pas uniquement liée au nom composé lui-même, mais aussi à la nature de la transformation. Ainsi, l'antéposition comme dans *traditionnels régulateurs à actions PID* dépend de la règle, puisqu'elle s'applique mieux sur des règles longues, mais aussi de la nature de l'adjectif puisqu'elle s'applique difficilement sur *électrique*. Il y a donc composition de deux acceptabilités:

- d'une part, l'acceptabilité au niveau de la règle telle que nous l'avions vue dans le cas des métarègles de base,
- d'autre part, l'acceptabilité liée à la nature des modifieurs introduits dans la règle par la métarègle *régulateurs à actions PID traditionnels* et sur laquelle opère la métatransformation.

Pour obtenir la métarègle (7) avec un modifieur postposé externe, on a cherché l'image de (1) qui ajoute un modifieur postposé interne par (Cxe). On pourrait relier l'acceptabilité de (7) à celle de (1).Mais de nombreux cas de formes figées nous indiquent que (7) peut être acceptable sans que (1) le soit comme pour * *tableau électronique de bord* et *tableau de bord électronique*.

[extension] des métarègles vers les structures supérieures

Sur un exemple, nous déduisons les métarègles sur la structure supérieure N de (N Adj) de celles sur la structure de base N de N (ces deux structures ont une partie d'arbre commune). Nous utilisons, pour cela, une métatransformation que nous appelons **[extension]**, qui pourrait également être définie pour le passage de N de N à N de (N de N) ou à (N Adj) de N...

[extension] produit une métarègle sur une structure supérieure à partir de celle définie sur la structure de base dont les arbres coïncident jusqu'à la profondeur 2.

(Exten) ($!N_1$ de $!N_2$) -> $!N_3$
=> ($!N_1$ de ($!N_2$ $!Adj_1$)) -> **subst**($!N_3$, $!N_2$, ($!N_2$ $!Adj_1$))

subst est une opération qui, à un triplet de termes associe un terme. Elle permet de remplacer $!N_2$ par ($!N_2$ $!Adj_1$) dans $!N_3$. Elle se définit aisément de façon récursive:

subst(!x, !x, !y) = !y
subst((!u !v), !x, !y) = (**subst**(!u, !x, !y) **subst**(!v, !x, !y))

La représentation, donnée pour [extension] au moyen de (Exten), n'est pas assez générale pour s'adapter à toutes les métarègles définies sur N de N. Dans la cas de transformations de catégorie 2 ou 3, le terme de droite $!N_3$ ne contient pas le terme $!N_2$. Nous ajoutons à l'expression ci-dessus une autre forme qui s'applique dans le cas de l'adjectivation (règle (5) de N de N):

(Exten) ($!N_1$ de $!N_2$) [!x] -> ($!N_1$!x)
=> ($!N_1$ de ($!N_2$ $!Adj_1$)) [!y] -> ($!N_1$!y)

Le terme !y reste à définir dans le cas où cette métarègle est acceptée pour qu'il soit compatible avec le (N_2 Adj_1). Cet ajout n'est pas suffisant car la forme initiale ne rend pas compte des diverses structures en profondeur du membre de droite. Nous nous en tiendrons à ces deux expressions pour ne pas alourdir l'exposé.

A l'aide de (Exten), nous produisons donc les neuf métarègles ci-dessous, candidates à être applicables sur une structure en N_1 de (N_2 Adj_1):

(9) = Exten[(1)] (N_1 de (N_2 Adj_1)) -> ((N_1 Adj_2) de (N_2 Adj_1))
(10) = Exten[(2)] (N_1 de (N_2 Adj_1)) -> (N_1 de ((N_2 Adj_1) Adj_2))
(11) = Exten[(3)] (N_1 de (N_2 Adj_1)) -> (N_1 (de (N_2 Adj_1) ConjCoord de GN))
(11') = Exten[(3')] (N_1 de (N_2 Adj_1)) -> (N_1 de (N_2 Adj_1) ConjCoord GN))
(12) = Exten[(4)] (N_1 de (N_2 Adj_1)) -> ((N_1 ConjCoord GN) de (N_2 Adj_1))
(13) = Exten[(5)] (N_1 de (N_2 Adj_1)) [Adj_2] -> (N_1 Adj_2)
(14) = Exten[(6)] (N_1 de (N_2 Adj_1)) -> (N_1 de Dét (N_2 Adj_1))
(15) = Exten[(7)] (N_1 de (N_2 Adj_1)) -> ((N_1 de (N_2 Adj_1)) Adj_2)
(16) = Exten[(8)] (N_1 de (N_2 Adj_1)) -> (Adj_2 (N_1 de (N_2 Adj_1)))

Nous donnons ci-dessous des transformations correspondant à chacune de ces métarègles:

(9) *applications de l'Intelligence Artificielle*
-> applications industrielles de l'Intelligence Artificielle

(10) *panneau de comptage électrique*
-> panneau de comptage électrique domestique

(11) *capteur de pression différentielle*
-> capteur de pression différentielle ou de mesure de déplacement

(11') *gestion d'espace disque*
-> gestion d'espace disque ou espace mémoire

(12) *conception de matériel informatique*
-> conception et fabrication de matériel informatique

(13) *capteur de pression différentielle*
-> capteur différentiel

(14) *calcul de conditions aux limites*
-> calcul des conditions aux limites

(15) *système d'équations différentielles*
-> système d'équations différentielles linéaire

(16) *Immeubles de Grande Hauteur*
-> futurs Immeubles de Grande Hauteur.

[croisement] des métarègles sur un nom composé imbriqué

Nous donnons les transformations possibles sur une structure N Adj telles qu'elles ont été recensées dans le tableau du II.1.a:

(17) $(N_1\ Adj_1) \rightarrow (N_1\ est\ Adj_1)$
(18) $(N_1\ Adj_1)\ [V_1\ N_2] \rightarrow (N_1\ V_1\ N_2)$
(19) $(N_1\ Adj_1) \rightarrow (N_1\ Adv\ Adj_1)$
(20) $(N_1\ Adj_1) \rightarrow (N_1\ (Adj_1\ ConjCoord\ Modif))$
(20') $(N_1\ Adj_1) \rightarrow ((N_1\ ConjCoord\ GN)\ Adj_1)$
(21) $(N_1\ Adj_1)\ [N_2] \rightarrow (N_1\ de\ N_2)$
(22) $(N_1\ Adj_1) \rightarrow (N_1\ non\ Adj_1)$
(23) $(N_1\ Adj_1) \rightarrow ((N_1\ Adj_2)\ Adj_1)$

(24) $(N_1\ Adj_1) \rightarrow ((N_1\ Adj_1)\ Adj_2)$ composition de (23) par [connexe]
(25) $(N_1\ Adj_1) \rightarrow (Adj_2\ (N_1\ Adj_1))$ composition de (24) par [antéposition]

Au II.1.c, nous avons détaillé les transformations de catégorie 1 sur une structure N_1 de $(N_2$ Adj) obtenue par juxtaposition d'un nom simple et d'un nom composé. Nous avons étudié les compositions de transformations entre la structure supérieure et la structure de base à l'aide d'une matrice de croisement. Nous avons remarqué qu'il y a une loi de transversalité

qui peut s'énoncer ainsi: un croisement d'une transformation T_1 sur une structure de base et T_2 sur une structure supérieure est vrai si et seulement si T_1 est vraie sur la structure de base seule et si T_2 est vraie sur la structure de base non modifiée.

Cette loi n'a pas été systématiquement explorée sur toutes les combinaisons possibles, puisque nous l'avons testée sur ce seul exemple. Nous avons remarqué qu'il y a plutôt affaiblissement de l'acceptabilité par passage à une structure plus large, donc l'application de la "loi de transversalité" tend à faire accepter plus de transformations que nécessaire. Les restrictions d'application que nous avons notées sont les suivantes:

- elle est vraie, aux compatibilités sémantiques près.

- Si T_2 ne conserve pas la structure de base, il se peut que le croisement des transformations ne puisse plus se faire, T_1 ne pouvant plus s'appliquer sur une structure modifiée. Plus précisément, si la structure de base est supprimée ou remplacée par un nom simple, le problème du croisement des transformations ne se pose plus. Si elle est transformée en une autre forme figée, il convient d'étudier le croisement de T_2 avec les transformations possibles sur cette nouvelle structure.

- Certaines transformations peuvent être redondantes, et, croiser T_1 avec T_2, reviendrait à composer une de ces deux transformations avec elle-même. L'ajout du modifieur sur la structure de base est traité dans les transformations de catégorie 1 sur la structure de base (voir la ligne "inutile" de la figure 29).

- Dans le cas où la structure supérieure est un recouvrement, certaines transformations telles que la coordination, ne peuvent être déduites uniquement à partir des croisements des coordinations de la structure supérieure et de la deuxième structure de base, mais nécessitent de considérer les transformations sur la première structure de base qui n'apparaît pas dans la règle.

Mise en oeuvre du croisement des transformations:

La composition de deux transformations peut être envisagée de deux manières:
- **statique:** les métarègles composées sont produites avant l'analyse,
- **dynamique:** toutes les métarègles sont appliquées en cours d'analyse sur la structure de base, puis sur la structure supérieure qui a fait appel à celle-ci.

Afin de ne pas faire une description exhaustive des transformations croisées acceptées par la structure imbriquée, nous avons supposé que la loi de transversalité s'applique automatiquement. La mise en oeuvre dynamique du croisement nous permet d'implanter automatiquement cette loi.

Exemple de croisement de transformation:

Il n'est pas possible d'étudier ici tous les croisements de toutes les transformations pouvant opérer sur les noms composés formés à l'aide d'un ou plusieurs noms composés de base. Nous allons donner un exemple sur le nom composé *générateur de (système expert)* de structure N_1 de (N_2 Adj_1). Pour cette structure, il y a 9x10 = 90 croisements potentiels entre les transformations de N_1 de (N_2 Adj_1) et de (N_2 Adj_1). Sur la figure 35, nous avons représenté le croisement des transformations correspondant à ce cas. Ce chiffre est approximatif car

- il prend en compte des croisements qui peuvent être impossibles si (N_2 Adj_1) n'est pas conservée dans la forme supérieure modifiée ou des croisements redondants,
- il omet les croisements des formes modifiées avec les formes initiales.

Nous étudions le cas particulier du croisement de:

- la transformation (9) sur la structure supérieure:

(9) (N_1 de (N_2 Adj_1)) -> ((N_1 Adj_2) de (N_2 Adj_1))

- et de l'ajout d'un modifieur sur le premier N dans une structure N Adj, composée avec la transformation [connexe] qui le rejette hors du nom composé:

(24) (N_2 Adj_1) -> ((N_2 Adj_1) Adj_3).

Ce croisement produit par composition, la nouvelle transformation:

(26) (N_1 de (N_2 Adj_1)) -> ((N_1 Adj_2) de ((N_2 Adj_1) Adj_3))

qui permet de reconnaître une modification du nom composé initial telle que:

générateur industriel de système expert financier.

Mécanisme de croisement des métarègles

Sur cet exemple, nous détaillons le mécanisme de croisement des métarègles. Il s'agit d'une opération interne notée [croisement] qui, à deux métarègles opérant sur deux structures dont l'une est incluse dans l'autre, associe une troisième métarègle sur la structure supérieure. Nous reprenons les notations utilisées pour les métatransformations. Le membre de gauche est constitué de couples de termes devant s'unifier avec les deux métarègles initiales. Le membre de droite est formé d'un couple de termes et produira la nouvelle métarègle.

(Croise) (!N_1 de (!N_2 !Adj_1)) -> !N_3
(!N_2 !Adj_1) -> !N_4
=> (!N_1 de (!N_2 !Adj_1)) -> **subst**(!N_3, (!N_2 !Adj_1), !N_4)

(9) (N_1 de (N_2 Adj_1)) -> ((N_1 Adj_2) de (N_2 Adj_1))
(24) (N_2 Adj_1) -> ((N_2 Adj_1) Adj_3)

L'unification du membre de gauche avec les deux métarègles (8) et (22), données ci-dessus, produit le système suivant:

!N_1 = N_1 !N_2 = N_2 !Adj_1 = Adj_1
!N_3 = ((N_1 Adj_2) de (N_2 Adj_1)) !N_4 = ((N_2 Adj_1) Adj_3)

Ce qui permet d'appliquer l'opération de substitution:

subst(!N_3, (!N_2 !Adj_1), !N_4)
= **subst**(((N_1 Adj_2) de (N_2 Adj_1)), (N_2 Adj_1), ((N_2 Adj_1) Adj_3))
= ((N_1 Adj_2) de ((N_2 Adj_1) Adj_3))

et de produire la nouvelle métarègle:

(26) (N_1 de (N_2 Adj_1)) -> ((N_1 Adj_2) de ((N_2 Adj_1) Adj_3))

Composition avec la transformation identique

Afin de parcourir toutes les possibilités de compositions de métarègles, il convient de rajouter à chacune des listes de métarègles la **transformation identique** qui, à toute structure, associe elle-même. Nous considérons que, d'une part, quelle que soit la structure figée, elle accepte la transformation identique, et d'autre part, que toute règle proposée en analyse est toujours composée par une métarègle qui est éventuellement cette identité. Ce qui donne pour N_1 de (N_2 Adj_1) (10 + 1) x (9 + 1) - 1 = 109 croisements potentiels.

Transformations sur N2 Adj1 / Transformations sur N1 de (N2 Adj1)	Transformation identique	10 transformations sur N2 Adj1
Transformation identique	Transformation identique	10 automatismes: extensions des transformations de N2 Adj1 à N1 de (N2 Adj1)
9 transformations sur N1 de (N2 Adj1)	9 nouvelles transformations : modifications de N1 de (N2 Adj1) quand N2 Adj1 est sous sa forme initiale	90 croisements des transformations de N2 Adj1 avec celles de N1 de (N2 Adj1)

Figure 35: croisement des transformations sur une structure en N_1 de (N_2 Adj_1)

- Le croisement de la transformation identique de N_1 de $(N_2\ Adj_1)$ avec les transformations de $(N_2\ Adj_1)$ permet d'étendre les transformations de $(N_2\ Adj_1)$ à N_1 de $(N_2\ Adj_1)$ (voir les **automatismes** définis au II.1.c dans les catégories 1, 2 ou 3).

- Le croisement de la transformation identique de $(N_2\ Adj_1)$ avec les transformations de N_1 de $(N_2\ Adj_1)$ permet de décrire les transformations de N_1 de $(N_2\ Adj_1)$ lorsque $(N_2\ Adj_1)$ est sous sa forme initiale (voir les **nouvelles transformations** définies au II.1.c dans les catégories 1, 2 ou 3).

Le croisement des transformations sur une structure de base avec celles sur une structure supérieure, en incluant la transformation identique, permet de produire tous les cas de transformations de la structure supérieure comme le montre le tableau de la figure 35.

Croisement des métarègles dans le système informatique

Dans notre système informatique, le croisement des métarègles est réalisé par la production dynamique des règles au moyen des métarègles. Toute règle analysée est proposée en trois étapes: la règle initiale, la règle transformée pa toutes les métarègles acceptées et la règle ayant subi des élisions s'il y en a.

Etudions, sur un cas particulier, le fonctionnement de la composition dynamique des métarègles. *générateur de système expert* est décrit par les deux règles ci dessous:

```
<générateur de système expert> -> {N, !1, !1}
   générat{NA, m, v, n} DE{P, v, v, v, n} <système expert>{N, m, v};
<système expert> -> {N, !1, !1}
   système{N, m, v, n} expert{NA, a1, a1, n};
```

A <générateur de système expert> nous appliquons toutes les transformations d'une structure N_1 de $(N_2\ Adj_1)$, déduites de celles de la structure de base N_1 de N_2 comme nous l'avons vu et qui sont autorisées sur cette forme particulière.

- Soit ces transformations conservent <système expert>, et, dans ce cas, nous appliquons à cette structure de base toutes les transformations autorisées, comme si elle était seule.

- Soit elles altèrent <système expert> (catégorie 2 ou 3).

S'il est transformé en un nom simple ou s'il est supprimé, il n'y a pas de problème de croisement de transformations.

S'il est transformé en une autre structure admettant des transformations, tous les croisements possibles des transformations de cette nouvelle structure avec la transformation courante de la structure supérieure seront produits. Ceci revient à ajouter au tableau de croisement des transformations une ligne externe qui représente l'acceptabilité de toutes les transformations de la nouvelle forme de base, pour la structure supérieure modifiée.

e Structuration du domaine de langue en trois niveaux

Nous rappelons les outils que nous avons utilisés pour définir les transformations que peuvent subir les noms composés:

- les métarègles qui opèrent sur les structures de base pour produire de nouvelles règles permettant de reconnaître les noms composés élémentaires sous une forme modifiée,

- les métatransformations de composition [antéposition] et [connexe] qui permettent de produire de nouvelles métarègles sur une structure de base, à partir de celles qui sont déjà définies sur celles-ci,

- la métatransformation [extension] qui déduit les nouvelles transformations applicables sur une structure supérieure à partir de celles existant sur une structure de base,

- l'opération [croisement] qui construit les transformations sur une structure supérieure par croisement entre les transformations déjà définies sur celle-ci et les transformations sur la structure de base.

Figure 36, nous présentons un schéma de la hiérarchie des niveaux de langue où interviennent ces différents outils.

- Au niveau de la description lexicale, se trouvent les règles d'énonciation des noms composés et les transformations de ces règles à l'aide de métarègles acceptées.

- Au niveau syntaxique, les métarègles, qui décrivent les transformations syntaxiques que peuvent subir les règles, opèrent sur le niveau lexical des règles. Les métarègles permettent de produire une nouvelle règle à partir de la règle d'origine.

- Au niveau supérieur se trouvent deux sortes de transformations sur les métarègles:

les métatransformations qui permettent de produire une nouvelle métarègle à partir d'une autre,

les opérateurs comme le croisement des métarègles dans un nom composé en (N de (N Adj)) qui permettent de produire de nouvelles métarègles à partir de plusieurs.

Afin de ne pas appliquer systématiquement les métarègles, dans l'énoncé de chaque règle du lexique, il est précisé la liste des transformations du niveau syntaxique qu'elles acceptent.

Afin de ne pas surcharger la présentation, nous avons omis les compositions de transformations sur une même structure, vues au II.1.b, qui augmenteraient le nombre des transformations définies sur une structure en le faisant passer de n à 2^n.

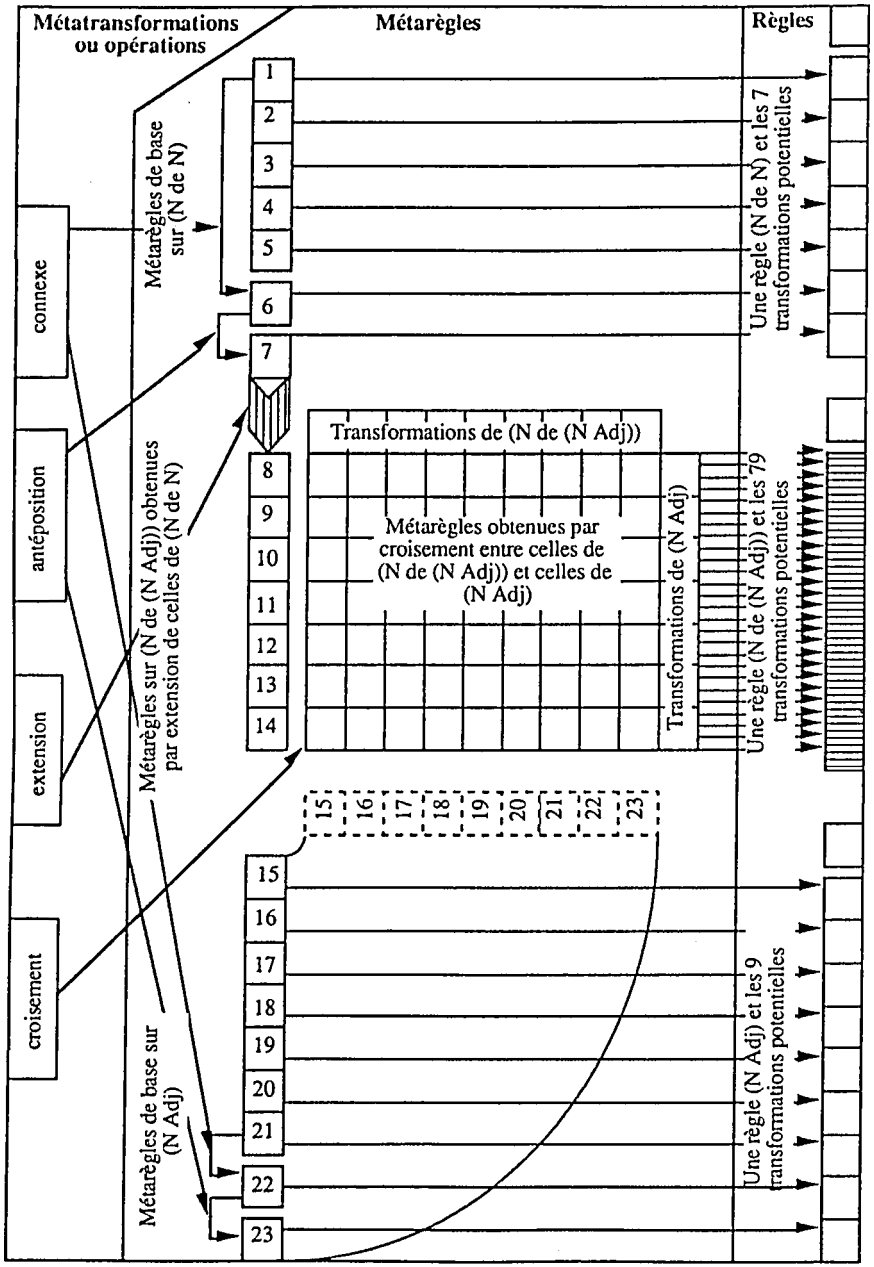


Figure 36: hiérarchie des niveaux de langue pour la description des transformations de (N de (N Adj))

2 ANALYSE DES NOMS COMPOSÉS TRANSFORMÉS

Dans ce chapitre, nous mettons en oeuvre l'analyse automatique des noms composés ayant subi des transformations. Nous reprenons les outils informatiques de reconnaissance morphologique développés dans la première partie (I), et les résultats de l'observation linguistique des transformations de noms composés établis dans le chapitre précédent (II.1).

Pour automatiser l'analyse des transformations, nous utilisons un mécanisme de métarègles. Elles transforment les arbres syntaxiques des règles afin de produire les arbres des règles transformées (II.2.a et b). Il s'agit d'une production dynamique, elle suit la loi de transversalité vue au II.1.d. Nous présentons également le mécanisme des élisions qui permet d'analyser la référence (II.2.c).

a Formalisme des métarègles

Dans ce paragraphe, nous reprenons le formalisme pour l'expression des transformations des noms composés, tel que nous l'avons présenté pour les structures `N de N` `N à N` et `N Adj`, au II.1.b.

Le terme de gauche de la métarègle doit s'unifier avec le terme de droite de la règle du nom composé courant. A cet effet, nous avons regroupé les métarègles par structures identiques de noms composés. Les contraintes sur l'unification sont suffisamment faibles, pour que toutes les règles ayant la même structure syntaxique puissent convenir. Le transfert d'informations, entre le terme de la règle initiale et le terme de la nouvelle règle, se fait au moyen de variables notées `!a`, `!b`, `!c`,... Elles permettent de reporter un identificateur ou une caractéristique lexico-syntaxique, d'un terme de la règle de base vers la règle transformée.

Pour certaines métarègles, telles que celles qui permettent de passer d'une forme adjectivale à une forme nominale, on a besoin de transmettre un paramètre à la métarègle. Cette notion a été présentée au II.1.b.

Exemple de métarègle: une métarègle de nominalisation

Nous décrivons, dans la figure 37, la métarègle N°2 du tableau `N Adj`, qui permet de passer d'une forme N_1 `Adj` à une forme N_2 `de` N_1. N_2 est une nominalisation de `Adj` passée en paramètre. Par exemple, à partir de *consommation spécifique*, on produit *la spécificité de consommation*, ainsi que toutes les flexions possibles telles que *les spécificités de consommation*...

En annexe II.2, nous donnons la forme extérieure des métarègles et un exemple de quelques métarègles.

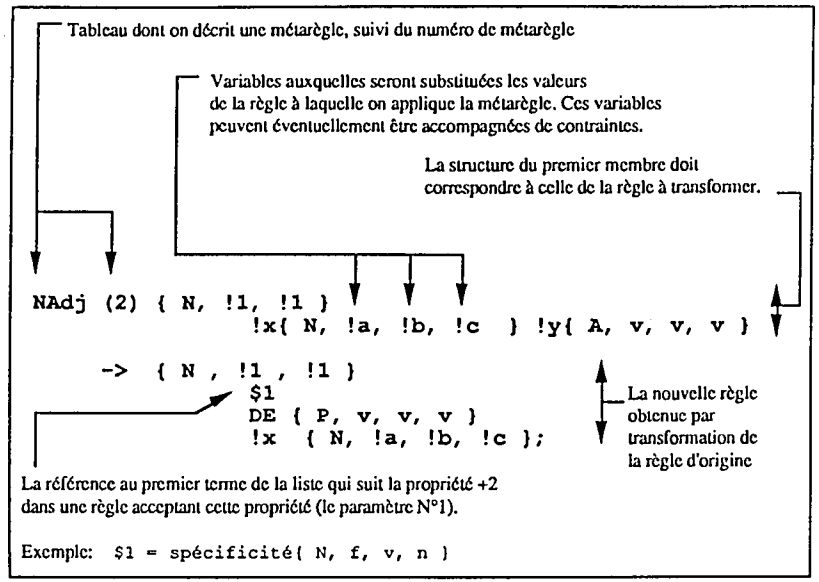


Figure 37: métarègle N°2 pour les structures N Adj: nominalisation en N de N

Ecriture des termes d'une métarègle

Nous présentons, aux figures 38 et 39, un récapitulatif des possibilités d'écriture des identificateurs et des caractéristiques lexico-syntaxiques dans une métarègle. Nous séparons celles d'un terme de gauche de métarègle (figure 38) de celles d'un terme de droite (figure 39).

0 ou 1 contrainte	Ecriture	0 ou 1 variable	Ecriture
- absence de contrainte	v ou rien	- pas d'unification avec le terme de droite	rien
- contrainte d'égalité sur la caractéristique lexico-syntaxique avec la terme de la règle	une caractéristique lexico-syntaxique (voir le tableau équivalent pour les règles)	- la valeur sera reprise dans le terme de droite	!a, !b... (reprise dans le terme de droite)

Les deux possibilités peuvent être simultanées

Figure 38: écriture d'un identificateur ou d'une caractéristique lexico-syntaxique dans le membre de gauche d'une métarègle

1 valeur	Ecriture	1 variable de métarègle	Ecriture
- la valeur est donnée littéralement sans référence à la partie gauche ou à la règle d'origine	une caractéristique lexico-syntaxique (voir le tableau précédent)	- unification avec un terme de gauche	!a, !b ... (existant dans le terme de droite)

Une et une seule des deux possibilités

Figure 39: écriture d'un identificateur ou d'une caractéristique lexico-syntaxique dans le membre de droite d'une métarègle

b Production des règles au moyen de métarègles

La **compilation d'une métarègle** comporte principalement une analyse syntaxique de deux termes: le terme de la règle d'origine (le terme de gauche) et le terme de la nouvelle règle (à droite). Cette analyse est similaire à celle que nous avons vue pour les règles, en II.2.c. On obtient, après compilation, deux arbres: l'arbre qui doit s'unifier avec la règle initiale proposée, et l'arbre de la nouvelle règle qui doit être complété en unifiant les variables.

- Les variables de métarègle sont unifiées avec la valeur trouvée dans le terme d'origine.
- Les paramètres de métarègle sont unifiées avec la valeur fournie dans l'énoncé de la règle.

La **production d'une nouvelle règle par une métarègle**, à partir d'une règle de base, se fait en trois étapes:

- une unification entre le terme de queue de la règle d'origine et le terme de gauche de la métarègle,
- une propagation de l'unification des variables du terme de gauche vers le terme de droite de la métarègle,
- une recopie du terme de droite de la métarègle pour constituer la queue de la nouvelle règle en ayant soin de recopier les valeurs des variables obtenues par l'unification de la première étape.

Ce mécanisme est similaire au *read, eval, print* du langage Lisp, il s'agit d'une application de fonction.

Etude d'un exemple: *consommation spécifique -> spécificité de consommation*

Règle initiale

```
<consommation spécifique> ->
        {N, !1, !1} consommation{N, f, v, n} spécifique{A, a1, a1, n}
        [() () (N Adj -1 +2(spécificité{ N, f, v, n }) -3 -4 -5 -6 +7) ];
```

L'arbre syntaxique de cette règle est donné à la figure 40.

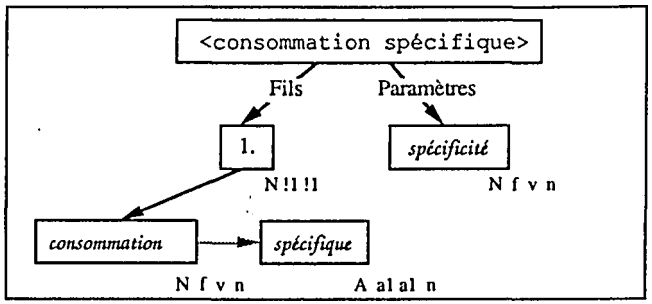


Figure 40: arbre de la règle <consommation spécifique>

Métarègle

```
N Adj (2) (N, !1, !1) !x(N, !a, !b, !c ) !y(A, v, v, v) ->
    (N, !1, !1) $1 DE(P, v, v, v) !x(N, !a, !b, !c);
```

Les deux arbres syntaxiques de cette métarègle sont donnés à la figure 41.

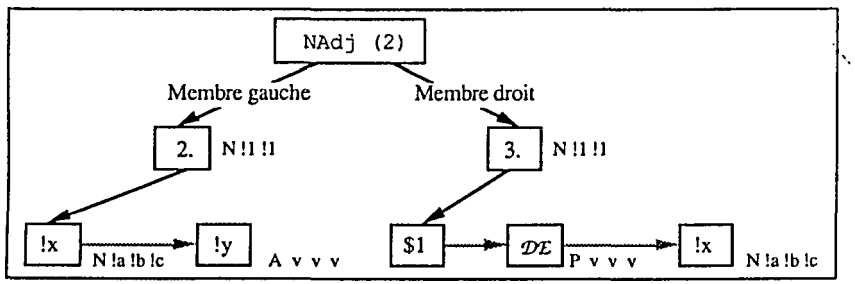


Figure 41: arbre de la métarègle N°2 de la structure N Adj

La nouvelle règle obtenue en remplaçant les variables par leurs valeurs et en ne gardant que le membre de droite de la métarègle

```
-> (N, !1, !1) spécificité(N, f, v, n) DE(P, v, v, v)
               consommation(N, f, v, n);
```

L'arbre syntaxique de cette nouvelle règle est donné à la figure 42.

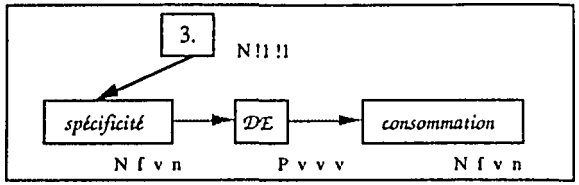


Figure 42: arbre de la nouvelle règle obtenue par application de la métarègle N°2 de la structure N Adj à la règle <consommation spécifique>

Le mécanisme d'unification, lors de l'application d'une métarègle

Il est décrit, par la figure 43, donnée ci-dessous.

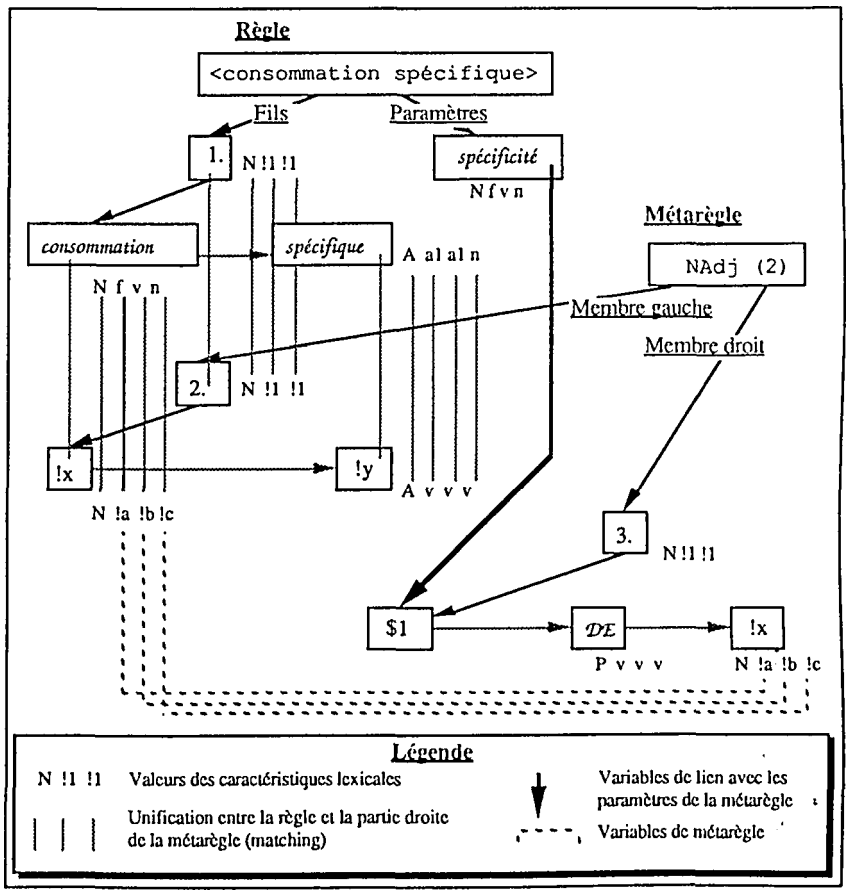


Figure 43: le mécanisme d'unification permettant la production de la nouvelle règle

Ecriture en pseudo-langage de la production d'une nouvelle règle au moyen d'une métarègle

```
Procédure ProduireParMétarègle( var règle; noMétaRègle )
Début
     métarègle = RechercheMétarègle( noMétaRègle,
                                     règle.identificateurTableau );
     (* elles sont compilées à l'ouverture de la session *)
     (* sauvegarde du membre de gauche de la métarègle *)
     (* ses valeurs seront modifiées durant l'unification *)
     arbreAux = RecopieArbre( métarègle.membreDeGauche );

     (* vérification de l'unification entre la règle et le membre de gauche
        de la métarègle *)
     Si( UnificationMéta( règle.fils, métarègle.membreDeGauche ) )
     Alors
         règle.transformée
             = RecopieTransformée( métarègle.membreDeDroite )
     Fin Si

     (* remise en état de la métarègle *)
     métarègle.membreDeGauche = RecopieArbre( arbreAux );
Fin
```

La structure de données des termes de la métarègle est légèrement différente de celle des termes des règles: elle comporte, en supplément, des couples formés d'un nom de variable et d'un pointeur comme, par exemple:

- un nom de paramètre de métarègles et un pointeur vers le terme donné dans la règle,
- un nom de variable de métarègle pour le genre et un pointeur vers le genre d'un terme du membre de gauche, etc...

Ces couples permettent de gérer les différentes possibilités de variables proposées dans l'énonciation des métarègles. Les pointeurs sont garnis au moment de la compilation en reliant les variables de métarègles ayant le même identificateur.

La procédure `UnificationMéta` vérifie que les arbres de la règle et du membre de gauche de la métarègle ont bien la même structure, et que leurs valeurs sont compatibles. Sans détailler, la compatibilité se résume ainsi: soit une des deux variables est libre, soit les deux variables ont des valeurs et dans ce cas elles sont égales. Dans le cas favorable où cette unification réussit, les valeurs du terme de la règle sont recopiées dans celle du terme de la métarègle.

La procédure `RecopieTransformée` permet de recopier l'arbre formé par le membre de droite de la métarègle. Pour chaque terme, on recopie les valeurs, champs par champs, en allant chercher les valeurs des pointeurs lorsqu'il s'agit d'une variable de métarègle, ou en recopiant le terme en entier, s'il s'agit d'un paramètre de métarègle donné dans la règle.

La procédure `RechercheMétarègle` permet de retrouver une métarègle qui a été compilée au début de la session et stockée en mémoire dans un tableau, et `RecopieArbre` est la duplication d'une arborescence en mémoire.

c Production des règles au moyen d'élisions

Afin de reconnaître les noms composés, dans des cas de référence où certains mots sont élidés, il est nécessaire de mettre en place un mécanisme spécifique qui s'apparente à-celui des métarègles.

L'élision ne peut s'appliquer qu'à des règles, dont le terme de queue est une concaténation de plus d'un terme. Il s'agit de la possibilité d'omission de certains termes (qui sont éventuellement des listes ou des concaténations) parmi les fils de la queue de règle, sans modification d'ordre, avec rétablissement des accords sur les nouveaux indices.

Exemple:

```
<ligne de communication spécialisée>
        ->  (N, !1, !1) ligne(N, f, v, n) DE(P, v, v, n)
                        communication(N, f, v, n) spécialis(V, al, al, n)
            (() ( 1-4 ) ())];
```

Elle admet comme élision la sous-chaîne composée uniquement des termes d'indices 1 et 4. Elle permet donc de produire la nouvelle règle suivante.

```
<ligne de communication spécialisée>
    -> (N, !1, !1) ligne(N, f, v, n) spéciales(V, al, al, n);
```

La figure 44 présente le mécanisme de l'élision sur cet exemple.

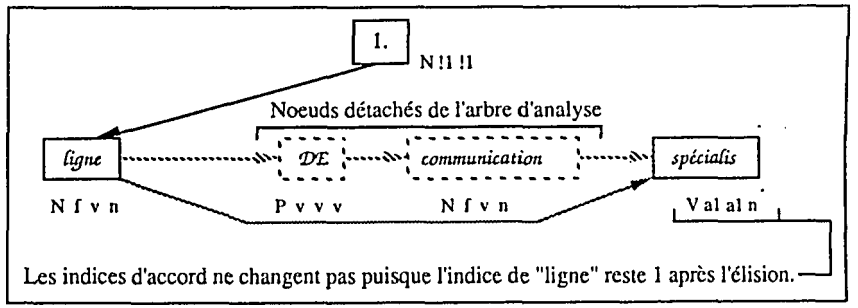


Figure 44: arbre de la règle obtenue par élision des termes 2 et 3 de `<ligne de communication spécialisée>`

Ecriture en pseudo-langage de la production d'une nouvelle règle au moyen d'une élision

```
Procédure ProduireParElision(var règle ; listeNumérosTermesElidés)
Début
    (* on recopie la règle initiale dans la transformée *)
    règle.transformée = RecopieArbre( règle.fils );
    décalage = 0;
    Pour tous les fils de la racine
    Faire
        Si indiceDuFils est un indice de terme à conserver
        Alors
            tableauDécalage[ indiceDuFils ] = décalage;
            réajuster les accords éventuels du fils courant;
        Sinon
            tableauDécalage[ indiceDuFils ] = -1;
            décalage = décalage + 1;
            supprimer le fils dans le chaînage et le désallouer;
        Fin Si
    Fin Pour
    réajuster les substitutions éventuelles des caractéristiques lexico-
        syntaxiques de la racine;
Fin
```


CONCLUSION

Remarques techniques sur l'application informatique

Les performances concernant l'application sur la complexité ont été données, en détail, au paragraphe I.3. Pour l'application linguistique, étudions les différentes possibilités offertes à l'utilisateur.

- La **compilation des règles** est une procédure longue, d'autant plus longue que le lexique est volumineux. L'évaluation ne peut être que statistique car les règles rattachées à des mots, en fin de liste alphabétique, sont insérées beaucoup plus rapidement que celles qui sont au début. L'ordre de grandeur de la vitesse est d'une règle par minute environ. Il ne peut s'agir que d'une procédure de traitement différé, il n'y a pas d'interruption en cas d'erreur. Une compilation sans insertion, très rapide, peut être effectuée avant de lancer la compilation proprement dite, afin de vérifier la syntaxe.

- **L'analyse lexicale** des textes est une procédure beaucoup plus optimisée, dont le temps dépend peu de la taille du lexique en raison des différents mécanismes d'accès aux données mis en place (B-Arbre, Hash Code, bufferisation des lectures sur disque...). Pour un lexique allant jusqu'à 10 000 mots simples et 50 000 noms composés, l'ordre de grandeur de la vitesse de l'analyse est de 2 pages à la minute. Elle pourrait diminuer pour un lexique de taille supérieure, puisqu'alors le B-Arbre ne permettrait pas d'accéder directement à tous les mots du lexique, mais nécessiterait un parcours séquentiel. Une autre solution consisterait à agrandir la taille du B-Arbre, en augmentant la capacité mémoire de la machine.

- L'**édition de liens** est une opération rapide, de l'ordre d'un millier de règles par minute.

- La **production** est une opération dont la complexité dépend essentiellement des règles proposées et du nombre de chaînes potentielles. Dans le cas de règles simples, la production est très rapide, dans le cas de règles couvrant un grand ensemble de formes, il serait intéressant de prévoir un coroutinage de l'unification.

Les ordres de grandeur donnés pour l'analyseur sont donnés sur PC AT, cadencé à 10 MHz, avec un temps d'accès disque de l'ordre de 50 ms.

Choix du langage

Dans le cadre du développement qui accompagne notre travail de recherche, plusieurs raisons nous ont fait choisir un langage de bas niveau (langage C)

- Il est nécessaire de prendre en compte l'aspect volumineux des données linguistiques et de ne pas négliger tous les problèmes qui surgissent sur un lexique de 100 000 mots et qui sont passés sous silence dans des traitements de données de l'ordre d'un millier de mots: évaluation de la complexité des traitements tels que la lexicalisation, optimisation des accès aux données en mémoire, stockage des lexiques sur disque et optimisation des accès aux disques.

- L'utilisation d'un matériel modeste (PC ou MacIntosh) pour ce type d'application nécessite une comptabilité précise de la mémoire utilisée, et ne peut supporter un langage avec un interpréteur volumineux ou avec un compilateur produisant un code trop grand.

Nous pensons que, dans le cadre d'une application linguistique, une modélisation ne peut suffire pour prouver la validité des méthodes et des représentations mises en jeu. C'est pourquoi, l'utilisation d'un langage de bas niveau a été inévitable, même si nous avions évalué le surcoût en développement que cela demandait.

Applications pratiques de ce développement informatique

L'annexe 4 donne un exemple de session d'analyse pouvant être réalisée avec ce développement informatique.

La première application qui vient à l'esprit pour un analyseur de structures lexico-syntaxiques locales, est évidemment **l'indexation automatique** des textes en vue de la constitution de thesaurus. La phase d'analyse est alors suivie de l'utilisation d'une base de données qui permet de faire le lien entre les formes reconnues, dans un texte, par l'analyseur, et les termes du thesaurus. Un travail de ce type est effectué à la Direction des Etudes et Recherche de l'EDF, et a déjà prouvé son intérêt.

D'autres réalisations, que nous n'avons pas abordées, tireraient également un grand profit de l'utilisation d'une grammaire telle que nous l'avons définie: la traduction automatique ou assistée par ordinateur, l'interfaçage en langage naturel,... [Pierrel 90] montre que la connaissance des expressions figées est utile en compréhension automatique de la parole.

En conclusion, comme l'a toujours prétendu l'équipe du L.A.D.L., la constitution de dictionnaires électroniques est une phase préalable à toute activité sur le langage naturel. Il s'agit là d'un travail gigantesque et irremplaçable qui nécessite une description volumineuse et détaillée, donc coûteuse en temps humain. Les noms composés comme les constructions des verbes, les conjonctions de subordination ou les compléments adverbiaux qui sont déjà réalisés ou en cours de l'être n'échappent pas à cette règle.

La figure 45 présente une vue globale du système informatique, et la figure 46 les différents fichiers utilisés.

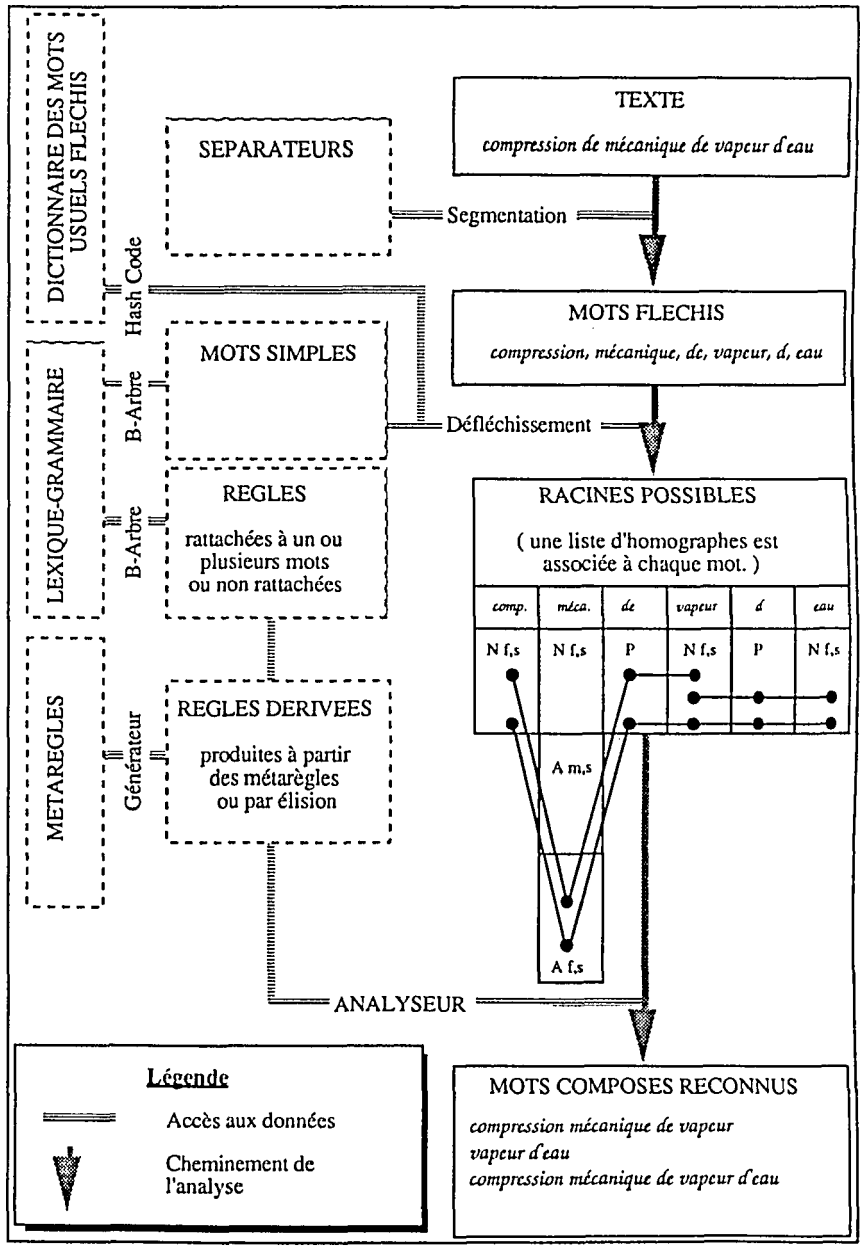


Figure 45: schéma de l'analyseur

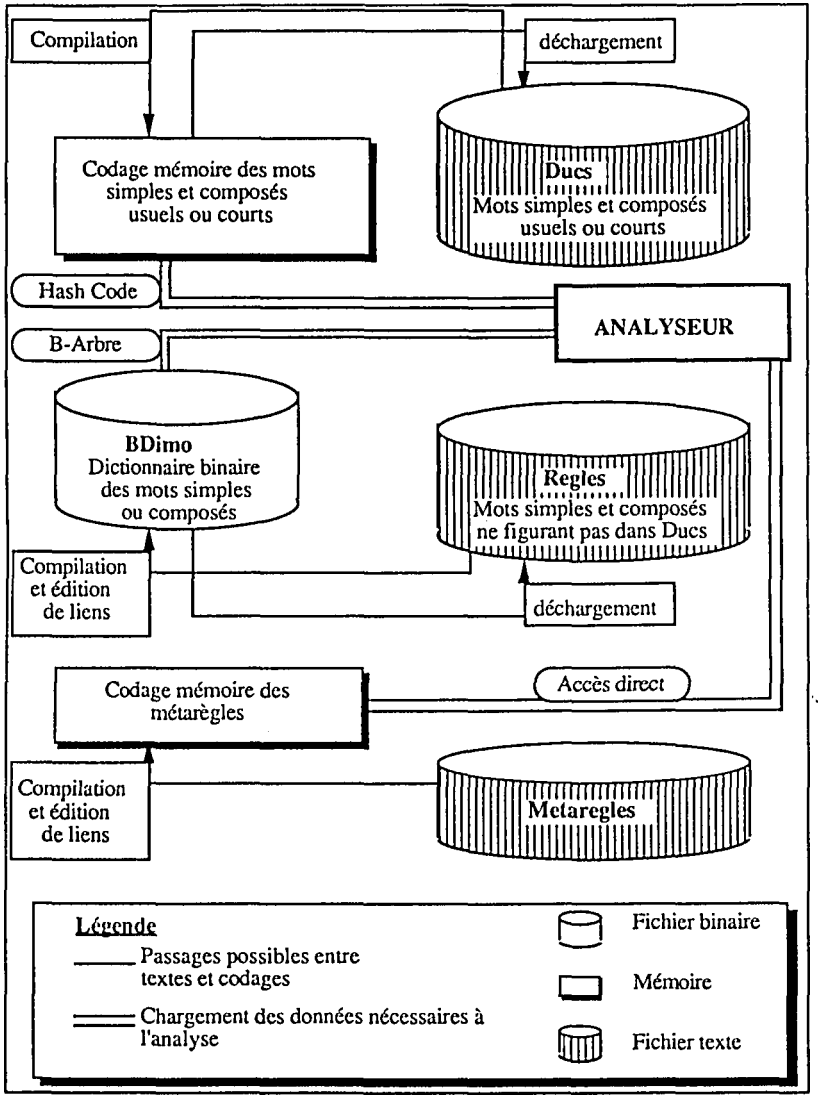


Figure 46: dictionnaires, fichiers binaires et mémoire

Bilan et perspectives

L'étude menée ici sur les noms composés, peut s'appliquer à d'autres structures syntaxiques locales telles que les expressions verbales, les conjonctions de subordination composées, les déterminants composés..., chaque fois qu'un assemblage de mots produit une structure suffisamment stable pour pouvoir être considérée comme une nouvelle entrée lexicale.

Cette étude ne concerne qu'une lucarne dans la description de la langue, est dépendante d'une description plus générale (construction à noms prédicatifs et verbes supports, coordination, relativation, pronominalisation...). Le point de jonction se trouve au niveau des métarègles.

Le parallèle entre la réalisation de l'analyseur / générateur et la description du formalisme d'énonciation, nous a permis de valider les choix pris (et de revenir sur certains éventuellement). Le formalisme dégagé doit permettre aux linguistes de décrire avec le plus de précision possible les noms composés de la langue française et leurs transformations éventuelles. Un travail de recensement est déjà bien avancé au niveau du L.A.D.L.

Toute application automatisée sur le traitement des langues ne pourra se faire sérieusement qu'en ayant connaissance de toutes les structures composées figées ou semi-figées, que ce soit pour la traduction, l'indexation ou la compréhension et la génération en vue d'interfaces en langage naturel. Les ignorer, c'est risquer de commettre de lourds contresens. Pour s'en convaincre, il suffit de lire des résultats de traduction automatique réalisées sur des systèmes dont les connaissances linguistiques sont insuffisantes.

Les trois facettes de ce travail: linguistique (tel que l'étude des transformations possibles sur les noms composés), mathématique (complexité du rattachement et recherche d'heuristiques), informatique (description de structures de données et d'algorithmes adaptés à le représentation et à l'analyse) sont trois aspects indissociables de l'analyse automatique de la langue...

La linguistique nous assure l'adéquation entre les résultats formels et les phénomènes de langue observable, les outils mathématiques nous permettent de formaliser et d'évaluer les algorithmes mis en oeuvre et l'informatique nous permet de faire la synthèse et de produire un système opérationnel.

Le système informatique que nous avons réalisé appartient aux systèmes d'intelligence artificielle classique (langages objets, programmation en logique, langage Lisp...). Il permet de rendre compte fidèlement des phénomènes linguistiques observés et de réaliser des outils d'analyse qui relèvent les noms composés sous leur forme initiale et transformée. La programmation en logique a fourni un cadre de représentation bien adapté à la finalité de cette étude, nous n'avons pas eu à le remettre en cause.

Un tel système aurait avantage à être complété d'un analyseur qui reconnaisse des structures proches de celles énoncées, et récupérer les cas non explicitement prévus. Pour cela, il est nécessaire d'élaborer un mécanisme assez souple pour identifier une information dégradée ou partielle. Avec un système d'intelligence artificielle classique, la seule façon de faire est d'essayer de façon exhaustive toutes les règles présentes dans le système, ce qui est rédhibitoire en raison de l'explosion combinatoire de la complexité.

Une réponse à ce problème peut être donnée par les réseaux connexionixtes qui sont des systèmes robustes, de complexité raisonnable et qui sont bien adaptés à la reconnaissance de motifs dégradés. [Rumelhart McClelland 86] présente un panorama des divers modèles de réseaux; ils ont été repris dans de nombreux travaux ultérieurs.
Ces réseaux, massivement parallèles sont constitués d'unités interconnectées réagissant selon des mécanismes similaires. On distingue ceux à information localisée où une unité est désignée pour représenter une partie de l'information (généralement un symbole) et ceux à information délocalisée où l'information est répartie sur plusieurs unités, chacune coopérant à sa représentation. Cette distinction est décrite dans [Béroule 89] qui sépare représentation codée (localisée) et représentation topographique (délocalisée). Il préconise une représentation mixte pour un système connexioniste de traitement de la langue.
Nous élaborons actuellement un réseau, qui doit permettre de reconnaître les noms composés sous des formes modifiées non prévues. Ce modèle s'accorde avec la notion de **diffusion de l'activation**, telles qu'elle est développée dans [Le Ny 89]. Il s'agit d'un réseau à information délocalisée, avec des neurones à fonction de seuil ayant une activité décroissante dans le temps. Ce travail est encore expérimental et nous ne pouvons en donner une validation précise, mais les premiers résultats sont encourageants.

Nous pensons poursuivre le travail dans l'observation de la langue et sa description au moyen de règles formelles, ainsi que dans la mise au point d'outils complémentaires pour récupérer les cas d'échec.

III ANNEXES

1 FORME EXTÉRIEURE DES RÈGLES, EXEMPLES

L'énonciation des noms composés doit se faire selon une grammaire Context Free dont les règles d'écriture sont données ci-dessous. Elles sont décrites un formalisme BNF où

```
a -> b c d;
```

signifie le terme **a** peut se récrire en **b c d**. Les termes notés entre guillemets doivent figurer littéralement, les autres représentent des termes figurant au moins une fois comme membre droit (ce sont des non terminaux).
Les commentaires sur ces règles sont écrits sur les lignes commençant par **!**.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! les règles des mots simples
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Règle ->      MotSimple "->" ";";
MotSimple -> Chaîne "(" CatLex "," Genre "," Nombre "," Forme ")";
MotSimple -> Chaîne "(" CatLex "," Genre "," Nombre "," Forme ")"
             FormesCanoniquesAnnexes;
! Certains mots tels que recevoir ont plusieurs formes canoniques énoncées ici:
FormesCanoniquesAnnexes -> "[" ListeChaînes "]";
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! les règles des noms composés
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Règle ->      TermeTête "->" Terme ";";
Règle ->      TermeTête "->" Terme  "[" "(" Abréviation ")"
              "(" ListeElisions ")"
              "(" IdentificateurTableau ListePropriétés ")" "]" ";";
TermeTête -> "<" IdentificateurListe ">";
TermeTête -> "<" IdentificateurMotComposé ">";
! Un terme est un terme simple (mot), ou un terme composé (appel d'une autre
! règle), ou une liste (choix entre plusieurs termes de même catégorie
! lexicale), ou une concaténation (la suite de plusieurs termes):
Terme ->      TermeSimple;
Terme ->      TermeComposé;
Terme ->      Liste;
Terme ->      Concaténation;
TermeComposé ->  "<" IdentificateurListe ">" "(" AccordCatLex ","
                     AccordGenre "," AccordNombre ")";
TermeComposé ->  "<" IdentificateurListe ">" "(" AccordCatLex ","
                     AccordGenre "," AccordNombre "," AccordForme ")";
TermeComposé ->  "<" IdentificateurMotComposé ">" "(" AccordCatLex
                     "," AccordGenre "," AccordNombre ")";
TermeSimple ->   Chaîne "(" AccordCatLex "," AccordGenre ","
                        AccordNombre "," AccordForme ")";
TermeSimple ->   "<>" "(" AccordCatLex "," AccordGenre ","
                        AccordNombre "," AccordForme ")";
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Liste et concaténation
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Une concaténation est formée d'une description des caractéristiques lexico-
! syntaxiques de la tête de la concaténation, et d'une suite de termes:
Concaténation ->     TêteConcaténation SuiteConcaténation;
SuiteConcaténation -> Terme SuiteConcaténation;
SuiteConcaténation -> Terme;
```

```
TêteConcaténation -> "{" AccordCatLex SubstConc "," AccordGenre SubstConc ","
                        AccordNombre SubstConc "}";
! Une liste est formée d'une description des caractéristiques lexico-
! syntaxiques de la tête de liste, suivie d'une suite de termes:
Liste ->     "(" TêteListe SuiteListe ")";
Suite Liste ->   Terme "+" SuiteListe;
Suite Liste ->   Terme;
TêteListe ->     "{" AccordCatLex SubstListe "," AccordGenre
                    SubstListe "," AccordNombre SubstListe "}";
TêteListe ->     "{" AccordCatLex SubstListe "," AccordGenre
                    SubstListe "," AccordGenre SubstListe ","
                    AccordForme SubstListe "}";
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Le bloc de fin de règle
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Pour chaque nom composé, on note son abréviation si elle existe:
Abréviation ->       Vide;
Abréviation ->       Chaîne;
! On décrit éventuellement deux types de métarègles:
! 1: les élisions permettant d'obtenir une règle plus courte à partir d'une
! règle de base qui doit être une concaténation dans ce cas:
ListeElisions ->     Vide;
ListeElisions ->     Nombre;
ListeElisions ->     Nombre "-" ListeElisions;
! 2: les propriétés qui font référence à des métarègles attachées à une
! structure lexicale et décrites dans un fichier séparé. Ces métarègles
! peuvent faire appel à des paramètres qui sont énoncés à la suite de la
! propriété:
ListePropriétés -> UnePropriété;
ListePropriétés -> UnePropriété ListePropriétés;
UnePropriété ->  Signe Nombre;
UnePropriété ->  Signe Nombre "(" ListeTermesSimplesOuComposés ")";
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!           Caractéristiques lexico-syntaxiques
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Les caractéristiques lexico-syntaxiques d'un terme simple ou composé:
AccordCatLex ->  Accord CatLex | CatLex | Accord;
AccordGenre ->   Accord Genre | Genre | Accord;
AccordNombre ->  Accord Nombre | Nombre | Accord;
AccordForme ->   Accord Forme | Forme | Accord;
! Accord avec un autre terme dans une concaténation:
Accord ->        "a" Nombre;
! 2 modes de substitution
! 1: sur une tête de concaténation, s'il y a substitution, on doit préciser
! quel est le terme avec lequel la substitution a lieu:
SubstConc ->     "!" Nombre;
! 2: sur une tête de liste, la substitution se fait avec le terme de la liste
! qui est retenu:
SubstListe ->    "!";
! Constantes des caractéristiques lexico-syntaxiques:
CatLex ->        "v" | "N" | "A" | "V" |...;
Genre ->         "v" | "m" | "f" ;
Nombre ->        "v" | "s" | "p" ;
Forme ->         UneForme;
Forme ->         UneForme Forme;
UneForme ->      "v" | "n" | "t" | "a" | "m";
```



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                    quelques règles auxiliaires
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Pour la lisibilité et la sécurité de saisie, nous avons choisi de distinguer
! l'identificateur d'une liste de celui d'une concaténation:
IdentificateurMotComposé -> Identificateur;
IdentificateurListe -> "LISTE" Identificateur;
Identificateur -> CaractAlpha SuiteAlphanumerique;
! Les formes canoniques annexes d'une unité lexicale:
ListeChaînes ->  Chaîne;
ListeChaînes ->  Chaîne "," ListeChaînes;
! Les mots ou les identificateurs pouvant être substitués dans une métarègle:
ListeTermesSimplesOuComposés -> TermeSimple;
ListeTermesSimplesOuComposés -> TermeComposé;
ListeTermesSimplesOuComposés -> TermeSimple "," ListeTermesSimplesOuComposés;
ListeTermesSimplesOuComposés -> TermeComposé "," ListeTermesSimplesOuComposés;
! Les tableaux de description des propriétés pouvant être
! attachées à une structure syntaxique de nom composé:
IdentificateurTableau -> "NAdj" | "NdeN" | "NaN" | "NN"...;
Signe ->         "+" | "-" | "?";
Nombre ->        Chiffre;
Nombre ->        Chiffre Nombre;
SuiteAlphanum -> CaractAlphaNum;
SuiteAlphanum -> CaractAlphaNum SuiteAlphanumerique;
Chiffre ->       "0" | ... | "9";
CaractAlpha ->   " " | "a" | ... | "z" | "A" | ... | "Z" |
                 "à" | "é" | "è" | "ê" | "î" | "ï" | "ô" | "ù" | "û" | "ü";
CaractAlphaNum ->    CaractAlpha | Chiffre;
```

Exemples de Règles

```
"*****************************************************************************
          MOTS SIMPLES
*****************************************************************************
agro{A, 1, v, n}->;                  agriculture{N, 1, f, n}->;
alimentaire{NA, 1, m, n}->;          air{N, 1, m, n}->;
alcool{N, 1, m, n}->;                algorithme{N, 1, m, n}->;
*****************************************************************************
    N ADJ SIMPLES
*****************************************************************************"
<accident nucléaire>
    -> {N, !1, !1} accident{N, m, v, n} nucléaire{NA, a1, a1, n}
        [() () ( NAdj -1 -2 -3 +4 -5 -6 +7 )];
<air ambiant>
    -> {N, !1, !1} air{N, m, s, n} ambiant{A, m, s, n}
        [() () ( NAdj -1 -2 -3 -4 -5 -6 +7 )];
<automate programmable>
    -> {N, !1, !1} automate{N, m, v, n} programmable{A, a1, a1, n}
        [() () ( NAdj +1 -2 -3 -4 -5 +6 +7 )];
<azote ammoniacal>
    -> {N, !1, !1} azote{N, m, s, n} ammoniaca{A, a1, a1, n}
        [() () ( NAdj +1 -2 -3 +4 +5(ammoniaque{N, m, s, n}) +6 +7 )];
<boucle expérimentale>
    -> {N, !1, !1} boucle{N, f, v, n} expérimenta{A, a1, a1, n}
        [() () ( NAdj +1 -2 +3 +4 +5(expérience{N, f, s, n}) -6 +7 )];
```

2 FORME EXTÉRIEURE DES MÉTARÈGLES, EXEMPLES

Nous réutilisons le même formalisme que celui pour la grammaire des règles, donné au 10.2. Nous ne répétons pas les termes qui y sont déjà définis.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                 les règles des noms composés
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MétaRègle ->      IdentificateurTableau "(" Nombre ")" TermeOrigine
                  "->" TermeNouveau ";";
! Pour TermeOrigine et TermeNouveau, on retrouve les mêmes règles de grammaire
! que celles qui avaient été énoncées pour les règles de noms composés.
! Seuls deux changements sont à noter: la description des mots ou des
! identificateurs et la description des caractéristiques lexico-syntaxiques
! des termes.
! Nous les indiquons ci-dessous, en séparant la description pour une règle
! d'origine de celle pour une nouvelle règle:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                    terme de nouvelle règle
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TermeComposéOrigine ->
             "<" IdentificateurListeOrigine ">"
             "{" AccordCatLexOrigine "," AccordGenreOrigine ","
                 AccordNombreOrigine "}";
TermeComposéOrigine ->
             "<" IdentificateurListeOrigine ">"
             "{" AccordCatLexOrigine "," AccordGenreOrigine ","
                 AccordNombreOrigine "," AccordFormeOrigine "}";
TermeComposéOrigine ->
             "<" IdentificateurMotComposéOrigine ">"
             "{" AccordCatLexOrigine "," AccordGenreOrigine ","
                 AccordNombreOrigine "}";
TermeSimpleOrigine ->
             ChaîneOrigine
             "{" AccordCatLexOrigine "," AccordGenreOrigine ","
                 AccordNombreOrigine "," AccordFormeOrigine "}";
TermeSimpleOrigine ->
             "<>"
             "{" AccordCatLexOrigine "," AccordGenreOrigine ","
                 AccordNombreOrigine "," AccordFormeOrigine "}";
! On a 4 possibilités pour un identificateur ou une chaîne de la
! règle d'origine:
IdentificateurListeOrigine -> VariableMétaRègle IdentificateurListe;
IdentificateurListeOrigine -> IdentificateurListe;
IdentificateurListeOrigine -> VariableMétaRègle;
IdentificateurListeOrigine -> Vide;
! C'est la même chose pour IdentificateurMotComposéOrigine et ChaîneOrigine.
VariableMétaRègle ->      "!" CaractAlpha;
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                caractéristiques lexico-syntaxiques règle origine
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! La référence reprise dans la nouvelle règle suivie d'une valeur attendue:
AccordCatLexOrigine ->    VariableMétaRègle AccordCatLex;
! La valeur attendue:
AccordCatLexOrigine ->    AccordCatLex;
! La référence reprise dans la nouvelle règle sans valeur attendue:
AccordCatLexOrigine ->    VariableMétaRègle;
```



```
! Il n'y aucune contrainte sur la valeur et cette caractéristique ne pourra
! pas être reprise en référence dans la nouvelle règle
AccordCatLexOrigine ->    Vide;
! Même chose pour AccordGenreOrigine AccordNombreOrigine et AccordFormeOrigine
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                    terme de nouvelle règle
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! La description est la même, sauf que l'on remplace TermeSimple par,
! TermeSimpleNouveau et TermeComposé par TermeComposéNouveau sachant que
! chacun de ces identificateurs peut être remplacé par une variable $1, $2...
! qui fait référence au ième terme de la liste qui suit la propriété dans la
! règle.
! On a 4 possibilités pour un terme composé ou une possibilité pour un terme
! simple repris dans la grammaire du 10.3. avec en plus:
TermeSimpleNouveau ->     VariableMétaRègle;
TermeComposéNouveau ->    VariableMétaRègle;
! Nombre est l'indice du terme dans la liste des paramètres de la métarègle:
VariableRègle -> "$" Nombre;
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!          caractéristiques lexico-syntaxiques nouvelle règle
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! On remplace également AccordCatLexOrigine par AccordCatLexNouveau,
! AccordGenreOrigine par AccordGenreNouveau,
! AccordNombreOrigine par AccordNombreNouveau et
! AccordFormeOrigine par AccordFormeNouveau puisque la possibilité d'une
! variable et d'une valeur ne peuvent plus être simultanées.
! Une caractéristique lexico-syntaxique est donnée, soit par sa valeur,
AccordCatLexNouveau -> AccordCatLex;
! soit par une valeur reprise du terme d'origine:
AccordCatLexNouveau -> VariableMétaRègle;
! idem pour AccordGenreNouveau, AccordNombreNouveau et AccordFormeNouveau
```

Exemples de Métarègles

```
"***********************************************************************
         EXPRESSION DES METAREGLES DE N Adj
***********************************************************************"
NAdj (1) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1} !x{N, !a, !b, !c} être{V, v, a1, v} !y{A, !d, !e, !f};
NAdj (2) {N, !1, !1} !x{N, !a, !b, !c } !y{A, v, v, v}
     -> {N, !1, !1} $1 DE{P, v, v, v} !x{N, !a, !b, !c};
NAdj (3) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1} !x{N, !a, !b, !c} <>{Av, v, v, v} !y{A, !d, !e, !f};
NAdj (4) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1}
          !x{N, !a, !b, !c} !y{A, !d, !e, !f} <>{C, v, v, v} <>{A, !d, !e, v};
NAdj (4) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1}
          !x{N, !a, !b, !c} <>{C, v, v, v} <>{N, v, v, v} !y{A, v, p, !f};
NAdj (5) {N, !1, !1} !x{N, !a, !b, !c} !y{A, v, v, v}
     -> {N, !1, !1} !x{N, !a, !b, !c} DE{P, v, v, v} $1;
NAdj (5) {N, !1, !1} !x{N, !a, !b, !c} !y{A, v, v, v}
     -> {N, !1, !1} !x{N, !a, !b, !c} DE{P, v, v, v} <>{Dd, a4, a4, v} $1;
NAdj (6) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1} !x{N, !a, !b, !c} non{Ne, v, v, v} !y{A, !d, !e, !f};
NAdj (7) {N, !1, !1} !x{N, !a, !b, !c} !y{A, !d, !e, !f}
     -> {N, !1, !1} !x{N, !a, !b, !c} <>{A, a1, a1, v} !y{A, !d, !e, !f};
```

3 EXEMPLE D'ANALYSE ET DE SYNTHÈSE

Règles utilisées (pour les métarègles de N_Adj se reporter à l'annexe précédente).

```
ammoniaca{ A, 7, v, n } ->;
ammoniaque{ N, 1, m, n } ->;
ammonium{ N, 1, m, n } ->;
azote{ N, 1, m, n } ->;
liquide{ NA, 1, m, n } ->;

<azote ammoniacal>
    -> { N, !1, !1 } azote{ N, m, s, n } ammoniaca{ A, a1, a1, n }
        [() ()
    ( NAdj +1 -2 -3 +4 +5(<LISTE ammoniaque>{ N, v, s, n }) +6 +7 )];
<LISTE ammoniaque>
    -> ({ N, !, !, ! } ammoniaque{ N, m, v, n }
            + ammonium{ N, m, v, n } );
```

Texte proposé

```
azote d'ammoniaque.
azote d'ammonium.
azotes d'ammoniaque.
```

Mots reconnus

```
azote d'ammoniaque.
azote d'ammonium.
```

Production de la règle `LISTE ammoniaque`

```
ammoniaques
ammoniaque
ammoniums
ammonium
```

Production de la règle `azote ammoniacal`

```
azote ammoniacal
azote est ammoniacal.
azote sera ammoniacal.
azote était ammoniacal.
azote soit ammoniacal.
azote serait ammoniacal.
azote ammoniacal C A
azote DE ammoniaque
azote DE ammonium
azote d ammoniaque
azote d ammonium
azote de ammoniaque
azote de ammonium
azote non ammoniacal
azote A ammoniacal
```


4 TESTS DE LEXICALISATION

Les tests

Les différents tests correspondent à des simulations des trois variantes de l'algorithme approché; sur des modèles de lexiques distincts. Ces lexiques sont créés aléatoirement, avec des variations sur la composition des mots composés et sur la répartition des mots composés sur les mots simples. Ils sont présentés sur la figure 47.

Lexique	Alpha	Bêta	Alpha N10	Bêta N10
Nombre de lettres	1000	100	1000	100
Nombre de couples	4000			
Répartition des lettres sur les ensembles	Uniforme		Loi Normale centrée sur la lettre 500 de coefficient s = 10 (20 pour N20...)	
Taille des couples	2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10			
Fréquences des tailles des couples	33% / 33% / 14% / 5% / 5% / 4% / 3% / 2% / 1%			
Seuil minimal	15	141	14	141

Figure 47: Les ensembles de données sur lesquels sont effectués les tests présentés figure 46

Les résultats

Le tableau de la figure 48 présente les résultats de l'application des variantes 1, 2 et 3 de l'algorithme approché sur les différents modèles de lexique proposés. A chaque application, nous donnons les quatre valeurs suivantes:

- dans la première colonne, nous indiquons le seuil minimal réalisable en répartissant le plus uniformément possible les couples sur les lettres, puis, en dessous, nous redonnons la valeur de ce seuil, calculée après l'étape 1, au cours de laquelle sont rattachés tous les couples sur les lettres non surchargées.
- Dans la deuxième colonne, nous inscrivons les résultats définitifs de l'algorithme: seuil de rattachement obtenu après incrémentation éventuelle du seuil initial, si celui-ci ne convenait pas, et le temps d'exécution. Le temps correspond à un programme écrit en langage C, et fonctionnant sur MacIntosh.

Les répartitions Alpha et Bêta sont des répartitions uniformes des lettres sur les ensembles des couples, ce qui explique que le seuil obtenu soit proche du seuil minimum. A l'opposé les répartitions Alpha N et Bêta N sont des répartitions suivant une loi normale, d'autant plus regroupées sur peu de lettres que le coefficient s est faible. Pour ces données, de nombreux couples ont une charge globale inférieure au seuil, ils sont rattachés dans l'étape 1. Il reste ensuite peu de couples, mais de charge globale élevée, ce qui explique que les seuils obtenus en fin de calcul soient plus hauts.

Le seuil final obtenu est, dans tous les cas, à comparer à la valeur du seuil calculé après les rattachements sur les lettres non surchargées. Les résultats sont en accord avec la propriété de seuil maximal établie au I.3.b, dans le cas où la répartition n'est pas trop mauvaise (s'il est possible de prendre, à chaque étape, tous les couples qui ne contiennent plus que la lettre courante comme lettre non rattachée).

Nom du lexique	Algorithme variante 1		Algorithme variante 2		Algorithme variante 3	
	Seuil min. possible avant et après rattachement sur lettres non surchargées	Seuil obtenu et temps d'exécution de la variante 1	Seuil min. possible avant et après rattachement sur lettres non surchargées	Seuil obtenu et temps d'exécution de la variante 2	Seuil min. possible avant et après rattachement sur lettres non surchargées	Seuil obtenu et temps d'exécution de la variante 3
Alpha	15 15	16 218 s	15 15	17 366 s	15 15	18 290 s
Alpha N10	14 466	467 302 s	14 466	467 322 s	14 466	467 304 s
Alpha N20	15 237	238 300 s	15 237	238 320 s	15 230	231 301 s
Alpha N40	14 117	118 287 s	14 117	118 307 s	14 119	121 440 s
Bêta	143 143	143 17 s	143 143	143 15 s	141 141	142 30 s
Bêta N10	141 466	468 47 s	141 466	467 30 s	140 462	463 31 s
Bêta N20	141 260	261 25 s	141 260	261 25 s	142 259	261 46 s
Bêta N40	141 143	144 17 s	141 143	144 17 s	142 148	149 29 s

Figure 48: Les résultats des trois variantes, décrites au I.2.g, testées sur les ensembles de données présentés figure 47

5 ANALYSEUR EN PSEUDO-LANGAGE

Nous utiliserons deux variables globales dont la valeur est sauvegardée dans l'environnement de tout terme réécrit:

```
DébutConnu: Booléen
IndiceDuMotCourant: Entier
```

Ces deux variables permettent de savoir, lorsque le terme courant est une unité lexicale, si le premier mot de la structure à analyser est identifié. Dans ce cas, la valeur de DébutConnu est Vrai. On note, alors, dans IndiceDuMotCourant, l'indice du premier mot de la séquence, et les mots sont pris séquentiellement à partir de celui-ci. Si le premier mot n'est pas encore connu, on recherche à partir du début de la phrase le premier mot qui peut s'unifier avec le terme courant.

Procédure auxiliaire ajoutant les pointeurs vers les termes T_2, T_3, T_4...

Cette procédure gère le nombre de choix (ou de concaténation) et affecte à l'indice du premier choix (ou concaténation) une valeur nulle. On chaîne le bloc courant avec le début du suivant.

```
Procédure AjouterTermesSuivants( var backTrackSuccès; var backTrackEchec;
                                 var blocCourant )
Début
    backTrackSuccès := Faux;
    backTrackEchec := Faux;
    Selon Type( blocCourant )
    Si _TermeRègle:
        relever la règle de même identificateur que le terme
        recopier dans T2 le pointeur vers la règle relevée en mémoire
        blocCourant.numéroChoix = 0;
        blocCourant.nombreChoix  = 1 + nombreMétaRègles + nombreElisions;
        blocCourant.blocSuivant = HautPile + Taille( blocCourant )
                                  + Taille( pointeur );
    Si ( _RègleNormale Ou _RègleMeta Ou _RègleElision ):
        blocCourant.numéroChoix = 0;
        blocCourant.nombreChoix = 1;
        blocCourant.blocSuivant = HautPile + Taille( blocCourant );
    Si ( _Concaténation Ou _Liste):
        Pour tous les fils de la liste ou de la concaténation
        Faire
            recopier le pointeur vers le fils suivant de T1 à la suite du
                bloc
        Fin  Pour
        blocCourant.numéroChoix = 0;
        blocCourant.nombreChoix = nombreDeFils;
        blocCourant.blocSuivant =  HautPile + Taille( blocCourant )
                                   + nombreDeFils * Taille( pointeur );
    Si _UnitéLexicale:
        Si le mot fléchi est trouvé dans la phrase
        Alors
            Pour tous les homographes trouvés pour ce mot
            Faire
                recopier l'homographe à la suite du bloc
            Fin  Pour
            blocCourant.numéroChoix = 0;
            blocCourant.nombreChoix = nombreDHomographes;
            blocCourant.blocSuivant = HautPile + Taille(blocCourant)
                        + nombreDHomographes * Taille( pointeur );
        Sinon
```

```
            backTrackEchec = Vrai;
        Fin  Si
    Si _MotLu:
        blocCourant.numéroChoix = 1;
        blocCourant.nombreChoix = 1;
        blocCourant.blocSuivant = HautPile + Taille( blocCourant );
        backTrackSuccès = Vrai;
    Fin  Selon
Fin AjouterTermesSuivants;
```

Procédure auxiliaire permettant la sauvegarde de l'environnement

Cette procédure permet de remettre les blocs dans l'état initial lors du dépilement du retour en arrière.

```
Procédure SauverEnvironnement ( var blocCourant; blocPère )
Début
    Selon Type( blocPère )
    Si _TermeRègle:
            (* au retour en arrière, T2 sert à stocker la métarègle suivante
               et à désallouer la règle courante *)
    Si ( _RègleNormale Ou _RègleMeta Ou _RègleElision ):
            (* lien automatique avec le bloc suivant *)
    Si ( _Concaténation Ou _Liste Ou _UnitéLexcicale):
        sauvegarder le pointeur vers le terme Ti+1 du bloc père, sachant que
            i est l'indice du choix ou de la concaténation du bloc père;
    Fin  Selon
    sauvegarder les valeurs des deux variables globales DébutConnu
        et IndiceDuMotCourant;
Fin SauverEnvironnement;
```

La restauration de l'environnement RestaurerEnvironnement est l'opération symétrique de la sauvegarde.

Procédure de réécriture

Cette procédure est non récursive, elle permet de descendre dans l'arbre d'analyse, d'afficher des résultats trouvés et d'effectuer le retour en arrière.

```
Procédure Réécriture
Début
    (* boucle infinie *)
    Tant  Que vrai
    Faire
        gestion du chaînage vers le bloc précédent;
        SauverEnvironnement ( blocCourant, blocPère );
        Selon Type( blocPère )
        Si _TermeRègle:
            (* Le pointeur vers la règle d'origine est dans le T2 du bloc
               père *)
            incrémenter l'indice de choix du père;
            Si l'indice de choix du père > 1
            Alors
                faire pointer la transformée de règle du père vers la
                    nouvelle règle produite par élision ou métarègle
                le type du bloc courant est _RègleElision ou _RègleMeta
            Sinon
```



```
                le type du bloc courant est _RègleNormale
            Fin  Sinon
            recopier la tête de règle dans T1;
        Si ( _RègleNormale Ou _RègleMeta Ou _RègleElision ):
            incrémentation de l'indice de choix du père: passage de 0 à 1;
            recopier la queue de règle dans T1
            (* son adresse se trouve dans la règle du T1 du bloc père *)
            Le type du bloc courant est le type de T1;
        Si ( _Concaténation Ou _Liste Ou _UnitéLexicale ):
            incrémenter l'indice de choix ou de concaténation du père;
            recopier le ième terme du choix ou de la concaténation ou le ième
                homographe dans T1;
            (* i: indice du choix ou de la concaténation *)
            faire pointer le terme Ti+1 du père vers le terme T1 du bloc
                courant;
            le type du bloc courant est le type de T1;
        Fin  Selon
        AjouterTermesSuivants( backTrackSuccès, backTrackEchec,
                               blocCourant );
        (* si le retour en arrière n'est pas demandé, on chaîne les blocs *)
        Si ( Non backTrackSuccès Et  Non backTrackEchec )
        Alors
            (* les termes à fils multiples tels que liste, concaténation ou
               unité lexicale, introduisent leur ième fils, i > 1, lors d'un
               retour en arrière sur succès ou sur échec *)
            le bloc père est le bloc précédent, et le haut de la pile est la
               fin du bloc courant
        Fin  Si
        Si backTrackSuccès
        Alors
            Si il n'y a pas de concaténation incomplète dans la pile
            Alors
                Si Unification()
                Alors
                    Afficher la séquence reconnue
                Sinon
                    Afficher la séquence non reconnue en précisant la raison
                        de l'échec
                Fin  Si

                (* on force le retour en arrière car on est sur un succès *)
                backTrackEchec = Vrai;
            Sinon
                (*il y a une concaténation incomplète dans la pile*)
                La concaténation incomplète devient le père du bloc suivant
            Fin  Si
        Fin Si  (* backTrackSuccès *)
        Si backTrackEchec
        Alors
            Si il n'y a pas de choix incomplet dans la pile
            Alors
                (* sortie de la boucle infinie *)
                Sortie  Fin de l'analyse;
            Sinon   (* il y a un choix incomplet dans la pile *)
```

```
                on dépile jusqu'à ce choix incomplet en restaurant
                    l'environnement à chaque étape;
                Selon Type( blocPère )
                    Si ( _Liste Ou UnitéLexicale ):
                        le bloc courant devient le père du bloc suivant
                    Si ( _TermeRègle ):
                        on désalloue la métarègle dont le pointeur est dans
                                le T2 du bloc courant
                        le bloc courant devient le père du bloc suivant
                Fin  Selon
            Fin  Si (* il y a un choix incomplet dans la pile *)
        Fin Si   (* backTrackEchec *)
        Fin Si  (* Type( blocCourant ) = _UnitéLexicale *)
    Fin  Tant  Que        (* boucle infinie *)
Fin  Réécriture
```

Procédure auxiliaire d'unification

Cette procédure, non récursive, effectue un parcours en post-ordre de l'arbre d'analyse pour remonter les valeurs des caractéristiques lexico-syntaxiques et vérifier que les contraintes sont respectées. Comme nous l'avons vu, ces valeurs sont stockées dans un champ des blocs de la pile au fur et à mesure de leur remontée.

```
Procédure Unification()
Début
    Pour tous les blocs blocCourant de la pile, en remontant du dernier au
         premier
        Selon Type( blocCourant )
            Si  _MotLu:
                recopier les caractéristiques du mot de la phrase dans celles
                    du bloc courant
                noter le mot fléchi pour l'affichage du résultat d'analyse
            Si  ( _Concaténation Ou _Liste ):
                Si un indice i de substitution est indiqué
                    recopier les caractéristiques du i^ème fils
                Sinon
                    donner des caractéristiques vides
                Fin  Si
            Si  _UnitéLexicale:
                recopier les caractéristiques du bloc fils (mot de la phrase)
            Si  ( _RègleNormale Ou _RègleMéta Ou _RègleElision ):
                recopier les caractéristiques du bloc fils (racine de la
                    queue de règle)
                Retourner Vrai
            Si  ( _TermeRègle ):
                recopier les caractéristiques du bloc fils (tête de règle)
        Fin  Selon
        Empiler le pointeur vers le bloc courant dans TableauPointeursNoeuds;
        Si ( il existe une contrainte de valeur
                Et cette contrainte n'est pas vérifiée )
        Alors_
            Retourner Faux
        Fin  Si
        Si ( Type( blocCourant ) = _Concaténation
                Et il existe une contrainte d'accord
                Et la valeur est différente de celle du terme auquel il faut
                    s'accorder )
        Alors
```



```
            Retourner Faux
        Fin  Si

    Fin  Pour (* tous les blocs de la pile *)
    Retourner Vrai
Fin Unification;
```

Production de noms composés

En reprenant l'expression de la procédure `AjouterTermesSuivants`, on obtient une production en modifiant le cas d'un bloc de type `UnitéLexicale` de la façon suivante:

```
Si _UnitéLexicale:
    Si  production
    Alors
        Pour toutes les flexions produites pour ce mot
        Faire
            recopier la flexion à la suite du bloc
        Fin  Pour
        blocCourant.nombreChoix = nombreDeFlexions;
        blocCourant.blocSuivant = HautPile + Taille( blocCourant )
                                + nombreDeFlexions * Taille( pointeur );
        Fin  Sinon
    Sinon
        Si le mot fléchi est trouvé dans la phrase
        Alors
            Pour tous les homographes trouvés pour ce mot
            Faire
                recopier l'homographe à la suite du bloc
            Fin  Pour
            blocCourant.nombreChoix = nombreDHomographes;
            blocCourant.blocSuivant = HautPile + Taille(blocCourant)
                                + nombreDHomographes * Taille( pointeur );
        Sinon
            backTrackEchec = Vrai;
        Fin  Si
    Fin  Si
```

Une telle production permet à un linguiste chargé de la création des règles d'avoir un retour, et de corriger les énoncés pour qu'ils produisent tous les termes qu'il souhaite, et ceux-ci seulement. On peut ainsi obtenir toutes les séquences qui sont reconnues au moyen des métarègles.

Lorsqu'un terme est une catégorie lexicale libre, le mécanisme de production se contente de remplacer les flexions produites par une seule flexion, dont la chaîne est celle qui sert à désigner la catégorie lexicale dans les règles. Ainsi la règle, qui permet de reconnaître *direction de propagation*, accepte la transformation qui insère un adjectif entre le premier nom et la préposition. En production, sachant que les adjectifs sont notés par A, on obtiendra: *direction* A *de propagation* qui représente, par exemple, *direction relative de propagation*.

6 EXEMPLES DE TRANSFORMATIONS DE NOMS COMPOSES

Nom composé formé par juxtaposition d'un nom composé et d'un adjectif ou un Prép N
(automate programmable) de BBC
(bases de données) relationnelles
(champ électrique) statique...

Nom composé formé par juxtaposition d'un nom simple et d'un nom composé
applications d'(Intelligence Artificielle)
arrêt (REP)
automate de (borication-dilution)...

Nom composé formé par juxtaposition de deux noms composés
(algorithme numérique) de (régulation de température)
(câbles d'énergie) sous (haute tension)
(campagne d'oscillation) de (combustible irradié)...

Nom composé formé par recouvrement de deux noms composés
(calcul des (conditions) aux limites)
(combustion assistée par (plasma) de charbon pulvérisé)
(direction de (propagation) du rayonnement)...

Coordination de noms composés
action (d'information et d'explication)
agent (d'exploitation et de conduite)
archivage (automatique ou explicite) de fichiers...

(à des revues et à des sociétés) savantes
(au formatage et à l'émission) des données
(de l'architecture et des protocoles) OSI...

Références par élision
circuit secondaire des centrales nucléaires
    -> circuit secondaire des centrales
conditions météorologique -> leur évolution prévue
cycle binaire eau / ammoniac -> cycle binaire...

Transformation de noms composés par ajouts de modifieurs
bon (retour d'expérience)
différents (régimes de fonctionnement)
différents (services opérationnels) concernés...

(acier inoxydable) (austéno-ferritique) moulé
(aide à la conduite) supplémentaire
(ammoniac liquide) aspiré...

BIBLIOGRAPHIE ET INDEX

[Aho 83] A. Aho J. Hopcroft J. Ullman. Data structures and Algorithms. Addison Wesley.

[Baase 78] S. Baase. Computer algorithms. Addison Wesley.

[Bach 74] E. Bach. Syntactic theory. Holt, Rinehart & Winston, inc.

[Béchade 86] H.-D. Béchade. Syntaxe du français moderne et contemporain. PUF.

[Béroule 86] D. Béroule. Traitement connexioniste du langage. Histoire, Epistémologie, Langage, Tome 11, Fascicule 1, P. 147-170. Presses Universitaires de Vincennes.

[Boons Guillet Leclere 76A] J.P. Boons A. Guillet C. Leclerc. La structure des phrases simples en français : Constructions intransitives. Droz.

[Boons Guillet Leclere 76B] J.P. Boons A. Guillet C. Leclerc. La structure des phrases simples en français : Constructions transitives. Rapport de recherche du L.A.D.L.

[Carlier 88] J. Carlier P. Chretienne. Problèmes d'ordonnancement. Modélisation / Complexité / Algorithmes. Masson.

[Changeux 83] J.P. Changeux. L'homme neuronal. Fayard.

[Chomsky 57] N. Chomsky. Syntactic structures. Mouton.

[Chomsky 67] N. Chomsky. Remarks on nominalization. Readings in English transformationnal grammar. R. Jacobs & P. Rosenbaum. Blaidsell.

[Colmerauer 78] A. Colmerauer. Metamorphosis Grammar. Natural Language Communication with Computers. Springer Verlag.

[Colmerauer 84] A. Colmerauer. Equations et inéquations sur les arbres finis et infinis. Rapport interne G. I. A. Université Aix Marseille II.

[Danlos 85] L. Danlos. *Génération automatique de textes en langue naturelle*. Masson.

[Danlos 88] L. Danlos. Les phrases à verbe support *être Prép*. Langages N°90, P. 23-38. Larousse.

[Darmesteter 94] A. Darmesteter. Traité de la formation des mots composés 2ème édition. Honoré Champion. 1894.

[D.E.R. 87] d.e.r. 87, Faits Marquants. Electricité de France. Direction des Etudes et Recherches.

[De Sutter 88] M. De Sutter. Vers un traitement unifié de l'ordre relatif et absolu des mots dans la phrase nominale française. Travaux de linguistique 16. P. 27-55.

[Dubois 65] J. Dubois. Grammaire structurale du français. Larousse.

[Dubois 70] J. Dubois, F. Dubois-Charlier. Eléments de linguistique française: syntaxe. Langue et langage. Larousse.

[Fauconnier 74] G. Fauconnier. La coréférence: syntaxe ou sémantique? Travaux linguistiques. Seuil.

[Gadzar 85] G. Gadzar, E. Klein, G. Pullum, I. Sag. Generalized phrase structure grammar. Basil Blackwell.

[Giry-Schneider 87] J. Giry-Schneider. Les prédicats nominaux en français. Les phrases simples à verbe support. Droz.

[Goldberg 83] A. Goldberg & D. Robson. SMALLTALK-80 the language and its implementation. Addison Wesley.

[Grevisse 86] M. Grevisse. Le bon usage. Duculot.

[G. Gross 86] G. Gross. "Typologie des noms composés" in rapport final de l'ATP "Recherches nouvelles sur le langage". Université Paris XIII.

[G. Gross 88A] G. Gross. Degrés de figement des noms composés. Langages N°90, P. 57-72. Larousse.

[G. Gross 88B] G. Gross. Structure des noms composés. Actes du colloque informatique et langue naturelle. LIANA Nantes.

[G. Gross 90] G. Gross. Définition des noms composés dans un lexique-grammaire. Langue Française N° 87, P. 84-90. Larousse.

[M. Gross 75] M. Gross. Méthodes en syntaxe. Herman.

[M. Gross 81] M. Gross. Bases empiriques de la notion de prédicat sémantique. Langages N°63, P.7-52. Larousse.

[M. Gross 86N] M. Gross. Syntaxe du Nom. Cantilène.

[M. Gross 86V] M. Gross. Syntaxe du Verbe. Cantilène.

[M. Gross 88] M. Gross. Lexiques électroniques. Ecole d'été "Traitement des langues naturelles". CNET Lannion.

[M. Gross 90] M. Gross. Sur la notion harrissienne de transformation et son application au français. Langages N°99, P.39-56. Larousse.

[M. Gross Tremblay 85] M. Gross D. Tremblay. Etude du contenu d'une banque terminologique. Rapport final du contrat N° 83.3.94.0201 à la MIDIST.

[Guillet LaFauci 84] Lexique grammaire des langues romanes. J. Benjamins.

[Harris 76] Z.S. Harris. Notes du cours de syntaxe. Seuil.

[Haugeland 85] J. Haugeland. Artificial Intelligence : the very idea. M.I.T. Press.

[Hénocque Jacquemin 86] L. Hénocque C. Jacquemin. Analyse syntaxique de rapports médicaux. Mémoire DEA. GIA. Université de Luminy. Marseille.

[Hillis 85] D. Hillis. The Connection Machine. M.I.T. Press.

[Hoback Haff 90] M. Hoback Haff. Coordonnants et éléments coordonnés: une étude sur la coordination en français moderne. L'information grammaticale N°46, P.17-21.

[Jacquemin 89] C. Jacquemin. Reconnaissance de mots composés à l'aide de règles et métarègles. Bigre N°68, P. 112-127. IRISA. Rennes.

[Jacquemin 91] C. Jacquemin. Modifications des formes figées: gestion dynamique des métarègles. Colloque ILN'91. Liana. Nantes

[Jayez 82] J.H. Jayez. Compréhension automatique du langage naturel. Cas du groupe nominal en français. Masson.

[Jung 90] R. Jung. Remarques sur la constitution du lexique des noms composés. Langue Française N° 87, P.91-97. Larousse.

[Kernigham 78] B.W. Kernigham D.H. Ritchie The C programming language. Prentice Hall.

[Kerridge 87] J. Kerridge. Occam programming : a practical approach. Blackwell.

[Le Ny 89] J.F. Le Ny. Accès au lexique et compréhension du langage: la ligne de démarcation sémantique. L'accès lexical. Lexique N°8, P. 75-86. Presses Universitaires de Lille.

[Lentin 90] A. Lentin. Quelques réflexions sur les références mathématiques dans l'oeuvre de Z. Harris. Langages N°99, P.85-91. Larousse.

[Mathieu-Colas 88] M. Mathieu-Colas. Typologie des mots composés. Rapport technique du Programme de Recherches Coordonnées "Informatique Linguistique". C.N.R.S. Paris.

[Mathieu-Colas 90] M. Mathieu-Colas. Orthographe et informatique: établissement d'un dictionnaire électronique des variantes graphiques. Langue Française N° 87, P.104-111. Larousse.

[Maurel 89] D. Maurel. Reconnaissance de séquences de mots par automates. Thèse de doctorat en informatique, L.A.D.L., Université Paris 7.

[McCawley 81] J.D. McCawley. Everything that linguists have always wanted to know about logic. Basil blackwell.

[Miller Torris 90] P. Miller, T. Torris. Formalismes syntaxiques pour le traitement automatique du langage naturel. Hermes.

[Nivat 88] M. Nivat. On a class of compound expressions in french. North Holland.

[Pierrel 89] J.M. Pierrel. Lexique et compréhension automatique de la parole. L'accès lexical. Lexique N°8, P. 137-165. Presses Universitaires de Lille.

[Pitrat 85] J. Pitrat. Textes ordinateurs et compréhension. Eyrolles.

[Pinkster 90] H. Pinkster. La coordination. L'information grammaticale N°46, P. 8-13.

[Popesco 86] L. Popesco. Analyse et génération de textes à partir d'un seul ensemble de connaissances pour chaque langue naturelle et de métarègles de structuration. Thèse Paris VI.

[Rady 83] M. Rady. L'ambiguïté du langage naturel est-elle source de non déterminisme dans les procédures de traitement. Thèse Paris VI.

[Robins 73] J.H. Robins. Linguistique générale: une introduction. Colin.

[Rumelhart McClelland 86] D.E. Rumelhart, J.L. McClelland. Parallel Distributed Processing Vol 1 &2. M.I.T. Press.

[Ruwet 67] N. Ruwet. Introduction à la grammaire générative. Plon.

[Sabah 88] G. Sabah. L'Intelligence Artificielle et le langage. Hermes.

[Salkoff 73] M. Salkoff. Une grammaire en chaîne du français. Dunod.

[Silberztein 89] M. D. Silberztein. Dictionnaires électroniques et reconnaissance lexicale automatique. Thèse de doctorat en informatique, L.A.D.L., Université Paris 7.

[Silberztein 90] M. D. Silberztein. Le dictionnaire électronique des mots composés. Langue Française N° 87, P.71-83. Larousse.

[Sommerhalder 87] R. Sommerhalder. The theory of computability. Programms, machines, effectiveness & feasability. Addison Wesley.

[Stroustrup 86] B. Stroustrup. The C++ programming language. Addison Wesley.

[Touratier 90] C. Touratier. Coordination et syntaxe. L'information grammaticale N°46, P. 13-16.

[Turner 84] R. Turner. Logics for Artificial Intelligence. Masson.

[Vives 90] R. Vives. Les composés nominaux par juxtaposition. Langue Française N° 87, P.98-103. Larousse.

[Vollat 89] P. Vollat. Calculabilité effective et algorithmique théorique. Eyrolles.

[Winograd 83] T. Winograd. Language as a cognitive process. Vol 1: syntax. Addison Wesley. .

[Winston 81] P.H. Winston. LISP 2nd edition. Addison Wesley. Masson.

[Wunderli 87] P. Wunderli. La place de l'adjectif: norme et infraction à la norme. Travaux de linguistique 14-15, L'ordre des mots en français, P. 221-235.

INDEX

1991 JAC