Contribution à l'étude du traitement des erreurs au niveau lexico-syntaxique dans un texte écrit en français

Vare Lucia Strube den Lima

▶ **To cite this version:**

Vare Lucia Strube den Lima. Contribution à l'étude du traitement des erreurs au niveau lexico-syntaxique dans un texte écrit en français. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1990. Français. NNT : . tel-00337073

HAL Id: tel-00337073
https://theses.hal.science/tel-00337073

Submitted on 6 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Tu 9072

THESE

présentée par

Vera Lúcia STRUBE DE LIMA

pour obtenir le titre de DOCTEUR
de l'UNIVERSITE JOSEPH FOURIER - GRENOBLE I
(arrêté ministériel du 5 juillet 1984)
Spécialité : **Informatique**

Contribution à l'étude du traitement des erreurs au niveau lexico-syntaxique dans un texte écrit en français

DATE DE SOUTENANCE : 15 mars 1990

COMPOSITION DU JURY :

M. Christian BOITET (président)
M. Jacques COURTIN (directeur)
M. Jean CAELEN (rapporteur)
M. Guy PERENNOU (rapporteur)
M. François PECCOUD (examinateur)

THESE PREPAREE DANS LE CADRE DE L'EQUIPE TRILAN
LABORATOIRE DE GENIE INFORMATIQUE
INSTITUT IMAG, UNIVERSITE JOSEPH FOURIER

Remerciements

Je remercie Monsieur Christian BOITET, professeur à l'Université Joseph Fourier et directeur adjoint du GETA (Groupe d'Etudes pour la Traduction Automatique), qui m'a fait l'honneur de présider le jury de cette thèse.

Je remercie Monsieur Jean CAELEN, chargé de recherche au CNRS et directeur de recherche à l'Institut de la Communication Parlée (INPG), et Monsieur Guy PERENNOU, professeur à l'Université Paul Sabatier de Toulouse, qui ont accepté d'être rapporteurs de cette thèse.

Je remercie Monsieur François PECCOUD, professeur à l'Université de Grenoble II et directeur du GETA, qui a bien voulu examiner cette thèse et m'a fait l'honneur de participer au jury.

Je remercie Monsieur Jacques COURTIN, professeur à l'Université de Grenoble II et responsable de l'équipe TRILAN, qui m'a accueillie dans son équipe de recherche et qui a assuré la direction de ma thèse.

Je remercie Madame Louise EICHENBON, professeur à l'Université de Langues et Lettres, qui a effectué la lecture de ce travail.

Je remercie Christian MICHAUX, Irène KOWARSKI, Danièle DUJARDIN et Damien GENTHIAL, membres de l'équipe TRILAN, pour leur sympathie et leur bonne humeur. Je remercie spécialement Irène, Danièle et Damien, pour le temps considérable qu'ils m'ont consacré au cours de ces années et pour l'intérêt qu'ils ont porté à ce travail.

Je remercie enfin de tout mon coeur Alvaro et Diogo, mes deux fils, pour leur gentillesse qui m'a permis de mener à terme, je l'espère, cette thèse.

TABLE DE MATIERES

INTRODUCTION 1
1 MOTIVATION 2
1.1 La correction de fautes d'orthographe 2
1.2 La vérification et correction d'erreurs au niveau syntaxique 4
1.3 Notre contexte 5
2 CONFIGURATION DU TRAVAIL REALISE 8
3 ORGANISATION DU TEXTE 9

Chapitre 1 :
TYPOLOGIE DES ERREURS
ET STRATEGIES DE CORRECTION 11
1 UNE APPROCHE INTRODUCTIVE A LA TYPOLOGIE DES ERREURS 12
1.1 Les niveaux d'erreur dans l'analyse d'un texte 12
1.2 La classification de Catach, Duprez et Legris 13
1.3 Erreurs de compétence et erreurs de performance 16
1.4 Les catégories orthographique et typographique dans le système VORTEX 17
1.5 Notre approche : les catégories d'erreurs lexicales dans le système DECOR 19
1.5.1 Les erreurs typographiques 20
1.5.2 Les erreurs phonétiques 20
1.5.3 Les erreurs de génération 21
2 METHODES DE CORRECTION LEXICALE 21
2.1 Méthodes locales et méthodes globales 22
2.2 Les premiers pas vers la correction automatique 23
2.3 L'édition de chaînes 25
2.3.1 Les études de Damerau 25
2.3.2 La distance de Levenshtein et l'algorithme de Wagner et Fischer 26
2.3.3 L'extension de Lowrance et Wagner 27
2.3.4 Autres propositions dans le domaine de la similarité 29
2.4 Les clés de similarité 30
2.4.1 Les travaux de Pollock et Zamora et la clé squelette 32
2.4.2 Les alphagrammes 34
2.5 Des mesures de similarité 37
2.6 Le traitement d'erreurs dans les mots composés 39
3 ERREURS SYNTAXIQUES 41
3.1 Les erreurs syntaxiques et leur traitement 41
3.2 Des réalisations en vérification et correction syntaxique 43
3.2.1 Le système EPISTLE 45
3.2.2 L'implantation informatique du français fondamental 46
3.2.3 La correction automatique des accords du participe passé 47
4 ERREURS SEMANTIQUES 50

Chapitre 2 :
L'ENVIRONNEMENT PILAF 51
1 L'ANALYSE MORPHOLOGIQUE 52
1.1 Le transducteur général d'états finis 52
1.1.1 Segmentation et transduction 53
1.1.2 Utilisation du transducteur 54
1.2 Fonctionnement de l'analyse morphologique 54
1.3 Les dictionnaires et leur organisation 56
1.3.1 Organisation du dictionnaire de bases morphologiques 56
1.3.2 Organisation du dictionnaire de suffixes et désinences 57
1.3.3 La recherche dans les dictionnaires 58
1.4 La grammaire 61

1.5 Les modèles 61
2 L'ANALYSE SYNTAXIQUE 62
2.1 La construction de dépendances et le modèle linguistique 62
2.2 Relations de dépendances 64
2.3 L'algorithme d'analyse syntaxique 65
3 TRAITEMENT D'ERREURS 67
3.1 Le module de correction par clé squelette 68
3.2 Le module de correction d'erreurs de génération 70
4 LES OUTILS DE PILAF ET LA LANGUE PORTUGAISE 71
4.1 Analyse morphologique et syntaxique 71
4.2 Le traitement d'erreurs 73

Chapitre 3 :
CORRECTION D'ERREURS LEXICALES PAR LA PHONETIQUE 75
1 INTRODUCTION 76
1.1 Etudes sur les erreurs du type phonétique 77
1.1.1 Le système VORTEX 77
1.1.2 Le logiciel TOPH 78
1.1.3 Le système ARCHIMEDE 79
1.1.4 Le système développé au CERIL/LADL 80
1.1.5 La transcription phonétique dans PIAF 82
1.2 Principe adopté 82
1.3 La correction d'erreurs 83
2 LA CHAINE ORALE ET LA CHAINE ECRITE 85
2.1 Graphèmes et phonèmes 85
2.2 Choix des phonèmes liés à la correction de fautes d'orthographe 86
3 STRUCTURES DE DONNEES UTILISEES PENDANT LA TRANSDUCTION PHONETIQUE 90
3.1 Règles, modèles et dictionnaire phonétiques 91
3.1.1 Les règles 91
3.1.2 Le dictionnaire de p-graphèmes 91
3.1.3 Les modèles phonétiques 92
3.1.4 Le code dérivation 93
3.2 Exemples de génération de chaînes orales à partir d'une chaîne écrite 94
4 GENERATION DE CHAINES GRAPHIQUES A PARTIR D'UNE CHAINE ORALE 94
4.1 Première version 94
4.1.1 Les bases morpho-phonétiques 97
4.1.2 La génération de chaînes graphiques 98
4.1.3 Contrôle de la génération 99
4.1.4 Vérification morphologique des chaînes engendrées 101
4.2 Deuxième version 101
4.2.1 Les règles de composition graphique 102
4.2.2 Les possibilités de concaténation (STATBOOL) 103
4.2.3 L'analyse des cas particuliers de composition graphique 104
4.2.4 L'algorithme de reconstitution graphique 105
5 RESULTATS OBTENUS 106
5.1 Potentialités d'une méthode de correction par phonétique 106
5.2 Comparaison entre la première et la deuxième version 108

Chapitre 4 :
VERIFICATION SYNTAXIQUE 111
1 LA VERIFICATION SYNTAXIQUE D'UNE PROPOSITION 113
1.1 Les langages et les règles 113
1.2 Les mécanismes de vérification syntaxique 114

1.2.1 Arbres de dépendances 114
1.2.2 Règles de vérification syntaxique 115
1.3 Le principe de la vérification 116
1.4 Grammaire syntaxique de l'arbre d'une phrase 120
1.5 Variables morphologiques 121
1.6 Catégories et informations lexico-syntaxiques 122
1.6.1 Informations lexico-syntaxiques 123
2 REGLES ET META-REGLES DE VERIFICATION SYNTAXIQUE 123
2.1 Règles de vérification syntaxique 124
2.2 Méta-règles de vérification syntaxique 127
3 CONSIDERATIONS A PROPOS DU TRAVAIL REALISE 130
3.1 Apports 131
3.2 Observations à propos des langages de programmation utilisés 132

Chapitre 5 :
DETECTION ET CORRECTION DES FAUTES D'ACCORD 135
1 FAUTES SYNTAXIQUES TRAITEES 136
1.1 Les types de fautes syntaxiques 136
1.2 Des cas d'absence de structure syntaxique 137
2 LES REGLES D'ACCORD 138
2.1 Règles d'accord pour la langue française 138
2.2 Le participe passé suivi d'un infinitif 141
3 MODELISATION DE REGLES D'ACCORD 143
3.1 Les règles modélisées 144
3.2 Les situations omises dans la modélisation 144
3.2.1 L'accord du verbe avec le groupe nominal sujet 144
3.2.2 L'accord de l'adjectif 145
3.2.3 Les accords du participe passé 145
3.3 Un exemple de vérification/correction syntaxique 146
4 CORRECTION DE FAUTES D'ACCORD 147
4.1 Approches face à la correction d'une faute d'accord 147
4.2 Paramètres adoptés dans le système 149
4.3 Une analyse post-syntaxique en vue de la correction 150
4.3.1 La correction du verbe et de l'attribut dans une relative 151
4.3.2 Les sous-arbres d'adjectifs gouvernés par la catégorie *coord* 154
5 LA SUITE D'UNE CORRECTION 157
6 INFORMATIONS NECESSAIRES A UNE MODELISATION EXHAUSTIVE DES REGLES D'ACCORD 158
6.1 Quelques propositions 159

Chapitre 6 :
CONSIDERATIONS A PROPOS D'UN SYSTEME DE TRAITEMENT D'ERREURS AU NIVEAU LEXICO-SYNTAXIQUE 161

1 CORSYNT : UNE MAQUETTE DE DETECTION/CORRECTION LEXICO-SYNTAXIQUE 162
1.1 L'organisation fonctionnelle 162
1.2 L'ordre d'application des méthodes de correction 165
1.3 Le système 166
2 UN SYSTEME DE TRAITEMENT D'ERREURS AU NIVEAU LEXICO-SYNTAXIQUE 166
2.1 Observations concernant l'étape d'analyse syntaxique 166
2.2 Une proposition d'application d'application des "Tree Adjoining Grammars" à l'analyse syntaxique en vue d'une vérification syntaxique 167

2.2.1 Généralités à propos des TAGs 168
2.2.2 TAGs et traitement d'erreurs syntaxiques 169

CONCLUSION 171

REFERENCES BIBLIOGRAPHIQUES 175

ANNEXE 1 187

ANNEXE 2 195

ANNEXE 3 219

ANNEXE 4 227

ANNEXE 5 235

ANNEXE 6 251

INTRODUCTION

1 MOTIVATION

1.1 La correction de fautes d'orthographe

Au début des années 50, ont commencé à apparaître les premiers travaux de recherche relatifs au traitement informatique des langues naturelles. Centrés sur la traduction automatique, les efforts engagés pour le traitement des langues naturelles ont permis d'adjoindre à ce domaine l'analyse d'autres caractéristiques de la langue, comme la syntaxe et la sémantique.

Parallèlement, un autre domaine du traitement des langues naturelles est apparu, qui se popularise rapidement et permet l'insertion de ses résultats dans la plupart des systèmes avancés de traitement de textes et des interfaces utilisateur. Il s'agit de la correction orthographique [GREANIAS 84].
La mise en oeuvre d'outils pratiques de traitement d'erreurs dans un texte se transforme en objet de recherche en vue d'algorithmes rapides et efficaces, pouvant être couplés à des systèmes plus complets de traitement et manipulation de la langue naturelle.

Ces outils ont permis, d'abord, le traitement des erreurs au niveau lexical.
Même si elles sont initialement tournées vers *l'appariement* de chaînes de caractères en général (et non spécifiquement vers la correction de fautes d'orthographe), quelques unes des méthodes développées ont pu aussi être utilisées pour la correction d'erreurs orthographiques dans des textes écrits en français [DAMERAU 64, WAGNER 74, POLLOCK 84, etc].
Mais la langue française présente des caractéristiques particulières, comme l'accentuation, ou la variété de possibilités graphiques à partir d'un modèle phonétique (exemple : ***karo*** est une graphie phonétiquement équivalente à *carreaux*), qui demandent des études spécifiques au sujet du traitement d'erreurs.

En ce qui concerne le traitement d'erreurs lexicales dans les textes écrits en français, plusieurs logiciels sont commercialement disponibles. Pour les ordinateurs de la ligne PC, nous pouvons citer WORD (de MICROSOFT), MANUSCRIPT (de LOTUS), ALPHA LEXIS (de BORLAND), HUGO (de LOGIDISQUE), parmi d'autres.
Sans considérer la rapidité de réponse (qui peut être un facteur souvent décourageant à l'utilisation de ces logiciels), nous pouvons observer, d'abord, que certains de ces outils, soit ne détectent pas tous les mots faux présents dans un texte, soit détectent comme faux des mots corrects. Par exemple, WORD a détecté comme faux le mot *complète*.

Certains types d'erreurs, comme les erreurs phonétiques, ne sont pas suffisamment traitées par ces logiciels. Par exemple, ALPHA LEXIS corrige ***farmacle*** en *pharmacie* [ICHBIAH 89], mais WORD suggère, pour ***ocuranse*** , les corrections *occurrents* ou *octantes*.
Les erreurs concernant une mauvaise désinence, comme ***disez*** au lieu de *dites*, sont très rarement corrigées.
Les erreurs concernant des mots composés sont faiblement détectées. Par exemple, ***stations-services*** est considéré correct par WORD, MANUSCRIPT ou SPRINT [ICHBIAH 89].

Au point de vue de la recherche scientifique, plusieurs études ont été réalisées à propos du traitement des erreurs lexicales dans un texte écrit en français [DEBILI 86, LAHENS 86, MARET 87, FOUQUERE 88, EMIRKANIAN 88a, VERONIS 88c, LAPORTE 89a, ...].
Certains de ces travaux ont déjà accordé une importance considérable au composant phonétique [LAHENS 86, MARET 87, VERONIS 88c, EMIRKANIAN 88a, LAPORTE 89a]. Parmi les approches utilisées, nous identifions deux lignes de conduite principales : soit l'équivalence phonétique est atteinte à travers une correspondance étroite, ce qui exige normalement un processus de transduction phonétique pour trouver toutes les possibilités de prononciation associées à une certaine chaîne graphique [LAPORTE 88, COURTIN 88], soit l'équivalence phonétique est obtenue par le biais d'un ensemble de relations de correspondance [LAHENS 86, VERONIS 88], ce qui peut être une solution valable dans un contexte bien circonscrit d'application.
Lorsque nous nous sommes intéressés à la correction d'erreurs dans des textes contenant des termes provenant d'un dictionnaire de grandeur réelle (200000 à 300000 formes), les méthodes employant des tables ou des règles de correspondance risquent de présenter des insuffisances. Elles s'exposent à l'inconvénient de ne pas effectuer un parcours exhaustif de l'ensemble des mots et, par conséquent, de ne pas produire comme résultat toutes les possibilités graphiques associées au terme erroné.
Dans ce cas, il est impératif d'employer une méthode qui présente le plus grand nombre de possibilités graphiques candidates à la correction.

La méthodologie mise en oeuvre par l'équipe de M. Gross [LAPORTE 88, LAPORTE 89a] pour la correction d'erreurs d'orthographe constitue une approche très généraliste. Elle permet la récupération d'erreurs au moyen d'un dictionnaire phonétisé, qui associe à chaque mot sa chaîne orale correspondante (cette proposition a été adoptée également dans le projet PILAF [COURTIN 88], dans une première approche de correction d'erreurs par la phonétique, avec un dictionnaire phonétisé d'essai).

Toutefois, dans un tel environnement, deux aspects peuvent être discutés, si nous envisageons le développement d'un logiciel de traitement d'erreurs :

1 - l'élaboration d'un dictionnaire phonétisé correspondant au dictionnaire de bases disponible dans le système est une tâche prolongée qui demande une étude minutieuse du comportement de la langue et de ses particularités phonétiques ;

2 - l'espace en mémoire secondaire exigé pour le stockage d'un tel dictionnaire, s'il s'agit d'un système multi-algorithmique pour la correction d'erreurs d'orthographe, peut être très vaste.

Il existe, donc, une lacune en ce qui concerne les méthodes de correction d'erreurs au niveau lexical, justement dans la présentation d'algorithmes qui produisent une gamme large et satisfaisante d'équivalents phonétiques, tout en réduisant les inconvénients causés par l'acquisition d'un dictionnaire phonétisé.

1.2 La vérification et correction d'erreurs au niveau syntaxique

Nous constatons que la grande variété de produits commerciaux de vérification/correction d'erreurs observée récemment, au niveau lexical, n'existe pas encore, pour la correction d'erreurs au niveau syntaxique dans un texte écrit. Le marché actuel est caractérisé par des produits qui dominent l'univers du mot, alors que nous n'avons pas connu une évolution significative, en ce qui concerne les systèmes commercialement disponibles, au niveau de la structure de la phrase et de sa signification.

La correction d'erreurs de syntaxe dans un texte exige la disponibilité d'informations au niveau morphologique, syntaxique voire sémantique, qui sont obtenues à travers une analyse morphologique, syntaxique et sémantique d'une amplitude considérable. En plus, il est normalement nécessaire de borner le sous-ensemble d'intérêt dans le domaine de la langue naturelle, laquelle se présente encore comme un champ trop vaste pour une vérification/correction syntaxique exhaustive.
Plusieurs équipes de recherche travaillent, actuellement, à la vérification/correction d'erreurs syntaxiques dans un texte écrit en français [RICHARD 86, LACOUTURE 88, entre autres], en utilisant soit des informations morpho-syntaxiques, soit des informations à caractère syntaxico-sémantique.

Nous considérons que l'utilisation d'informations sémantiques dans le traitement des erreurs de syntaxe, pour un domaine générique de la langue, est une tâche qui présentera pour longtemps certaines difficultés. Assurément toutes les erreurs de syntaxe ne peuvent pas être corrigées sans tenir compte d'informations sémantiques ; plusieurs types de fautes ne pourront même pas être détectés (par exemple, dans l'analyse syntaxique de la phrase *les chiens de la voisine qui **abole** incessamment*, au verbe *aboyer* pourrait être attribué, comme sujet, *la voisine*, et aucune faute ne serait observée).
D'autre part, la non utilisation d'informations sémantiques permet la vérification/correction syntaxique d'un ensemble beaucoup plus large de constructions, dans une première étape d'études (il nous est possible de traiter des phrases comme *elle avalait ses mots en parlant* en leur attribuant normalement des structures de dépendances).

1.3 Notre contexte

Ce travail s'insère dans le contexte du projet PILAF (**P**rocédures **I**nteractives **L**inguistiques **A**ppliquées au **F**rançais [COURTIN 86]), équipe TRILAN (**TR**aitement **I**nformatique des **LA**ngues **N**aturelles), du Laboratoire de Génie Informatique de l'IMAG.
Le projet PILAF a comme objectif général le développement d'une boite à outils linguistiques pour le traitement de la langue française. Spécifiquement, un des objectifs de PILAF est le développement d'outils de traitement d'erreurs aux niveaux lexical, syntaxique voire sémantique.
Avec les outils disponibles, nous avons développé un système de détection/correction d'erreurs au niveau lexical - le système DECOR (DEtection/CORrection) [COHARD 88].
Au niveau syntaxique, une pré-étude a été conçue pour la vérification syntaxique d'une phrase [SAIDI 86].

Le présent travail a donc pour objectif de réaliser une étude des erreurs pouvant apparaître dans un texte et de mettre en oeuvre, dans l'environnement PILAF, des méthodes de détection et correction d'erreurs aux niveaux lexical et syntaxique permettant la conception et le développement d'un système multi-algorithmique de traitement d'erreurs. Ce système, pourrait s'insérer dans des interfaces homme-machine aussi bien que dans des logiciels de traitement de textes en français.

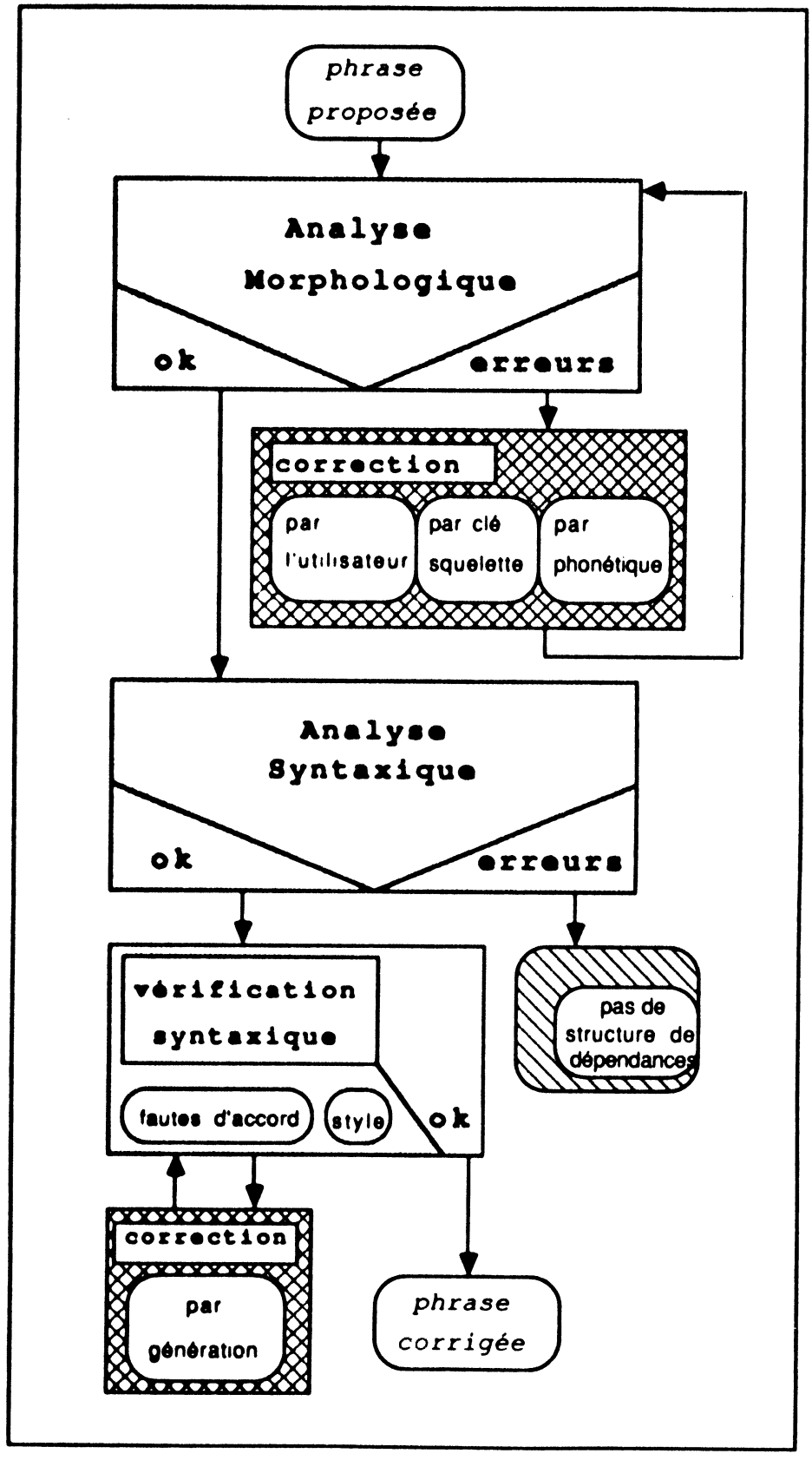


Figure I-1

Architecture d'un système de détection/correction au niveau lexico-syntaxique

Dans ce contexte, nous avons étudié le traitement des erreurs aux niveaux lexical et syntaxique et nous avons établi l'architecture d'un système de traitement d'erreurs au niveau lexico-syntaxique - le système *CORSYNT* - présentée dans la figure I-1.
Ce système utilise comme base les outils disponibles de PILAF (analyseurs morphologique et syntaxique, correction lexicale par clé squelette, génération morphologique) et aussi d'autres outils à développer.

Du projet de cette maquette ressort l'intérêt du développement d'une méthode de correction d'erreurs par la phonétique, qui soit applicable à un dictionnaire de grandeur réelle et qui soit utilisable en tant que composant d'un système de traitement d'erreurs dans un texte en français.
Une telle méthode de correction, appliquée à un dictionnaire de grandeur réelle, n'existe que dans un seul système (celui du LADL [LAPORTE 89a]) dont nous ne connaissons pas de détails face à l'utilisation comme partie d'un environnement de détection/correction.

L'intérêt se présente également pour la mise en oeuvre d'un module de vérification et de correction syntaxique, cette dernière centrée sur le thème des erreurs d'accord.
Disposant d'un analyseur morphologique et d'un analyseur syntaxique capables de construire les structures de dépendances associées à une phrase, nous avons envisagé d'effectuer une étude en vue de la vérification et correction syntaxiques applicables à un dictionnaire de vraie grandeur.
Parmi les travaux au sujet du traitement des erreurs syntaxiques, nous ne connaissons qu'une seule réalisation basée sur un ensemble expressif de la langue française - celle développée par Richard et Lapalme [RICHARD 86], bien que l'objectif principal de ces auteurs soit de corriger les fautes d'accord du participe passé et que le dictionnaire utilisé ne soit que celui du "français fondamental".

Finalement, il faut signaler que, dans cette étude, nous ne nous sommes pas encore préoccupés des restrictions concernant le temps de réponse *envisagé* par l'utilisateur.
Nous pouvons constater que ce dernier, souvent découragé par la longue attente d'une proposition de correction, préfère se servir des outils de traitement d'erreurs commercialement disponibles plutôt comme des outils de détection que comme des outils de correction.
Certainement, les réponses obtenues par nos algorithmes au niveau lexical de correction ne seront pas plus décourageantes que celles proposées ailleurs. Au niveau syntaxique, cependant, le traitement de

phrases plus longues, facilement trouvées dans un texte écrit, pourrait s'avérer, dans un premier temps, fastidieux à l'utilisateur d'un système interactif.
Ce questionnement, peut-être présent dans tout développement concernant le traitement des langues naturelles, ne met pas en cause les objectifs de nos travaux.

2 CONFIGURATION DU TRAVAIL REALISE

Ce travail aborde le thème du traitement des erreurs aux niveaux lexical et syntaxique dans un texte écrit en français.

Le contexte présente initialement les éléments de base d'un ensemble de méthodes utilisées actuellement dans le traitement d'erreurs au niveau lexical. Ensuite, il décrit l'environnement PILAF de traitement de la langue naturelle, où s'insère l'étude en question.

Dans le cadre du projet PILAF, nous avons proposé et mis en oeuvre un algorithme de correction orthographique au niveau lexical par la phonétique. L'algorithme proposé réalise la transduction phonétique du mot faux suivie de sa reconstitution graphique.
Cet algorithme a été mis en oeuvre en deux versions différentes. La première version se sert d'un dictionnaire de bases phonétisées pour reconstituer la graphie d'un mot à partir de sa transcription phonétique. Ayant comme objectif l'application sur un dictionnaire de grandeur réelle, nous avons considéré, de cette manière, la création d'un dictionnaire phonétisé associé à un dictionnaire de bases de taille réelle. Nous avons pu constater la complexité du processus de création d'un dictionnaire phonétisé de telle grandeur.
En conséquence, nous avons conçu et mis en oeuvre une deuxième réalisation de l'algorithme de correction d'erreurs lexicales par la phonétique. Cette deuxième version ne requiert pas un dictionnaire phonétisé. A la place d'une telle structure de données, elle emploie des listes de règles de reconstitution graphique, définies à partir d'une analyse des formes stockées dans le dictionnaire de formes de PILAF.

Egalement dans le cadre du projet PILAF, nous proposons et décrivons la mise en oeuvre du pré-prototype d'un module de vérification syntaxique et de correction des erreurs d'accord.
Nous parcourons les structures de dépendances produites par l'analyseur syntaxique de PILAF réalisant la vérification des accords et l'unification des traits lexico-syntaxiques associés aux noeuds.
A la détection d'une faute d'accord, nous relevons de la phrase les éléments essentiels à sa correction, et nous utilisons le générateur

morphologique de PILAF pour engendrer des formes capables de substituer le mot syntaxiquement erroné.
Cette étude, même si elle est centrée sur un dictionnaire d'essai au point de vue pratique, introduit des observations et des propositions concernant l'utilisation d'un dictionnaire en vraie grandeur.

La plausibilité des modules développés est démontrée par le biais d'une maquette qui réalise la vérification/correction lexico-syntaxique d'une phrase : la maquette *CORSYNT*.

Quelques considérations ont été aussi réalisées en vue d'un traitement de la langue portugaise : son analyse morphologique et syntaxique, la détection et correction d'erreurs lexicales et syntaxiques par moyen des outils disponibles de PILAF et des outils développés au long de ce travail.

3 ORGANISATION DU TEXTE

Ce document est organisé en six chapitres, précédés d'une introduction et suivis d'une conclusion et d'une liste de références bibliographiques. Séparément, nous présentons une liste d'annexes.

Le premier chapitre contient une approche générale des erreurs pouvant apparaître dans un texte. Nous y décrivons différentes typologies des erreurs et aussi des méthodes de correction et mesures de similarité proposées dans les principales études réalisées dans le domaine de la correction.

Le deuxième chapitre offre une description de l'environnement PILAF de traitement de la langue naturelle. Nous présentons dans ce chapitre le transducteur d'états finis de PILAF et ses analyseurs lexical et syntaxique.
Nous y présentons également les mécanismes de traitement d'erreurs disponibles dans le système DECOR .

Du troisième jusqu'au sixième chapitre nous détaillons la réalisation effectuée.
Le troisième chapitre concerne la correction de fautes d'orthographe par la phonétique. Nous y analysons les solutions adoptées par des systèmes qui traitent les erreurs du type phonétique et nous y présentons nos réalisations.
Le quatrième chapitre expose la vérification syntaxique et le cinquième, la correction de fautes d'accord.
Le sixième chapitre présente d'abord la maquette de détection/correction lexico-syntaxique réalisée avec les modules

développés et ensuite des considérations vers l'alternative de réalisation d'une analyse/vérification/correction syntaxiques intégrées.

CHAPITRE 1

TYPOLOGIE DES ERREURS

ET

STRATEGIES DE CORRECTION

Dans ce chapitre nous présentons une approche générale des erreurs pouvant apparaître dans un texte.

Nous mettons en évidence une classification linguistique des erreurs, en trois niveaux : lexical, syntaxique et sémantique.

Au niveau lexical de cette classification, nous proposons une approche des différents types d'erreurs trouvées dans un texte, suivie d'une synthèse des stratégies de correction les plus répandues.
Au niveau syntaxique, nous présentons quelques systèmes existants et commentons leurs résultats.

1 UNE APPROCHE INTRODUCTIVE A LA TYPOLOGIE DES ERREURS

Les études faites dans le domaine de la classification des erreurs trouvées dans un texte ont été tout d'abord dirigées vers les erreurs provoquées par la mauvaise reconnaissance des caractères optiques et vers les erreurs typographiques dans le code des programmes.

Aujourd'hui, la correction automatique d'une chaîne de caractères fait partie de plusieurs applications - la télématique, la bureautique, la lecture optique, l'assistance à la mise au point de programmes, la Biologie Moléculaire, la Génétique, et la préparation des interfaces homme-machine pour les systèmes de quatrième et cinquième génération.
Parmi ces applications, se trouve la mise au point de textes écrits en langue naturelle.
C'est sur la typologie et le traitement des erreurs trouvées dans un texte écrit en langue naturelle que nous centrons le présent chapitre de ce travail.

1.1 Les niveaux d'erreur dans l'analyse d'un texte

Au point de vue linguistique classique, au cours de l'analyse d'un texte les fautes peuvent être trouvées à trois niveaux différents : le *niveau lexical*, le *niveau syntaxique* et le *niveau sémantique*, c'est-à-dire respectivement au niveau du mot, de la construction ou du sens.

Au *niveau lexical*, nous pouvons distinguer [COURTIN 86] les erreurs d'orthographe (comme la transposition des lettres **r** et **é** dans *aréoport*), les erreurs phonétiques (comme la substitution de **au** par **eau** dans ***beaume***) et les erreurs de génération (comme l'utilisation

d'une mauvaise racine dans le participe passé ***valnquu***). Les erreurs typographiques (comme la transposition des lettres **c** et **h** dans ***hcapeau***) sont également insérées dans cette catégorie.
Les erreurs produites dans les éléments constituants des mots composés pourraient aussi être traitées au niveau lexical.

Au *niveau syntaxique* nous considérons la correction en ce qui concerne les règles de construction de la phrase ainsi que les accords entre ses composants.
Par exemple, la phrase *garçon mangé le pomme* peut être considérée syntaxiquement fausse à cause de l'absence d'un gouverneur (probablement le verbe *avoir*), à cause d'une faute d'accord entre l'article *le* et le nom *pomme* et aussi à cause de l'absence d'un déterminant associé au mot *garçon*.

Au *niveau sémantique*, une interprétation de la phrase est envisagée en permettant sa vérification selon sa signification, qui peut être suivie d'une correction, si nécessaire.
Par exemple, le traitement au niveau sémantique de la phrase *Le mètre fait travailler l'élève* fait apparaître certainement une faute : le sujet *mètre* ne peut pas pratiquer l'action de *faire travailler*. Pour le corriger nous pouvons faire référence à un ensemble d'équivalents phonétiques de *mètre* qui contient comme alternative le mot *maître*

En ce qui concerne la typologie des erreurs d'orthographe, plusieurs auteurs ont énoncé leurs points de vue. Nous présentons, dans le texte qui suit, des travaux intéressants dans le domaine de la détection/correction.

1.2 La classification de Catach, Duprez et Legris

Une classification d'erreurs envisagée pour la langue française et centrée sur le problème de l'enseignement de l'orthographe est celle de Catach, Duprez et Legris [CATACH 80c].
Les auteurs proposent une typologie des erreurs orthographiques en six catégories :

I. ERREURS A *DOMINANTE PHONETIQUE*

Les erreurs présumées à dominante phonétique peuvent être commises par ceux qui entendent mal ou/et prononcent mal.

a) Omission ou adjonction d'une lettre ou d'une syllabe : comme ***maltenant*** au lieu de *maintenant*, ***arbruste*** au lieu de *arbuste* ou ***bienteur*** au lieu de *bienfaiteur*.
b) Confusion : comme ***pupller*** (p ↔ b), ***tortoir*** (t ↔ d), ***nimer*** (m ↔ n), ***défint*** pour *défunt* ou ***quillier*** pour *cuiller*.
Ce genre d'erreurs, il faut l'observer, ne conserve pas, normalement, la forme orale du mot.

II. ERREURS A *DOMINANTE PHONOGRAMMIQUE*

Ces erreurs se produisent quand on connaît les sons sans connaître leur transcription. C'est le maniement des phonogrammes, c'est-à-dire des unités de base de l'écrit, qui est ici en cause.
Cette catégorie comprend les erreurs traitées, ailleurs, comme des erreurs "phonétiques".

A Altérant la valeur phonique :

a) Omission ou adjonction : comme ***boef*** pour *boeuf*, ***cheuveu*** pour *cheveu*, ***exès*** pour *excès*.
b) Confusion : ***oisis*** pour *oasis*, ***nè*** pour *né*.
c) Inversion : ***idoit*** pour *idiot*, ***èlève*** pour *élève*, ***ceullllr*** pour *cueillir*.

B N'altérant pas la valeur phonique :

a) Omission ou adjonction : ***sin*** pour *sein*, ***oeull*** pour *oeil*, ***tiket*** pour *ticket* ou comme dans ***pensser***, ***enffermer***, ***allourdir***, ***aléger***.
b) Confusion : ***noiller*** pour *noyer*, ***invantère*** pour *inventaire*, ***pharmatie*** pour *pharmacie*.

III. ERREURS A *DOMINANTE MORPHOGRAMMIQUE*

A Les morphèmes grammaticaux :

Ce groupe tient compte des relations mal établies entre les catégories grammaticales (comme dans ***des ombres passes***), de la confusion dans les formes du pluriel (***chevaus*** pour *chevaux*) et entre nom et déterminant (comme dans ***la routes***, ***les rue***) ou entre nom ou adjectif et complément (comme ***sac de bille***).

B Les morphèmes lexicaux :

a) Non-reconnaissance des mots : comme dans ***un névier***.
b) Ignorance de la famille lexicale (comme dans ***inabilté***), des préfixes ou suffixes (comme dans ***anterremant***), du maintien ou non du radical (comme dans ***nous vogons***) ou des lettres finales justifiables (exemple : ***couvers***, ***blan***).

Catach, Duprez et Legris signalent que, pour approfondir les recherches dans ce genre d'erreurs, il serait utile de posséder une grille ordonnée et complète des principaux **morphogrammes grammaticaux** (signes de genre, signes de nombre, signes de flexion verbale) et des **morphogrammes lexicaux** (signes dérivatifs, graphies figées de préfixes et de suffixes, etc).

IV. ERREURS CONCERNANT LES HOMOPHONES

Elles touchent à la reconnaissance du mot.

a) Homophones de discours : comme ***l'arme*** pour *larme*, ***encore sage*** pour *en corsage*.
b) Homophones lexicaux : comme ***chant*** pour *champ*, ***voie*** pour *voix*, ***vain*** pour *vin*.
c) Homophones grammaticaux : comme ***où*** pour *ou*, ***sont*** pour *son*, ***ni*** pour *n'y*, ***ces*** pour *ses*, etc.

V. ERREURS CONCERNANT *LES IDEOGRAMMES*

Les idéogrammes sont des signes extérieurs à l'alphabet proprement dit, hautement informatifs cependant, pour la plupart d'entre eux.
Cette catégorie d'erreurs concerne la majuscule (***l'état*** n'est pas *l'Etat*), l'apostrophe et le trait d'union, la ponctuation et les erreurs de signes.

VI. ERREURS CONCERNANT DES LETTRES NON FONCTIONNELLES

Sont incluses dans cette catégorie les consonnes doubles non fonctionnelles, les lettres latines et grecques qui ne servent pas, par exemple, à distinguer des homophones (comme dans ***sculter*** et ***téâtre***), le *h* muet dans la plupart des cas, le *e* muet (comme dans *asseoir*) qui ne sert pas, par exemple, à noter les consonnes finales prononcées (comme *faite*) ou à noter le morphème du féminin après

voyelle (comme ***amie***), et des finales particulières (comme dans ***abrit***, ***frai*** ou ***peure***).

1.3 Erreurs de Compétence et Erreurs de Performance

Veronis [VERONIS 88b] utilise les concepts introduits par Chomsky (dans *Aspects of the Theory of Syntax*, 1965) en considérant la distinction entre les erreurs de **compétence** et les erreurs de **performance** : alors que la compétence est la connaissance des structures de la langue, la performance se rapporte à l'emploi effectif de la langue dans des situations concrètes. Les fautes typographiques, par exemple, dues à une mauvaise frappe sur le clavier, sont des erreurs de performance, alors que les erreurs phono-graphiques (comme ***hortodoxe*** pour *orthodoxe*) sont des erreurs de compétence.
Ces travaux se réalisent dans le cadre du système expert ARCHIMEDE (groupe Représentation et Traitement des Connaissances, Marseille, destiné à l'enseignement de la géométrie euclidienne plane) avec l'aide du pôle Langage Naturel du PRC Communication Homme-Machine.

Comme l'indique Veronis, les notions de performance et de compétence peuvent, de plus, s'appliquer aux deux interlocuteurs, l'utilisateur et le système, alternativement émetteur et récepteur, bien que l'un d'eux ne soit qu'une machine. Les quatre grandes classes d'erreurs ainsi induites :
- erreurs de compétence de l'utilisateur,
- erreurs de performance de l'utilisateur,
- erreurs de compétence du système,
- erreurs de performance du système,

nécessitent des stratégies de correction différentes.

Veronis propose l'application de ces notions à tous les niveaux du traitement d'erreurs dans le dialogue homme-machine en langage naturel : soit le niveau lexical, le niveau syntaxique ou le niveau sémantique.

Au niveau lexical, une stratégie de correction des erreurs phono-graphiques tient compte des erreurs typographiques et des erreurs phonétiques.
Cette stratégie représente une extension aux erreurs phono-graphiques des méthodes destinées aux erreurs purement graphiques, comme les erreurs d'édition de chaînes (insertion, suppression, transposition ou substitution de lettres dans le mot). Parallèlement aux erreurs d'édition, une nouvelle opération de *substitution entre deux sous-chaînes phonologiquement semblables* est définie (exemple : *eau* et *o*).

Un algorithme calcule l'indice de dissimilarité entre deux chaînes : le coût de la séquence la moins coûteuse permettant de passer d'une chaîne à l'autre, en considérant les opérations de substitution, insertion et suppression typographique et la nouvelle opération de substitution entre deux chaînes phonologiquement semblables.
Cet algorithme permet de calculer le coût de la séquence d'opérations la moins coûteuse, et aussi de déterminer la séquence elle même.

Le problème pratique est de déterminer, le plus rapidement possible, les sous-chaînes semblables en tout point de la comparaison. Veronis propose pour cela un transcodage préalable des deux chaînes, qui consiste à remplacer chaque partie x[i] d'une chaîne x par un code c[i] représentant la plus longue sous-chaîne ux[i] telle que x = a ux[i] b

Pour réaliser la transcription phonie → graphie, l'auteur utilise des tables de correspondance entre sous-chaînes semblables, construites à partir de l'analyse d'un dictionnaire d'environ 4000 mots.

En vue des problèmes liés à l'utilisation d'un grand dictionnaire, Veronis propose une clé de similarité phonétique analogue à celle définie par Pollock et Zamora [POLLOCK 84] (détaillée dans le paragraphe 2.4.1 de ce chapitre) pour minimiser le nombre des accès disque.
Cette clé phonétique prend en compte la similarité de prononciation, à partir de règles de calcul données dans [VERONIS 88b].

1.4 Les catégories orthographique et typographique dans le système VORTEX

Dans le cadre du système VORTEX [LAHENS 86, PERENNOU 86], développé au laboratoire CERFIA de l'Université Paul Sabatier de Toulouse, une classification des erreurs est aussi proposée. Les fautes qui peuvent s'introduire dans un texte sont regroupées en deux catégories : le niveau *orthographique* et le niveau *typographique*.

Les *fautes typographiques* sont introduites à la saisie des textes et souvent liées à l'usage des claviers de machines à écrire. Le système VORTEX considère également dans ce niveau les fautes matérielles diverses dont l'origine est indépendante des difficultés de l'orthographe : erreurs de reconnaissance dans les lecteurs optiques, erreurs de transmission télématique, erreurs d'encodage dans les bases de données textuelles, parmi d'autres.

Les fautes *orthographiques* comprennent les fautes d'usage, les fautes d'accord et les fautes qui sont situées sur la limite entre ces deux types.
Les fautes d'usage proviennent essentiellement du manque de concordance entre l'écrit et l'oral. Exemple : ***acoustic*** au lieu de *acoustique*, ***kmer*** au lieu de *khmer*.
Pour traduire les risques de fautes d'orthographe d'usage, VORTEX définit des groupes de lettres posant des problèmes orthographiques (exemple : **t/th**, **xion/ction**).

Les fautes typographiques suivantes sont traitées dans VORTEX :
- effacement d'une lettre (**D**),
- insertion d'une lettre (**I**),
- substitution d'une lettre par une autre (**S**),
- transposition de deux lettres consécutives (**T**),
- coupure d'un mot ou insertion de blanc (Γ) et
- soudure de deux mots consécutifs ou effacement de blanc (Σ).

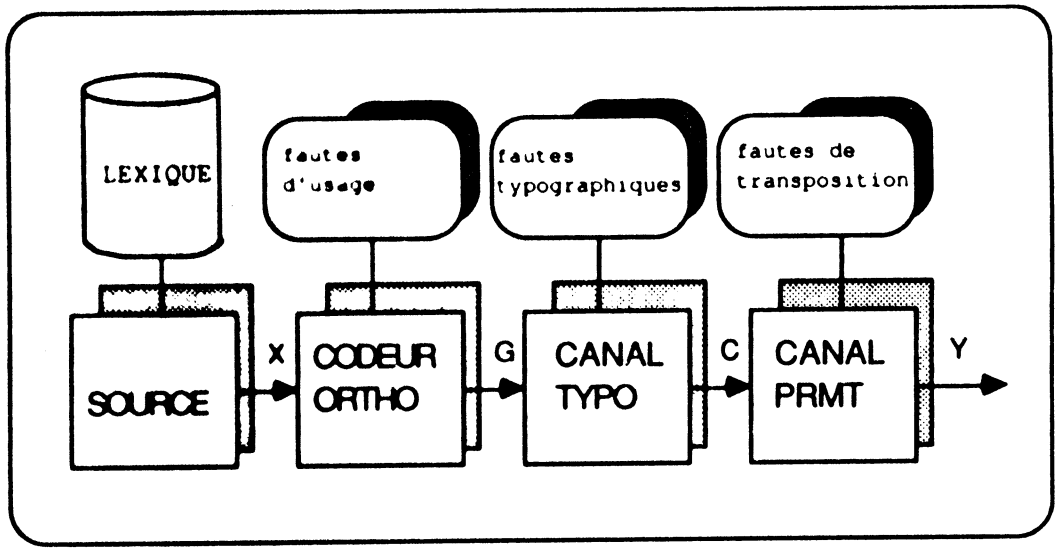


Figure 1.1 - Modèle théorique de VORTEX

Le modèle de correction adopté dans ce système comporte quatre sources de connaissances (voir figure 1.1, extraite de [PERENNOU 86]) :

- une source lexicale, qui inclut la base de connaissances lexicales,
- un codeur orthographique *ORTHO* où sont modélisées les fautes d'usage,
- un canal typographique *TYPO* où sont modélisées les fautes typographiques **D**, **I**, **S**, Γ et Σ,
- un canal de permutations *PRMT* où sont modélisées les fautes de transposition **T**.

La source lexicale émet des mots, ou des suites de mots, sans fautes. Ceux-ci proviennent d'un vocabulaire **V**. Ils sont écrits avec l'alphabet **A**, constitué de symboles représentant chacun un *gpo* (groupe de lettres posant des problèmes orthographiques) et d'un symbole spécial # destiné à marquer les *fins de mots*.
Le lexique représente le vocabulaire **V** sous forme d'un arbre factorisant les préfixes communs.

Le codeur orthographique *ORTHO* rend compte des fautes d'usage en associant à chaque *x* de **A** un code *c* de **B***, où **B** est l'alphabet usuel complété par #.
Le recodage orthographique tient compte des erreurs du type phonétique.

Exemple (extrait de [LAHENS 86]) :
Soit le gpo *o* de **A** dont la graphie sans erreur est «**o**». A ce gpo correspondent les règles de recodage suivantes, qui traduisent les fautes possibles sur un «**o**» interne (les valeurs de probabilité, données entre parenthèses, sont variables suivant les applications) :
o → ***o*** (0,97) | ***au*** (0,02) | ***eau*** (0,01)

> Conformément à l'auteur, le codage est supposé aléatoire pour rendre compte des comportements, des hésitations et des performances effectives lors de la création de textes.

Les fautes typographiques modifient aléatoirement les codes orthographiques.
Dans le canal *TYPO*, une faute n'affecte qu'un seul code orthographique. L'ensemble de codes que *TYPO* peut produire en sortie est un sous-ensemble de **B***.

Conformément à [LAHENS 86, PERENNOU 86], le rôle du canal de permutation *PRMT* dans le modèle VORTEX se limite à la concaténation des sorties successives du canal *TYPO* avec introduction d'éventuelles transpositions. Les transpositions introduites rendent compte des interversions de caractères à la frappe des textes.

Une description détaillée du modèle théorique VORTEX est donnée dans [LAHENS 86].

1.5 Notre approche : les catégories d'erreurs lexicales dans le système DECOR

Le système DECOR, développé avec les outils de PILAF, traite les erreurs sous une optique la plus linguistique possible. Au niveau

lexical, les erreurs produites dans un texte sont considérées comme appartenant à une des trois catégories : erreurs typographiques, erreurs phonétiques ou erreurs de génération. Le chapitre suivant de ce document contient une présentation détaillée des composants de DECOR.

1.5.1 Les erreurs typographiques

Dans cette catégorie on regroupe les erreurs d'orthographe (par exemple, ***consomne*** pour *consonne*) et les erreurs purement typographiques d'édition de lettres dans un mot.
Comme erreurs purement typographiques, nous avons :

- les erreurs d'omission : une lettre est omise dans le mot (exemple : ***solire*** pour *solaire*) ;
- les erreurs d'insertion : une lettre supplémentaire est insérée dans le mot (exemple ***bâtioment*** pour *bâtiment*) ;
- les erreurs de substitution : une lettre du mot est remplacée par une autre (exemple ***avzncer*** pour *avancer*) ;
- les erreurs de transposition : deux lettres voisines sont échangées (exemple : ***particuleir*** pour *particulier*).

Nous incluons aussi dans cette catégorie les erreurs d'accentuation et les coupures ou liaisons anormales.
Exemples :

- accent aigu : ***ecraser*** pour *écraser* ;
- accent grave : ***siecle*** pour *siècle* ;
- accent circonflexe : ***chaine*** pour *chaîne* ;
- coupure : ***par fois*** pour *parfois* ;
- liaisons anormales : ***s'enaller*** pour *s'en aller*.

En vue des méthodes de correction utilisées, nous traitons les erreurs d'orthographe également comme des erreurs d'omission, d'insertion, de transposition et de substitution de lettres dans un mot.
Par exemple, ***aréoport*** peut être une erreur de transposition du mot *aéroport*, ***assetion*** peut être une erreur d'omission du mot *assertion*, ***chirn*** peut être une erreur de substitution du mot *chien* et ***lidvre*** peut être une erreur d'insertion du mot *livre*.

1.5.2 Les erreurs phonétiques

Cette classe d'erreurs prend en compte la correspondance entre la prononciation et l'orthographe du mot.

A certains phonèmes peuvent correspondre plusieurs graphèmes de la langue française, par exemple :

Phonème		Graphème
[o]	----------	*au , ô , aux , eau , o , ...*
[k]	----------	*k , qu , c , ...*
[an]	----------	*an , en , ans , ent , ...*

Cette relation de correspondance rend fréquentes des erreurs que nous appelons *phonétiques* : la chaîne proposée comme écriture présente des fautes d'orthographe, mais elle garde la forme orale du mot. C'est-à-dire que la prononciation de la chaîne erronée peut être utilisée pour proposer des graphies alternatives.

Ainsi, nous considérons que ***asserslon*** peut être une erreur phonétique de *assertion*, ***okurance*** peut être une erreur phonétique de *occurrence* et ***ortografe*** peut être une erreur phonétique de *orthographe*.
Dans la classification de Catach [CATACH 80c], ce groupe est associé aux *erreurs à dominante phonogrammique n'altérant pas la valeur phonique du mot.*

1.5.3 Les erreurs de génération

Les erreurs dites "de génération" comprennent les situations de méconnaissance de certains éléments grammaticaux, comme les règles de formation du pluriel, les désinences verbales, etc.
Comme exemples, on peut citer ***allerons*** à la place de *irons* ou ***chevals*** à la place de *chevaux*.
Dans la classification de Catach, ces erreurs appartiennent à la catégorie des *erreurs à dominante morphogrammique*.

Parmi les trois types d'erreurs considérés dans DECOR, il n'y a pas de frontière stricte : une erreur d'un certain type peut être considérée comme une erreur d'un autre type. Par exemple, ***asserslon*** peut être vu soit comme une erreur phonétique, soit comme une erreur de substitution de *assertion*.

2 METHODES DE CORRECTION LEXICALE

La correction d'une chaîne de caractères inconnue dans le lexique, et de ce fait, présumée mal écrite, consiste à proposer en remplacement un mot ou une suite de mots la corrigeant au mieux, selon un critère donné [LAHENS 86].

Cette relation entre le mot faux et les mots qui le corrigent amène à une formalisation du concept d'équivalence entre deux chaînes de caractères.

Les travaux de Dominique Maret [MARET 87] dans le domaine des comparaisons de chaînes de caractères et des accès lexicaux tolérants, effectués au Centre de Recherches en Informatique appliquée aux Sciences Sociales (CRISS) - Département Informatique et Mathématiques en Sciences Sociales, Université des Sciences Sociales de Grenoble, contiennent une discussion approfondie de ce problème.

Deux chaînes **a** et **b**, construites sur un même alphabet **A**, seront déclarées équivalentes s'il existe une fonction d'équivalence *f* telle que *f*(a) = *f*(b), *f* étant une fonction n'agissant que sur la forme des chaînes.
Soit **S** un ensemble quelconque de caractères éventuellement égal ou inclus dans **A**.
Soient A^* et S^*, respectivement, les ensembles de toutes les chaînes que l'on peut construire sur **A** et **S**, y compris la chaîne vide.
Plus formellement, *f* est une fonction de A^* vers S^* [MARET 87].

Le problème qui se présente est donc de discuter le choix de la fonction *f*, compte tenu de l'application envisagée.

2.1 Méthodes locales et méthodes globales

Fouqueré [FOUQUERE 88] présente une analyse des méthodes de correction d'erreurs lexicales. L'approche présentée par Fouqueré distingue deux types de méthodes en correction lexicale : les méthodes locales et les méthodes globales.

Les méthodes *locales* (ou méthodes statistiques) sont caractérisées par l'emploi de *règles statistiques* sur la construction des chaînes de caractères formant les mots. La correction des mots consiste à parcourir un réseau de transitions valué. Dans ce cadre, on trouve l'ensemble des méthodes utilisant des n-grammes, dont on peut citer celle mise en oeuvre dans le système VORTEX.

Les méthodes *globales* (parfois nommées méthodes combinatoires) nécessitent un moyen de comparaison direct entre deux chaînes de caractères. Une fonction de comparaison détermine numériquement l'écart qui sépare deux chaînes de caractères. Ces techniques permettent d'extraire du lexique un sous-ensemble de chaînes de caractères dont l'écart avec la chaîne d'entrée est inférieur à un seuil donné.

S'agissant des méthodes globales de correction, le problème général est [FOUQUERE 88] :

*Soit un lexique **L**, où chaque entrée lexicale, ou mot, est une suite de symboles d'un alphabet **A**.*
*Soit **e** la chaîne d'entrée.*
*Comment extraire de **L** l'ensemble des mots **m** "proches" de **e** ?*

Par exemple, soit l'entrée **e** = ***divre***. Alors, il faut extraire du lexique les mots : *livre*, *vivre*, *ivre*, *dire*, parmi d'autres.

Pour résoudre ce problème, on peut avoir comme contraintes, par exemple :

→ rechercher les *x* meilleures solutions

→ rechercher toutes les solutions à un seuil d'éloignement de **e** fixé

Il est encore possible de limiter le nombre de solutions en utilisant des contraintes syntaxico-sémantiques. Par exemple, dans la phrase ***Le bo garçon va à l'école*** la catégorie lexico-syntaxique de la correction de ***bo*** pourrait être restreinte à adjectif.

Les stratégies présentées dans la suite de ce chapitre sont des stratégies globales de correction d'erreurs.

2.2 Les premiers pas vers la correction automatique

Dans un article publié en 1960, Charles Blair [BLAIR 60] affirme que, *si on propose à la machine un critère adéquat pour calculer la "similarité" entre deux mots, elle peut "corriger" une erreur d'orthographe en remplaçant le mot faux par les mots corrects les plus "similaires" à ce premier.*

A cette époque il existe déjà une méthode d'archivage (créée par Remington Rand) qui fait l'association parmi les mots "similaires", en prenant comme critère un code basé sur la prononciation des mots - le code SOUNDEX.
Toutefois, l'approche SOUNDEX de "similarité" associait plusieurs formes correctes, alors que Blair s'intéressait à l'association entre le mot erroné et son équivalent correct unique.
En vue de ces problèmes, Blair établit son propre critère de "similarité", basé sur des **abréviations** : *nous sommes amenés à considérer que deux mots sont similaires quand leurs abréviations sont identiques.*

La méthode proposée par Blair comprend, initialement, la construction des abréviations associées à chaque mot du dictionnaire.

Etant donné que la première lettre est de la plus grande importance, que la dernière lettre est la deuxième en importance et que toutes les autres lettres sont plus importantes à la mesure qu'elles s'approchent de la première et de la dernière, les abréviations sont construites en observant ces règles.
Blair associe aux lettres des poids entiers entre 0 et 7, en fonction de l'intérêt à les rejeter. Il associe aussi des poids aux positions où se trouvent les lettres dans le mot, en fonction de l'intérêt à les rejeter.
Une abréviation de longueur **m** est calculée en effectuant, pour chaque lettre d'un mot de longueur **n**, la somme **s** de ses deux poids associés (respectivement, le poids associé à cette lettre et le poids associé à sa position dans le mot) et en rejetant les **m-n** lettres avec les plus grandes valeurs de **s** associées.
Exemple de calcul d'une abréviation de 4 caractères pour le mot *intéressant* (les poids associés aux lettres sont obtenus dans [BLAIR 60]) :

	I	N	T	E	R	E	S	S	A	N	T
poids associés aux lettres	6	3	3	7	4	7	5	5	5	3	3
poids associés aux positions	0	2	4	5	6	7	6	5	4	3	1
somme des poids	6	5	7	12	10	14	11	10	9	6	4
abréviation calculée		I	N	N	T						

Pour corriger un mot erroné, on calcule son abréviation **c** et on cherche les mots ayant une même abréviation **c**, dans le dictionnaire.
Si à une abréviation correspond, par coïncidence, plus d'un mot, la méthode calcule des abréviations plus longues pour le mot faux et pour les entrées du dictionnaire.

Dans son système, Blair utilise des abréviations composées de 4 lettres, et son processus d'association permet de trouver, normalement, un seul mot correct associé à l'abréviation du mot erroné. Mais il faut bien noter que le dictionnaire utilisé est composé uniquement des 117 mots qui font partie d'un ensemble de test de la méthode appliquée. Dans l'ensemble des 117 mots à corriger, la méthode a identifié correctement 89 mots, soit près de 76 % des erreurs.

Cette méthode, même si elle est simplifiée, présente une caractéristique intéressante : par le biais des abréviations calculées, elle tient compte d'erreurs humaines d'écriture.

2.3 L'édition de chaînes

2.3.1 Les études de Damerau

Damerau [DAMERAU 64] a présenté une des premières études sur le sujet de la correction d'erreurs typographiques dans un texte. En analysant la nature des fautes d'orthographe trouvées dans un texte, il a pu remarquer que 80 % des mots faux présentaient *une seule faute*, qui peut être : la substitution d'une lettre par une autre, l'omission d'une lettre, l'insertion d'une lettre ou la transposition de deux lettres.

Damerau a proposé une méthode de correction pour ce genre de fautes : un mot faux est corrigé à partir de nouvelles recherches dans le dictionnaire, en supposant à chaque fois l'existence d'un des types de faute d'omission, d'insertion, de substitution ou de transposition et en essayant de les éliminer.

Premièrement, le nombre de caractères du mot faux est comparé aux nombre de caractères de l'entrée du dictionnaire.
Si la différence est *plus grande que 1* alors l'entrée en analyse ne peut pas être une correction pour le mot faux en question.
Si les longueurs sont acceptables, les caractères des deux mots sont comparés.
Si les caractères des deux mots diffèrent par plus de deux lettres, aucune comparaison n'est possible.
Ces deux tests ont pour objectif d'éviter des opérations inutiles qui consommeront du temps sans mener à un résultat satisfaisant.

Si la comparaison est considérée comme possible, les deux chaînes sont donc comparées caractère à caractère.
Suivant le nombre de caractères des deux chaînes, l'algorithme prend un des trois chemins :

C1-	les deux chaînes ont le même nombre de caractères ;
C2-	le mot faux est le plus long ;
C3-	l'entrée du dictionnaire est la plus longue.

Les chaînes ayant le même nombre de caractères et présentant la différence d'une seule position sont considérées comme étant le même mot.

Les essais faits par Damerau avec sa méthode de correction ont permis de corriger 95 % des erreurs d'insertion, d'omission, de substitution ou de transposition trouvées dans un texte.

2.3.2 La distance de Levenshtein et l'algorithme de Wagner et Fischer

Dans le domaine de la similarité, des techniques plus abstraites ont été proposées, comme la distance de Levenshtein, proposée par son auteur dans le périodique *Soviet Physics Doklady* nº 8 de 1966. Cette distance se définit comme le coût minimum nécessaire pour transformer une chaîne en une autre, à l'aide des opérations d'insertion, d'omission, de substitution ou de transposition.

Wagner et Fischer [WAGNER 74] ont développé un algorithme pour calculer la distance de Levenshtein dans le cas où les transpositions sont exclues.
L'algorithme donné utilise la programmation dynamique.

Ces auteurs ont analysé le problème de la distance entre deux chaînes en proposant un coût pour les opérations d'édition qui transforment une chaîne en une autre.
Les opérations d'édition considérées sont la substitution, l'insertion et l'omission de caractères.

Soit **x** l'opération de substitution d'une lettre par une autre. Exemple : *a* **x** *q* représente la substitution de *a* par *q* dans ***chqîne***.
Soit $\Sigma+$ l'opération d'insertion d'une lettre dans un mot. Exemple : $\Sigma+r$ représente l'insertion de la lettre *r* dans ***livrre***.
Soit $\Sigma-$ l'opération d'omission d'une lettre dans un mot. Exemple : $\Sigma-r$ représente l'omission de la lettre *r* dans ***vere***.
Exemple :
La transformation de la chaîne ***eaukurense*** en *occurrence* peut être obtenue par la séquence suivante d'opérations d'édition :

e **x** o, Σ +a, Σ +u, c **x** k, Σ -c, Σ -r, c **x** s

Wagner et Fischer ont associé un coût à chacune de ces opérations et ils ont proposé un algorithme pour déterminer la distance minimale d'édition entre deux chaînes.

Soient deux chaînes $x = x_1,\ldots, x_n$ et $y = y_1,\ldots, y_p$.
Désignons par x_i et y_j leurs préfixes respectifs d'ordre i et j (i dans x et j dans y correspondant à l'indice de la position où il y a une erreur), et par *C* une fonction de pondération.

$C(x_i, y_j)$ représente le coût associé à la substitution du caractère x_i par y_j.

Soit **d(x,y)** la distance entre les chaînes **x** et **y**.
L'algorithme de calcul de la distance **d(x,y)** est basé sur la relation :

$$d(\lambda, \lambda) = 0$$

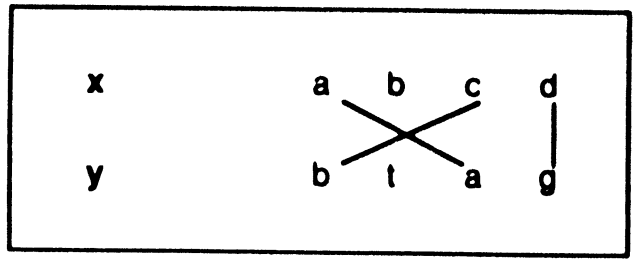
$$d(x_i, y_j) = \min \{ \begin{array}{l} d(x_{i-1}, y_{j-1}) + C(x_i, y_j), \\ d(x_{i-1}, y_j) + C(x_i, \lambda), \\ d(x_i, y_{j-1}) + C(\lambda, y_j) \end{array} \}$$

où λ désigne la chaîne vide, et $C(x_i,y_j)$, $C(x_i,\lambda)$ et $C(\lambda,y_j)$ sont les coûts respectifs de substitution de x_i par y_j, d'omission de x_i et d'insertion de y_j.

2.3.3 L'extension de Lowrance et Wagner

Lowrance et Wagner [LOWRANCE 75] ont proposé une extension pour l'algorithme donné précédemment. L'extension proposée prend en compte les transpositions entre deux caractères adjacents.

Soient w_i, w_o, w_s et w_t, respectivement, les poids associés à l'insertion d'un caractère, à l'omission d'un caractère, à la substitution d'un caractère par un autre et à la transposition de deux caractères adjacents.
Soit **s** une séquence d'opérations d'édition.
Un *tracé* est un diagramme qui permet d'arriver à une chaîne **y** à partir d'une chaîne **x**, comme le montre le dessin :



Ce tracé présente une méthode de transformation de **x** en **y** :

- Si $x_i \neq y_j$, une ligne de x_i à y_j dénote que x_i doit être changé en y_j.
- Si $x_i = y_j$, une ligne de x_i à y_j signale que x_i ne doit pas être changé (il devient y_j).
- Les caractères de **x** non touchés par une ligne sont à omettre et les caractères de **y** non touchés par une ligne sont à insérer.

Soit |a| le nombre de caractères de la chaîne **a** (|a| peut être égal à zéro).
Formellement, T = [U, **a**, **b**] est un tracé de **a** à **b** si :

1) $U \subseteq X_a \times X_b$

où $X_c = \{ i \mid i \text{ est entier et } 1 \leq i \leq |c| \}$

2) si $(i_1, j_1) \in U$ et $(i_2, j_2) \in U$, alors

$i_1 = i_2$ si et seulement si $j_1 = j_2$.

Alors, X_a et X_b sont des ensembles de points dérivés de **a** et de **b**, et U est une collection de paires ordonnés (i, j) interprétés comme des segments de ligne qui vont de a_i à b_j.

Si $u \in U$, on établit $u = (u_1, u_2)$.
Soit $u = (i, j) \in U$. Nous disons que *u est une ligne balancée de U* si $a_i = b_j$. Nous disons que *u est une ligne non balancée de U* si $a_i \neq b_j$.

A partir de ces propriétés, nous pouvons présenter la définition de ***tracé restreint*** donnée par Lowrance et Wagner.
Un tracé est T restreint s'il satisfait les propriétés :

i) aucune ligne de T n'intercepte plus d'une autre ligne de T ;

ii) toute ligne de T qui intercepte une autre ligne est *balancée* ;

iii) si $u = (u_1, u_2)$ et $v = (v_1, v_2)$ sont des lignes de T qui s'interceptent, avec $u_1 < v_1$, il n'existe pas d'entiers i (ou j) tels que

1) $u_1 < i < v_1$ et $a_{u_1} = a_i$ ou

2) $v_2 < j < u_2$ et $b_{v_2} = b_j$.

Lowrance et Wagner concluent que, si $w_i = w_o = w_s = w_t$, le problème de trouver une séquence de longueur minimum d'opérations d'édition se réduit à :

trouver le tracé restreint de coût minimum
pour lequel les poids sont $w_i = w_o = w_s = w_t = 1$.

Dans [LOWRANCE 75] les auteurs présentent un algorithme qui résout ce problème en trouvant un tracé restreint de coût minimum quand $2 w_t \geq w_i + w_o$.

2.3.4 Autres propositions dans le domaine de la similarité

Quand la similarité doit être calculée sur un grand dictionnaire et en temps réel, l'algorithme de Wagner et Fischer et l'algorithme de Lowrance et Wagner présentent des problèmes de coût. Ces problèmes ont été analysés par Owolabi et McGregor [OWOLABI 88], qui proposent un processus en deux étapes pour calculer la similarité entre deux chaînes de caractères.
La première étape utilise une table de n-grammes compacte pour sélectionner un ensemble de chaînes *grossièrement similaires*. La deuxième étape compare ces dernières avec la chaîne d'entrée. Pour réaliser la comparaison, les auteurs ont défini une nouvelle mesure de similarité, basée, comme les précédentes, sur la métrique de Levenshtein.

Le processus proposé par Owolabi et McGregor présente deux caractéristiques intéressantes :

- il réduit l'espace mémoire nécessaire au stockage des n-grammes et
- il passe à la programmation dynamique seulement l'ensemble des chaînes isolées par le filtre initial.

Selon ces auteurs, des tests effectués sur un dictionnaire de 25000 mots ont apporté de bonnes performances, surtout quand le dictionnaire est ordonné par rapport à leur clé de recherche.

Okuda, Tanaka et Kasai [OKUDA 76] ont proposé une autre version de correction basée sur la distance de Levenshtein et la distance de Levenshtein pondérée.
Leur étude, apparue à la même époque que les travaux de Lowrance et Wagner, apporte en plus la capacité de corriger les erreurs de transposition de caractères.
Leur méthode permet la correction d'une ou plusieurs fautes d'insertion, d'omission ou de substitution dans la chaîne de caractères. Cela rend possible la correction des erreurs de transposition de caractères (on peut considérer une erreur de transposition comme une séquence d'opérations d'édition).

Soient **x** et **y** deux mots et soit **WLD** une fonction de distance entre ces deux mots. Si **x** est transformé en **y** après la substitution de k_i lettres, l'insertion de m_i lettres et l'omission de n_i lettres, la distance de **x** à **y** est défini par :

$$WLD(x \rightarrow y) = \min_{i} (pk_i + qm_i + m_i) \qquad (1)$$

... où p, q et r sont des poids non négatifs associés aux opérations de substitution, d'insertion ou d'omission de caractères.

Alors, nous avons :

$$WLD(y \rightarrow x) = \min_{i} (pk_i + qm_i + m_i) \qquad (2)$$

Pour résoudre ce problème, les auteurs proposent soit l'utilisation de la programmation dynamique, soit une réalisation par "hardware".
Dans le cas d'une résolution par "hardware", les poids des branches représentent les temps de délai proportionnels à p, q et r.
Dans [OKUDA 76], Okuda, Tanaka et Kasai montrent comment mettre en oeuvre un *circuit de délais* correspondant à une branche de la solution.
Pour une réalisation pratique d'un tel système de correction d'erreurs, chaque mot dans le dictionnaire doit avoir son propre décodeur ainsi qu'un détecteur pour identifier le décodeur dont la sortie présente le temps de délai le plus faible.
Ce circuit serait, selon leur opinion, réalisable dans un futur proche. Toutefois, nous ne connaissons aucune annonce d'une telle réalisation jusqu'à présent.

2.4 Les clés de similarité

Le concept de clé de similarité a pour objectif de produire, à partir d'une chaîne **a** donnée, une autre chaîne **c**, dénommée clé, qui concentre les principales caractéristiques de **a**.
La clé **c** est obtenue à partir de **a** en appliquant des règles de transformation.

Par exemple, la séquence ***mnndtnloa*** peut être une clé associée à la chaîne ***lmnondation***. Dans ce cas, nous avons appliqué deux règles de transformation :

1 PRENDRE LES CONSONNES DANS LEUR ORDRE D'APPARITION ;

2 CONCATENER LES VOYELLES DANS LEUR ORDRE D'APPARITION, SANS REPETITION .

La technique des clés de similarité est utilisée pour compenser des variations dans la forme des mots. L'une des deux chaînes, **a**, subit un certain nombre de déformations et c'est la forme **b**, entachée d'erreurs, qui est connue.

b est comparée à une forme de référence, issue d'un lexique, et l'identification de **a** est réalisée par la forme de référence équivalente à **b**.

Par exemple, soit **a** la chaîne *chapeau* et soit **b** la mauvaise graphie ***hcapeau*** utilisée à la place de la chaîne correcte **a**. Soit C_b la forme de référence (c'est-à-dire, la clé) associée à la chaîne erronée **b** et soit C_a la forme de référence ou clé associée à la chaîne correcte **a**. La clé C_b est utilisée pour identifier **a**.

Utilisons, comme transformations, les règles 1 et 2 données ci-dessus. HCPAEU sera donc la clé associée à ***hcapeau*** et CHPAEU sera la clé associée à *chapeau*. A l'aide d'une transposition des deux premières lettres de la clé HCPAEU, nous pouvons identifier la clé CHPAEU, qui fait référence à une entrée du lexique. Cette entrée permettra de proposer une correction (*chapeau*) pour le mot faux.

Parmi les études qui ont été réalisées à propos de la similarité entre chaînes de caractères, une des premières doit probablement être celle présentée par Blair [BLAIR 60] (paragraphe 2.2 de ce chapitre). Les résultats apportés, plus tard, par Riseman et Hanson [RISEMAN 74], contiennent des conclusions importantes dans le domaine des clés de similarité.

Comme résultat, ces auteurs énoncent que deux propriétés importantes d'une chaîne sont l'identité et l'inter-relation des lettres qui la composent.
Ces deux critères sont antagonistes : la clé doit être suffisamment discriminante, pour bien garder l'identité de la chaîne, en même temps qu'elle doit être insensible aux erreurs qui ont pu se produire. Plus la clé retient d'informations, plus son pouvoir discriminant est élevé (à la limite, la clé est égale à la chaîne initiale!), mais plus sa sensibilité aux erreurs est accrue.

En utilisant ces propriétés, on peut envisager une série de clés de similarité qui commence par la clé la plus rigoureuse (la chaîne elle-même) et diverge vers la ressemblance la plus faible (par exemple : le même nombre de caractères).

Prenons par exemple la clé constituée par une seule occurrence des lettres composantes du mot faux dans leur ordre original (comme CHAPEU pour le mot *chapeau*). Cette clé est relativement précise car elle conserve beaucoup d'informations de la chaîne originale.
Une clé plus rigide serait celle qui conserve toutes les inter-relations originales (exemple : CHAPEAU associée à *chapeau*).

2.4.1 Les travaux de Pollock et Zamora et la clé squelette

Pollock et Zamora [POLLOCK 84] ont développé un algorithme de correction de fautes d'orthographe qui fait partie de SPEEDCOP (Spelling Error Detection/Correction Project) - un projet supporté par la *National Science Foundation* des Etats-Unis.

Selon ces auteurs, le problème de la correction de fautes d'orthographe peut être traité par deux stratégies différentes - une stratégie absolue ou une stratégie relative.

Si on utilise une méthode absolue, la correction est déduite directement des caractéristiques du mot faux. Un dictionnaire n'est utilisé que pour confirmation.
Dans les méthodes relatives, la forme corrigée est sélectionnée dans un dictionnaire.

La méthode absolue plus simple est l'approche historique. Cette approche demande la construction d'un dictionnaire de formes mal orthographiées non-ambiguës qui ont été fréquemment utilisées auparavant.
Par exemple, si la forme ***chqîne*** est fréquemment utilisée à la place de *chaîne*, alors au moment où cette première est trouvée elle peut être transformée automatiquement en *chaîne*. Cette méthode est aussi incorporée dans le programme de correction de SPEEDCOP.
Des méthodes absolues moins extrêmes utilisent les caractéristiques associées aux lettres de la forme mal orthographiée. Par exemple, trois occurrences de la même lettre en séquence (comme *ttt* dans ***chattte***) pourraient être substituées par deux occurrences seulement de cette lettre (en produisant la forme ***chatte***) et la nouvelle forme pourrait être directement acceptée si elle se trouve dans le dictionnaire.

La stratégie relative est caractérisée par l'utilisation d'un dictionnaire où des clés de localisation sont associées aux mots. Ces clés permettent de trouver une correspondance entre les mots mal orthographiés et les mots dans le dictionnaire. La correction est sélectionnée parmi les correspondants dans le dictionnaire.

Dans SPEEDCOP, une clé de similarité est engendrée pour chaque mot dans le dictionnaire. Les mots du dictionnaire sont alors classés dans l'ordre des clés engendrées.
L'algorithme de correction calcule une clé pour la forme mal orthographiée, et cherche dans le dictionnaire les mots dont les clés sont proches de la clé calculée. De cet ensemble de mots, une ou plusieurs alternatives sont sélectionnées comme correction.

Pollock et Zamora ont proposé une clé spécifique, appelée clé squelette, en gardant la première lettre, puis en concaténant les consonnes (une seule occurrence) dans leur ordre d'apparition et en concaténant de la même manière une seule occurrence des voyelles du mot mal orthographié.

Voici quelques exemples de chaînes avec les clés respectives :

chaîne	**clé**	*erreur introduite*
OCCASION	OCSNAIO	
OCASION	OCSNAIO	(omission du C)
OCAZION	OCZNAIO	(substitution du S par Z)
OCCASIN	OCSNAI	(omission du deuxième O)
INONDER	INDROE	
INNONDER	INDROE	(insertion du N)
IMNONDER	IMNDROE	(insertion du M)
AEROPORT	ARPTEO	
AEROPOR	ARPEO	(omission du T)
AREOPORT	ARPTEO	(transposition entre R et E)
AEEROPORT	ARPTEO	(insertion du E)
AEORPORT	ARPTEO	(transposition entre O et R)
AEROPOFT	ARPFTEO	(substitution du deuxième R par F)
AIROPORT	ARPTIO	(substitution du E par I)

De cette liste ressort que les variantes mal orthographiées du mot *aéroport* ont souvent la même clé que *aéroport* (ARPTEO) ou une clé proche (exemple : ARPTIO, ARPEO). La recherche dans le dictionnaire des mots ayant la même clé ou une clé proche de *aréoport*, permettra de trouver ce mot parmi les possibilités de correction.

Généralement le mot mal orthographié n'a pas exactement la même clé que sa correction (AREOPORT et AEROPORT constituent un exemple extrême), mais leurs clés sont proches l'une de l'autre. Par exemple, la clé squelette calculée pour ***Imnonder*** (IMNDROE) diffère de la clé squelette calculée pour *inonder* (INDROE) par une lettre.

Selon Pollock et Zamora, quatre raisons justifient la création de ce type de clé :

(1). La première lettre est la plus rarement mal orthographiée.
(2). Les consonnes portent plus d'informations que les voyelles.
(3). L'ordre des consonnes originales est très souvent conservé.
(4). Doubler ou dédoubler les lettres ne change pas la clé et la plupart des transpositions ne la changent pas, non plus.

L'aspect le plus discutable de cette clé est l'importance donnée aux consonnes du début du mot. Plus une consonne incorrecte est proche du début du mot, plus la clé du mot faux et la clé de la correction seront distantes dans le dictionnaire.
Exemples :

I) Soit ***actre*** une mauvaise écriture du mot *astre*.
La clé associée au mot faux est ACTRE, alors que la clé associée à la correction est ASTRE.
Au cours d'une collation, la distance entre les deux clés peut rendre impraticable la correction par cette méthode.

II) Soit ***satre*** une erreur de transposition du mot *astre*.
Dans ce cas, la clé calculée (SATRE) et la clé associée au mot correct (ASTRE) sont excessivement éloignées, ce qui rend ici la méthode inapplicable.

Suivant les observations rapportées dans [POLLOCK 84] :
- le programme SPEEDCOP a corrigé 85 à 95 % des erreurs pour lesquelles il était prévu (les erreurs élémentaires d'édition : insertion, omission, transposition ou substitution), en considérant des mots ayant au maximum une erreur et
- SPEEDCOP a corrigé 65 à 80 % du total des erreurs trouvées.

2.4.2 Les alphagrammes

Un autre type de clé de similarité est celui des **alphagrammes**. Imaginé et mis en oeuvre pour le français par Débili, Andreewsky et Fluhr en 1978, ce type de clé a été repris en 1986 par Débili et son équipe [DEBILI 86] pour l'étendre à l'arabe.

Une constatation est à l'origine de cette clé : *les anagrammes sont rares en français*. Or, avec un même ensemble de caractères on ne peut obtenir, en général, qu'un seul mot en français.

Un **alphacode** est une chaîne constituée par l'ensemble des lettres qui composent un mot, classées en ordre alphabétique.
Débili et son équipe ont observé que 85 % des alphacodes du français ne peuvent produire qu'un seul mot. Ils ont utilisé les alphacodes, alors, comme clé de similarité en vue de la correction d'un mot erroné.
En extension au concept d'alphacode et par analogie aux termes digramme, trigramme, n-gramme, les auteurs suggèrent d'appeler **alphagramme** la liste des caractères qui constituent une graphie quelconque, pris dans l'ordre alphabétique (l'*alphagramme* est

applicable non seulement aux mots corrects mais également aux graphies erronées).
Exemples :

graphie	*alphagramme*
papier	aeippr
porte	eoprt
arc	acr
car	acr
premettre	eeemprrtt

On appelle ***cardinal*** *d'un alphagramme* le nombre de mots qu'il permet d'obtenir (par exemple, le cardinal de **acr** est égal à 2).
Un alphagramme est ambigu si son cardinal est supérieur à 1.

En constatant que les anagrammes sont rares, Débili et son équipe remarquent que les alphagrammes ambigus sont par conséquent rares en français de même qu'en anglais ou en arabe (les mesures de cette rareté sont présentées dans [DEBILI 86]). Les mots de ces lexiques seraient donc caractérisés presque exclusivement par les lettres qui les forment.

Conformément à [DEBILI 86], la méthode de correction proposée permet le traitement des suppressions d'un ou de deux caractères (comme dans ***lorsqe*** ou ***documention***), de la transposition de deux caractères adjacents (comme dans ***naltonale***), de la substitution d'un caractère (comme dans ***btessé***), de l'insertion d'un ou de deux caractères (comme dans ***satellllites*** ou ***démultitiplication***), de la combinaison *transposition + omission* (comme dans ***portetion*** au lieu de *protection*) et de la combinaison *transposition + insertion* (comme dans ***premlettre*** au lieu de *permettre*).
Toutefois, nous observons que les règles employées pour la correction des erreurs de *transposition + omission* et des erreurs de *transposition + insertion* se sont basées sur un ensemble réduit de règles de réécriture du mot erroné (voir l'algorithme extrait de [DEBILI 86] donné ci-après).

L'algorithme proposé pour la correction des mots erronés prévoit la construction préliminaire d'un dictionnaire contenant les alphagrammes associés à un lexique donné. Ces alphagrammes sont non-accentués et à chaque alphagramme est associé l'ensemble de mots accentués qui sont à son origine.
La méthode est opérationnelle au CIRCE (Conseil d'Etat). Nous ne connaissons pas des données à propos de l'organisation ou de la taille du dictionnaire utilisé par cette application.

Pour corriger un mot erroné, l'algorithme employé est le suivant :

1) calcul de l'alphagramme α de la graphie erronée ;
2) accès, par le moyen de cette clé, à l'ensemble des mots du dictionnaire dont l'alphagramme associé est α (nous obtenons ainsi un premier ensemble de graphies qui peuvent remplacer le mot faux, lequel tient compte des erreurs de transposition) ;
3) génération, à partir de la clé α, des alphagrammes obtenus par omission, insertion ou substitution, lesquels permettent d'accéder à un deuxième ensemble de graphies ;
4) transformation du mot erroné en utilisant quelques règles de réécriture :

ai	→	é, è, ê
ct	→	x
ph	→	f
qu	→	c
qu	→	k
ss	→	c

Les nouvelles graphies obtenues ont leurs alphagrammes calculés et ces alphagrammes sont utilisés comme clés d'accès aux mots du dictionnaire, en vue d'un enrichissement de l'ensemble de solutions proposées.

Exemple :

Soit ***trionphalement*** la graphie à corriger.

L'alphagramme α calculé pour cette graphie est **aeehilmnnoprtt.**

On accède aux mots du dictionnaire dont l'alphagramme associé est **aeehilmnnoprtt** et on obtient l'ensemble E_1 de corrections.

A partir de la clé α on engendre des nouvelles clés. On obtient :

- 14 clés par suppression (longueur de α = 14 caractères) ;
- 26 clés par insertion de lettres ;
- 14 * 25 clés par substitution.

Les 390 nouveaux alphagrammes obtenus permettent d'accéder à un deuxième ensemble E_2 de mots du dictionnaire.

Finalement, une règle de réécriture est appliquée au mot erroné (la règle *ph* → *f*) et nous obtenons la graphie ***trionfalement***, dont l'alphagramme associé est **aeefilmnnortt**. L'accès aux mots du dictionnaire ayant l'alphagramme **aeefilmnnortt** associé permet d'obtenir un troisième ensemble E_3 de corrections à proposer.

L'ensemble de corrections proposées par la méthode sera $E_1 \cup E_2 \cup E_3$.

Les auteurs s'intéressent, aussi, au problème du classement des graphies (que nous appellerons simplement *candidates*), ces graphies étant celles qui sont candidates à la correction.
A chaque candidate est associée une note. L'algorithme de calcul de cette note prend en compte les sous-chaînes communes entre la graphie fautive et la candidate à la correction, la longueur de ces sous-chaînes et l'égalité entre les premières et les dernières lettres des deux graphies comparées.

2.5 Des mesures de similarité

En correction orthographique, quelle que soit la stratégie de correction utilisée, un problème se pose au moment de présenter à l'utilisateur l'ensemble de candidates obtenues : *dans quel ordre les candidates doivent être présentées ?*
Cette question est à l'origine d'une autre : *comment mesurer la similarité entre une candidate et le mot à corriger ?*
Ce problème est parfois traité par la méthode de correction elle-même, qui peut engendrer des corrections avec des poids associés. Mais il est aussi objet de recherches.

En mesurant la similarité entre deux chaînes de caractères, comme le rappellent Angell, Freund et Willet [ANGELL 83], trois critères peuvent être analysés :

- **l'identité** des caractères (*combien de caractères identiques y-a-t-il dans les chaînes comparées ?*) ;

- **l'ordre** des caractères dans les deux chaînes (*les caractères identiques, sont-ils dans le même ordre ?*) et

- **la position** des caractères identiques (*combien de caractères, parmi les caractères identiques, sont en positions correspondantes ?*).

La correction par similarité ordinale a été déjà largement utilisée (exemple : dans le système de correction proposé par Blair [BLAIR 60], dans SPEEDCOP [POLLOCK 84]). Une abréviation est engendrée pour chaque mot dans le dictionnaire, et on suppose qu'un mot erroné aura la même abréviation ou une abréviation relativement proche de son correspondant correct.

La *similarité matérielle* est une approche qui concerne une définition beaucoup moins rigide de similarité, proposée par Angell, Freund et

Willet [ANGELL 83] et basée sur les trigrammes communs entre un mot faux et un terme du dictionnaire.
Ce concept de similarité a été aussi employé par Adamson et Boreham [ADAMSON 74], dans le développement d'une technique de classement automatique basée sur la structure des mots. A la place des trigrammes, Adamson et Boreham utilisent des digrammes, mais le coefficient de similarité choisi est le même (le coefficient de Dice).

Soit **d** un mot du dictionnaire contenant n caractères. Supposons que, en vue des recherches futures à réaliser sur le dictionnaire, **d** doit être stocké sous la forme d'une liste de sous-chaînes de longueur λ dans **d**.
Alors, si nous insérons $\lambda-1$ espaces devant et derrière **d** pour assurer que chaque caractère apparaît en exactement λ sous-chaînes, **d** peut être représenté par le vecteur

$$(d_1, d_2, \ldots\ldots, d_{n+\lambda-1})$$

où d_i $(1 \leq i \leq n+\lambda-1)$ correspond à la sous-chaîne qui commence par le $i^{ème}$ caractère dans la chaîne expansée.

Un mot mal orthographié **m** de longueur n' peut être décrit par un vecteur similaire : $(m_1, m_2, \ldots\ldots, m_{n'+\lambda-1})$ contenant des sous-chaînes m_j $(1 \leq j \leq n'+\lambda-1)$.

Une mesure de similarité entre **d** et **m** peut être obtenue par le calcul de c, qui est le nombre de sous-chaînes dont $d_i = m_j$, où ni d_i ni m_j ont été présentes dans une combinaison antérieure.
Le coefficient utilisé est :

$$2c/(n+n')$$

Si S_d note la similarité entre **m** et **d**, alors la correction associée à **m** doit être l'option qui maximise S_d :

$$\max \{S_d\}$$

Nous présentons un exemple d'application de la méthode.
Soit ***carttable*** le mot à corriger et *cartable* une entrée du dictionnaire.
La liste de trigrammes associés à ***carttable*** sera (le symbole § représente un espace) :

§§C, §CA, CAR, ART, RTT, TTA,
TAB, ABL, BLE, LE§, E§§

La liste de trigrammes associés à *cartable* sera :

§§C, §CA, CAR, ART, RTA,
TAB, ABL, BLE, LE§, E§§

La première liste contient 11 trigrammes et la deuxième liste contient 10 trigrammes. Les trigrammes communs sont au nombre de 9 :

§§C, §CA, CAR, ART, TAB, ABL, BLE, LE§, E§§

Alors la valeur du coefficient de similarité sera (2 * 9) / (10 + 11) soit ≈ 0,86.

Si 0,86 est le coefficient le plus grand parmi les coefficients calculés, alors le mot ***cartable*** sera choisi comme une correction pour le mot faux ***carttable***.

Les auteurs observent que le calcul et l'espace disque exigés pour obtenir et stocker les inversions associées aux entrées du dictionnaire sont les points critiques de la méthode exposée.

Angell, Freund et Willet ont aussi analysé la longueur des mots par rapport au type de faute qu'ils présentent [ANGELL 83]. L'analyse a été réalisée à partir d'une base de 1718 mots erronés obtenus de trois sources : SHEFFIELD (382 mots contenant des erreurs de frappe collectionnées dans le *Departement of Information Studies* de l'Université de Sheffield, Angleterre), AMERICAN (788 mots erronés obtenus de sources américaines diverses) et BRADFORD (548 mots erronés collectionnés à l'Université de Bradford dans un autre projet de correction orthographique). Selon ces auteurs :

- la longueur moyenne des mots erronés est de 8,4 caractères ;
- des erreurs de transposition arrivent dans des mots plus courts, avec une longueur moyenne de 7,6 caractères ;
- de multiples erreurs ont tendance à apparaître dans des mots plus longs, avec une longueur moyenne de 8,8 caractères.

2.6 Le traitement d'erreurs dans des mots composés

Parmi les études exposées dans ce document, nous observons que très peu d'intérêt est porté au traitement des erreurs dans des mots composés.

Par la "composition", la langue forme des mots nouveaux, soit en combinant des mots simples avec des mots déjà existants, soit en faisant précéder ces mots simples de syllabes sans existence propre (exemples : contredire, gendarme, hôtel de ville, chou-fleur). Un mot est composé dès le moment où il évoque dans l'esprit, non des images distinctes répondant à chacun des mots composants, mais une image unique [GREVISSE 69].

Les mots composés sont constitués par des séquences qui incluent un ou plusieurs mots simples (par exemple : *sauf-conduit, cerf-*

volant, caméra vidéo, larmes de crocodile, chiffre d'affaires, cheval vapeur, pomme de terre, etc), dont la notion de composition est fondée soit sur des critères formels, soit sur des considérations lexicales.

Laporte et Silberztein [LAPORTE 89] présentent une série de considérations à propos de la vérification orthographique lexicale automatique des mots composés. Parmi ces considérations, nous rappelons :

I Il est difficile de distinguer un mot composé d'une expression quelconque, une fois que certains mots composés peuvent contenir plusieurs blancs comme séparateurs (exemple : *salle à manger*).

II Le trait d'union n'est pas une marque qui appartient exclusivement aux mots composés (exemple : *rendez-vous*).

III L'apostrophe peut être utilisé dans des mots composés mais aussi ailleurs (exemple : *d'abord, qu'il vienne*).

IV Certaines séquences, même si nous les traitons comme des mots simples, sont considérées traditionnellement comme des mots composés (exemple : *électromagnétique*).

Laporte et Silberztein évaluent à 300000 le nombre de mots composés en français, sans compter les termes techniques.
Cependant il est notable que le phénomène de composition dans la langue française manque encore de systématisation.

En ce qui concerne la correction des erreurs dans des mots composés, Frisch et Zamora [FRISCH 88] proposent une méthode d'assistance à la correction orthographique des mots composés en allemand. Bien que les règles de composition soient plus strictes en allemand (normalement par agglutination), les possibilités de composition restent très nombreuses, ce qui présente des difficultés de stockage et d'accès.

La technique proposée consiste à analyser un mot inconnu en vue de la détermination de ses composants, à l'aide d'un dictionnaire qui associe aux composants des mots des codes décrivant leurs particularités de composition (par exemple, ***franco*** dans *franco-russe*, *aéro* dans *aérogare, aéroport* ou *aérosol*).
La réalisation de Frisch et Zamora prend en compte des transformations morphologiques spécifiques de l'allemand.
Les résultats obtenus sont spécialement affectés par :

- la procédure de décomposition utilisée ;
- les procédures de traitement d'erreurs des mots simples ;
- les codes de composition stockés dans le dictionnaire.

Les auteurs remarquent que l'assistance à la correction des mots composés est moins efficace que celle touchant les mots simples (leur méthode a corrigé 70 % des erreurs dans les tests effectués).

3 ERREURS SYNTAXIQUES

3.1 Les erreurs syntaxiques et leur traitement

Les erreurs syntaxiques dans un texte constituent un problème plus délicat, pour deux raisons :

a) leur traitement suppose un traitement morphologique antérieur (c'est-à-dire que le vérificateur/correcteur syntaxique doit disposer d'un ensemble consistant de renseignements linguistiques sur chaque mot) ;

b) leur traitement suppose souvent un traitement sémantique : il est fréquent de trouver des situations d'impasse quand on arrive à la frontière entre Syntaxe et Sémantique - les traits morpho-syntaxiques sont parfois insuffisants pour arriver à une vérification/correction syntaxique précise.

Analysons comme exemple la phrase :

Les comédiens qu'on a empêchés de jouer.

Supposons que nous nous intéressons ici à l'examen de l'accord du participe passé.
Nous trouvons dans cette phrase un participe passé (*empêchés*) suivi d'un infinitif (*jouer*).
Selon Grevisse, l'accord du participe passé conjugué avec le verbe *avoir* et suivi d'un infinitif est réglé en tenant compte du rapport entre ce participe et le pronom qui le précède [GREVISSE 69]. Or, le participe s'accorde lorsque le pronom objet direct *que* se rapporte à ce participe. Il faut donc décider si *empêchés* se rapporte oui ou non à *les comédiens*, pour savoir si la phrase est correcte.

Dans ce cas on pourrait considérer comme "sémantiques" les renseignements exigés à propos du verbe *empêcher* : ce sont les verbes de perception (*écouter, entendre, sentir, regarder, voir, ...*) et quelques autres (*laisser, envoyer, empêcher,...*) qui se comportent d'une telle manière quand ils sont suivis d'un infinitif pur.

J. Carbonell et P. Hayes [CARBONELL 83] font des observations en ce qui concerne la couverture d'une analyse syntaxique, dans des interfaces en langue naturelle. Ils constatent que, dans une application comme le traitement de textes, les erreurs grammaticales apparaissent moins souvent - les textes sont soigneusement préparés et édités, ce qui élimine la plupart des erreurs grammaticales.
Ceci n'est pas valable pour les systèmes acceptant un langage produit spontanément par l'utilisateur, qui auront besoin d'une analyse syntaxico-sémantique beaucoup plus robuste.

Veronis [VERONIS 88c] fait remarquer qu'il n'est pas primordial de corriger les erreurs de performance aux niveau syntaxique.
Ces erreurs (mots oubliés, intervertis, ajoutés) sont des erreurs que l'utilisateur sait corriger lui-même après une lecture de son texte, tandis qu'on aurait intérêt à traiter automatiquement les erreurs de compétence orthographique de l'utilisateur détectées au niveau syntaxique (comme les erreurs d'accord).

Selon Emirkanian et Bouchard [EMIRKANIAN 88a, EMIRKANIAN 88b], l'expérience a montré que les erreurs de syntaxe sont relativement peu fréquentes en français.
Ces auteurs ont réalisé une étude à propos de la syntaxe dans des textes écrits par des enfants en école primaire. Dans 6580 phrases analysées, seulement 79 (1,2 %) étaient non grammaticales.
Le problème le plus fréquemment trouvé a été l'utilisation de la subordination (53 % des erreurs).
Nous observons que ces données ne se vérifient certainement pas dans de textes écrits par des étrangers en apprentissage de la langue française.

La construction des structures syntaxiques de dépendance associées à une phrase permet de procéder à une détection d'erreurs au niveau syntaxique.
Nous pouvons effectuer cette détection en vérifiant l'adéquation d'une phrase aux règles de syntaxe de la langue.
Cependant, au point de vue pratique, l'état de l'art du traitement des langues naturelles ne permet pas encore l'analyse complète d'une langue : la plupart des systèmes n'en traitent que des sous-ensembles qui, dans la plupart des cas, sont déterminés pour un domaine d'application particulier dont une analyse de corpus permet de délimiter le vocabulaire et la couverture de la grammaire [LACOUTURE 88].

Dans un système visant à la correction syntaxique automatique de la langue française, le traitement des fautes d'accord entre les éléments

composants d'une phrase est normalement envisagé. Toutefois, la détection et correction des fautes d'accord doivent être subordonnées à une analyse syntaxique puissante qui fournisse les éléments morphologiques, syntaxiques et même sémantiques nécessaires à la vérification des accords.

3.2 Des réalisations en vérification et correction syntaxique

Des règles d'accord au niveau du groupe nominal (accord article-nom-adjectif) et groupe verbal (accord du verbe avec son ou ses sujets) sont appliquées pour corriger des fautes d'accord dans le système ESOPE, développé par un groupe de l'IRISA, à Rennes [ST-DIZIER 84]. En cas d'échec de l'analyse d'une phrase, un mécanisme de détection et de correction automatique des fautes d'accord est déclenché : il consiste à réactiver l'analyse en inhibant les contraintes syntaxiques et à prendre connaissance du genre et du nombre de chaque mot.
Lorsqu'une faute d'accord a été détectée, une règle permet de dire comment la corriger. Exemple : dans *le réunion*, lorsque l'incompatibilité syntaxique entre *le* et *réunion* est détectée, la règle appliquée est : "Lorsqu'il y a incompatibilité en genre entre le nom et l'article qui introduit ce nom, le nom impose son genre à l'article". L'article le est donc remplacé par son correspondant féminin *la*.
Dans ESOPE, 8 règles d'accord sont définies concernant l'accord en genre et en nombre des différents constituants d'une phrase.

Le système NOMAD [GRANGER 83], dont l'objectif est la représentation non-ambiguë des connaissances, accepte des textes en anglais "télégraphique" (des messages provenant de la communication navale *bateau - terre*) où sont souvent omis les noms, les verbes voire la ponctuation et où sont souvent utilisées des abréviations ad-hoc. De plus, ces textes peuvent contenir des problèmes d'interprétation.
NOMAD intègre le traitement syntaxique et sémantique. Les informations concernant le mot et son rôle syntaxique permettent d'arriver à un nombre d'interprétations possibles associées à un texte.
Il considère deux types d'erreurs : les erreurs *de surface* et les erreurs *d'interprétation*.
Les erreurs de surface sont les erreurs lexicales et syntaxiques (mots inconnus ou mots faux, omission de mots, mauvaise syntaxe, utilisation d'abréviations, confusion dans la présentation des idées, manque de ponctuation).
Ces erreurs sont fréquemment à l'origine des erreurs d'interprétation (une représentation incomplète, une violation de but, un auteur ou un objet hors de séquence).

La méthode de correction employée est basée sur des prévisions syntaxiques et sémantiques, fondées sur la connaissance de l'anglais de surface et sur la connaissance de la situation traitée dans le texte.

Nous analysons maintenant trois réalisations : Le système EPISTLE (IBM), l'implantation du français fondamental par Lacouture et Lapalme et le système de correction automatique des accords de Richard et Lapalme.

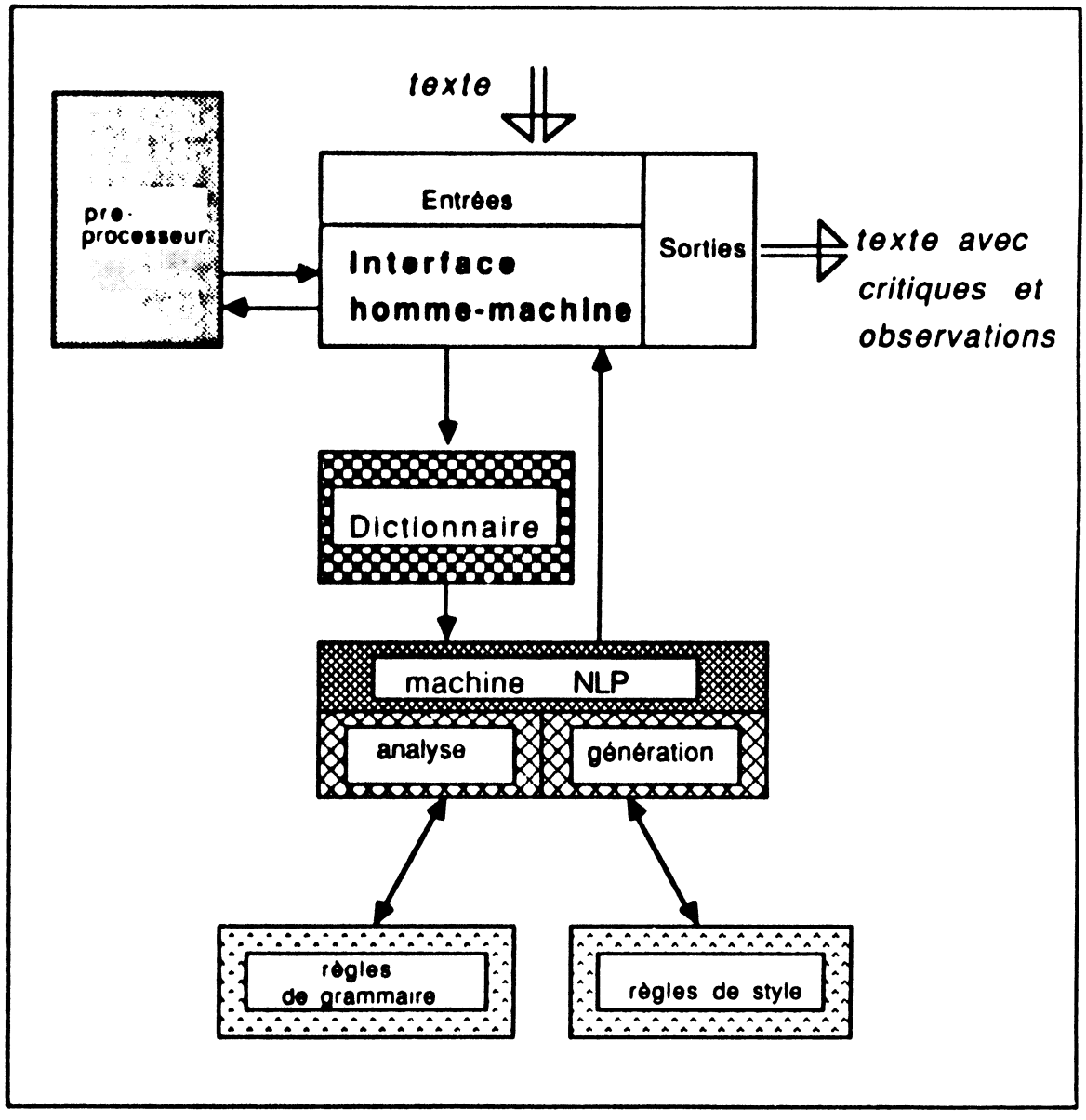


Figure 1.2

Structure générale du système EPISTLE

3.2.1 LE SYSTEME EPISTLE

Le système EPISTLE (IBM, [HEIDORN 82, JENSEN 83, JENSEN 84]) envisage le traitement des problèmes de grammaire et de style dans des textes écrits en anglais ordinaire, c'est-à-dire, des lettres, des rapports, etc, par opposition à des textes écrits en anglais littéraire, comme des poèmes ou des romans. La figure 1.2 présente un schéma simplifié de ce système (inspiré de [JENSEN 84]).

Au niveau de la grammaire, ce système traite des erreurs syntaxiques telles que l'accord en nombre entre le sujet et le verbe, des problèmes avec les pronoms dans des phrases prépositionnelles ou placés comme objet direct, formes verbales avec auxiliaire, infinitif, etc. Au niveau du style, il signale des problèmes comme des phrases trop complexes, avec un nombre excessif de mots, des répétitions, double négation, contractions informelles, etc.

La grammaire utilisée est écrite en NLP, un langage défini par Heidorn et mis en oeuvre en LISP/370.

Le système EPISTLE, d'après les descriptions faites dans [JENSEN 83], n'utilise pas de renseignements sémantiques, mais uniquement des informations lexicales et syntaxiques. Ce point de vue est toutefois contestable : parmi les informations associées à un mot se trouvent sa classe (qui peut être une information à caractère sémantique [JENSEN 84]), sa catégorie syntaxique, etc.

Le dictionnaire utilisé contient près de 130000 entrées

La grammaire utilisée est basée sur des phrases
Le corpus de phrases de test est composé de 2254 phrases relevées dans 411 lettres d'affaires. La phrase la plus longue contient 63 mots, mais la longueur moyenne est de 19,2 mots

L'analyseur syntaxique employé dans l'EPISTLE se divise en 3 parties :

1 un ensemble de règles appelé "core grammar", qui définit précisément les structures plus importantes du langage ;

2 des procédures de traitement des structures ambiguës créées lors de l'analyse syntaxique (ces procédures sélectionnent la structure à prendre, en cas d'ambiguïté) ;

3 des procédures "périphériques" de traitement des échecs pendant l'analyse syntaxique.

3.2.2 L'implantation informatique du français fondamental

Lacouture et Lapalme [LACOUTURE 88] ont conçu et réalisé un analyseur qui traite un sous-ensemble de la langue française : le "français fondamental". Le sous-ensemble de la langue nommé "français fondamental" a été défini durant les années cinquante en France par une commission spéciale suite au voeu de l'UNESCO concernant la diffusion des grandes langues de la civilisation.
Le français fondamental est formé de deux niveaux, le premier portant sur la langue parlée et le deuxième portant plutôt sur la langue écrite. Ces niveaux se présentent sous la forme de deux documents divisés chacun en deux parties : le vocabulaire et la grammaire.

Le système de Lacouture et Lapalme est composé d'un analyseur lexical et d'un analyseur syntaxique. L'analyseur lexical valide l'entrée et identifie chaque mot de la phrase à l'aide des dictionnaires qui font partie du système. L'analyseur syntaxique utilise les informations produites par l'analyse lexicale et reconnaît l'organisation syntaxique de la phrase et des locutions. Il signale également certaines erreurs d'accord simple (comme l'accord entre un adjectif et le nom auquel il se réfère) et traite les mots inconnus en tentant de leur affecter une catégorisation syntaxique permettant de poursuivre l'analyse.

L'analyseur syntaxique de Lacouture et Lapalme est formé d'une suite de règles PROLOG traduisant la grammaire du français fondamental.
Les auteurs signalent comme un avantage l'absence d'indication sémantique dans ce système : l'utilisation de renseignements sémantiques tend habituellement à restreindre le type de phrases acceptables par un système [LACOUTURE 88]. Ils considèrent que, en définitive, pour une phrase prise hors de son contexte, on ne peut rien affirmer de son intention, du message qu'elle veut faire passer. Exemple (extrait de [LACOUTURE 88]) : on ne pourrait rien décider à propos du sens ou du non-sens d'une phrase comme *Des atomes verts et bleus dansaient*).

De cette étude, Lacouture et Lapalme avancent deux conclusions importantes :

a) il est possible d'effectuer une analyse syntaxique sans analyse sémantique, à condition d'y inclure toutes les informations indépendantes du contexte sémantique que contient un mot ;
b) il est possible de concevoir un analyseur servant de "noyau" à des systèmes dédiés à une application particulière.

Les auteurs considèrent toutefois comme discutable le choix d'exclure toute sémantique dans le cadre d'une application concrète.

3.2.3 La correction automatique des accords du participe passé

D. Richard et G. Lapalme [RICHARD 86] ont développé à l'Université de Montréal un système de correction automatique des accords des participes passés dont le choix des règles de grammaire est basé sur le Français Fondamental [LACOUTURE 88].
Les fautes dans un texte sont divisées en deux catégories : les fautes d'orthographe et les fautes d'accord. Les dernières comprennent les fautes d'accord entre le sujet et le verbe et entre le verbe, l'adjectif et le déterminant et les fautes d'accord du participe passé.

Pour la correction des accords du participe passé, ce système prend en compte des règles générales et des cas particuliers.
Les règles générales concernent le participe passé sans auxiliaire (traité comme adjectif qualificatif), le participe passé conjugué avec l'auxiliaire *être* (qui s'accorde avec le sujet) et le participe passé conjugué avec l'auxiliaire *avoir* (invariable sauf si le complément d'objet direct le précède - dans ce cas il s'accorde en genre et en nombre avec ce complément).
Même si ces règles sont les seules utilisées dans le Français Fondamental, l'acceptation d'un infinitif comme complément oblige à connaître d'autres règles pour traiter des cas particuliers, parfois assez fréquents.

Les auteurs ont concentré leurs efforts sur les règles impliquées dans les groupes suivants de cas particuliers :

1 le participe passé précédé d'un nom collectif (exemple : *foule, tas, clientèle...*), d'un adverbe de quantité (*beaucoup, davantage, peu...*) d'antécédents liés par une conjonction de comparaison (*ainsi que, comme...*) ou en rapport avec *un des* ;
2 le participe passé suivi d'un infinitif ;
3 le participe passé d'un verbe impersonnel ou d'un verbe pronominal.

Les règles d'accord regroupées par Grevisse dans son oeuvre *Savoir accorder le participe passé* sont programmées en PROLOG.
Pour corriger les fautes d'accord du participe passé les auteurs considèrent comme informations nécessaires :

a) genre et nombre du sujet et du complément d'objet direct ;

b) traits sémantiques des sujets et compléments acceptables pour le verbe ;

c) traits sémantiques des sujets et compléments acceptables pour l'infinitif qui suit le verbe ;

d) compléments du verbe qui sont présents dans la phrase ;

e) compléments de l'infinitif qui sont présents dans la phrase ;

f) espèce du verbe (selon Grevisse [GREVISSE 69] : *transitif direct, transitif indirect, intransitif, impersonnel, pronominal*).

Les informations provenant du lexique sont obtenues en deux étapes. Dans une première phase, sont obtenus le genre, le nombre et la forme de base du mot. Une deuxième phase permet de retrouver les traits sémantiques attachés à la forme de base du mot et qui sont utilisés, à la fois, pour s'assurer d'un minimum de sens dans la phrase et pour corriger les participes passés.

Toutefois, même disposant d'un tel ensemble d'informations syntaxico-sémantiques, il reste encore des situations impossibles à corriger. Prenons l'exemple :

Les fillettes que Marie a vu chanter...

Cette phrase présente un participe passé (*vu*) suivi d'un infinitif (*chanter*) dans une relative. Le verbe *voir* étant un verbe de perception, son participe s'accorde avec le nom qui précède la relative (*les fillettes*) lorsqu'il se rapporte a ce groupe nominal.

Le nom qui précède la relative (*les fillettes*) peut pratiquer l'action de l'infinitif (*chanter*). Mais il est impossible de dire, sans ambiguïté, le sens réel de la phrase, à moins de disposer d'informations supplémentaires, de contexte.
L'infinitif *chanter* pourrait avoir, ici, le sens de *vanter*. Dans ce cas, les fillettes auraient été *chantées* par quelqu'un d'autre. Le participe *vu* doit alors rester invariable.
L'infinitif *chanter* pourrait signifier, simplement, "former avec la voix une suite de sons musicaux". Dans ces cas, ces sont les fillettes qui ont chanté. Le participe *vu* doit alors s'accorder avec *les fillettes*, puisqu'elles réalisent l'action de l'infinitif.
Par conséquent, cette phrase présenterait une faute d'accord.

Les fillettes peut être complément d'objet direct (dans ce cas, le participe *vu* devrait être mis au féminin pluriel), mais aucun

complément de l'infinitif dans la phrase ne permet de rejeter aucun des deux rôles possibles de ce nom. Alors, le système de Richard et Lapalme fournit les deux accords et c'est l'utilisateur qui va choisir entre les options proposées.

Des ambiguïtés sont aussi présentes dans des phrases comme (extrait de [RICHARD 86]) :

*Les chats de Marie que Jean a **vus** sont lavés*

Il est impossible de rejeter aucun des noms du groupe nominal (*les chats* ou *Marie*) pour rattacher la relative qui, sémantiquement, peut s'appliquer à l'un ou l'autre des noms du groupe nominal. A la détection d'une telle structure, la possibilité d'écriture

*Les chats de Marie que Jean a **vue** sont lavés*

serait aussi proposée.

En ce qui concerne l'accord du participe passé d'un verbe du premier groupe, un problème supplémentaire se pose : le participe passé étant phonétiquement identique à l'infinitif, l'utilisateur peut avoir employé un infinitif là où il aurait dû employer un participe passé. Exemple : *Il a **manger** le poisson.*

En fait, l'infinitif comme adjectif ou comme participe passé n'est pas une forme syntaxique acceptable par la grammaire, mais il est nécessaire de l'identifier afin de la transformer en forme acceptable.

Richard et Lapalme incorporent à la grammaire deux alternatives pour tenir compte de ces fautes très fréquentes. Voici la BNF (Bachus-Naur form) correspondante :

adjectif ::= adjectif qualificatif | participe passé | [] | *infinitif*

prédicat ::= verbe | auxiliaire, participe passé | auxiliaire, *infinitif*

Dans ces deux cas spéciaux, l'*infinitif* est remplacé par le participe passé de base correspondant.

Les corrections effectuées par le système de Richard et Lapalme sont "définitives" : l'analyse syntaxique n'est pas reprise à l'occasion d'une correction. Cette mesure, d'un coté, élimine de coûteux retours en arrière. Mais elle peut empêcher l'utilisateur d'exprimer directement son impression personnelle de correction, et elle conserve toutes les structures ambiguës proposées par l'analyse.

Cette recherche a mis en évidence la nécessité d'avoir recours à une analyse sémantique complète pour tenir compte de plusieurs règles d'accord reliées au sens de la phrase.

4 ERREURS SEMANTIQUES

Au niveau sémantique, la détection et correction d'erreurs se pose comme un vaste champ de recherche, dans lequel il y a encore très peu de résultats. Ce qu'on peut trouver c'est le traitement sémantique de sous-ensembles de la langue, insérés dans des applications spécifiques.

Les erreurs détectées au niveau sémantique sont des erreurs de compétence [VERONIS 88b] qui peuvent provenir soit du système soit de l'utilisateur.
Dans le cas des erreurs provenant du système, ce dernier possède une représentation contradictoire ou incomplète des connaissances.
Les erreurs provenant de l'utilisateur sont classées en deux types : erreurs conceptuelles et erreurs pragmatiques.

Les erreurs conceptuelles proviennent d'une mauvaise représentation des connaissances, contradictoire ou incomplète, dans l'esprit de l'utilisateur. Ces erreurs apparaissent tout particulièrement en Enseignement Assisté par Ordinateur (E.A.O.), l'utilisateur étant, par définition, en cours d'apprentissage du domaine.
Exemple (extrait de [VERONIS 88b]) :
La phrase

[AB] est l'hypoténuse du cercle C

est le reflet d'une représentation erronée du domaine par l'utilisateur (l'expression *l'hypoténuse de x* présuppose que l'objet *x* est un triangle rectangle).

Les erreurs pragmatiques proviennent de l'emploi impropre, dans une situation donnée, d'une phrase ou d'une expression qui en elle-même n'est pas erronée. Elles consistent généralement en une violation des lois du discours, ou une violation d'une procédure, d'un scénario, d'un script, propre au domaine considéré.
Exemple :

Quand mon papa était petit, moi, j'allais travailler tous les jours à l'usine.

Dans le cadre des recherches de l'équipe TRILAN, une étude est en cours pour prendre en compte quelques connaissances sémantiques permettant d'améliorer la détection et la correction d'erreurs.

CHAPITRE 2

L'ENVIRONNEMENT PILAF

*Le projet **PILAF** (**P**rocédures **I**nteractives **L**inguistiques **A**ppliquées au **F**rançais), issu du projet PIAF [COURTIN 77], dispose de deux analyseurs - un analyseur morphologique et un analyseur syntaxique - qui calculent la ou les structures de dépendances associées à une phrase. L'analyseur morphologique de PILAF est constitué d'un transducteur d'états finis. Ce transducteur a déjà été employé dans plusieurs réalisations, parmi lesquelles la transduction phonétique [COURTIN 88].*
Des modules de détection/correction d'erreurs y sont aussi disponibles, au niveau lexical. Avec les outils de PILAF, nous avons développé un système de DEtection/CORrection d'erreurs dans un texte - le système DECOR.

Dans le présent chapitre nous présentons brièvement le transducteur d'états finis de PILAF de même que l'organisation générale de ses analyseurs morphologique et syntaxique. Pour plus de détails, le lecteur pourra se reporter à l'annexe 6.
Nous décrivons aussi les modules de correction d'erreurs lexicales disponibles et nous effectuons de considérations concernant l'applicabilité des outils de PILAF dans le traitement de la langue portugaise.

1 L'ANALYSE MORPHOLOGIQUE

1.1 Le transducteur général d'états finis

Nous présentons dans ce paragraphe le transducteur d'états finis utilisé dans l'analyseur morphologique de PILAF, décrit en détail dans [COURTIN 77]

Si on envisage l'analyse morphologique, le transducteur de PILAF a pour buts :

- de segmenter une chaîne de caractères (phrase) pour obtenir ses chaînes composantes ;
- de déterminer un certain nombre de renseignements linguistiques sur les chaînes obtenues par la segmentation.

Ce transducteur est composé de deux dictionnaires, d'une grammaire et d'une liste de modèles.
Nous effectuons ici quelques remarques concernant la segmentation et la transduction dans l'analyse morphologique. Ensuite, nous exposons le fonctionnement du transducteur de PILAF.

Les concepts de *vocabulaire*, *chaîne*, *chaîne vide*, *longueur*, *préfixe*, *relation inférieure* et *grammaire à validations et saturations* sont repris de [COURTIN 77].

1.1.1 Segmentation et transduction

Etant donnée une phrase

$$a_1 a_2 .... a_n \quad (\forall\, i \in [1,n],\ a_i \in V)$$

on doit déterminer des mots ou groupes de mots m_i ($m_i \in V^+$) tels que :

$$m_1 m_2 ..... m_k = a_1 a_2 ..... a_n \quad \text{avec } k \leq n.$$

Le principe de la segmentation employée dans le transducteur de PILAF comprend la détermination de m_1 et contrôle par application de règles de grammaire, puis itération du processus jusqu'au point de fin de phrase.
La décomposition d'un m_i n'est pas unique. Exemple : *livre* peut être décomposé comme le substantif *livre* ou comme *livr* + *e* (verbe régulier *livrer*).

La transduction donne les interprétations de toutes les décompositions d'un m_i.
Par exemple, les décompositions associées à la chaîne *livre* ont pour interprétation :

a) substantif commun, masculin, singulier ;
b) substantif commun, féminin, singulier ;
c) verbe *livrer* à la troisième personne du singulier du présent de l'indicatif ou à la troisième personne du singulier du présent du subjonctif.

Cette transduction est réalisée par la grammaire, qui vérifie la segmentation d'un m_i étant donné que la plupart des éléments constituant un m_i sont porteurs d'information.

Le transducteur de PILAF utilise les règles de la même façon qu'une grammaire à validations et saturations [COURTIN 77] mais en tenant compte du modèle et du code morphologique. Le code morphologique est une liste de règles. Cette liste est associée à un système de désinences qui pourra être activé dans certaines conditions. Pour passer d'un état i à un état i+1 en appliquant la règle *r*, et pour le modèle *m*, on utilise les formules :

$$SAT_{i+1} = \text{union} \quad (SAT_i, SAT_r, SAT_m)$$
$$VAL_{i+1} = \text{union} \quad (VAL_r, VAL_m) - SAT_{i+1}$$

où

SAT_i : les saturations associées à l'état i ;
SAT_{i+1} : les saturations associées à l'état i+1 ;
SAT_r : les saturations associées à la règle *r* ;
SAT_m : les saturations associées au modèle *m* ;
VAL_{i+1} : les validations associées à l'état i ;
VAL_r : les validations associées à la règle *r* ;
VAL_m : les validations associées au modèle *m* ;

1.1.2 Utilisation du transducteur

Des compilateurs et des éditeurs de règles et de modèles autorisent une mise au point interactive des paramètres linguistiques du transducteur de PILAF.
Ce transducteur est entièrement paramétré, ce qui a permis de l'utiliser dans de nombreuses applications, parmi lesquelles :
- traduction d'un texte en braille abrégé,
- traduction d'un texte en morse abrégé,
- traduction de programmes écrits en PL360 en programmes équivalents écrits en LP80,
- traduction d'une chaîne en un codage pseudo-phonétique,
- analyse morphologique d'une autre langue.

Ce transducteur est réversible. Il permet ainsi d'engendrer toutes les formes associées à une même base morphologique. La génération morphologique sera utilisée pour corriger certains types d'erreurs dans DECOR (paragraphe 3.2 de ce chapitre).

1.2 Fonctionnement de l'analyse morphologique

Nous présentons ici un exemple d'analyse morphologique réalisée par le transducteur de PILAF :

Elle prépare son rapport au fur et à mesure.

se décompose en

Elle + prépare + son + rapport + au fur et à mesure + .

Les segments obtenus sont considérés comme indécomposables par le système et, pour une phrase donnée, on obtient un nombre unique de segments.

Nous observons que l'espace n'est pas un séparateur. Cette particularité permet de considérer *au fur et à mesure* comme une unité lexicale ou

segment, ce qui est cohérent puisque les cinq mots apparaissent sous la forme d'une locution.

En français, un segment ***s*** peut être composé d'une base ou racine, d'un ou plusieurs suffixes et d'une désinence. Les parties suffixe et désinence peuvent être vides.
Par exemple, *aimera* se décompose en :

la base :	/a i m/
le suffixe :	/e r/
la désinence :	/a /

Pour la décomposition d'un segment ***s***, nous disposons :

- de deux dictionnaires (un dictionnaire de *bases morphologiques* et un dictionnaire de *suffixes et désinences*);
- d'une *grammaire* chargée d'infirmer ou de confirmer la concaténation des divers éléments du segment ;
- de *modèles* permettant de réaliser l'interface entre la grammaire et les dictionnaires.

L'algorithme simplifié de l'analyseur morphologique est le suivant :

1 *Déterminer un segment* ***s***
Avec le dictionnaire de bases et le dictionnaire de suffixes et désinences, déterminer la décomposition d'une partie contiguë de la chaîne d'entrée.
Si aucun segment ne peut être déterminé, l'analyse est terminée
La détermination d'un segment est suivie de sa validation

2 *Valider le segment* ***s*** *obtenu*
Avec la grammaire, valider le segment **s** déterminé en **1**.
S'il est correct :
2.1 si toute la chaîne d'entrée a été analysée, nous concluons l'analyse ;
sinon nous passons à la détermination du segment suivant (nous retournons en **1**).
Si le segment déterminé n'est pas correct :
2.2 nous reprenons la chaîne d'entrée pour une nouvelle décomposition.

Comme résultat de la transduction, nous avons une interprétation du segment ***s*** obtenu : à chacun de ses composants sont associées des informations linguistiques.

Par exemple, l'analyse morphologique de *allée* produit les résultats :

/allée /	-->	base allée	:	substantif commun féminin
		désinence / /	:	singulier
/allée /	-->	base all	:	verbe irrégulier (modèle *aller*)
		désinence /ée /	:	participe passé, féminin, singulier

Comme nous pouvons l'observer, la décomposition d'un segment n'est pas unique. Les ambiguïtés provenant de l'analyse morphologique seront souvent levées par des outils syntaxiques voire sémantiques.

1.3 Les dictionnaires et leur organisation

L'organisation des dictionnaires de PILAF est inspirée de la technique des B-arbres.

1.3.1 Organisation du dictionnaire de bases morphologiques

Ce dictionnaire est constitué d'un ensemble de bases morphologiques réparties en fichiers selon la technique suivante :

on part d'un fichier F_1 vide dans lequel on insère des éléments (la taille de ce fichier est actuellement paramétrée à 600 éléments maximum) ;

dès que F_1 est plein, on le partage en deux fichiers F_1 et F_2 de 300 éléments : les 300 premiers dans F_1 et les 300 suivants dans F_2 (premiers et suivants dans l'ordre alphabétique inverse) ;

si à la suite d'autres insertions un de ces fichiers est saturé, on le découpe de nouveau en deux, etc.

Les bases dans les fichiers sont rangées en ordre alphabétique inverse pour permettre d'obtenir en premier, lors d'une recherche, l'élément le plus long.
Chaque fichier a en moyenne 450 éléments.

Le dictionnaire de bases est actuellement réparti en 57 fichiers. C'est-à-dire que nous avons, environ, 26000 bases stockées, permettant la reconnaissance d'environ 220000 formes.
L'accès aux fichiers F_i est assuré par une table appelée TABMAITRE, chargée en mémoire, qui contient pour chaque entrée :

- le nom f_i du fichier ;
- BINF et BSUP : deux chaînes, de longueur dix, correspondant aux bornes inférieure et supérieure de l'ensemble des éléments du fichier ;
- NBELE : le nombre d'éléments du fichier.

La taille de TABMAITRE est fixée, par paramètre, à 99 éléments au maximum, ce qui nous permet d'avoir jusqu'à 99 fichiers du type F_i (fichiers contenant les bases morphologiques).

Un élément du dictionnaire de bases morphologiques est un enregistrement constitué de 6 champs se rapportant à une base :

1	*INDECOUPABLE :*	booléen signalant que la base est non-réductible ;
2	*MODELE :*	numéro du modèle correspondant à cette base ;
3	*LETTRES :*	chaîne de caractères qui constitue la base ;
4	*OCCLETTRES :*	numéro d'occurrence de cette base dans le dictionnaire ;
5	*CHCIRC :*	chaîne de caractères ;
6	*OCCCIRC :*	numéro d'occurrence de la chaîne dans le dictionnaire.

Le champ *occlettres* distingue deux éléments homographes.
Les champs *chcirc* et *occcirc* permettent d'accéder à l'élément suivant dans le chaînage circulaire. Ces informations sont utilisées actuellement par la génération morphologique, pour engendrer, par exemple, toutes les formes d'un même verbe. Nous présentons comme exemple la chaîne circulaire correspondant au nom *monsieur* :

monsieur → mesdames
mesdames → madame
madame → messieurs
messieurs → monsieur

La présence de **vrai** dans l'indicateur INDECOUPABLE permet d'arrêter la recherche, c'est-à-dire de ne pas chercher d'autre décomposition de la chaîne d'entrée, au cours de l'analyse morphologique.

1.3.2 Organisation du dictionnaire de suffixes et désinences

La composition des enregistrements du dictionnaire de suffixes et désinences est similaire à celle du dictionnaire de bases de PILAF : les suffixes et désinences ont des modèles associés, aussi bien qu'un indicateur de non-réductibilité. Néanmoins, le dictionnaire de suffixes et désinences comporte un nombre limité d'éléments, ce qui permet d'opérer sur un fichier unique. La taille actuelle de ce dictionnaire est d'environ 100 enregistrements.
A cause de sa taille limitée, le dictionnaire de suffixes et désinences est chargé en mémoire.

Un élément du dictionnaire de suffixes et désinences est un enregistrement composé de 4 champs :

1	*INDECOUPABLE :*	booléen signalant que l'élément est non-réductible ;
2	*MODELE :*	numéro du modèle correspondant ;
3	*LETTRES :*	chaîne de caractères ;
4	*OCCLETTRES :*	numéro d'occurrence de la chaîne dans le dictionnaire.

1.3.3 La recherche dans les dictionnaires

Considérant qu'on ne dispose pas de séparateur de mot, l'entrée de l'algorithme de recherche dans les dictionnaires est alors un pointeur sur le premier caractère de la chaîne restant à analyser.
Exemple :

L'eau bout
↑

L'algorithme :
1 détermine le fichier F_i dans lequel on va effectuer la recherche ;
2 essaye de faire coïncider un élément de F_i avec le début de la chaîne d'entrée.

Pour déterminer le fichier F_i, on effectue une recherche dans TABMAITRE.
On extrait de la chaîne d'entrée la chaîne **a** de longueur dix par troncature ou ajout de blancs (les BINF et BSUP de TABMAITRE ont été ramenés, eux aussi, à une longueur 10). Il suffit ensuite de trouver l'entrée I de TABMAITRE telle que :

$$BINF_i \leq \mathbf{a} \leq BSUP_i$$

et on a dans TABMAITRE[i].*f* la chaîne de caractères qui permet de déterminer le nom du fichier F_i.

Par exemple, soit */bout /* la sous-chaîne à analyser.

bout
↑

Par ajout de blancs on obtient de cette sous-chaîne une autre : */bout / * (longueur de 10 caractères) permettant de déterminer le fichier F_i dans lequel doit commencer la recherche.

Une fois déterminé F_i, on recherche à l'aide d'un algorithme de dichotomie, l'élément le plus long du dictionnaire qui soit un préfixe de la chaîne donnée. Comme on ne sait pas a priori où s'arrête le mot dans la chaîne d'entrée (on peut avoir, par exemple, une locution composée de plusieurs mots), l'ordre alphabétique inverse permet d'obtenir d'abord les éléments les plus longs.

Supposons que les dictionnaires disponibles sont :

Dictionnaire de bases morphologiques (fichier F_i) :

.....
/bouture/**volture**/
/bouturage/**homme**/ *indécoupable*
...
/bout/**homme**/
/bout/**pos**/
...
/bou/**boullllr**/ *indécoupable*
.....

(on n'a pas mentionné ici les informations sur le chainage circulaire, car elles ne présentent pas d'intérêt pour l'exemple)

Dictionnaire de suffixes et désinences :

.....
/,/,/ *indécoupable*
/ // *indécoupable*

L'accès au fichier F_i sera réalisé conformément à l'algorithme suivant (pour plus de détails, voir [COURTIN 77, COURTIN 86]) :

1) *Recherche dans le fichier F_i*
On cherche dans F_i la sous-chaîne */bout/*.
Le premier élément de F_i qui a le même préfixe que la chaîne d'entrée est /bouture/, qui est plus long que cette chaîne.
On réalise un parcours séquentiel dans le fichier F_i à partir de /bouture/, jusqu'à ce qu'on trouve /bout/ qui coïncide exactement avec la chaîne d'entrée de l'algorithme.

2) *Appel de la grammaire*
On appelle la grammaire pour valider /bout/ et calculer les informations linguistiques concernées.
Le dictionnaire fournit le modèle associé à l'élément (/bout/ est défini sur le modèle /**homme**/ - un modèle de substantif). Deux cas sont alors possibles :

a) *la grammaire ne valide pas cet élément*
La chaîne reconnue ne peut donc pas faire partie du segment en cours de construction. On lance alors la recherche d'une chaîne éventuellement plus courte ;

b) *la grammaire valide cet élément*
En fait, la grammaire valide la chaîne /bout/. Mais on a de nouveau deux cas possibles :

b.1) La concaténation des éléments déjà reconnus peut constituer un segment incomplet.
Dans la validation de /bout/, nous avons besoin de valider aussi l'espace, qui indique la marque de singulier (sinon les informations linguistiques resteront incomplètes). On relance alors la recherche sur les caractères qui suivent la chaîne courante (cette nouvelle recherche est faite sur le dictionnaire de suffixes et désinences, où on trouvera l'espace).

b.2) La concaténation des éléments déjà reconnus constitue un segment complet.
La concaténation de /bout/ (définie sur le modèle /**homme**/) avec l'espace est un segment complet et ce résultat est ajouté à la liste des résultats de l'analyse morphologique.
Ensuite, si l'indicateur INDECOUPABLE de l'entrée courante du dictionnaire est positionné à **vrai**, on arrête la recherche, sinon on lance la recherche d'une chaîne plus courte.

La base /bout/ définie sur le modèle /**homme**/ n'est pas non-réductible. La grammaire relance alors la recherche d'un élément plus petit que (ou éventuellement égal à) ce dernier élément analysé.
Comme résultat, on trouve la base /bout/ définie sur le modèle /**pos**/, un modèle verbal dont la conjugaison suit celle du verbe *poser*. Il s'agit de la base associée au verbe *bouter*. La grammaire ne valide pas cet élément : le segment /bout/ + / / n'est pas validé comme une forme du verbe *bouter*. On donne suite à l'analyse.

Comme la base /bout/ définie sur le modèle /**pos**/ n'est pas non-réductible, on relance la recherche d'un élément plus petit, et on trouve la base /bou/ définie sur le modèle /**bouillir**/. La grammaire valide la sous-chaîne /bou/ et on lance alors la recherche de la sous-chaîne suivante :

bout
↑

On lance la recherche des caractères /t / (sur le dictionnaire de suffixes et désinences) et on valide le segment complet */bout /*.

Ce résultat est rajouté à la liste de résultats de l'analyse morphologique.

La base /bou/ définie sur le modèle /**bouIIIIr**/ est non-réductible. On arrête la recherche et l'analyse morphologique.
Les résultats obtenus pour le segment */bout /* sont :
1 - substantif masculin singulier ;
2 - verbe *bouillir* à la troisième personne du singulier du présent de l'indicatif.

1.4 La grammaire

Le modèle théorique de base est celui des grammaires à validations et saturations (GVS, paragraphe 1.1.3 de ce chapitre). Ce modèle est équivalent formellement aux grammaires d'états finis mais permet une formulation plus concise [COURTIN 86].

Dans l'analyseur morphologique de PILAF les règles de la grammaire sont nommées (RIB, FOR, ...) et le vocabulaire est constitué d'informations linguistiques. Les règles ont la forme :

nom : informations linguistiques ;
VAL := (liste de noms de règles) ou valeur prédéfinie ;
SAT := (liste de noms de règles) ou valeur prédéfinie ;
indicateurs

VAL et SAT définissent respectivement les validations et les saturations.
On a deux indicateurs possibles :
- **FIM** qui indique une règle terminale ;
- **ND** qui indique une non-désinence.

Le mot clé **VAL00** désigne la liste prédéfinie des règles initiales.

1.5 Les modèles

Pour assurer l'interface entre le dictionnaire et la grammaire on utilise des modèles. Ils représentent des classes de mots ayant le même comportement linguistique (exemple : les verbes du 1er groupe). Ils sont de la forme :

/nom/: **REG := (liste de noms de règles) ou valeur prédéfinie ;**
informations linguistiques ;
VAL := (liste de noms de règles) ou valeur prédéfinie ;
SAT := (liste de noms de règles) ou valeur prédéfinie ;

REG contient une liste des règles applicables à ce modèle.

Les informations linguistiques sont de deux sortes :

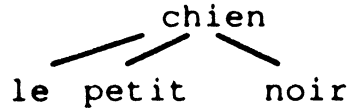
1) **classe (CL)**
   CL est la classe lexicale du mot. Exemple : pour le mot *roi* on a CL = *substantif*.
2) **variable (VAR)**
   Elle peut prendre plusieurs valeurs. Exemple : *roi* a MAS SIN (*masculin singulier*) comme variables.

2 L'ANALYSE SYNTAXIQUE

Cette analyse est réalisée par un module qui construit les structures de dépendances associées à une phrase, en fonction des catégories délivrées par l'analyse morphologique.

2.1 La construction de dépendances et le modèle linguistique

Le concept fondamental associé aux structures de dépendances est celui de relation entre les mots. Etant donnés deux mots du langage, on établit entre eux une relation de dominé (dépendant) à dominant (gouverneur) et on peut représenter cette relation par un arc entre deux noeuds, chaque noeud étant étiqueté par un mot.
Exemple :
Dans le groupe *le petit chien noir*, *chien* est le gouverneur et *le*, *petit* et *noir* sont ses dépendants, d'où la structure :



La représentation arborescente met en valeur la hiérarchie que les relations établissent entre un gouverneur et ses dépendants, étant entendu que ces dépendants peuvent avoir eux-mêmes des dépendants.

Une grammaire de dépendances sur un vocabulaire V est constituée [COURTIN 77] :
- d'une famille de parties C_i de V que constituent des catégories syntaxiques, telle que l'union des C_i soit égale à V et
- d'un ensemble de règles des deux formes suivantes :

  i) $*(X)$

  ii) $X(X_1 \ldots X_i * X_{i+1} \ldots X_n)$

Les c_i sont les classes de mots définies, désignées par leur nom : ART (article), SUBC (substantif commun), ADJQ (adjectif), etc. Les X_i sont des noms de classes. L'astérisque sert à indiquer la place du gouverneur par rapport à ses dépendants. Ainsi, dans une règle de type *ii)* :

$X_1 \ldots X_i$ sont les dépendants gauches du gouverneur X et

$X_{i+1} \ldots X_n$ ses dépendants droits.

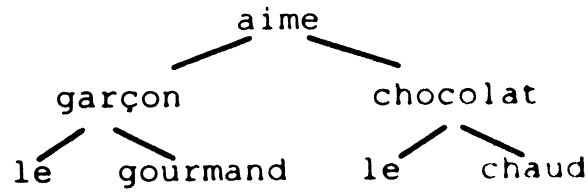
Si n=0, la règle s'écrit X(*) et est une règle terminale ; les règles de type *i)* sont des règles initiales.

Exemple :

*(VERB)
VERB(SUBC,*,SUBC)
SUBC(DET,*)
SUBC(DET,*,ADJQ)
DET(*) VERB(*) ADJQ(*) SUBC(*)

VERB = {*aime, marche*}
SUBC = {*fille, garçon, bonbon, chocolat*}
DET = {*le, les*}
ADJQ = {*noir, gourmand, chaud*}

Avec cette grammaire on peut construire :



...soit la dérivation :

*(VERB)
(VERB(SUBC,,SUBC))
(VERB(SUBC(DET,),*,SUBC))
(VERB(SUBC(DET,,ADJQ),*,SUBC(DET,*,ADJQ)))
(VERB(*) (SUBC(DET,),*,SUBC(DET,*,ADJQ)))
:
(VERB(*)(SUBC(*)(DET(*),,ADJQ(*)),*,SUBC(*)(DET(*),*,ADJQ(*))))

Les grammaires de dépendances sont des grammaires génératives.
Le principe de génération est le suivant :

a) on choisit une règle de type *i)* (c'est le choix du gouverneur principal) ;
b) on choisit des règles de type *ii)* jusqu'à obtenir une structure mise complètement entre parenthèses ne contenant que des règles terminales.

Pour un gouverneur donné, la grammaire de dépendances doit comporter autant de règles qu'il y a de combinaisons possibles de dépendants sous ce gouverneur.
Si on veut par exemple engendrer des groupes nominaux comprenant au moins un nom et un article, mais 0, 1 ou 2 adjectifs avant le nom et 0 ou 1 adjectif après le nom, on aura la grammaire [COURTIN 77] :

SUBC(DET, *)
SUBC(DET, *, ADJQ)
SUBC(DET, ADJQ, *)
SUBC(DET, ADJQ, ADJQ, *)
SUBC(DET, ADJQ, *, ADJQ)
SUBC(DET, ADJQ, ADJQ, *, ADJQ)

Comme on peut l'observer, ces grammaires imposent la description de toutes les configurations possibles des dépendants pour un gouverneur donné. Pour pallier l'inconvénient de cette multiplicité on introduit la notion de relation de dépendances entre deux catégories syntaxiques (paragraphe suivant de ce chapitre).
Exemple : la relation SUBC*ADJQ indique que la catégorie SUBC gouverne la catégorie ADJQ.

2.2 Relations de dépendances

Les relations de dépendances doivent représenter :

- la relation gouverneur-dépendant ;
- la position du dépendant par rapport au gouverneur (à gauche ou à droite),
- les positions relatives des dépendants, au cas où un gouverneur peut avoir plusieurs dépendants du même côté.

Exemple :
Dans *le petit chien noir*, l'analyse morphologique indique les catégories DET, ADJQ, SUBC et ADJQ, soit déterminant, adjectif, substantif commun et adjectif, respectivement, pour les composants de la phrase. Les relations de dépendances doivent préciser que l'ADJQ peut se trouver à gauche ou à droite du SUBC, et que le DET est plus à gauche que l'ADJQ.

On attache à chaque relation un vecteur de poids (entiers positifs ou négatifs) et on écrit :

$$\text{GOUV} \quad * \quad \text{DEP} \quad := \quad (x_1,\ldots,x_n)$$

ce qui signifie qu'il peut y avoir 0,1,..n dépendants de la catégorie DEP du gouverneur de la catégorie GOUV.

La valeur des poids permet également, pour deux catégories dépendant d'un même gouverneur, de déterminer les positions relatives de ces deux catégories sous le gouverneur.
Exemple :

```
SUBC*ADJQ:=(-15,-14,+18)
SUBC*DET:=(-16)
```

Les poids positifs désignent les dépendants droits et les poids négatifs les dépendants gauches. Le poids -16 dans la deuxième relation précise que le déterminant sera toujours placé avant les adjectifs. Dans la première relation, on voit qu'on pourra avoir deux adjectifs avant le nom, mais que rien ne peut s'insérer entre les deux (aucun entier entre -15 et -14).

Les deux relations ci-dessus sont donc équivalentes à la grammaire de dépendances :

```
SUBC(*)
SUBC(DET,*)
SUBC(*,ADJQ)
SUBC(DET,*,ADJQ)
SUBC(ADJQ,*,ADJQ)
SUBC(DET,ADJQ,*,ADJQ)
SUBC(ADJQ,ADJQ,*,ADJQ)
SUBC(DET,ADJQ,ADJQ,*,ADJQ)
SUBC(DET,ADJQ,ADJQ,*)
SUBC(ADJQ,ADJQ,*)
SUBC(DET,ADJQ,*)
SUBC(ADJQ,*)
```

2.3 L'algorithme d'analyse syntaxique

Nous décrivons ensuite l'algorithme de l'analyse syntaxique conformément aux remarques contenues dans [COURTIN 86].
Les entrées de cet algorithme sont :

- la suite $X_1 \ldots X_n$ des catégories syntaxiques calculées par l'analyseur morphologique,
- l'ensemble des relations de dépendance avec leurs vecteurs de poids associés.

On ajoute aux catégories syntaxiques la catégorie PHRA (phrase) afin de déterminer les gouverneurs possibles de la phrase (pour démarrer l'algorithme). Si les gouverneurs possibles sont, par exemple, la conjonction de coordination (COCO) et le verbe (VERB), on aura les deux relations :

```
PHRA*VERB =(+1)
PHRA*COCO =(+1)
```

Comme on ne peut avoir qu'un seul gouverneur pour une phrase, ces deux relations sont exclusives ; on le signale en mettant le même poids aux dépendants possibles de la catégorie PHRA.

L'analyseur commence par construire la matrice de la figure 2.1.

Gouv.	PHRA				
Dép.	X_0	X_1	X_2	...	X_n
X_0	∅	P_{01}	P_{02}	...	P_{0n}
X_1	P_{10}	∅	P_{12}	...	P_{1n}
X_2	P_{20}	P_{21}	∅	...	P_{2n}
.	..	..		..	
X_n	P_{n0}	P_{n1}	P_{n2}	...	∅

Figure 2.1 - Matrice d'analyse

P_{ij} : ensemble des poids déterminés par les relations entre X_i (dépendant) et X_j (gouverneur).
P_{ii} : ∅.

Nous éliminons du triangle supérieur droit de la matrice tous les poids positifs, et du triangle inférieur gauche tous les poids négatifs (nous nous sommes intéressés à ne construire que des structures projectives).
Nous avons donc les deux propriétés :

$1 \le i, \quad j \le n, \quad i > j, \quad$ si $P_{ij} \neq \emptyset \quad$ alors $\forall \quad p \in P_{ij}$, on a $p>0$.

$1 \le i, \quad j \le n, \quad i < j, \quad$ si $P_{ij} \neq \emptyset \quad$ alors $\forall \quad p \in P_{ij}$, on a $p<0$.

La construction des structures de dépendances est récursive et descendante :

1 Pour tous les gouverneurs possibles de la phrase (colonne PHRA) :
- *1.1* construire tous les sous-arbres gauches puis tous les sous-arbres droits (récursivement) ;
- *1.2* construire les structures finales avec les structures partielles obtenues.

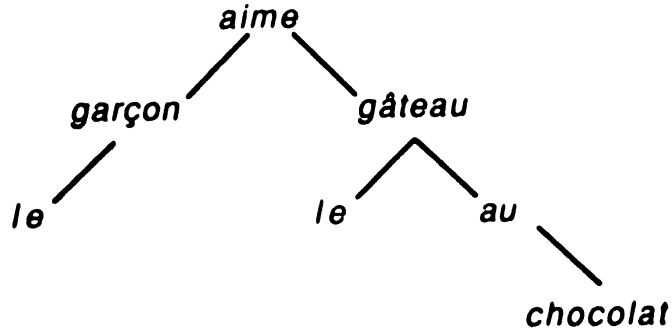
Nous présentons comme exemple l'analyse de la phrase :

Le	*garçon*	*aime*	*le*	*gâteau*	*au*	*chocolat.*
DET	SUBC	VERB	DET	SUBC	PREP	SUBC

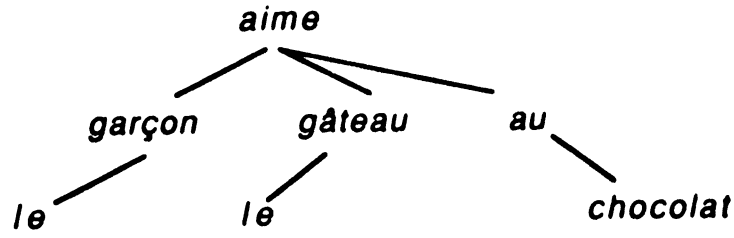
On a les relations (PREP indique préposition) :

PHRA*SUBC	≔	(+1)
PHRA*VERB	≔	(+1)
VERB*SUBC	≔	(-20,+20)
VERB*PREP	≔	(+30)
SUBC*DET	≔	(-16)
SUBC*PREP	≔	(+20)
PREP*SUBC	≔	(+7)

On obtient :



et



3 TRAITEMENT D'ERREURS

Le système DECOR a pour but de réaliser la détection/correction d'erreurs au niveau lexical dans un texte. Pour l'atteindre, il dispose d'un ensemble de modules qui réalisent :

- la détection/correction par *clé squelette* ;
- la correction par *génération* ;
- la correction par *phonétique* ;

Dans le cadre des recherches menées par l'équipe TRILAN, nous nous intéressons également au traitement d'erreurs au niveau syntaxique, disposant de deux modules qui effectuent :

- la détection d'erreurs syntaxiques, par l'application d'un ensemble de règles de vérification syntaxique ;
- la correction de fautes d'accord par génération.

Dans ce chapitre nous présentons les modules de détection/correction par clé squelette et correction par génération. La correction par phonétique et la détection/correction syntaxique sont décrites dans les chapitres 3, 4 et 5.

3.1 Le module de correction par clé squelette

Pour corriger les erreurs du genre typographique et orthographique nous avons mis en oeuvre une méthode basée sur la *clé squelette* (clé de similarité proposée par Pollock et Zamora [POLLOCK 84] décrite dans le paragraphe 1 du chapitre 1), présentée en détail dans [LU 86, COHARD 88, BAINIER 89].
L'algorithme s'appuie sur une idée fondamentale : si une chaîne peut être transformée en un mot appartenant au dictionnaire par l'inversion d'une des opérations d'édition, alors ce mot est une correction possible pour la chaîne en question.

Etant donnée une chaîne à corriger, nous calculons sa clé squelette, puis nous recherchons cette clé squelette dans le dictionnaire de *clés/formes*.
Ce dictionnaire est organisé en trois niveaux (voir figure 2.2) pour restreindre la taille des fichiers et permettre un accès plus rapide aux informations.

Un *fichier maître* permet d'accéder aux divers fichiers de clés (son organisation est identique à celle du fichier maître concernant le dictionnaire de bases morphologiques).

Un *fichier de clés* contient au maximum 2000 éléments.
A chaque fichier de clés correspond un *fichier bloc* où on trouve les formes associées aux clés contenues dans le fichier de clés correspondant.

Dans un fichier de clés, chaque enregistrement est composé :
- de la clé elle-même ;
- d'un *numéro de bloc*, qui permet d'accéder à l'enregistrement **b** où se trouve la première forme correspondant à cette clé dans le fichier bloc correspondant ;
- d'un *numéro d'indice*, qui permet d'accéder au premier caractère de la première forme correspondant à cette clé.

Dans un fichier bloc, les enregistrements correspondent à des tableaux de 512 caractères où les formes sont stockées caractère par caractère. Des caractères spéciaux séparent deux formes distinctes ou deux groupes de formes possédant des clés différentes.

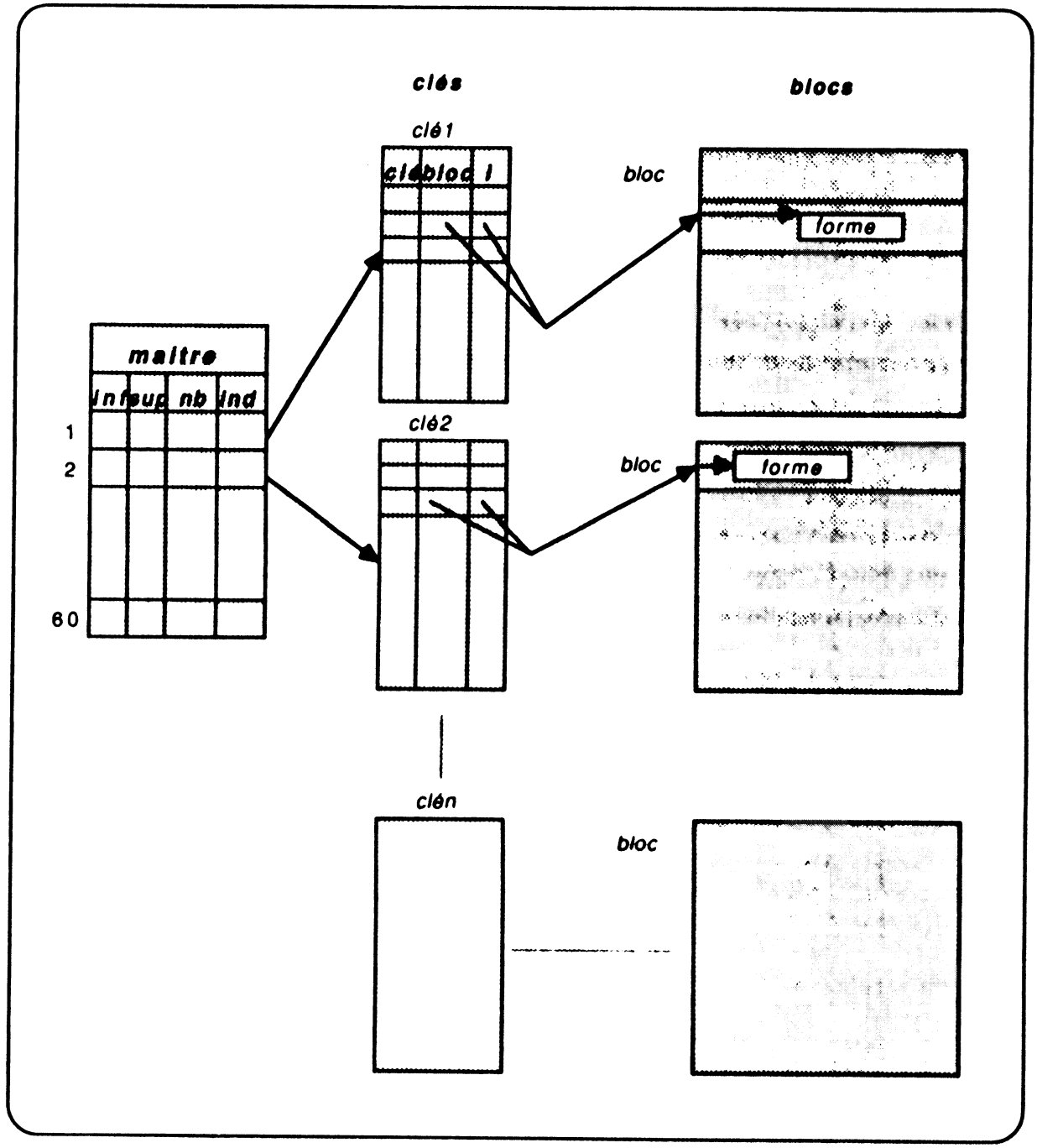


Figure 2.2 - Structure du dictionnaire de clés/formes

Pour une correction :

1 si la clé calculée pour le mot faux existe dans le dictionnaire de clés/formes, alors :

1.a on recherche la/les formes correspondant à la clé ;

2 sinon :

2.a on mémorise la clé immédiatement inférieure ;

2.b on sélectionne en deux niveaux les formes associées à cette nouvelle clé en analysant la correspondance entre les candidats et le mot faux comme une erreur d'omission, insertion, transposition ou substitution.

3.2 Le module de correction d'erreurs de génération

L'analyse morphologique d'un mot par l'analyseur de PILAF détermine une racine (base morphologique) et un ensemble de flexions.
Ayant détecté une erreur sur une chaîne dont la racine est correcte, on peut traiter cette erreur en supposant que les caractères qui suivent la racine contiennent les informations correspondant à la flexion envisagée pour ce mot. Ceci permet d'obtenir les variables grammaticales associées au mot erroné. Quant à la catégorie lexicale, elle est fournie par la racine.

Le module de correction d'erreurs par génération [COHARD 88, BAINIER 89] permet de traiter le pluriel irrégulier et les fautes de conjugaison effectuées sur les formes conjuguées des verbes qui peuvent présenter plus d'une racine (une mauvaise racine est employée à la place d'une autre). Ce genre d'erreurs est corrigé en utilisant le générateur morphologique. Pour une racine détectée, on engendre toutes les formes correspondant à l'ensemble de variables grammaticales trouvées dans le mot mal orthographié.
Exemples :

I. analyse de ***chevals*** :

racine :	**chev**	substantif commun
désinences :	**al**	masculin
	s	pluriel

chevals est une chaîne incorrecte, mais la racine ***chev*** est correcte et donne les informations *substantif commun.* Ensuite, l'analyse de la désinence ***als*** à partir de l'état initial permet de détecter des marques de *masculin* et de *pluriel.*

II. analyse de ***allerons*** :

racine :	**all**	verbe
désinence :	**erons**	1ère personne, pluriel, futur de l'indicatif

allerons est une chaîne incorrecte dont la base ***all*** permet d'induire la catégorie ***verbe*** et le modèle de conjugaison. Ensuite, l'analyse de ***erons*** fournit les renseignements suivants : *première personne du pluriel du futur de l'indicatif.*

Pour le premier exemple proposé, on applique la génération à partir de la base **chev** avec les renseignements linguistiques *substantif masculin pluriel*, ce qui permet d'obtenir la chaîne *chevaux*. De même, à partir de la base **all** et des variables *première personne du pluriel du futur de l'indicatif*, on obtient la chaîne *irons*.

L'algorithme de correction par génération réalise :

a) la *reconnaissance de la plus longue base* β du mot erroné ;

b) la *reconnaissance de la terminaison* τ(une partie intermédiaire placée éventuellement entre la base β et la terminaison τ n'est pas traitée par l'algorithme) ;

c) la *génération*, à partir de la base β, du mot ayant comme racine cette base et comme variables les mêmes variables associées à la terminaison τ.

4 LES OUTILS DE PILAF ET LA LANGUE PORTUGAISE

Nous avons réalisé un essai d'utilisation des outils de PILAF en vue du traitement de la langue portugaise employée au Brésil [COURTIN 89b]. Dans les paragraphes 4.1 et 4.2, nous abordons cette expérience.

4.1 Analyse morphologique et syntaxique

Les analyseurs de PILAF étant paramétrables, nous avons envisagé d'utiliser ces outils pour le traitement de la langue portugaise.
En ce qui concerne les paramètres nécessaires au traitement du portugais, nous avons créé une grammaire composée de plus de trente règles et d'une centaine de modèles permettant d'accepter des bases représentatives de différentes catégories de substantifs communs, adjectifs qualificatifs et possessifs, verbes de première, deuxième et troisième conjugaison, quelques verbes irréguliers, conjonction de coordination, conjonction de subordination, locutions de subordination, locutions prépositionnelles, déterminants, pronoms, la négation, des signes de ponctuation, etc.

Nous avons relevé de cette étude vingt trois catégories morphologiques que nous avons traité aussi au niveau de l'analyse syntaxique.
Nous avons détecté l'existence de nouvelles catégories morphologiques, comme les *collectifs*, lesquelles ne sont pas représentés dans la version française de PILAF et qui pourraient pourtant recevoir un traitement systématique en portugais.

Nous avons analysé les traits morphologiques associés aux mots et nous avons configuré un ensemble élémentaire de traits en portugais. Cet ensemble est composé des variables et des valeurs utilisés pour le français, mais aussi d'autres variables.
Par exemple, nous trouvons en portugais la variable *degré* associée régulièrement aux adjectifs qualificatifs et aux substantifs communs. Comme valeurs possibles pour cette variable nous avons le *diminutif*, le *normal* et l'*augmentatif*.
Nous trouvons également une quatrième valeur de *degré* associée aux adjectifs qualificatifs : le *superlatif*.

Les structures définies ont permis la création d'un dictionnaire de bases et d'un dictionnaire de terminaisons et l'obtention de nos premiers résultats en analyse morphologique du portugais.

Nous avons analysé les dépendances entre les composants fondamentaux d'une phrase et nous avons relevé, dans un premier temps, des rapports de dépendance essentiels en portugais. Ceci nous a permis de dégager neuf catégories lexico-syntaxiques élémentaires.
Nous avons écrit une trentaine de règles de dépendances modélisant des rapports de dépendance en portugais. Soumises à l'analyseur syntaxique de PILAF, ces règles nous ont permis d'obtenir des résultats cohérents et complets pour l'ensemble envisagé de cette langue.

Cet ensemble étant toutefois réduit pour une expérience d'ampleur pratique, d'autres règles de dépendances doivent être conçues. Nous envisageons de déterminer, tout d'abord, une partie de la langue portugaise à nommer "fondamentale" (en correspondance à l'idée de "français fondamental"). Nous pourrons ensuite passer à la détermination des rapports syntaxiques de dépendance faisant partie de ce sous-ensemble et effectuer, ainsi, l'écriture des règles de dépendance concernées.

Cette expérience, même si limitée à un ensemble bien élémentaire de règles, modèles et dictionnaires, nous a permis d'obtenir des résultats satisfaisants. Nous avons pu constater la faisabilité du traitement de la langue portugaise par moyen des analyseurs de PILAF, ce qui nous a permis d'envisager, à partir des résultats obtenus, une série d'applications.

4.2 Le traitement d'erreurs

Parmi les applications envisagées, nous pouvons citer le traitement d'erreurs aux niveaux lexical et syntaxique effectué par les outils de traitement d'erreurs de PILAF.

L'algorithme de correction d'erreurs par clé squelette est directement applicable au portugais, dès que nous disposons d'un dictionnaire de clés/formes du portugais.

La grammaire construite pour l'analyse morphologique du portugais peut également être utilisée pour la génération morphologique. De cette manière, nous pouvons appliquer au portugais la méthode de correction d'erreurs par génération morphologique.
Cette méthode se montre d'un intérêt vital au traitement d'erreurs en portugais, puisque nous y pouvons constater une large gamme de désinences.
Il serait toutefois souhaitable d'élargir les possibilités de l'algorithme de correction par génération, en corrigeant notamment les féminins irréguliers (engendrés à travers une mauvaise désinence), qui constituent une erreur assez fréquente en portugais.

L'algorithme de correction d'erreurs lexicales par la phonétique, décrit en détail dans le chapitre suivant de ce document, peut aussi être appliqué au portugais (nous revenons à ce sujet dans le chapitre 3, paragraphe 5.1).
Des nuances dans la phonologie de cette langue, parlée différemment en Afrique, en Europe ou en Amérique, et encore bien différemment au nord ou au sud du Brésil, exigeraient d'abord une analyse de l'amplitude à donner à une *modélisation phonétique* de la langue portugaise

Au niveau syntaxique, l'étude réalisée pour la langue française (chapitres 4 et 5) peut servir comme point de départ à une étude en vue de la vérification syntaxique et correction des erreurs d'accord en portugais.
Il faut cependant préciser que les accords du participe passé, thème de travaux en ce qui concerne la détection/correction d'erreurs au niveau syntaxique et sémantique en français, suivent des règles plus souples en portugais.

CHAPITRE 3

CORRECTION D'ERREURS LEXICALES PAR LA PHONETIQUE

Ce chapitre est consacré à une approche détaillée de la correction de fautes d'orthographe par la phonétique.

Nous commençons par étudier quelques systèmes basés sur une technique de type phonétique pour corriger des textes écrits en langue naturelle ; ensuite, nous décrivons les deux versions de la méthode que nous avons mise au point en utilisant le transducteur général d'états finis de PILAF. Cette méthode a pour objet de reconstruire des graphies correspondant aux transcriptions phonétiques engendrées à partir de la chaîne erronée.

1 INTRODUCTION

Les erreurs du type phonétique ou du type phonogrammique (conformément à la classification de Catach, Duprez et Legris, chapitre 1, paragraphe 1 2) sont considérées comme de plus en plus importantes [EMIRKANIAN 88, VERONIS 88c] et, suivant les applications, comme les plus fréquentes dans un texte écrit en français.

Les erreurs du type typographique, considérées, il y a vingt ans, comme les plus habituelles, prennent une place moins importante à mesure que la fréquence de bruit dans les transmissions de données se réduit et le profil des applications change.
L'ordinateur est aujourd'hui accessible à une vaste gamme d'utilisateurs, et le traitement de la langue naturelle est inséré dans des domaines d'applications les plus divers. Les erreurs dues à une mauvaise compétence concernant la langue deviennent courantes et leur traitement demande des algorithmes spécifiques de correction.

Un utilisateur qui connaît moins bien la langue, en ce qui concerne l'orthographe, mais qui connaît bien la langue parlée, peut avoir des difficultés au cours de l'écriture, en proposant une mauvaise graphie pour un mot qu'il sait bien prononcer.

Par exemple, la graphie ***ocuranse*** peut être considérée comme une mauvaise écriture de la chaîne *occurrence*. Dans ce cas, le mot erroné et son correspondant correct se prononcent de la même manière.

On remarquera que les algorithmes de correction d'erreurs typographiques ne conviennent généralement pas à ce type de problème (dans un mot faux comme ***ocuranse***, existent plusieurs erreurs d'édition).

Pour cette raison, il semble important, dans un système qui réalise la correction d'erreurs dans un texte écrit en langue naturelle, de disposer d'une méthode de correction qui tient compte de la prononciation du mot.

1.1 Etudes sur les erreurs du type phonétique

La correction utilisant des invariants phonétiques est pratiquée par d'autres systèmes de traitement d'erreurs ([LAHENS 86, MARET 87, EMIRKANIAN 88a, VERONIS 87, LAPORTE 89a]).

Le traitement des erreurs du type phonétique est effectué soit par des méthodes *stochastiques*, comme celle employée dans le système VORTEX, soit par des méthodes *déterministes*, comme celles employées dans le logiciel TOPH et dans les systèmes ARCHIMEDE, CERIL/LADL et DECOR.

Au point de vue correspondance phonie ↔ graphie, deux situations se présentent. Soit cette correspondance est réalisée au moyen d'une table de *références des graphèmes interchangeables* (des graphèmes qui se prononcent de la même manière), comme par exemple dans le système ARCHIMEDE, soit cette correspondance est réalisée au moyen d'une *transduction phonétique* (comme dans le système du CERIL/LADL ou dans l'environnement PILAF).

Les paragraphes 1.1.1 à 1.1.5 présentent en détail les systèmes cités.

1.1.1 Le système VORTEX

En vue de la correction des fautes d'usage, le système VORTEX [LAHENS 86, PERENNOU 86] applique, dans le codeur orthographique ORTHO, des règles de codage orthographique (voir chapitre 1, paragraphe 1.4).
Ces règles correspondent à un ensemble de règles de réécriture avec des probabilités associées.
Nous présentons un exemple extrait de [LAHENS 86] :

Règles qui traduisent les fautes possibles sur un "o" interne :

o	→	*o*	: 0,97	*au*	: 0,02	*eau*	: 0,01

Règles qui traduisent les fautes possibles sur un "o" terminal (*OF*) :

OF	→	*o*	: 0,20	*au*	: 0,01	*eau*	: 0,01
		aud	: 0,06	*aut*	: 0,02	*ot*	: 0,35
		oth	: 0,34	.....			

Règles qui traitent le "o" initial par rapport à l'insertion possible d'un "h" (HO) :

HO → *vide* : 0,95 *h* : 0,05

Règles qui traitent le "o" initial par rapport à l'omission d'un "h" (H) :

H1 → *vide* : 0,10 *h* : 0,90

La source émettra *ostrogoth* sous la forme HO-o-s-t-r-o-g-OF, ce qui permet le recodage en *austrogaud*, *hostrogot*, etc.
Un dictionnaire spécialisé stocke les mots sous la forme de sous-séquences orthographiques ayant chacune une valeur phonétique (exemple : *orthodoxe* est stocké comme o-r-th-o-d-o-xe).

Comme Laporte l'a déjà remarqué [LAPORTE 88], nous ne connaissons pas en détail les principes de choix des éléments composants de ce codage.

1.1.2 Le logiciel TOPH

Le logiciel TOPH [MARET 87] est un transducteur qui permet de réécrire une chaîne d'entrée quelconque en une chaîne de sortie quelconque. Une expérimentation a été faite, avec ce logiciel, dans le cadre du PRC "Communication Homme Machine", pôle "Langage Naturel", ayant comme intérêt particulier la comparaison de deux mots d'orthographe différente possédant plus ou moins le même invariant phonétique.

La comparaison de chaînes est faite par des règles de substitution généralisées, du type

(***X*** : sous-chaîne de référence , ***Y*** : sous-chaîne observée).

La base de règles contient environ un millier de règles.

Un programme effectue la comparaison des deux chaînes proposées. Il fournit la dissimilarité et les alignements optimaux.
L'algorithme utilisé procède par énumération des sous-chaînes terminales X, Y de *a* et *b* ($a = a_1 a_2 \ldots\ldots a_i$ et $b = b_1 b_2 \ldots\ldots b_j$).
Cette énumération est limitée par la longueur maximale des sous-chaînes X, Y répertoriées dans la base de règles.

Une valeur de dissimilarité est toujours calculée par défaut (règles générales de substitution, omission, insertion, transposition). La mesure *m* de similarité employée est

$$S(a, b) = s(a, b) \,/\, \max\{|a|, |b|\}$$

où

s(*a*, *b*) : longueur de la plus longue sous-chaîne commune entre *a* et *b* et
|*a*| , |*b*| : longueurs de *a* et *b*, respectivement.

Les formes voisines d'un terme d'entrée sont extraites d'un lexique (sur la base de la mesure *m* de similarité) et ces solutions sont réordonnées suivant le score de dissimilarité fourni.

Exemple (extrait de [MARET 87]) :

chaîne de référence : ***ostrogoth***

chaîne observée : *estragon*
alignement optimal :

o	s	t	r	o	g	o	t	h
e	s	t	r	a	g	o	n	

20+	0+	0+	0+	80+	0+	0+	100+10 = 210

chaîne observée : *austraugaud*
alignement optimal :

o	s	t	r	o	g	o	t	h
au	s	t	r	au	g	au	d	

10+	0+	0+	0+	10+	0+	10+	20+	10 = 60

1.1.3 Le système ARCHIMEDE

Veronis [VERONIS 88b], dans son système (voir chapitre 1, paragraphe 1.3), emploie également une méthode de réécriture pour corriger les erreurs phonétiques.
La réécriture est réalisée à l'aide d'une matrice de règles de réécriture, élaborée à partir d'un dictionnaire d'environ 4000 mots.
Considérons le mot à corriger *pautto*. Une partie de la matrice de règles de réécriture utilisée est présentée dans la figure 3.1 (cet exemple, aussi bien que la figure 3.1, sont extraits de [VERONIS 88a]).

Soit l'alphabet **A** = {a, e, o, p, t, u} et la relation de *similarité entre sous-chaînes* décrite dans la matrice de la figure 3.1. Soit **E** l'ensemble des sous-chaînes qui peuvent être mises en relation :

E = {a, au, e, eau, o, p, pp, t, tt, u}

Le transcodage est effectué par un automate construit à partir de **E** (ce qui produit les graphies *poteau*, *potteau*, etc). Ensuite, un algorithme de

calcul de la dissimilarité est appliqué. Finalement, une recherche est opérée dans le dictionnaire.

Ce genre de correction peut satisfaire un sous-ensemble de la langue employé dans une application bien circonscrite, comme c'est le cas du système ARCHIMEDE [VERONIS 88b]. Il serait vraisemblablement insuffisant dans le cas du traitement d'erreurs phonétiques d'un texte en général.

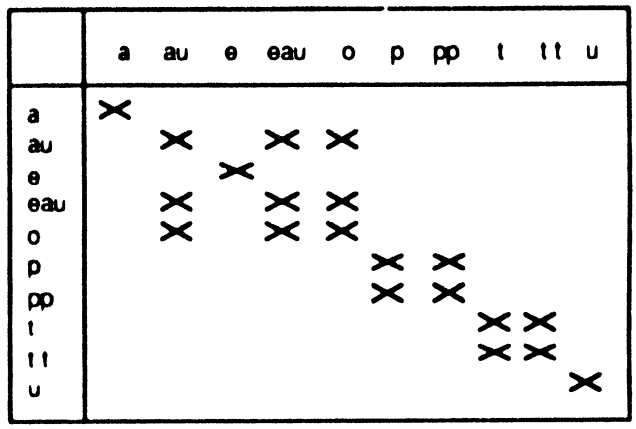


Figure 3.1
Quelques règles de réécriture
(Système ARCHIMEDE)

1.1.4 Le système développé au CERIL/LADL

Laporte et Silberztein [LAPORTE 88, LAPORTE 89a, LAPORTE 89b] emploient une technique différente des précédentes dans l'aide à la correction orthographique par phonétisation. Une ou plusieurs transcriptions phonétiques sont calculées pour le mot faux. A partir de ces transcriptions, le système cherche dans le dictionnaire les mots corrects qui se prononcent de la même manière ou d'une manière proche du mot à corriger.
Le système de Laporte et Silberztein, développé au CERIL / LADL (Centre d'études et de recherches en Informatique Linguistique) à Paris VII, se sert d'un dictionnaire phonétisé qui permet de chercher directement les correspondants corrects du mot erroné.
Ce dictionnaire fait partie du DELA (Dictionnaires Electroniques du LADL) - un système de données linguistiques qui regroupe des algorithmes et des dictionnaires morphologiques et phonémiques du français.

Le DELAP, un des dictionnaires appartenant à la base de données DELA, donne des informations phonémiques sur les mots. Il compte environ 64000 mots simples [LAPORTE 88].
A chaque entrée du DELAP correspondent une représentation orthographique et une représentation de la prononciation du mot, celle-ci étant exprimée dans un alphabet phonémique. Ces deux représentations sont suivies d'informations grammaticales et flexionnelles sur le mot.

Le DELAP est à l'origine du DELAP-F, qui rassemble toutes les formes phonémiques fléchies, rangées par ordre alphabétique (environ 530000 formes). Pour chaque transcription phonétique, ce dictionnaire donne une liste de formes fléchies correspondantes (verbes conjugués, pluriels, féminins, etc).

Le programme de vérification et correction orthographique construit par Laporte et Silberztein est constitué de deux modules
1 - la production d'une ou plusieurs transcriptions phonétiques du mot erroné ;
2 - la recherche du ou des mots corrects qui correspondent à ces transcriptions, dans le DELAP-F.

Sur un ordinateur VAX 730, avec des algorithmes programmés en langage C, le phonétiseur de mots inconnus produit ses résultats en moins d'une seconde par mot erroné. Quant à l'étape suivante de la correction, on n'a pas de mesures de temps rapportées

1.1.5 La transcription phonétique dans PIAF

Une méthode similaire à celle adoptée par Laporte et Silberztein a été proposée dans le projet PIAF [COURTIN 77].
Le transducteur d'états finis de PIAF, muni d'un ensemble de règles, modèles et dictionnaires phonétiques, peut produire la ou les transcriptions phonétiques associées à un mot faux. Ces transcriptions permettent d'engendrer un ensemble de graphies correctes candidates à corriger le mot erroné.
La génération des graphies correctes pourrait être faite à l'aide d'un dictionnaire de bases phonétisées, par un nouveau processus de transduction, celui-ci étant très simplifié. Le dictionnaire de bases phonétisées sert alors comme entrée du transducteur, et le nouveau processus de transduction produit des graphies phonétiquement équivalentes ou voisines du mot erroné.

Les graphies candidates engendrées par cette deuxième transduction sont ensuite analysées morphologiquement, ce qui permet d'éliminer des candidates qui ne correspondent pas à des mots français.

Emirkanian et Bouchard [EMIRKANIAN 88a, EMIRKANIAN 88b] réalisent, eux aussi, la correction d'erreurs du type phonétique en français, mais nous ne connaissons pas de détails à propos de la réalisation effectuée.

1.2 Principe adopté

Nous pensons que la correction d'erreurs phonétiques par des règles de réécriture ne peut couvrir qu'une partie des rapports phonie ↔ graphie. Le traitement des chaines phonétiques plus longues (celles qui groupent plusieurs phonèmes, comme dans les sous-chaînes de fin de mot */tions/*, */ill/*, etc) impose l'utilisation d'un nombre considérable de relations Une transduction permettrait d'arriver à l'ensemble complet des transcriptions phonétiques associées à un mot.

Nous avons donc adopté une technique qui réalise la transduction phonétique du mot faux, pour obtenir ses différentes transcriptions phonétiques

La transduction phonétique, faite par le biais du transducteur de PILAF, présente toutefois des difficultés par rapport à la modélisation phonétique de la langue française.
La complexité du système phonologique du français et ses correspondances phono-graphiques non seulement rendent difficile une description très détaillée de ce système, mais aussi obligent à une simplification, si nous envisageons l'application constituée par la correction de fautes d'orthographe.

En vue d'une transcription phonétique des mots erronés, nous avons analysé les phonèmes et les séquences de phonèmes fréquents en français (conformément à Catach, Meissonnier, Demeyère, Delgado, Lallias, etc [CATACH 79, CATACH 80a, CATACH 80b, CATACH 80c, DELGADO 86, GRAMMONT 63]) et nous avons proposé des groupes de graphèmes les plus importants en vue d'une correction, pour le processus de transduction phonétique.
Le paragraphe 3 de ce chapitre présente en détail la modélisation conçue pour la détermination des transcriptions phonétiques associées à un mot.

1.3 La correction d'erreurs

Notre algorithme de correction est basé sur la transduction phonétique du mot faux, suivie d'une reconstruction graphique [COURTIN 88, COHARD 88].

La reconstruction graphique, fortement liée à la structure des données disponibles, a été l'objet de deux expérimentations différentes.
D'abord, nous avons adopté la méthode suggérée dans [COURTIN 77] qui se sert d'un dictionnaire de bases morphologiques phonétisées pour engendrer, à travers un processus de transduction simplifié, des graphies associées aux transcriptions phonétiques.
Toutefois, la création d'un tel dictionnaire associé à l'ensemble d'environ 26000 bases morphologiques de PILAF, nous a découragé d'employer cette méthode de reconstitution graphique (les difficultés liées à la construction d'un dictionnaire de bases phonétisées correspondant au dictionnaire de bases de PILAF sont considérées dans l'introduction de ce travail, paragraphes 2 et 3, et sont reprises sous les titres **Première version** et **Deuxième version** qui suivent).
Alors, en vue de l'intérêt à l'application de cette méthode de correction sur un dictionnaire en vraie grandeur, nous avons proposé et mis en oeuvre un deuxième algorithme de reconstruction graphique. Ce deuxième algorithme n'utilise plus de bases morphologiques phonétisées, mais des listes de règles de reconstitution graphique. Ces règles ont été conçues en analysant l'ensemble de formes contenues dans nos dictionnaires, soit environ 220000 formes.

Nous exposons ensuite les principes de fonctionnement des versions mises en oeuvre pour la reconstruction graphique. Ces deux versions sont détaillées dans les paragraphes suivants (4.1 et 4.2).

Première version

Elle prévoit, comme nous l'avons spécifié dans le paragraphe précédent, l'utilisation d'un dictionnaire de bases morphologiques phonétisées (ou dictionnaire *morpho-phonétique*) pour obtenir, à travers un nouveau processus de transduction, les graphies correctes associées aux transcriptions phonétiques.

Cette méthode permet d'engendrer, à partir d'une transcription phonétique, un ensemble de graphies constituées soit de la base phonétisée elle-même (une base morpho-phonétique correspondant à cette chaîne phonétique) soit de la concaténation des bases morpho-phonétiques qui la composent (les segments correspondant à des bases morphologiques phonétisées trouvées dans le dictionnaire morpho-phonétique).

Toutefois, pour l'application une telle méthode, il faut disposer d'un dictionnaire phonétisé, dont la création n'est pas un processus automatique.
A chaque base morphologique, doit correspondre une seule base morpho-phonétique.
La création du dictionnaire morpho-phonétique exige, pour une base morphologique :

- sa transduction phonétique préalable, qui engendre une ou plusieurs transcriptions phonétiques ;
- le choix de la transcription phonétique qui caractérise le mieux cette base.

Nous avons créé le dictionnaire morpho-phonétique correspondant au mini-dictionnaire de PILAF. Ce dictionnaire est constitué par 688 entrées (bases et terminaisons morphologiques), à partir desquelles nous avons engendré 640 bases morpho-phonétiques. Ceci nous a permis de programmer et de tester la première version de correction par la phonétique [COURTIN 88] (présentée en détail dans le paragraphe 4.1).
Cependant, étant intéressés au développement d'une application sur un dictionnaire de taille réelle, nous avons considéré la création d'un dictionnaire de bases morpho-phonétiques de taille réelle.
Nous avons constaté la complexité du processus de création d'un tel dictionnaire, liée à l'espace mémoire nécessaire au stockage de toutes ces données. Ceci nous a fait concevoir une deuxième méthode permettant la reconstruction graphique du mot faux sans avoir besoin d'un dictionnaire de bases phonétisées.

Deuxième version

La deuxième méthode de reconstruction graphique proposée engendre des graphies alternatives à partir des listes de règles de composition graphique associées aux éléments phonétiques.
Par cette méthode on élimine la construction d'un dictionnaire morpho-phonétique. A la place, nous utilisons des listes stockées en mémoire qui contiennent des règles de composition graphique associées aux phonèmes PILAF.
Comme exemple nous donnons les règles de composition graphique pour le phonème *m* de PILAF (pour plus de détails voir paragraphe 4.2.1) :

graphie	*début de mot*	*milieu de mot*	*fin de mot*
m	oui	oui	oui
me	non	non	oui
mm	non	oui	non
mme	non	non	oui

Ayant pour but de réduire le nombre des graphies composées, le programme de concaténation des divers éléments graphiques tient compte de la composition des mots contenus dans le dictionnaire de formes de PILAF (voir paragraphe 4.2.2).

En plus, certains cas spéciaux ont été relevés parmi les formes stockées dans les dictionnaires de PILAF, ce qui nous a permis de créer une structure (CP) contenant des *cas particuliers* : des formes dont les règles de composition graphique sont particulières.
La génération graphique réalise alors une recherche dans cette structure, pour trouver des graphies particulières qui peuvent être associées à la transcription phonétique calculée.

Finalement nous sélectionnons, parmi les graphies engendrées, celles qui appartiennent au dictionnaire de formes du système.

2 LA CHAINE ECRITE ET LA CHAINE ORALE

2.1 Graphèmes et Phonèmes

Un graphème peut être caractérisé comme la plus petite unité distinctive ou significative de la chaîne écrite [CATACH 80], composée d'une lettre ou d'un groupe de lettres ou signes auxiliaires, ayant une référence phonique dans la chaîne parlée.

Le concept de phonème est plutôt lié à l'oral : un phonème est la plus petite unité distinctive de la chaîne orale.

En envisageant la correction de fautes d'orthographe par la phonétique, nous avons étendu le concept de graphème pour permettre des chaînes plus longues, traitées ici comme pseudo-graphèmes ou, simplement, ***p-graphèmes***. Exemple : on utilise la chaîne *enou* (la concaténation de plusieurs graphèmes élémentaires) comme un p-graphème.

Le concept de pseudo-phonème a été aussi créé, pour permettre des sous-chaînes orales plus longues.
Les pseudo-phonèmes ou, simplement, ***p-phonèmes***, sont constitués soit de phonèmes élémentaires soit de chaînes de phonèmes élémentaires. Exemple : *c.s.y.on* (comme dans *action*).

La relation ***p-phonème --> p-graphème*** est du type **1 : n**, $n \geq 1$. A un p-phonème peuvent correspondre un ou plusieurs p-graphèmes. Ici, **n** représente le nombre de p-graphèmes associés à un p-phonème.

Exemples :

Au p-phonème **o.n** sont associés les p-graphèmes *onne*, *one*, etc.

Au p-phonème **s** sont associés les p-graphèmes *s*, *ss*, *c*, *ç*, etc.

Au p-phonème **muet** correspondent le p-graphème *h* et les p-graphèmes *s*, *x*, *e*, ... en fin de mot (c'est-à-dire, des graphèmes qui peuvent être muets lors de la prononciation du mot).

La relation ***p-graphème --> p-phonème*** est du type **1 : m**, m ≥ 1. A un p-graphème peuvent correspondre un ou plusieurs p-phonèmes. Ici, **m** représente le nombre de p-phonèmes associés à un p-graphème.

Exemple :

Au p-graphème *ch* sont associés les p-phonèmes *ch* et *k*.

Soit le p-graphème *s* de fin de mot. A ce p-graphème sont associés les p-phonèmes *s* et *muet*.

2.2 Choix des phonèmes liés à la correction de fautes d'orthographe

Le problème des graphèmes liés à l'oral a été déjà étudié chez plusieurs linguistes comme Catach, Lallias, Demeyère, etc. En regardant les propositions présentées par ces auteurs, nous avons proposé un ensemble de 33 p-phonèmes élémentaires, associés :

- aux voyelles orales ;
- aux voyelles nasales ;
- aux consonnes et
- aux semi-voyelles.

Le tableau 1 présente les p-phonèmes élémentaires proposés, avec des exemples. Le tableau 2 présente la correspondance entre les p-phonèmes élémentaires et l'Alphabet Phonétique International.

Certains p-phonèmes élémentaires combinés produisent des sous-chaînes orales importantes, associées à des groupes de p-graphèmes souvent utilisés. Ceci nous a permis de concevoir des p-phonèmes et de p-graphèmes non-élémentaires qui permettront d'améliorer le temps de réponse d'une transduction phonétique.

Exemple :

Le p-phonème *y.on*, fréquemment trouvé dans des flexions verbales (comme *soyons*), est composé du phonème élémentaire *y* suivi du phonème élémentaire *on*.

Les 33 p-phonèmes élémentaires combinés permettent d'arriver à plus de deux cents p-phonèmes non-élémentaires, associés à des chaînes graphiques plus longues.

Exemples :

Le p-phonème *o.in* s'applique aux p-graphèmes *oin*, *oint*, etc.

Le p-phonème *é.t* est applicable aux p-graphèmes *ette*, *ettes*, *ête*, *êtes*, *ète*, *ètes*, etc.

Il est nécessaire d'observer que nous n'avons pas considéré, dans la modélisation effectuée, le niveau phonologique d'analyse du français. Une étude de la fonction des phonèmes dans la langue, réalisé avec l'aide de phonéticiens, permettrait d'incorporer au système développé ces caractéristiques indispensables au point de vue pratique.

P-PHONEMES ELEMENTAIRES

---

a .	appel, arbre, pâte	*b.*	boucle, brebis
é.	lait, blé, après, manger,tête	*ch.*	tâche, couche, cheval
e.	recherche, petit	*d.*	sud, deux
i.	ici, vie	*f.*	faveur, oeuf
o.	beau, mot, gauche	*g.*	guichet, gant
u.	tu, sûr	*j.*	jouet, genou
ou.	doux, tout	*k.*	kilo, cave, psychopathe, qui
oe.	oeil, leur	*l.*	lourde, balle
		m.	sommet, mouche
		n.	nez, annonce
		p.	pile, coupe
y.	fille, mention, yoga	*r.*	route, arrosoir, finir
w.	squash, week-end, loin	*s.*	passion, nation, cil, ça, son
		t.	thème, tard
		v.	vif, wagon
		z.	maison, zone
in.	sapin, pain		
an.	dans, temps, vent, lampe		
on.	coupon, tombé		
un.	un, jungle		
		gn.	baigner, châtaigne
		ing.	parking

---

muet haut, pommes

TABLEAU 1

LES P-PHONEMES ET
L'ALPHABET PHONETIQUE INTERNATIONAL

a.	[a]	[ɑ]	*b.*	[b]	
é.	[e]	[ɛ]	*ch.*	[ʃ]	
e.	[ə]		*d.*	[d]	
i.	[ı]	[ȷ]	*f.*	[f]	
o.	[o]	[ɔ]	*g.*	[g]	
u.	[y]	[ɥ]	*j.*	[ʒ]	
ou.	[u]		*k.*	[k]	
oe.	[ø]	[œ]	*l.*	[l]	
			m.	[m]	
y	[ȷ]	[i]	*n.*	[n]	
w	[w]		*p.*	[p]	
			r.	[ʀ]	
			s.	[s]	
in.	[ɛ̃]		*t.*	[t]	
an.	[ɑ̃]		*v.*	[v]	
on.	[ɔ̃]		*z.*	[z]	
un.	[œ̃]				
			gn.	[ɲ]	
			ing.	[iŋ]	

TABLEAU 2

Le tableau 3 présente quelques autres p-phonèmes non élémentaires avec des exemples.

QUELQUES P-PHONEMES NON-ELEMENTAIRES	
a.n	â**ne**, ba**nane**, pays**anne**
an.p	**lampe**, **temple**
a.s.y.on	**passion**, continu**ation**
a.y	**ail**, trav**ail**, bat**aille**
é.m.an	**sémant**ique, **cément**
i.l	**ville**, **fil**
k.a.r	**car**otte, **car**, **carré**, **karm**a
k.s.y.on	**action**, fle**xion**
s.t.y.on	**gestion**

NOTE : On a utilisé des mots corrects comme exemples pour permettre une association plus facile avec la phonétique de la langue française

TABLEAU 3

Validation

Les p-graphèmes choisis pour effectuer la transduction phonétique d'un mot sont réunis dans un dictionnaire, utilisé par le transducteur de PILAF.
Ces p-graphèmes sont associés à un ensemble de modèles phonétiques qui sont validés ou non pendant le traitement du mot faux. La validation d'un modèle donne lieu à une sous-chaîne orale : le p-phonème associé à ce modèle.
La transduction phonétique d'un mot est faite dans le sens

gauche → droite

en prenant en compte le plus long p-graphème trouvé.
Exemple :

mot : ***baze***

séquence de validations :

1 chaîne analysée : /baze /
le plus long p-graphème trouvé : /b/ (modèle **b**)
Sous-chaîne orale : *b*

2 chaîne analysée : /aze /
le plus long p-graphème trouvé : /aze / (modèle **asefin**)
Sous-chaîne orale : *a.z*

TRANSCRIPTION PHONETIQUE RESULTANTE : *b.a.z*

Lors du découpage phonétique d'un mot, plusieurs modèles peuvent être validés à partir d'un même p-graphème, ce qui peut produire plusieurs transcriptions phonétiques associées à un mot faux.

3 STRUCTURES DE DONNEES UTILISEES PENDANT LA TRANSDUCTION PHONETIQUE

La transduction phonétique d'un mot est faite par le transducteur d'états finis de PILAF en utilisant trois types principaux de structures de données : les règles, les modèles phonétiques et le dictionnaire de p-graphèmes.

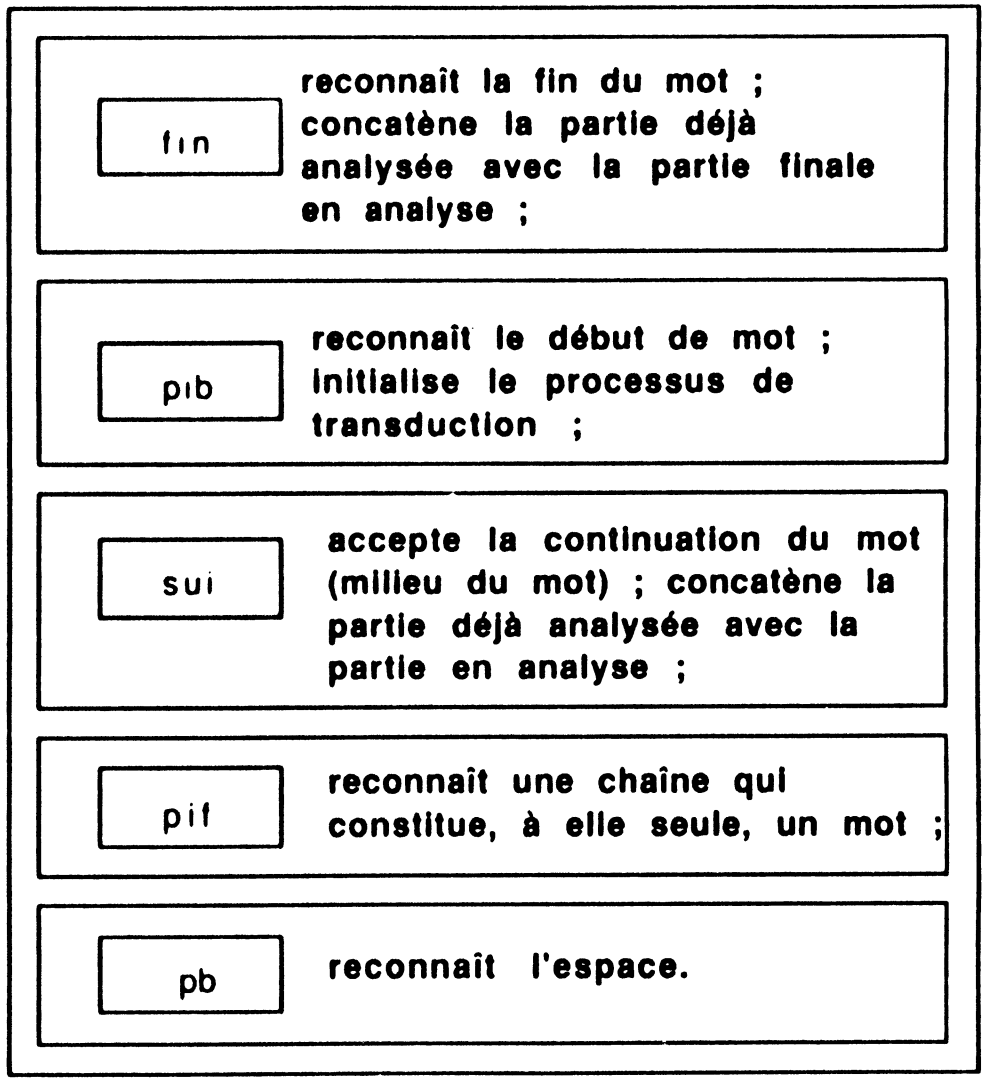


Figure 3.2

Règles de transduction phonétique d'un mot

3.1 Règles, modèles et dictionnaire phonétiques

3.1.1 Les règles

Les règles de transduction phonétique permettent d'analyser le mot en reconnaissant la position de ses graphèmes composants.

Nous avons défini des règles qui permettent la reconnaissance de la fin du mot, du début, du milieu du mot et aussi de l'espace.
Dans la figure 3.2, nous décrivons le rôle de chaque règle de transduction phonétique.

Par l'utilisation d'une règle spécifique - la règle **plf**, il est aussi possible de reconnaître une chaîne qui constitue, à elle seule, un mot. Cette situation peut être intéressante pour les mots composés d'un seul p-phonème.

Exemples :	aux	(modèle **o**)
	un	(modèle **un**)
	oeil	(modèle **oeil**).

3.1.2 Le dictionnaire de p-graphèmes

Les p-graphèmes sont stockés dans un dictionnaire, actuellement avec plus de mille entrées. Pour chacun des p-graphèmes, on trouve un modèle phonétique associé.
La gestion du dictionnaire de p-graphèmes est réalisée de manière analogue à celle du dictionnaire de bases morphologiques de PILAF (voir chapitre 2).

Les p-graphèmes sont distribués en fichiers dictionnaires de 100 enregistrements au maximum, triés en ordre alphabétique inverse.
L'accès à un enregistrement est obtenu par le moyen d'une table qui contient les entrées correspondantes à chaque fichier dictionnaire, avec ses p-graphèmes limitatifs inférieur et supérieur (la PHTABMAITRE).

A la tentative d'insertion d'un centième élément, le fichier dictionnaire est éclaté en deux. Une moitié du fichier d'origine est stockée dans un nouveau fichier dictionnaire et l'autre reste dans ce propre fichier.
Toutefois, nous remarquons que le dictionnaire de p-graphèmes fait partie des paramètres du système. Il ne doit pas être modifié par l'utilisateur.

Hormis le chaînage circulaire, un enregistrement du dictionnaire de p-graphèmes a un format similaire à celui des bases morphologiques.

L'annexe 1 présente la liste des p-graphèmes contenus dans le dictionnaire de p-graphèmes.

3.1.3 Les modèles phonétiques

Ces modèles contiennent les informations qui se réfèrent à leur applicabilité, dans un état donné, par l'automate.

Les informations associées à un modèle phonétique sont :

- le nom du modèle ;
- les règles d'arrivée dans cet état par l'automate ;
- les règles qui valident ce modèle et
- son p-phonème associé (nommé "code dérivation").

Exemple :

Nom du modèle :	**a**
Règles d'arrivée :	pib sui
Validation :	sui fin
Code dérivation :	***a***

Observations :

1) le modèle **a** peut s'appliquer au début ou au milieu du mot (règles pib ou sui)
2) pour valider ce modèle, la chaîne peut avoir ou non une continuation (règles sui ou fin)
3) après la validation du modèle **a**, le code dérivation ***a*** est utilisé comme résultat de cette étape de la transcription phonétique (le modèle **a** est le modèle phonétique associé au p-phonème ***a***).

Pour chaque p-phonème existe un modèle associé. Ce modèle aura pour code dérivation la transcription du p-phonème concerné. La transduction phonétique réalisée se sert de plus de deux cents modèles phonétiques.
Pour chaque p-graphème existe au moins un modèle phonétique associé, ce qui garantit la correspondance entre le p-graphème et le(s) p-phonème(s) associé(s).
Exemples :

Au p-graphème /**a**/ est associé le modèle **a**.
Au p-graphème /**à** / est aussi associé le modèle **a**.
Au p-graphème /**w**/ sont associés le modèle **v** et le modèle **w**.
Au p-graphème /**qu**/ sont associés le modèle **k** et le modèle **qw**.
Au p-graphème /**ch**/ sont associés le modèle **ch** et le modèle **k**.

Quand un modèle est considéré comme applicable, le p-phonème associé à ce modèle est utilisé par l'automate pour la génération d'une sous-chaîne orale correspondante au p-graphème analysé.
Prenons, par exemple, la graphie ***sous***. On y retrouve les p-graphèmes /**s**/, /**ou**/ et /**s** /, qui ont des modèles phonétiques associés. La transduction phonétique de cette graphie produit deux transcriptions phonétiques : *s.ou* et *s.ou.s*.
La production de deux transcriptions phonétiques différentes est due à l'existence de plusieurs occurrences du p-graphème /**s**/ dans le dictionnaire de p-graphèmes. Pour chaque occurrence, ce p-graphème est associé à un modèle différent : **muet** ou **s**.

Une liste complète des modèles phonétiques utilisés est proposée dans l'annexe 2.

3.1.4 Le code dérivation

Les p-phonèmes associés aux modèles phonétiques sont gardés comme *codes dérivation*, ce qui permet au transducteur d'états finis d'engendrer une chaîne résultante de la transduction Le code dérivation est la sous-chaîne orale associée à un certain modèle phonétique.
Exemples :

Au modèle **k** est associé le code dérivation *k*.
Au modèle **asefin** est associé le code dérivation *a.z*.

A chaque modèle phonétique correspond soit un, soit aucun code dérivation. Les cas d'absence du code dérivation sont dus à l'existence de lettres muettes ou qui peuvent se comporter comme des lettres muettes.
Exemple :

Au modèle **hmil** n'est associé aucun code dérivation (ce modèle est utilisé pour accepter le p-graphème /**h**/ en milieu de mot).

Un même code dérivation peut être utilisé par plusieurs modèles, s'il existe des différences concernant les combinaisons de règles utilisées.
Exemples :

modèle **sz**
règles d'arrivée : sui
code dérivation : *z*
validation : sui
modèle **z**
règles d'arrivée : pib sui
code dérivation : *z*
validation : fin sui

Remarque :
Le modèle **sz** est défini pour accepter le p-graphème /s/ quand il peut avoir le même son que *z* (entre deux voyelles), alors que le modèle **z** est défini pour accepter le p-graphème /z/.

La réussite, après l'analyse d'un modèle, détermine la concaténation du code dérivation du modèle à la chaîne orale engendrée.

3.2 Exemples de génération de chaînes orales à partir d'une chaine écrite

L'analyse phonétique du mot faux peut permettre la génération d'une ou plusieurs formes orales ses différentes possibilités de prononciation, selon les modèles phonétiques du système. Ces formes orales (ou chaines orales) sont les transcriptions phonétiques résultant du processus de transduction
Voici quelques exemples

1) mot faux : **vinku**
chaîne orale : *v.in.k.u.*

2) mot faux : **okurranse**
chaîne orale : *o.k.u.r.an.s.*

3) mot faux : **peti**
chaînes orales : *p.é.t.i.* *p.e.t.i.*

4) mot faux : **echo**
chaînes orales : *e.ch.o.* *e.k.o.*
é.ch.o. *é.k.o.*

4 GENERATION DE CHAINES GRAPHIQUES A PARTIR D'UNE CHAINE ORALE

Nous décrirons ici les deux versions mises en oeuvre pour la reconstitution graphique des transcriptions phonétiques.

4.1 Première version

Cette version résout en trois étapes le problème de la correction de fautes d'orthographe par la phonétique, comme le montre la figure 3.3.

Dans une première étape, le système effectue un découpage phonétique du mot faux, réalisé par un processus de transduction qui produit une ou plusieurs transcriptions phonétiques associées à ce mot. L'objectif ici

est de produire les différentes formes orales possibles correspondant au mot faux.

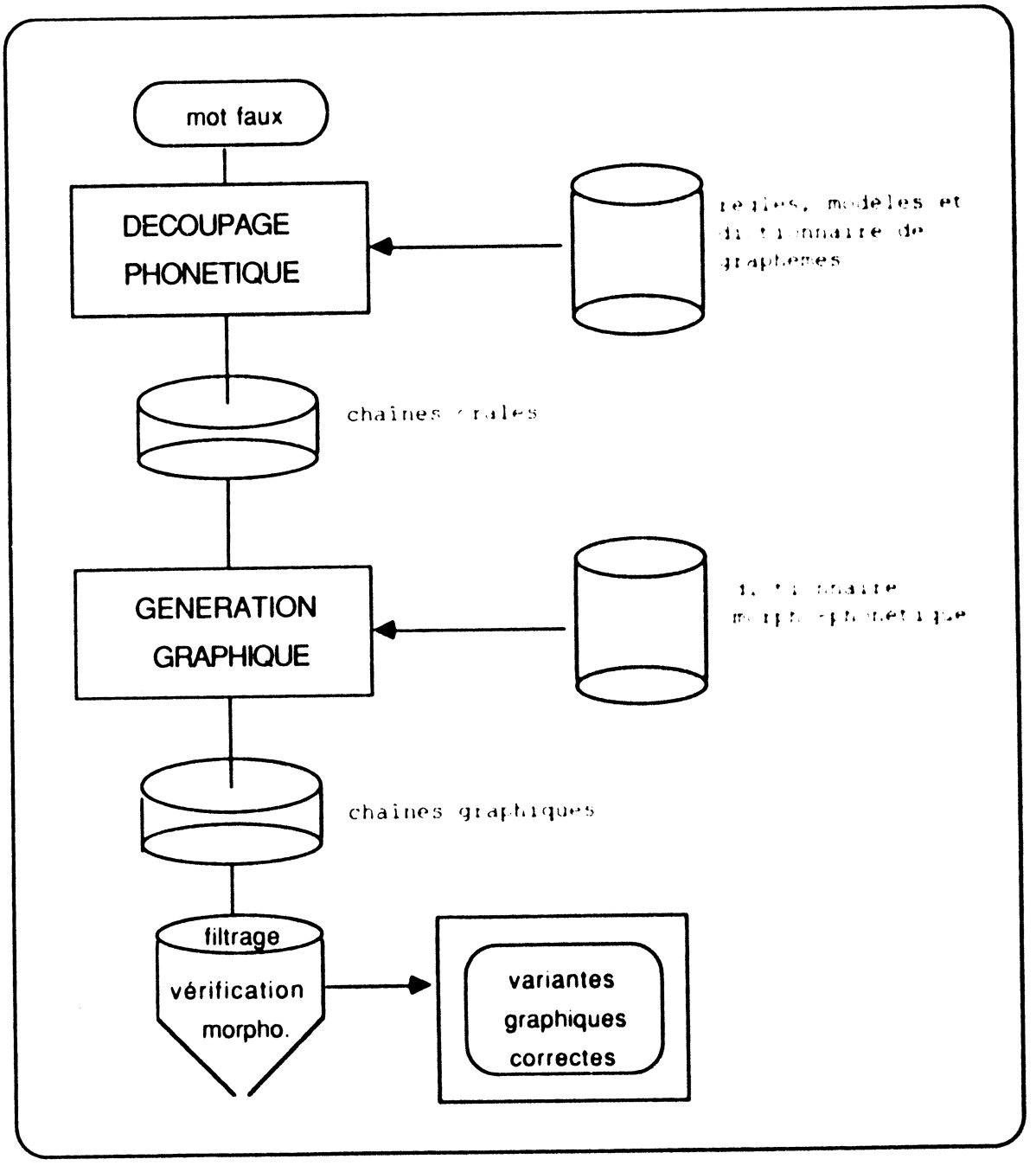


Figure 3.3

Correction de fautes d'orthographe par la phonétique - 1ère version

Dans une deuxième étape, chaque forme orale est à l'origine d'un processus de reconstruction graphique lequel crée des graphies alternatives pour le mot faux.

La reconstitution de graphies à partir d'une transcription phonétique peut produire une ou plusieurs chaînes graphiques, mais toutes les chaînes graphiques produites ne sont pas des mots français. Leur appartenance au lexique doit ensuite être vérifiée.
La troisième étape de l'algorithme de correction réalise un filtrage dont le but est de garder, parmi les chaînes graphiques engendrées, celles qui constituent des mots corrects.

Nous présentons un exemple de correction par cet algorithme (les symboles utilisés dans une transcription phonétique sont décrits dans le paragraphe 3.2) Le mot à corriger est ***plassé***.

transcription phonétique *p.l.a.s.é*

graphies candidates

placé placée placés placées plaçé plaçée plaçés plaçées placer plaçer placais plaçais placait plaçait placaient plaçaient placez plaçez ...

graphies choisies après vérification

placé placée placés placées placer placez plaçais plaçait plaçaient

Les trois étapes qui composent la première version se servent de l'automate conçu pour l'analyse morphologique de PILAF, en préservant le principe de fonctionnement sur des données spécifiques.

Cette réalisation est programmée en PASCAL standard sous UNIX. Elle fonctionne actuellement sur un ordinateur GOULD UTX/32 et se sert d'un dictionnaire d'essai de 688 bases morphologiques.

Le dictionnaire de bases de PILAF est à l'origine d'une structure de données spécifiquement créée pour engendrer des chaînes graphiques à partir des chaînes orales : le dictionnaire morpho-phonétique, ou simplement, dictionnaire M-F. Si l'on dispose d'une forme orale, le dictionnaire M-F permet de réaliser sa reconstitution graphique, à travers un deuxième processus de transduction qui utilise toujours le transducteur de PILAF.
Cette version se sert d'un dictionnaire de 640 bases morpho-phonétiques.

4.1.1 Les bases morpho-phonétiques

Le dictionnaire M-F est constitué par les chaînes orales associées aux bases morphologiques. Il existe une chaîne orale associée à chaque base morphologique.
Si plusieurs bases identiques ont une même chaîne orale associée, le dictionnaire M-F contiendra une seule référence à cette chaîne.
Exemple :

base morphologique /**savant**/ (définie sur un modèle morphologique d'*adjectif*)
chaîne orale associée : *s.a.v.an*
base morphologique /**savant**/ (définie sur un modèle morphologique de *substantif*)
chaîne orale associée : *s.a.v.an*

Dans le dictionnaire M-F sont gardées seulement les informations :

chaîne orale : ***s.a.v.an***
base associée : **/savant/**

Ce fait est dû à la prononciation : les deux bases **/savant/** se prononcent de la même manière.

Analysons maintenant une autre situation. Prenons par exemple la base **/arrêt/**.
Cette base apparaît également deux fois dans le dictionnaire de bases. Elle est définie soit sur un modèle de substantif, soit comme une base verbale qui permet d'arriver à toutes les formes du verbe *arrêter*
Dans ce cas, les deux occurrences de la base **/arrêt/** n'ont pas la même prononciation. Au substantif ***arrêt*** est associée la chaîne orale ***a r é***, alors qu'à la base verbale **/arrêt/** est associée la chaîne orale ***a r é.t*** (le **t** doit être présent parce qu'il est prononcé dans toutes les formes conjuguées du verbe *arrêter*).
Comme conséquence, les deux chaînes orales apparaîtront dans le dictionnaire M-F.

Les enregistrements du dictionnaire M-F ont la forme suivante :

°
°
°

s.a.v.an	**/savant/**

°
°

a.r.é	**/arrêt/**
a.r.é.t	**/arrêt/**

°
°

Chaque enregistrement est constitué de deux champs :

1 une *base morpho-phonétique* (chaîne orale) ;
2 sa *base morphologique* associée.

Les enregistrements sont classés en ordre alphabétique inverse de la base morpho-phonétique.

Comme nous l'avons déjà observé, une chaîne orale peut apparaître plusieurs fois dans le dictionnaire M-F, chaque fois associée à une base différente.
Cette situation rend possible la génération de différentes chaînes graphiques à partir d'une base morpho-phonétique.
Exemple :

chaîne orale : ***s.on***

Cette chaîne apparaît plusieurs fois dans le dictionnaire M-F, associée aux bases /**son** /, /**sont** /, etc.
Une telle situation arrive pour toutes les bases homophones.

4.1.2 La génération de chaînes graphiques

La génération graphique est effectuée par une version simplifiée du transducteur d'états finis
Cette version n'utilise pas de règles ni de modèles, puisqu'elle opère sur un dictionnaire qui contient déjà les chaînes orales et leurs bases associées : le dictionnaire M-F.

Une chaîne orale est parcourue en y recherchant les combinaisons possibles de concaténation de bases morpho-phonétiques. Si une base est trouvée qui correspond à une sous-chaîne de la chaîne orale, cette base morpho-phonétique est remplacée par les bases morphologiques correspondantes (ce qui produit des variantes graphiques pour le mot à corriger), et le processus continue avec la partie restante de la sous-chaîne à analyser.
Pendant ce processus, on observe, pour les graphies engendrées, les mêmes clés phonétiques, ou des clés phonétiques proches du mot à corriger.

Si la combinaison de sous-chaînes orales ne permet pas une transduction, la génération graphique ne produit pas de résultats.

Nous présentons quelques exemples qui décrivent le reconstitution graphique par des bases morpho-phonétiques.

Exemple 1 :

Mot à corriger : ***arête***
Clé phonétique engendrée : *a.r.é.t*

a) En effectuant une recherche sur le dictionnaire M-F, on retrouve la sous-chaîne orale *a.r.é.t* comme la plus longue base dans la clé phonétique engendrée.
La base morphologique associée à la base morpho-phonétique *a.r.é.t* est alors utilisée pour engendrer la première partie de la graphie résultante : ***arrêt.***
b) La suite de la graphie résultante est obtenue en analysant la suite de la clé phonétique (ici, la clé phonétique a été entièrement prise en compte dans la sous-chaîne initiale).
c) La génération graphique finit par ajouter quelques p-graphèmes muets à la graphie résultante : **s**, **ent**, **e**, **es**, etc, selon des fondements linguistiques.

Comme résultats, avec le dictionnaire de test de PILAF, nous avons les équivalents ***arrête***, ***arrêtent*** et ***arrêtes***. Nous avons aussi ***arrêt*** et ***arrêts***.
En utilisant un dictionnaire plus complet, les mots ***arète*** et ***arètes*** seraient aussi retrouvés.

Exemple 2 :

Mot à corriger : ***émé***.
Clé phonétique engendrée : *é.m.é*

a) La clé phonétique en question n'existe pas dans le dictionnaire M-F. La reconstitution graphique prend donc en compte les plus longues sous-chaînes composantes de la clé phonétique.
b) Dans l'exemple, les bases morpho-phonétiques trouvées sont ***é.m*** (associée à la racine verbale /aim/ du verbe *aimer*) et ***é*** (associée aux désinences /é/, /er /, etc).

Nous obtenons alors les graphies ***aimé***, ***aimée***, ***aimés***, ***aimées***, ***aimer***, ***aimait***, ***aimais***, ...

4.1.3 Contrôle de la génération

Nous avons observé que le processus de reconstitution graphique utilisé produit, parfois, un nombre considérable de graphies, ce qui rend plus longue l'étape suivante de vérification morphologique.

Le principe de transduction utilisé prend en compte toutes les alternatives de découpage de la clé phonétique, en produisant, parfois, des graphies moins probables en tant que corrections du mot faux.

En analysant les résultats obtenus, nous avons constaté que les meilleures alternatives de correction par la phonétique sont souvent les premières retrouvées dans le dictionnaire M-F.
Ce fait est dû au classement des entrées du dictionnaire (par ordre alphabétique inverse de la clé phonétique). Ce classement permet de trouver en premier les chaînes orales les plus longues et, par conséquence, les bases morphologiques les plus longues.
Dans le cas de correction d'un substantif, par exemple, la graphie la plus adéquate est souvent la première à être retrouvée.
Ainsi, à l'occasion de la correction de ***jirraphe*** par la méthode proposée, la première alternative phonétique retrouvée est *girafe* (associée à la clé phonétique *j.i.r.a.f*). Nous ajoutons ensuite quelques graphèmes muets, vérifions si les graphies engendrées appartiennent aux dictionnaires du système et finissons par proposer, comme graphies correctes, ***girafe*** et ***girafes***.

Mais cette propriété n'est pas valable pour toutes les corrections réalisées. Elle ne se vérifie pas, par exemple, pour les formes verbales.
Considérons la correction de ***alez***.
La clé phonétique engendrée, dans ce cas, est *a.l.é.*. La recherche dans le dictionnaire M-F permet de trouver directement cette clé phonétique (*a l é*), mais elle est associée à la base */allée/*, qui n'est pas suffisante pour la correction.

Nous devons continuer à produire les graphies possibles à partir des sous-chaînes phonétiques trouvées dans *a.l.é* et nous arrivons de cette manière à la base verbale associée au verbe *aller*, dont la clé phonétique est *a.l* (base /all/).
Avec la sous-chaîne phonétique *é* nous obtenons les formes du verbe *aller* dont la désinence se prononce en *é* (*aller*, *allez*, *allait*, etc).
Comme résultat, nous avons un ensemble de graphies contenant ***allée***, ***allées***, ***allé***, ***allés***, ***aller***, ***allez***, ***allait***, etc.

Pour faire face à cette situation particulière, nous proposons comme option deux possibilités de génération graphique :

a) la *génération de la première graphie* (l'algorithme s'arrête à la première base morpho-phonétique retrouvée);
b) la *génération de toutes les graphies possibles* pour le mot erroné (l'algorithme réalise une transduction complète et retrouve toutes les

combinaisons possibles de bases morpho-phonétiques dans la transcription phonétique analysée).

C'est à l'utilisateur de choisir la modalité de travail la mieux adaptée.

4.1.4 Vérification morphologique des chaînes graphiques engendrées

La génération graphique, même si elle est réalisée à partir des bases morpho-phonétiques, peut produire un mauvais enchaînement des bases, introduisant, parmi les résultats corrects, des graphies qui ne constituent pas des mots. Par exemple, on peut citer la graphie ***vainquu*** apparue pendant la correction de **vinku** (le verbe *vaincre* possède deux bases différentes : */vainc/* et */vainqu/*).
Il est donc nécessaire d'opérer une vérification et de déterminer si ces chaînes constituent oui ou non des mots.

Nous pouvons vérifier ces résultats, soit en utilisant l'analyse morphologique elle-même, soit en opérant une recherche sur le dictionnaire de formes du système. La première méthode a été choisie pour le développement de cette expérimentation, alors que la deuxième a été exploitée lors de la construction du système DECOR [COHARD 88].

Par le biais d'une telle vérification, nous proposons comme alternatives de correction seulement des graphies qui constituent des mots corrects, en éliminant du résultat final celles considérées comme morphologiquement erronées ou introuvables dans le dictionnaire de formes.

4.2 Deuxième version

La deuxième version de notre méthode de correction a pour but de produire un module (PHONE) susceptible d'être inséré à l'intérieur d'un système de traitement d'erreurs dans un texte écrit en français. Elle est programmée en TURBOPASCAL version 5.0.

Elle diffère essentiellement de la première par l'élimination du dictionnaire morpho-phonétique.
Dans PHONE, l'utilisation du transducteur de PILAF est restreinte à l'étape de transduction phonétique, et la vérification des graphies est opérée directement sur le dictionnaire de formes du système.

4.2.1 Les règles de composition graphique

Nous employons, dans cette réalisation, des règles de composition graphique pour calculer les graphies alternatives du mot erroné.
A chaque p-phonème élémentaire, nous associons une liste de règles de composition graphique (lrcg) :

<lrcg> ::= <règle de composition> <règle terminale> |

<règle de composition> <lrcg>

<règle de composition> ::= <p-graphème> <liste d'indicateurs>

<règle terminale> ::= **3*****

<p-graphème> ::= *tout p-graphème du dictionnaire de p-graphèmes*

<liste d'indicateurs> = <indicateur 1> <indicateur 2> <indicateur 3>

Pour chacun des indicateurs 1, 2 et 3 nous avons, respectivement, une valeur logique, correspondant à la possibilité de concaténer ce p-graphème au début d'une graphie, au milieu d'une graphie ou comme graphème final d'une graphie.

Nous présentons comme exemple la liste des règles de composition graphique associée au p-phonème **y** :

p-graphème	début de mot	milieu de mot	fin de mot
i	non	oui	non
ll	non	non	oui
i	non	oui	non
ll	non	oui	non
lle	non	non	oui
y	oui	oui	oui

Les listes de règles de composition graphique sont réunies dans une structure de fichier texte (le fichier PHCHAIN). Chaque règle de composition correspond à une ligne de ce fichier, et chaque liste de règles est précédée d'une ligne contenant le p-phonème concerné.
Cette structure est chargée dynamiquement en mémoire pour l'exécution de l'algorithme.

Les règles de composition graphique, à l'intérieur d'une liste, sont classées par ordre alphabétique du p-graphème, pour permettre d'engendrer des graphies en ordre alphabétique.
Les listes de règles de composition sont classées en ordre alphabétique inverse du p-phonème concerné.

Nous avons cherché à réduire le temps de réponse de notre algorithme de correction. Pour cela, nous avons enrichi la structure PHCHAIN avec quelques listes de règles supplémentaires, associées à des p-phonèmes non-élémentaires plus ou moins fréquents (exemple : ***l.t.é***).

En vue de la reconstitution graphique, nous commençons par effectuer une recherche dans PHCHAIN. Cette recherche amène à toutes possibilités d'écriture de chaque composant phonétique d'une transcription phonétique en analyse, par rapport aux dictionnaires de PILAF (nous rappelons que les situations particulières sont à l'origine d'entrées dans la structure de cas particuliers).

Nous présentons comme exemple les étapes de la recherche des composants graphiques possibles pour la transcription phonétique ***k.a.r.o.***

a) règles de composition graphique associées au p-phonème ***k*** :

c	oui	oui	oui
cc	non	oui	non
ch	oui	oui	oui
cqu	non	oui	non
k	oui	oui	oui
kh	oui	oui	non
qu	oui	oui	non
que	non	non	oui

p-graphèmes sélectionnés : *c, ch, k, kh, qu*

b) règles de composition graphique associées au p-phonème ***a*** :

a	oui	oui	oui
ha	oui	oui	oui
hâ	oui	non	non
â	oui	oui	non

p-graphèmes sélectionnés : *a, ha, â*

c) règles de composition graphique associées au p-phonème ***r*** :

r	oui	oui	oui
rd	non	non	oui
re	non	non	oui
rh	oui	oui	non
rr	non	oui	non
rre	non	non	oui
rt	non	non	oui

p-graphèmes sélectionnés : *r, rh, rr*

d) règles de composition graphique associées au p-phonème ***o*** :

au	oui	oui	oui
aud	non	non	oui
aux	non	non	oui
eau	non	oui	oui
eaux	non	non	oui
hau	oui	non	non
o	oui	oui	oui
ot	non	non	oui
ô	oui	oui	non

p-graphèmes sélectionnés : *au, aud, aux, eau, eaux, o, ot*

4.2.2 Les possibilités de concaténation (STATBOOL)

Une analyse des concaténations possibles entre tous les couples de p-graphèmes utilisés dans la génération graphique nous a permis de réduire le nombre de graphies engendrées par la méthode.

Nous avons réuni plus de 200 composants graphiques appartenant aux listes de règles de composition graphique, et nous avons analysé les mots du dictionnaire (environ 220000 formes), par rapport à leur composition. Dans chaque mot nous avons cherché toutes les chaînes constituant des combinaisons possibles de deux composants graphiques concaténés
Une table, construite à partir de cette analyse, signale l'existence ou non des n grammes constitués par la concaténation de deux composants graphiques Par exemple, à l'aide de cette table, nous savons que la concaténation des composants *mm + amp* n'amènera pas à un mot valable, alors que la combinaison *mm + aient* existe dans au moins un mot du dictionnaire

La structure de données résultante est aussi chargée en mémoire, sous forme dynamique Chaque information occupe l'espace d'un bit.

4.2.3 L'analyse des cas particuliers de composition graphique

Afin de réduire encore plus le nombre de graphies engendrées par cette méthode de correction, nous éliminons quelques règles particulières des listes de règles de composition, tout en ajoutant à une structure de données les mots concernés

La structure de données contenant des cas particuliers, nommée CP, est gardée sous la forme d'un fichier texte qui contient des enregistrements constitués par :

1 la chaine orale ,
2 la forme associée

Par exemple, nous observons que les seuls cas d'occurrence de la graphie î en début de mot, dans le dictionnaire de formes de PILAF, sont : île, îles, îlot, îlots.
Ceci nous permet d'insérer les quatre mots concernés dans la structure CP et d'avoir, parmi les règles associées au phonème *i* de PILAF, la simplification :

1 non oui non

4.2.4 L'algorithme de reconstitution graphique

Nous décrivons ici l'algorithme employé dans l'étape de génération graphique correspondant à la deuxième version mise en oeuvre.

Une chaîne orale produite par l'étape de transduction phonétique est à l'origine d'un ensemble de graphies. Cet ensemble peut contenir aucune, une ou plusieurs graphies.
Les graphies sont composées conformément à l'algorithme suivant :

I *La chaîne orale est parcourue en vue de la détermination de ses p-phonèmes composants et des p-graphèmes concernés.*

A chaque p-phonème détecté, nous associons une liste de p-graphèmes alternatifs (unité COMPOSE). C'est-à-dire, nous effectuons une recherche dans la structure des listes de règles de composition (provenant de PHCHAIN, chargée en mémoire), et sélectionnons les p-graphèmes applicables à chaque p-phonème.

La recherche des p-graphèmes applicables à chaque p-phonème tient compte de la position du p-phonème dans la chaîne orale.
Par exemple, si nous analysons le p-phonème */* dans la chaîne orale ***i.b.ou***, alors nous aurons, parmi les p-graphèmes sélectionnés, *hi*, *i*, etc. Mais nous n'aurons pas le p-graphème *i*, puisque ce p-graphème n'apparaît pas en début de mot (sauf dans quatre cas particuliers, voir ce chapitre, paragraphe 5.2.3).

Comme résultat, nous obtenons une liste de groupes de p-graphèmes dont chaque composant contient les p-graphèmes à utiliser à la place du p-phonème de la chaîne orale analysée.

Ayant comme objectif de rendre plus efficace la composition graphique, quelques préfixes phonétiques ont été relevés (exemple : *f.o.t.o, m.i.n.i, t.r.an.s,...*). L'accès à ces préfixes permet de déterminer directement leurs chaînes graphiques associées (exemple : *f.o.t.o* → *photo*).

II *Les p-graphèmes sont assemblés.*

Nous réalisons l'assemblage des p-graphèmes (unité ASSEMBL) selon les possibilités de concaténation prévues dans la table STATBOOL (paragraphe 5.2.2 de ce chapitre).

Nous prenons aussi certaines précautions pendant la concaténation de p-graphèmes qui peuvent changer excessivement la chaîne orale.

Par exemple, considérons la chaîne orale ***s.o***. Parmi les alternatives d'écriture du p-phonème ***s*** en début de mot, nous avons *c*, *s*, etc. Toutefois, le p-graphème *c* suivi de certaines voyelles ou des consonnes n'aura plus la même prononciation : il sera prononcé comme ***k***. Nous évitons, alors, ce genre de concaténation.
Ces situations sont traitées actuellement dans une procédure spécifique du programme d'assemblage des p-graphèmes. Le nombre de cas concernés étant considérable, il est même envisageable d'utiliser une méthode plus organisée de traitement, soit sous la forme d'une structure de données soit sous la forme d'un automate.

Nous finissons par ajouter des p-graphèmes muets aux graphies assemblées. Les graphies complètes sont gardées dans une liste, classées par ordre alphabétique.

III *Les graphies résultantes sont vérifiées.*

Parmi les graphies engendrées, certaines ne sont pas des formes valables. Il est donc nécessaire de réaliser un filtrage et d'éliminer de cet ensemble les graphies qui ne constituent pas des mots français.

Cette vérification comporte une recherche dans le dictionnaire de formes de PILAF (unité RECFORMES). Ce dictionnaire est organisé actuellement en 26 fichiers correspondant aux 26 lettres de l'alphabet. Chacun d'eux contient les formes placées par ordre alphabétique.
Nous vérifions l'existence d'une graphie dans les fichiers du dictionnaire de formes à l'aide d'une recherche dichotomique, suivie d'une recherche séquentielle.

5 RESULTATS OBTENUS

5.1 Potentialités d'une méthode de correction par phonétique

La méthode de correction d'erreurs par la phonétique permet la correction de mots simples erronés dès qu'ils gardent leur forme orale ou une forme orale proche de la forme orale originale.
Elle permet la correction d'erreurs souvent impossibles à corriger avec les algorithmes de correction d'erreurs d'édition :

1) Cette méthode permet la correction de mots contenant plusieurs erreurs. Exemples :
Inbessillité est corrigé en *imbécillité, imbécillités.*
karo est corrigé en *carreau, carreaux.*

2) Elle permet aussi la correction des mots dont la première lettre est erronée (impossible avec la clé squelette). Exemples :
ibou est corrigé en *hibou, hiboux.*
klore est corrigé en *chlore, chlores.*

On peut également traiter quelques accents. Exemples :
paraître aura comme correction le mot *paraître.*
chevre peut être corrigé en *chèvre, chèvres.*

Destinée aux mots simples, comme nous l'avons déjà fait remarquer, la méthode proposée n'envisage pas le traitement des erreurs dans des mots composés. Nous pouvons toutefois obtenir quelques résultats avec des mots composés en employant la première version développée pour la correction par phonétique (la transcription phonétique associée au mot composé étant celle associée à la base correspondante dans le dictionnaire de bases phonétisées). Exemple : ***porte-chapo*** peut être corrigé en *porte-chapeaux.*

Les erreurs dits de liaison et causées par la mauvaise transcription graphique d'une chaîne orale comme, par exemple, ***les zabeilles*** ou ***un névier***, ne sont pas traitées par la méthode proposée. Ce traitement présupposerait une étude des cas d'insertion de consonnes au début de mot suite à une mauvaise transcription écrite d'une chaîne orale.

En ce qui concerne les homophones employés erronément, comme *l'arme* pour *larme* ou *plus tôt* pour *plutôt*, nous n'effectuons aucun traitement particulier puisque, par principe, la correction par phonétique ne manipule que des mots morphologiquement faux.
Cependant, il s'avère possible d'employer, lors d'un traitement des erreurs au niveau syntaxique voire sémantique, les outils développées (le transducteur phonétique peut produire des transcriptions phonétiques à reconstituer selon des règles de reconstitution spécifiques).

Au niveau lexical, nous pouvons utiliser la méthode de correction par phonétique en association avec d'autres méthodes de correction, comme celles applicables aux erreurs d'édition et aux erreurs de génération.
Nous avons adopté cette structure pour le système DECOR, y exploitant trois algorithmes différents de correction d'erreurs lexicales (phonétique, clé squelette et génération). Des priorités peuvent être attribuées aux algorithmes de correction, en fonction des compétences et des performances de l'utilisateur (le paragraphe 1.2 du chapitre 6 aborde l'ordre d'application des méthodes de correction).

Finalement, nous remarquons l'applicabilité d'une telle méthode de correction à d'autres idiomes, notamment ceux qui possèdent des

combinaisons orales dont la représentation écrite peut être obtenue par des différentes combinaisons écrites, comme c'est le cas, par exemple, du portugais.
L'utilisation pour le portugais de la méthode développée prévoit une modélisation phonétique de cette langue (étape de transcription phonétique) et, dans le cas d'utilisation de la deuxième version de l'algorithme de correction d'erreurs par phonétique, une modélisation de l'ensemble de règles de réécriture à adopter.

5.2 Comparaison entre la première et la deuxième version

Les deux versions développées pour la correction d'erreurs par la phonétique diffèrent en deux points : les structures de données utilisées et, par conséquent, l'algorithme suivi lors d'une correction.
La première version mise en oeuvre exige la disponibilité d'un dictionnaire morpho-phonétique concomitant avec un dictionnaire morphologique. Si nous considérons que cet algorithme sera utilisé en combinaison avec l'algorithme de correction par clé squelette (comme c'est le cas dans le système DECOR), nous devons additionner à ces deux dictionnaires le dictionnaire squelette.
Avec la deuxième version mise en oeuvre, on n'a plus besoin d'un dictionnaire morpho-phonétique : la composition graphique est réalisée par moyen d'un ensemble de règles de composition lequel est chargé en mémoire. C'est-à-dire qu'on diminue l'espace mémoire exigé pour cette technique de correction.

La première version de cette méthode (en utilisant un dictionnaire morpho-phonétique) a permis de considérer le mot faux comme la concaténation de plusieurs mots, sans séparateur.
Exemple : ***equivalait*** peut être considéré comme une séquence de mots (*et qui valait*) par omission des séparateurs.
Cette possibilité n'a pas été exploitée dans notre système à cause du nombre excessif de graphies engendrées et du temps de réponse observé.

La figure 3.4 présente les temps observés (en secondes) pendant la correction de quelques mots, par le module PHONE. Cette expérimentation a été réalisée sur un ordinateur TANDON 386 avec un dictionnaire d'environ 220000 formes.

L'omission de données comparatives entre les deux versions de correction par la phonétique est due aux différents environnements de test disponibles, soit un dictionnaire de test par rapport à un dictionnaire de taille réelle.

mot	transduction phonétique	reconstitution graphique	vérification	corrections proposées
aute	0,54	0,06	2,25	haut haute hautes hauts ote otes ote ôtee ôtees ôtes
inbecilité	1,54	1,91	0,61	inbécillite inbecillites
bou	0,28	0,05	2,20	boue boues bous bout bouts
aurtografique	1,87	0,22	1,04	orthographique orthographiques
den	0,43	0,01	2,09	dan dans dans dent dents
alez	0,60	0,27	5,95	allai allaient allais allait aller allez alle allee allees alles balai balaier balais balait baler balez bale balee balees bales balai balaier balais balait baler balez bale balee balees bales

Figure 3.4

Quelques mesures de temps de correction

(deuxième version)

CHAPITRE 4

VERIFICATION SYNTAXIQUE

Une fois que l'on dispose des structures de dépendances associées à une phrase (calculées par l'analyseur syntaxique de PILAF), le vérificateur syntaxique détermine si elles sont valables en français, c'est-à-dire, si la phrase est ou non correcte.
Exemples :
- ***le soupe** est considéré incorrect (erreur d'accord en genre entre le déterminant et le nom) ;*
- ***je le vous donne** est considéré incorrect (erreur de "style" : mauvais placement des pronoms **vous** et **le**) ;*
- ***Il mangeras** est considéré incorrect (erreur d'accord sur personne entre le sujet et le verbe) ;*
- ***la souris mange le chat** est considéré syntaxiquement correct.*

*La vérification syntaxique effectuée concerne l'applicabilité d'un ensemble de règles qui modelisent les rapports les plus fondamentaux entre les composants d'une proposition écrite en français ayant pour but la détection d'erreurs de syntaxe. Parmi les rapports modélisés, nous considérons la liaison entre le nom et son déterminant, le nom et son adjectif, le sujet et le verbe, le verbe et son attribut, aussi bien que la conjonction de coordination **et**, les subordinations à partir d'un pronom relatif et à partir de conjonctions de subordination.*
*Cette vérification est réalisée par unification de traits. Nous parcourons un arbre de dépendances en vue de l'analyse des rapports entre gouverneur et dépendant. L'analyse d'une paire de noeuds dont les rapports syntaxiques sont corrects est à l'origine d'une unification de traits syntaxiques, produisant comme résultat un noeud unique auquel sont attachés les renseignements lexico-syntaxiques de la liaison analysée. Par exemple, l'analyse du déterminant **le** (article défini, masculin singulier) avec son gouverneur **belge** (substantif commun, masculin féminin singulier) dans **le belge** unifie les caractéristiques de ces deux noeuds produisant un seul noeud auquel sont associés le substantif **belge** et les valeurs masculin singulier.*
A l'occasion d'une faute d'accord, le vérificateur syntaxique procède au stockage de paramètre permettant à un module de correction d'engendrer un ensemble de possibilités pour le mot erroné (ceci sera l'objet du chapitre suivant de ce mémoire).

Même si nous envisageons d'obtenir un système de traitement d'erreurs au niveau lexico-syntaxique, la vérification syntaxique proposée est encore loin d'être un outil pratique, applicable à la vérification de textes en général.
Après la conception et la mise en oeuvre de cette maquette, plusieurs questions peuvent se poser par rapport à son utilisation comme partie d'un système de détection/correction. Parmi celles-ci, l'adéquation de

cette méthode au traitement d'un vocabulaire significatif de la globalité de la langue et, peut-être, l'adéquation du mécanisme propre d'analyse syntaxique .

Dans le présent chapitre, nous décrivons en détail la vérification syntaxique réalisée.

1 LA VERIFICATION SYNTAXIQUE D'UNE PROPOSITION

1.1 Les langages et les règles

Tous les langages sont gérés par un ensemble de règles de formulation de propositions.

Néanmoins, dans l'univers des langues naturelles, la définition de grammaires sensibles aux phénomènes observés n'est pas totalement mise au point. Les études existantes se limitent aux phénomènes considérés comme les plus importants, soit la partie considérée comme "les fondements de la langue", soit des sous-ensembles bien limités de la langue, utilisés dans un domaine spécifique d'application
En outre, se pose la question de la faisabilité de la définition d'une grammaire exhaustive, pour les langues naturelles.

Dans le système développé, nous essayons de modéliser un sous-ensemble limité des relations possibles entre les composants lexico-syntaxiques d'une proposition donnée, qui prend en compte des fondements de la langue française.
La vérification syntaxique se sert des informations morpho-syntaxiques provenant de l'analyse syntaxique d'une phrase, sans faire référence à des variables sémantiques.

Cette mesure présente des aspects de simplification. On n'exclut pas des analyses pour une phrase prise hors de son contexte, car on ne considère que la structure syntaxique de la phrase, sans se préoccuper du sens.
Exemples :

Le chêne ***pleurait*** *à regarder le roseau* est une phrase qui sera considérée correcte par le vérificateur syntaxique alors que sémantiquement, elle devrait normalement être refusée.
Elle mange lire est considérée également correcte, car analogue, du point de vue syntaxique, à la phrase *Elle aime lire*.

Toutefois, l'inclusion de traits sémantiques est prévue et actuellement étudiée dans le projet PILAF, en envisageant une application bien

déterminée. Elle permettra de modéliser plus précisément les rapports entre les catégories lexico-syntaxiques.

1.2 Les mécanismes de vérification syntaxique

Le système de vérification syntaxique se sert de deux structures importantes :
- les *arbres de dépendances* associés à une phrase ;
- les *règles de vérification* qui modélisent un sous-ensemble de la langue.

1.2.1 Arbres de dépendances

Les arbres de dépendances associés à la phrase sont calculés par l'analyseur syntaxique de PILAF, en tenant compte d'un ensemble de règles de dépendance.
Ils contiennent pour chaque noeud une décoration, c'est-à-dire un ensemble d'informations lexico-syntaxiques :
- la catégorie lexico-syntaxique du noeud ;
- les listes des valeurs associées aux groupes de variables morphologiques ;
- le mot correspondant à ce noeud dans la phrase.

La figure 4.1 présente comme exemple l'arbre de dépendances engendré pour la phrase *La souris aime le chat noir* (les codes utilisés dans la décoration sont décrits dans les paragraphes 1.5 et 1.6 de ce chapitre).

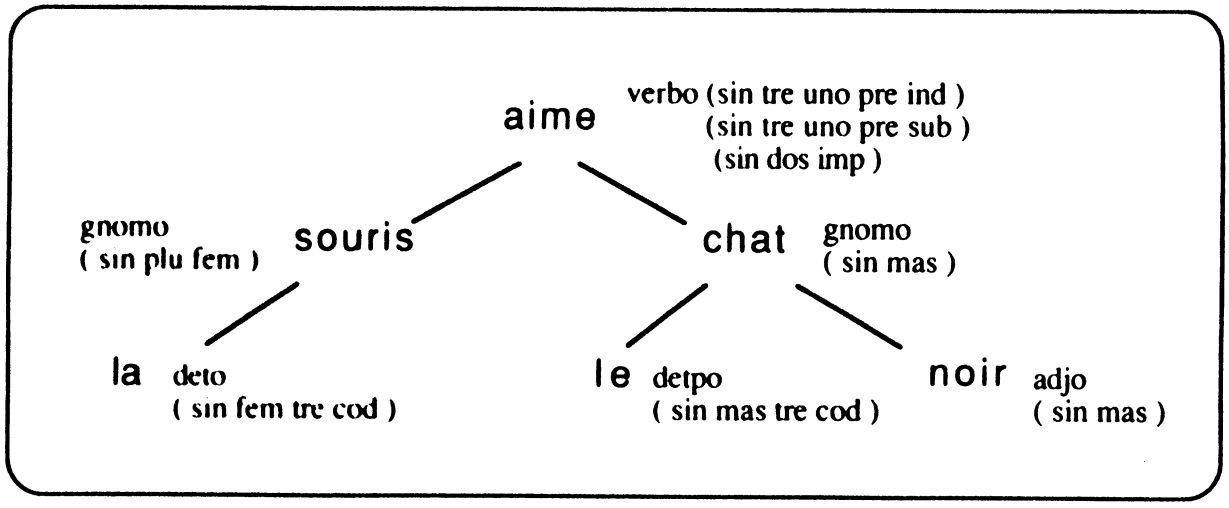


Figure 4.1 - Un arbre décoré

Remarques :

1) Nous utilisons ici une représentation graphique d'un arbre dont l'objectif est d'illustrer cette structure de données. Une *grammaire d'édition* concernant un arbre a été conçue pour la vérification syntaxique et elle sera détaillée dans la suite de ce chapitre. L'arbre passé au vérificateur syntaxique est réédité conformément à cette grammaire.
2) Nous pouvons observer dans l'exemple l'existence de trois listes de valeurs associées au mot *aime*, se référant aux trois cas possibles : première ou troisième personne du singulier du présent de l'indicatif, première ou troisième personne du singulier du présent du subjonctif, deuxième personne du singulier de l'impératif. La vérification syntaxique est faite, dans ce cas, en analysant chaque liste séparément.
3) A mesure que la vérification syntaxique opère sur un arbre, nous ajoutons aux listes de valeurs morphologiques des informations lexico-syntaxiques associées aux noeuds.
Exemple - pendant la vérification syntaxique de l'arbre associé à la phrase *La souris aime le chat noir* :
- l'analyse des noeuds *la* et *souris* produit comme résultat un noeud unique, dont le mot attaché est *souris* et auquel sont associées des informations parmi lesquelles une valeur qui spécifie que *souris* a déjà eu son article vérifié ;
- l'analyse des noeuds *souris* et *aime* produit comme résultat un noeud dont le mot attaché est *aime* et auquel sont associées des informations parmi lesquelles une valeur qui spécifie que le verbe *aime* a déjà eu son sujet vérifié.

Le vérificateur syntaxique ne manipule qu'un arbre à la fois. Si plusieurs arbres sont associés à une phrase, chacun est analysé individuellement. Une correction syntaxique opérée par le système amène à un changement de la phrase et de la décoration de l'arbre syntaxique. Cette correction est suivie d'une nouvelle vérification.

1.2.2 Règles de vérification syntaxique

La vérification syntaxique est réalisée par un ensemble de clauses associées à un prédicat PROLOG, qui sont analysées pour déterminer si une construction est syntaxiquement correcte ou non en français.

Cette vérification est organisée en deux niveaux :
- un niveau plus général (les *méta-règles*) qui sert à reconnaître la catégorie lexico-syntaxique des deux noeuds dont la liaison doit être analysée et qui sert à sélectionner la règle à appliquer ;

- un niveau plus fonctionnel (les *règles*) qui réalise la vérification syntaxique proprement dite et unifie les traits lexico-syntaxiques des noeuds en analyse.

Les *méta-règles*, en plus grand nombre que les règles, permettent de grouper les situations traitées identiquement au point de vue linguistique par le système.
Le corps d'une méta-règle est assez simplifié, puisque son objectif est uniquement de diriger l'analyse vers l'application d'une règle spécifique.
Si deux paires différentes (k_1, k_2), (k_3, k_4) de catégories lexico-syntaxiques suivent une même règle de vérification syntaxique, cette règle sera définie une seule fois dans le système, et deux méta-règles seront définies (respectivement, pour chacune des paires (k_1, k_2), (k_3, k_4)) canalisant l'analyse de ces deux situations vers une même règle.
Exemple : le traitement des liaisons
adjectif qualificatif → *substantif commun* (exemple : *petit* → *chien*)
et *substantif commun* → *adjectif qualificatif* (exemple : *soupe* → *chaude*)
est canalisé vers une même règle.
Effectivement, ces deux situations seront traitées de manière identique, par rapport à la vérification d'accord (voir chapitre 5 paragraphe 2 1 pour les règles d'accord) et par rapport à l'unification de traits.
Ceci peut rendre moins fastidieuse l'étape d'écriture de règles : les linguistes et les professionnels concernés n'auront pas besoin de réécrire des règles de vérification syntaxique identiques.

Les *règles* de vérification syntaxique ont pour objectif d'accepter une liaison déterminée entre deux noeuds de l'arbre syntaxique, soit une liaison *fils gauche* → *racine*, soit une liaison *racine* → *fils droit*, en produisant deux résultats : une *liste d'informations lexico-syntaxiques* et un *indicateur de détection de faute d'accord*.
Nous avons développé un *ensemble élémentaire de règles de vérification syntaxique*, qui constitue le noyau de la vérification syntaxique.
L'accès à cet ensemble de règles est coordonné par un *ensemble de méta-règles de vérification syntaxique*.

Nous présentons dans les paragraphes 2.1 et 2.2 les règles et les méta-règles de vérification syntaxique.

1.3 Le principe de la vérification

La vérification syntaxique est faite en parcourant les arbres de dépendances en ordre *postfixé*, comme le montre la figure 4.2.

Nous désignons par **(gauche * droit)** une paire de noeuds en analyse : un fils gauche avec une racine ou une racine avec un fils droit

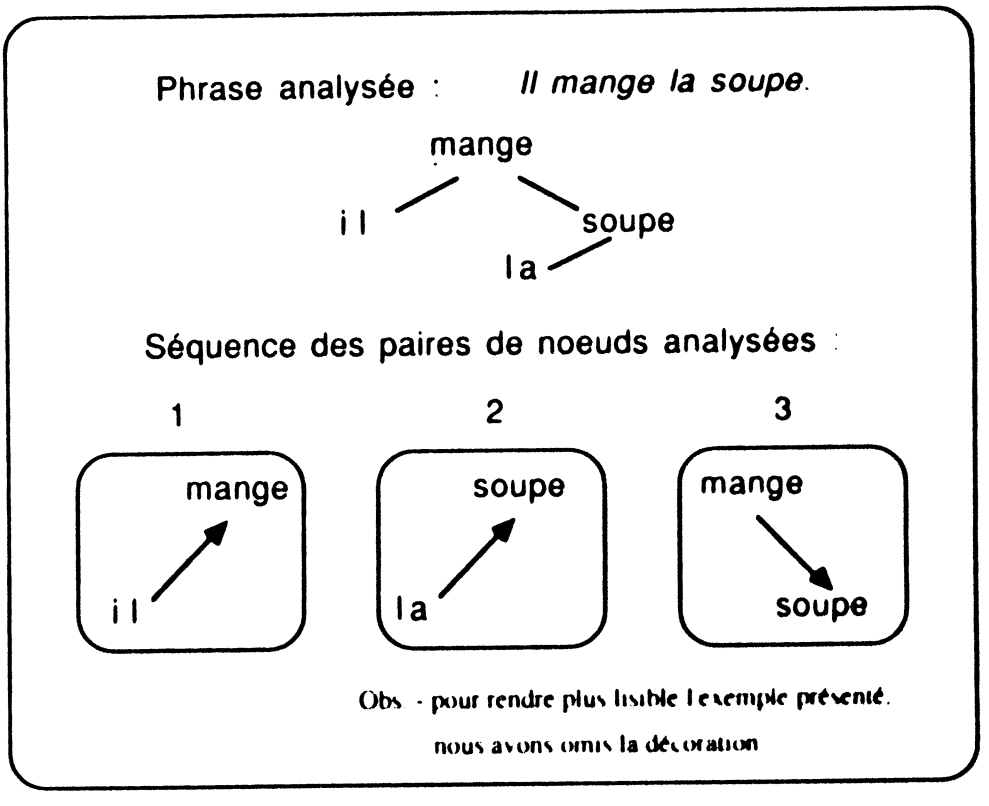


Figure 4.2 - Analyse d'une phrase

La vérification syntaxique est faite en essayant de trouver, d'abord, une méta-règle de vérification syntaxique qui puisse s'appliquer au couple **(gauche * droit)**.
Trouvée cette méta-règle, nous passons directement à l'application d'une règle de vérification syntaxique.

La recherche d'une règle à appliquer s'arrête à la première règle *candidate* trouvée.
Une règle est considérée candidate à l'application quand elle concerne la paire **(gauche * droit)** en analyse (conformément à l'en-tête de la règle).
S'il y a plusieurs règles définies pour un couple **(gauche * droit)**, c'est la première de ces règles qui sera utilisée. On ne passera à la prochaine candidate que dans le cas d'impossibilité d'application de la candidate en analyse.

On considère qu'une règle est *applicable* si :
1 elle concerne les catégories lexico-syntaxiques associées respectivement aux noeuds **gauche** et **droit** du couple analysé (c'est une règle candidate) et
2 toutes les conditions qu'elle contient sont remplies.

Le résultat de l'analyse d'un couple **(gauche * droit)** est un noeud unique, qui garde les informations lexico-syntaxiques provenant de la vérification effectuée.

A l'application d'une règle à un couple **(gauche * droit)**, si toutes les conditions contenues dans cette règle sont remplies, des informations lexico-syntaxiques résultant de l'application sont calculées par la règle.
Cette opération produit une liste qui contient les informations lexico-syntaxiques actualisées se référant à une branche analysée.

Si une règle candidate est trouvée mais qu'une condition dans cette règle n'est pas remplie, on tombe dans un cas d'*échec*.
Dans le système de vérification mis en oeuvre, les cas d'échec provoqués par l'absence d'accord entre les deux composants d'une paire de noeuds analysée font l'objet d'un traitement spécifique : parmi les valeurs morphologiques concernant les mots traités, celles considérées d'intérêt pour une correction sont gardées.
Quels que soient les cas d'échec, le système se poursuit, à la recherche d'une règle applicable.

Si aucune règle applicable n'est trouvée pendant la vérification d'une paire de noeuds, et qu'aucune faute d'accord n'a été trouvée, l'utilisateur est averti de l'inexistence de règles pour traiter la phrase proposée.
Une telle situation peut être causée par un des deux motifs :

A - soit les règles définies sont insuffisantes (dans ce cas le système présenterait une défaillance fonctionnelle par rapport aux phrases qu'il se dispose à traiter) ;
B - soit la construction proposée n'est pas acceptée en français.

La situation A ne saurait être que transitoire (récemment nous avons réalisé l'acquisition d'un grand dictionnaire, et nous effectuons une analyse des catégories lexico-syntaxiques et règles de dépendances), la situation B devra être la seule à se présenter.

Un échec causé par une faute d'accord est à l'origine d'un processus de correction. Dans ce cas, un module de correction de fautes d'accord engendre des alternatifs syntaxiquement corrects pour les mots faux. Par exemple, *Il partiront* est corrigée en *Il partira*.

La correction est réalisée en prenant les mots syntaxiquement faux et les valeurs voulues pour la correction, et en utilisant le module de génération morphologique de PILAF.
Le chapitre suivant de ce travail présente une description détaillée du module de correction de fautes d'accord.

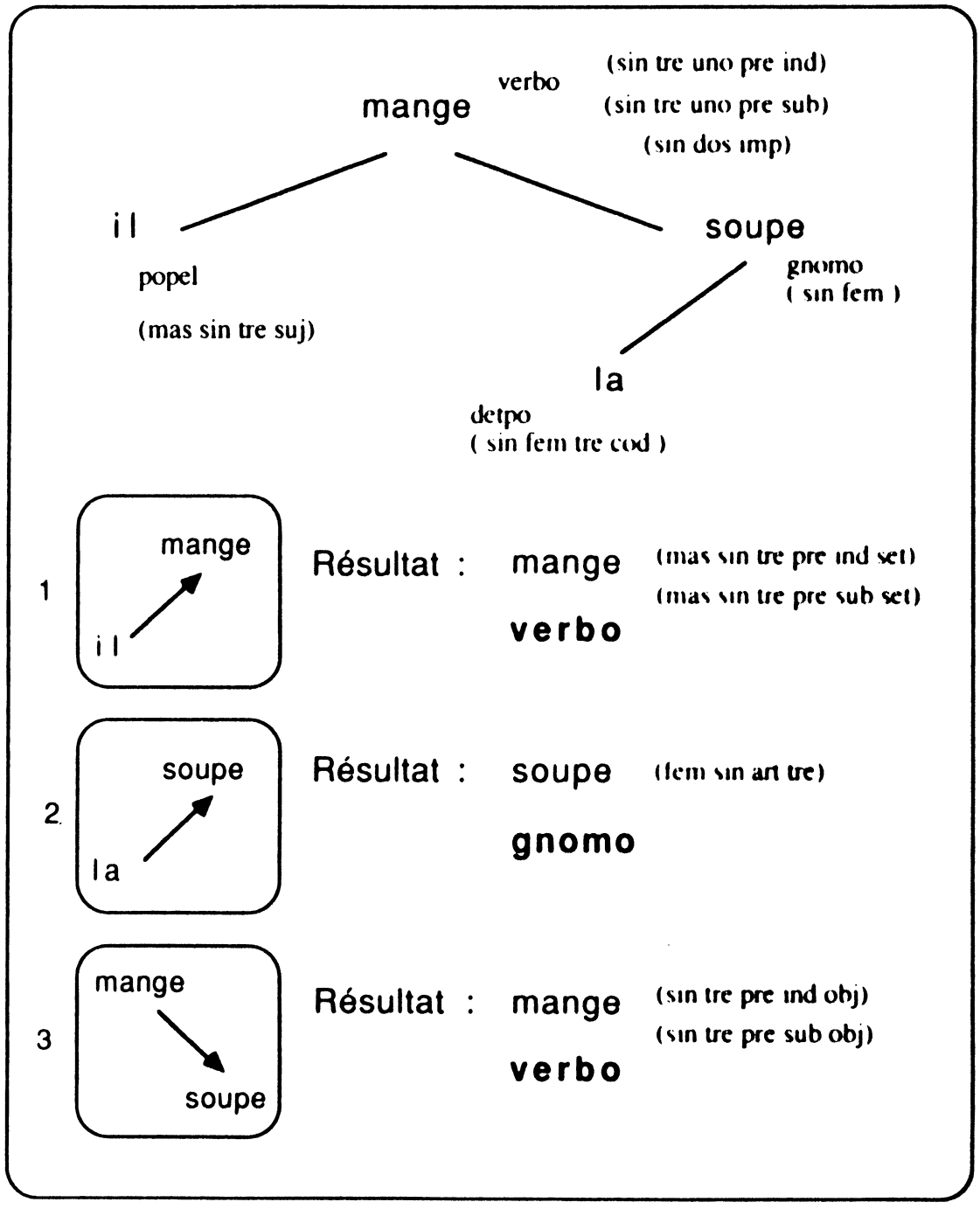
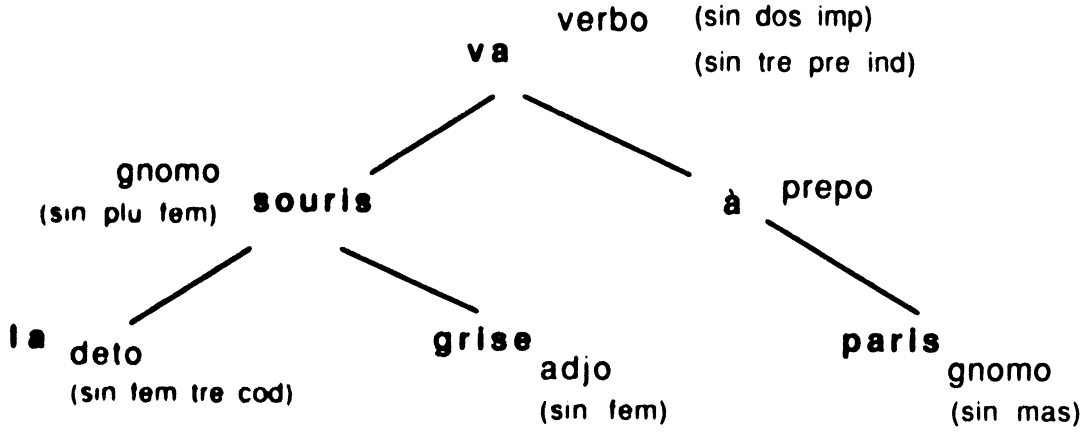


Figure 4.3 - Exemple d'application de règles

Nous présentons dans la figure 4.3 le même exemple de la figure 4.2 (analyse de la phrase *Il mange la soupe*), avec les listes de valeurs morphologiques et les catégories lexico-syntaxiques associées à chaque noeud et le résultat de chaque étape du parcours de l'arbre.

Voici un autre exemple dans lequel nous vérifions syntaxiquement la phrase *la souris grise va à Paris*.
L'arbre calculé par l'analyseur syntaxique est :



Vérification syntaxique :

1)	la	•	souris	→	souris	*gnomo* (sin fem tre art)
2)	souris	•	grise	→	souris	*gnomo* (sin fem tre art)
3)	souris	•	va	→	va	*verbo* (sin tre pre ind set art)
4)	à	•	Paris	→	Paris	*gnomo* (sin mas prp)
5)	va	•	Paris	→	va	*verbo* (sin tre pre ind set)

Les paragraphes 1 5 et 1 6 de ce chapitre contiennent une description détaillée des codes utilisés.

1.4 Grammaire syntaxique de l'arbre d'une phrase

Nous présentons ici la grammaire qui définit la syntaxe de l'arbre d'une phrase, telle qu'elle est acceptée par le vérificateur syntaxique pour effectuer les opérations d'unification de traits sur les noeuds.
Pour présenter cette grammaire nous utilisons une BNF (Bachus-Naur Form).

Tous les caractères ou symboles écrits en **gras** sont des terminaux.
Des commentaires sont donnés entre deux symboles §.

GRAMMAIRE DE L'ARBRE D'UNE PHRASE

<phrase>	::=	<noeudecor>
<noeudecor>	::=	**noeud** (<décoration> , [<pgd>] , [<pgd>]) .
		§ pgd : partie gauche/droite §
<pgd>	::=	vide | <pgd_non_vide>.
<pgd_non_vide>	::=	<noeudecor> <suite_pgd>.
<suite_pgd>	::=	vide | , <pgd_non_vide>.
<décoration>	::=	**Infms** (<lettres> , <cls> , <liste_var>) .
<cls>	::=	un symbole TURBOPROLOG représentant une catégorie lexico-syntaxique (gnomo, verbo, detpo, etc).
<lettres>	::=	une chaîne (symbole TURBO-PROLOG) qui garde : - le numéro du mot correspondant dans la phrase ; - le mot correspondant.
<liste_var>	::=	[<liste_v_morph> { , <liste_v_morph> }]
<liste_v_morph>	::=	[ ] | [<var_morph> { , <var_morph> }] .
<var_morph>	::=	un symbole TURBO-PROLOG représentant une valeur de variable morphologique (sin, plu, mas, fem, uno, dos, tre, etc).

Observation - les codes utilises sont decrits dans 1.5 et 1.6.

1.5 Variables morphologiques

Les analyseurs de PILAF utilisent 7 groupes concernant variables morphologiques : genre, nombre, personne, temps, mode, cas et mot. Voici les valeurs possibles de ces variables :

GROUPE	VALEURS
GNR ou genre	mas (masculin), fem (féminin)
NBR ou nombre	sin (singulier), plu (pluriel)

PRS ou personne	uno (première personne), dos (deuxième personne), tre (troisième personne)
TPS ou temps	pre (présent), imi (imparfait), pas (passé simple), fut (futur)
MOD ou mode	ind (indicatif), imp (impératif), cdl (conditionnel), sub (subjonctif)
CAS ou cas	suj (sujet), cod (complément d'objet direct), dat (complément d'objet indirect), cpr (complément de préposition) cdn (complément de nom)
MOT ou mot	int (interrogatif), exc (exclamatif), ref (réfléchi), ord (ordinal), pat (partitif), ide (indéfini), xav (auxiliaire avoir), xet (auxiliaire être)

1.6 Catégories et informations lexico-syntaxiques

Dans la liste qui suit nous présentons les catégories lexico-syntaxiques utilisées et ses constituants.

CATEGORIE	CONSTITUANTS
adjo	adjectifs qualificatifs, adjectifs indéfinis
ppaso	participe passé
advo	adverbes, pronom adverbial (y), locutions adverbiales
ce	ce
cocod	conjonctions de coordination
cosub	conjonctions de subordination, locutions de subordination
deto	déterminants
detpo	déterminants possessifs
gnomo	substantifs communs, substantifs propres
infio	infinitifs
neg1	ne (première partie d'une négation)
neg2	pas (deuxième partie d'une négation)
ppto	participe présent
ponct	point, deux-points
popel	pronoms personnels, pronoms possessifs
porel	pronoms relatifs
prepo	prépositions, locutions prépositionnelles
relco	pronoms relatifs conjonctifs
verbo	verbes

virgo	virgule
xavo	auxiliaire avoir
xavso	auxiliaire avoir dans une relative
xeto	auxiliaire être
xetso	auxiliaire être dans une relative
phrao	phrase
verso	vers

1.6.1 Informations lexico-syntaxiques

Nous avons deux groupes d'informations lexico-syntaxiques : **GNO** et **VBO**.

GNO est le groupe d'informations associées à un *gnomo*, qui peut prendre les valeurs **art** et **prp**.
L'information **art** est utilisée pour signaler qu'un *gnomo* a déjà eu son article vérifié (il a déjà "pris" son article).
L'information **prp** est utilisée pour signaler qu'un *gnomo* a déjà eu sa préposition vérifiée (il a déjà "pris" sa préposition).

VBO est le groupe d'informations associées à un *verbo*, qui peut prendre les valeurs **set** et **obj**.
L'information **set** est utilisée pour signaler qu'un *verbo* a déjà eu son sujet vérifié (il a déjà "pris" son sujet).
L'information **obj** est utilisée pour signaler qu'un *verbo* a déjà eu son objet vérifié (le verbo a déjà "pris" son objet).

2 REGLES ET META-REGLES DE VERIFICATION SYNTAXIQUE

Pour procéder à la vérification syntaxique des arbres de dépendances, nous avons mis au point une base composée de règles et de méta-règles de vérification syntaxique qui traitent les rapports élémentaires entre les composants d'une proposition.

Parmi les cas les plus généraux, nous traitons les rapports dans un groupe nominal (les rapports entre le nom et son déterminant, entre le nom et son adjectif), le verbe et son sujet, le sujet et son attribut, etc. Nous traitons aussi la conjonction de coordination *et*, les subordinations à partir d'un pronom relatif (*que*, *qui*) et de conjonctions de subordination (*que*, etc) et nous traitons le participe passé.

Lors de l'écriture des règles de vérification syntaxique, nous avons pu observer que les rapports entre deux paires différentes de catégories lexico-syntaxiques sont parfois traités d'une manière analogue.

Les restrictions de taille du code source disponible pour exprimer les clauses d'un même prédicat PROLOG nous ont aussi obligés à limiter le nombre de règles de vérification syntaxique.
En conséquence, nous avons conçu et mis en oeuvre un niveau additionnel de traitement inséré dans le processus de vérification syntaxique : les méta-règles.

Au point de vue du créateur de règles, comme nous l'avons déjà observé dans le paragraphe 1.2.2, les méta-règles simplifient l'écriture et la mise au point d'un système de vérification syntaxique.

A partir de ces deux ensembles, les règles et les méta-règles de vérification, nous avons construit notre vérificateur syntaxique.

2.1 *Règles* de vérification syntaxique

Une règle de vérification syntaxique opère sur deux groupements de traits lexico-syntaxiques en produisant :

- une liste de traits lexico-syntaxiques résultants de l'unification des deux noeuds analysés et
- un indicateur d'existence ou non-existence de problème d'accord.

Une règle de vérification syntaxique est constituée de trois parties :

- un **en-tête** (ou une proposition syntaxique) qui tient compte des catégories lexico-syntaxiques des noeuds auxquels cette règle est applicable ;

- une **proposition conditionnelle** (appelée **SI**) qui contient les conditions nécessaires à l'applicabilité de cette règle ;

- une **expression d'affectation** (appelée **ALORS**) qui calcule les informations lexico-syntaxiques associées au noeud résultant de l'application de cette règle.

Nous présentons dans la figure 4.4 un exemple de règle de vérification syntaxique

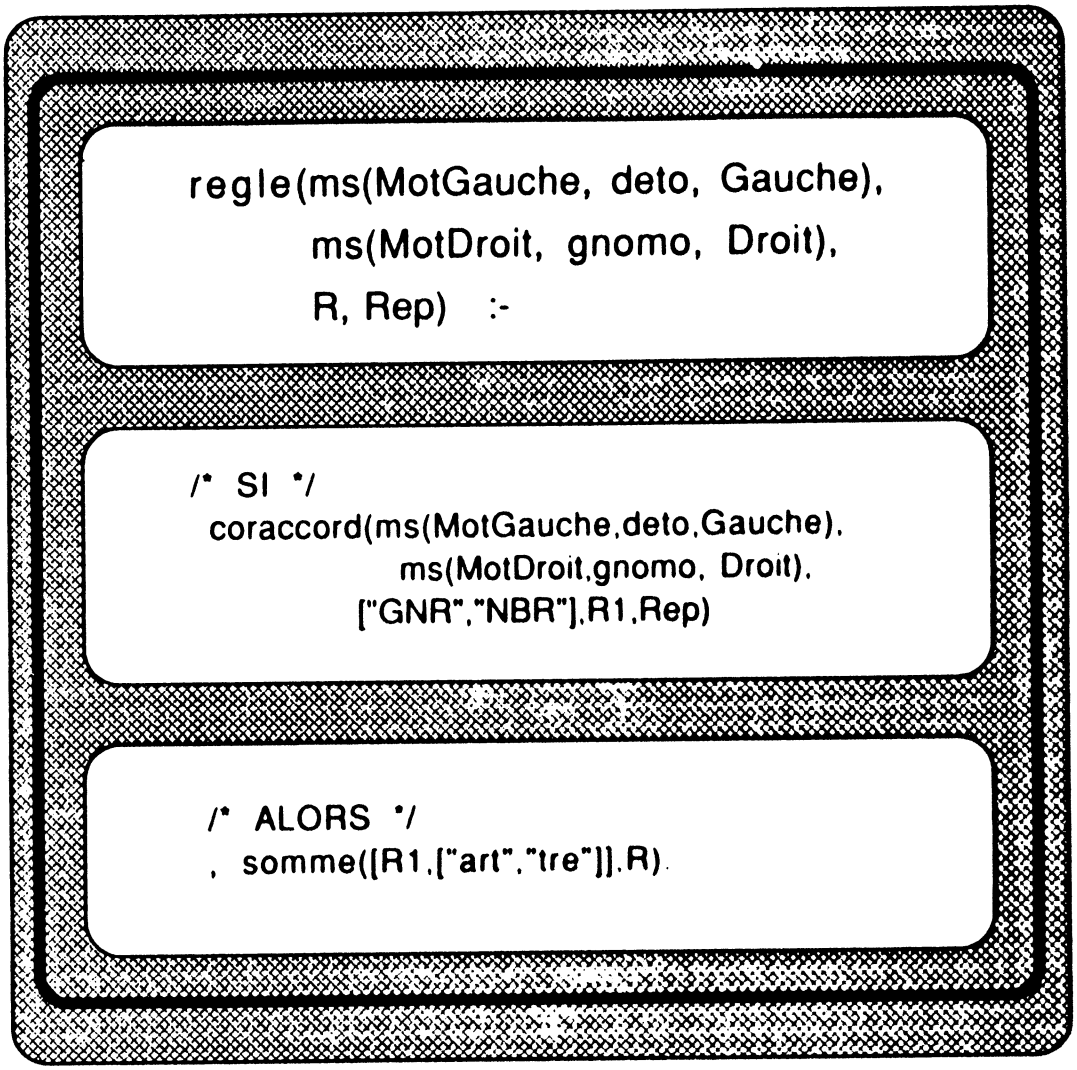


Figure 4.4
Une règle de vérification syntaxique

Commentaires à propos de la figure 4.4 et à propos des règles en général :

1) Une règle opère sur deux groupements de traits lexico-syntaxiques. Un groupement de traits lexico-syntaxiques se rapporte à un noeud de l'arbre en vérification et il a la forme :

ms (*mot*, *catégorie*, *liste_de_traits*)

où :
- *mot* est le mot associé au noeud en question ;
- *catégorie* est sa catégorie lexico-syntaxique ;
- *liste_de_traits* est la liste de traits lexico-syntaxiques associée à ce noeud.

Un groupement ayant cette structure sera traité simplement par *groupement du type ms*.
A l'écriture d'une règle les groupements de traits lexico-syntaxiques associés aux noeuds gauche et droit sont notés respectivement par :

ms(MotGauche, **catégorie1**, Gauche) et

ms(MotDroit, **catégorie2**, Droit).

L'applicabilité de la règle à une paire de noeuds en analyse est décidée en fonction des catégories lexico-syntaxiques **catégorie1** et **catégorie2**.
La règle donnée en exemple est applicable à une paire de noeuds constituée par un *deto* à gauche et un *gnomo* à droite : (**deto * gnomo**).

2) Dans une règle, la vérification d'accord entre les deux noeuds s'effectue à travers le prédicat **coraccord.** Ce prédicat est aussi chargé de garder les paramètres de correction, dans les cas de faute d'accord.
Sont passés comme paramètre à **coraccord** : les deux groupements du type ms associés respectivement aux noeuds gauche et droit de l'application et la liste des variables dont l'accord doit être vérifié. La règle présentée comme exemple vérifie l'accord en genre (GNR) et nombre (NBR) entre les deux noeuds.
Comme résultat, **coraccord** produit :
- une liste **R1** de résultats concernant les valeurs associées aux variables accordées (exemple : *le* → *belge* produira R1 = [mas sin]) et
- l'indicateur **Rep** qui signale l'existence ou non de faute d'accord.

A la détection d'une faute d'accord, **coraccord** garde des paramètres de correction dans un fichier de communication avec le module de correction.
Evidemment, toutes les règles ne comportent pas une vérification d'accord. Les règles ne contenant pas de vérification d'accord ont comme résultat de **Rep** une réponse **ac**, toujours positive. Actuellement, ce genre de réponse a pour but de remplir la position d'un paramètre - pour la continuation de ces travaux, nous envisageons la correction d'une gamme plus vaste de fautes, et les réponses **ac** toujours positives pourront être remplacées par différents paramètres de correction.

3) Les résultats calculés par une règle sont la liste de traits lexico-syntaxiques **R** associée au noeud produit par cette analyse et l'indicateur de problème d'accord **Rep**.
Dans l'exemple, font partie de **R** :

- la liste **R1** calculée pendant la vérification d'accord (les valeurs de genre et de nombre du gnomo) ;
- la constante lexico-syntaxique **art**, qui signale que *le gnomo a déjà absorbé son article* ;
- la valeur **tre**, qui souligne que *le gnomo est à la troisième personne* (information importante dans la suite de la vérification).

4) Il faut observer que l'analyse d'une paire de noeuds pour laquelle on ne fait pas de vérification d'accord a pour objectif le déplacement dans l'arbre syntaxique, en tenant compte des rapports existants entre les branches et en unifiant les traits syntaxiques. Par exemple, dans l'analyse de la phrase *Pierre a mangé un bonbon et un chocolat* nous n'effectuons pas de vérification d'accord pour les paires ***bonbon*** → ***et*** ou ***et*** → ***chocolat*** : nous simplement signalons que les mots à gauche et à droite de la conjonction de coordination ***et*** ont été trouvés, et nous gardons les valeurs des variables de genre et nombre associées au groupe *un bonbon et un chocolat.*

Nous avons mis au point un ensemble de règles qui doivent modéliser les rapports les plus fondamentaux de la langue française, compte tenu de l'environnement morpho-syntaxique PILAF. Cet ensemble couvre une partie considérable des règles du "français fondamental" [LACOUTURE 86, LACOUTURE 88, RICHARD 86] bien que le "français fondamental" n'ait pas été envisagé comme objectif premier de nos travaux.
La liste complète des règles de vérification syntaxique développées se trouve dans l'annexe 3 de ce document.

2.2 *Méta-règles* de vérification syntaxique

Une méta-règle de vérification syntaxique a pour objectif de diriger dans un sens déterminé l'application des règles de vérification syntaxique : en analysant les catégories lexico-syntaxiques des noeuds en traitement, une méta-règle canalise le traitement des situations similaires vers une même règle.
Par exemple, un adjectif qualificatif qui suit un substantif commun, comme dans *robe jaune* (liaison représentée par ***gnomo * adjo***) est traité de la même manière qu'un participe passé qui suit un substantif commun (***gnomo * ppaso***), comme dans *air réfléchi*. Ces deux situations sont alors traitées par deux méta-règles spécifiques, qui déclencheront l'application d'une même règle.

La figure 4.5 présente une liste de quelques méta-règles définies dans le système de vérification syntaxique.

catégories d'origine		catégories calculées		Catégorie résultante	Mot résultant
noeud gauche	noeud droit	noeud gauche	noeud droit	(ClsRes)	(MotRes)
deto	gnomo	deto	gnomo	gnomo	MotDroit
adjo	gnomo	gnomo	adjo	gnomo	MotDroit
gnomo	adjo	gnomo	adjo	gnomo	MotGauche
gnomo	verbo	gnomo	verbo	verbo	MotDroit
verbo	gnomo	verbo	gnomo	verbo	MotGauche
verbo	prepo	verbo	prepo	verbo	MotGauche
popel	verbo	popel	verbo	verbo	MotDroit
detpo	verbo	detpo	verbo	verbo	MotDroit
detpo	xavo	detpo	verbo	xavo	MotDroit
popel	xavso	popel	verbo	xavso	MotDroit
prepo	gnomo	prepo	gnomo	gnomo	MotDroit
detpo	gnomo	deto	gnomo	gnomo	MotDroit
gnomo	prepo	gnomo	prepo	gnomo	MotGauche
popel	xeto	popel	verbo	xeto	MotDroit
gnomo	xeto	gnomo	verbo	xeto	MotDroit
xeto	gnomo	xeto	gnomo	xeto	MotGauche
xeto	adjo	xeto	adjo	xeto	MotGauche
gnomo	cocod	gnomo	cocod	cocod	MotDroit
cocod	gnomo	cocod	gnomo	gnomo	MotDroit
popel	xavo	popel	verbo	xavo	MotDroit
gnomo	xavo	gnomo	verbo	xavo	MotDroit
xavo	gnomo	verbo	gnomo	xavo	MotGauche
xavo	ppaso	xavo	ppaso	ppaso	MotDroit
xavso	ppaso	xavso	ppaso	xavso	MotGauche
xetso	ppaso	xeto	adjo	xetso	MotGauche
xetso	adjo	xeto	adjo	xetso	MotGauche
ppaso	gnomo	ppaso	gnomo	ppaso	MotGauche
xeto	ppaso	xeto	adjo	ppaso	MotDroit
gnomo	ppaso	gnomo	adjo	gnomo	MotGauche
ppaso	prepo	ppaso	prepo	ppaso	MotGauche
verbo	cocod	verbo	cocod	cocod	MotDroit
cocod	verbo	cocod	verbo	verbo	MotDroit
xeto	cocod	verbo	cocod	cocod	MotDroit
cocod	xeto	cocod	verbo	xeto	MotDroit
adjo	cocod	gnomo	cocod	cocod	MotGauche
cocod	adjo	cocod	adjo	adjo	MotDroit
xavo	popel	popel	verbo	xavo	MotGauche
xavo	detpo	detpo	verbo	xavo	MotGauche
relco	ppaso	relco	ppaso	ppaso	MotDroit
porel	ppaso	relco	ppaso	ppaso	MotDroit
porel	verso	relco	ppaso	verso	MotDroit
gnomo	verso	gnomo	verso	verso	MotDroit

Figure 4.5

Quelques méta-règles de vérification syntaxique

Analysant les catégories lexico-syntaxiques des noeuds d'une application :
- une méta-règle tient compte de ces catégories et détermine la catégorie lexico-syntaxique du noeud résultat (ClsRes) ;
- elle détermine le mot qui sera gardé comme résultat de la paire de noeuds analysée, en vue d'une correction (MotRes).

La vérification syntaxique est réalisée par le prédicat **mregle**, qui effectue l'analyse/unification de traits appelant d'abord les méta-règles (*prep_appl_reg*) et ensuite les règles (*regle*) de vérification syntaxique :

mregle (MsGauche, MsDroit, ms(MotRes, ClsRes, R), Rep) -

prep_appl_reg(MsGauche, MsDroit,

MsNouveauGauche, MsNouveauDroit, MotRes, ClsRes)

, *regle*(MsNouveauGauche, MsNouveauDroit, R, Rep)

... où :
- MsGauche et MsDroit sont les structures du type ms (voir paragraphe 2.1) associées respectivement aux noeuds gauche et droit de l'application ;
- ms(MotRes, ClsRes, R) est la structure du type ms associée au résultat de l'application ;
- MsNouveauGauche et MsNouveauDroit sont les nouvelles structures du type ms calculées, respectivement, pour les noeuds gauche et droit ;
- MotRes, ClsRes et R sont, respectivement, le mot, la catégorie lexico-syntaxique et la liste de traits lexico syntaxiques associés au noeud résultat de l'application.

Ce sont les clauses associées au prédicat *prep appl reg* qui établissent les nouvelles catégories lexico syntaxiques à utiliser, le mot résultant et la catégorie lexico-syntaxique du résultat, pour une paire de noeuds déterminée. Par conséquent, elles sont l'essence des méta-règles.
Nous avons mis au point une cinquantaine de clauses de préparation d'application de règles.

Exemple :
Considérons la vérification syntaxique de la proposition *le fils aimé*, comprenant le traitement d'un substantif (***gnomo***) à gauche avec un participe passé (***ppaso***) à droite.

Déroulement :

→ La vérification syntaxique examine d'abord les méta-règles (*prep_appl_regle*).

→ Une méta-règle est trouvée pour la paire (***gnomo * ppaso***) :

prep_appl_reg (ms(MotGauche,gnomo,Gauche), ms(MotDroit, ppaso, Droit),

ms(MotGauche,gnomo,Gauche), ms(MotDroit, adjo, Droit),

MotGauche, gnomo)

Cette méta-règle change la catégorie lexico-syntaxique du composant à droite en ***adjo*** La paire résultante est (***gnomo * adjo***).
La méta-règle utilisée détermine aussi la catégorie du résultat de l'analyse (***gnomo***) et le mot associé à ce résultat (MotGauche, c'est-à-dire, *fils*).

→ L'analyse se poursuit par la recherche d'une règle pour la paire (***gnomo * adjo***).

Il faut observer que, si aucune règle n'est trouvée qui puisse être appliquée à partir d'une méta-règle, le système met en évidence *l'absence de règles pour l'application en question*, et conduit l'utilisateur à une révision de la phrase proposée.

De la même manière mais dans un niveau supérieur de traitement, si aucune méta-règle n'est trouvée qui puisse être appliquée à une paire déterminée de noeuds, le système met en évidence *l'absence de règle pour l'application en question.*

La description complète des méta-règles de vérification syntaxique définies est donnée dans l'annexe 4 de ce document.

3 CONSIDERATIONS A PROPOS DU TRAVAIL REALISE

Pour développer le module vérificateur syntaxique tel qu'il se présente actuellement nous avons pris, comme point de départ, les travaux de S. Saidi [SAIDI 86].

Saidi a programmé en PROLOG-CRISS (version 4 de multics) un module qui réalisait le parcours d'un arbre syntaxique en ordre *postfixé*, en employant des règles appartenant à une base de règles élémentaires de test. Le module s'arrêtait au moment où il ne trouvait pas de règle

applicable pour une paire déterminée de noeuds (sans discerner la présence de fautes d'accord dans une application).

3.1 Apports

Le système d'analyse d'un arbre a donc été premièrement modifié en ce qui concerne les fautes d'accord. Nous avons mis au point un prédicat qui vérifie l'accord entre les deux noeuds d'une application par rapport à un ensemble de groupes de variables morphologiques (le prédicat *coraccord*).
Le prédicat développé réalise aussi le stockage des variables qui permettent d'engendrer la correction d'une faute d'accord.

Ensuite, nous avons modifié le système original d'écriture de règles de syntaxe, en y ajoutant des paramètres concernant la correction.
Jusqu'à présent, ces paramètres ont été utilisés seulement pour signaler les problèmes d'accord. Mais il est possible d'élargir les alternatives de détection, par des paramètres associés, par exemple, aux fautes de transposition du pronom (*je **lui le** donne* au lieu de *je le lui donne*), ou des paramètres associés à l'omission de composants importants de la phrase (par exemple, dans *Je promène dans le parc*).

Finalement, nous avons procédé à la structuration d'une base de règles syntaxiques élémentaires. Le noyau de cette base a été développé ; de nombreuses autres règles à écrire suivront certainement.

Par ailleurs, le processus d'acquisition d'un grand dictionnaire pour le système, récemment conclu, entraîne la nécessité d'une réévaluation des règles de dépendance et des catégories lexico-syntaxiques initialement définies pour l'analyse morpho-syntaxique. A mesure que les règles de dépendances seront réévaluées et que l'ensemble des catégories lexico-syntaxiques deviendra stable, nous pourrons avoir une base de règles syntaxiques plus complète et stabilisée.

Le module de vérification syntaxique résultant (écrit en TURBOPROLOG version 1.0) a été lié à l'analyseur morpho-syntaxique de PILAF (écrit en TURBOPASCAL version 5.0), ce qui permet de réaliser une analyse et vérification en continu.
En utilisant la génération de PILAF et les paramètres de correction de fautes d'accord calculés pendant la vérification syntaxique, nous avons construit une unité de correction de fautes d'accord par génération.
Le problème de la correction de fautes d'accord sera considéré en détail dans le chapitre suivant de ce document.

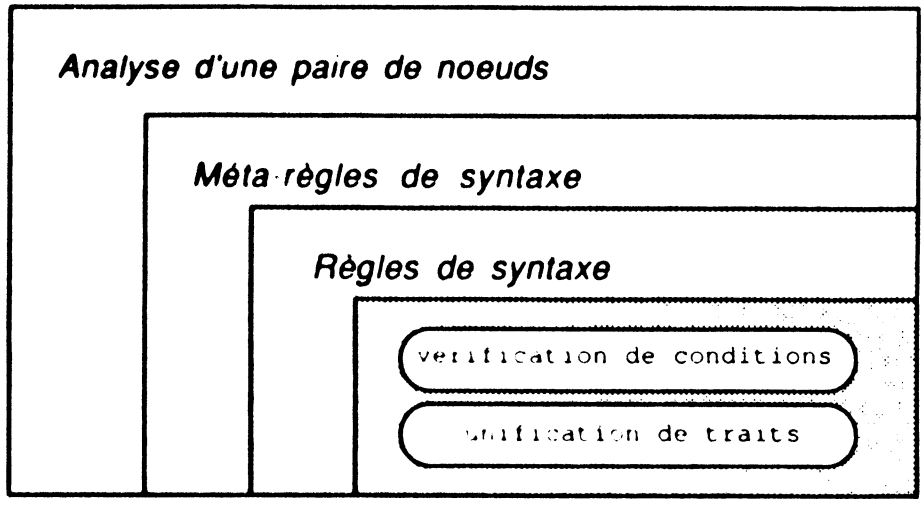


Figure 4.6 - Vérification syntaxique

La figure 4 6 présente l'organisation modulaire de la vérification syntaxique

Pour démontrer la faisabilité des méthodes de correction proposées aux niveaux lexical et syntaxique, nous avons mis en rapport ces méthodes avec le programme d'analyse morpho-syntaxique de PILAF, et avons obtenu comme résultat une maquette de vérification/correction morpho-syntaxique (la maquette CORSYNT, exposée en détail dans le chapitre 6)

3.2 Observations à propos des langages de programmation utilisés

Dans le présent travail, nous avons utilisé le langage TURBOPROLOG pour programmer le vérificateur syntaxique proposé. Le choix du PROLOG comme langage de programmation représente, d'une part, une suite aux travaux réalisés par S Saidi, et donc disponibles, dans le cadre du projet PILAF D'autre part, cette approche facilite le traitement des langues naturelles

PROLOG possède des mécanismes de retour arrière qui pourraient permettre l'évaluation de toutes les possibilités d'analyse pour une même phrase aussi bien qu'une écriture claire des règles de vérification syntaxique Par ailleurs, PROLOG permet la confection d'une maquette rapidement, ce qui est spécialement adéquat à la réalisation d'un prototype.

En ce qui concerne l'écriture des règles de vérification syntaxique en PROLOG, nous avons observé un rapport entre l'énoncé et la codification qui rend lisibles et directes les règles écrites.

Cependant, en ce qui concerne l'applicabilité des règles écrites, nous n'utilisons pas de retour arrière : dès qu'on a trouvé une règle applicable, nous écartons toute autre candidate. Alors, l'adéquation d'un langage comme PROLOG ne peut être observée que partiellement dans les unités de notre système.
L'algorithme du vérificateur est déterministe, mais la programmation de certains prédicats est parfois longue et répétitive, en raison du non-déterminisme du contrôle PROLOG.

Il nous paraît intéressant, pour faciliter le traitement de ce genre de problèmes, d'utiliser un langage plus déterministe (par exemple, le langage C), associé au langage PROLOG dans le module de vérification syntaxique.

CHAPITRE 5

DETECTION ET CORRECTION DES FAUTES D'ACCORD

Nous présentons dans ce chapitre les problèmes rencontrés lors de l'établissement de paramètres pour la détection et correction de fautes d'orthographe au niveau syntaxique, et les solutions adoptées dans notre système.

Les principales règles d'accord en français sont énoncées (selon transcription de [BESCHERELLE 84]) et ensuite analysées en vue de l'environnement lexico-syntaxique dans lequel nous avons travaillé. Un ensemble de mesures est proposé et réuni dans une unité d'analyse post-syntaxique, ayant pour objectif de rendre les structures de dépendances construites par l'analyseur syntaxique de PILAF plus adaptées au traitement des fautes d'accord.

1 FAUTES SYNTAXIQUES TRAITEES

1.1 Les types de fautes syntaxiques

Les erreurs syntaxiques détectées dans le système sont essentiellement de deux types :

I. les *erreurs d'accord* entre les éléments composant la proposition, par exemple :

Les souris ***mange*** *du fromage* au lieu de
Les souris mangent du fromage ou

La soupe que j'ai ***mangé*** au lieu de
La soupe que j'ai mangée ou

Le chat mange ***le*** *souris* au lieu de
Le chat mange la souris ou *Le chat mange les souris.*

II. les *erreurs de style*, c'est-à-dire, des constructions considérées comme impossibles ou inadéquates dans la langue française et qui, par conséquent, ne sont pas modélisées par des règles de vérification syntaxique :

a) certaines erreurs de placement du pronom, comme :
Je ***lui le*** *donne* au lieu de *Je le lui donne.*

b) certaines constructions impropres, comme :
Père et le fils sont partis au lieu de
Le père et le fils sont partis ou *Père et fils sont partis.*

1.2 Des cas d'absence de structure syntaxique

Dès lors que l'analyse syntaxique précède la vérification syntaxique, certaines constructions complètement inadaptées sont déjà éliminées, faute de proposition d'une structure.
Exemple :

Le chat et mangent la souris.

===> le terme à droite (ou le terme à gauche) de la conjonction de coordination est absent.

Cette analyse peut s'avérer parfois trop sélective, éliminant à tort des structures de dépendances que la vérification syntaxique pourrait bien traiter. Mais elle est justifiable dès qu'elle permet de faire passer à la vérification syntaxique seulement des phrases avec un minimum de cohérence, possédant un ensemble essentiel de termes composants.
L'analyseur syntaxique de PILAF est toujours objet d'études en ce qui concerne les cas particuliers de l'analyse. Au fur et à mesure que l'analyseur syntaxique évolue, ce genre de constructions sera vraisemblablement traité convenablement.

Il faut aussi observer que, avec les règles utilisées actuellement, notre vérificateur syntaxique ne fait pas de distinction entre les causes d'une absence de règle : soit une omission délibérée de règle, soit une omission occasionnelle, comme il arrive souvent dans une étape d'implantation d'un jeu de règles élémentaires.
Par exemple, citons l'omission délibérée de la règle **ppaso * gnomo**, qui gère le rapport entre les catégories *participe passé* et *substantif*, dans la version actuelle du vérificateur : on n'admet pas un participe passé utilisé comme adjectif placé avant un nom.
En fait, dans [GREVISSE 69] nous trouvons des *remarques particulières* concernant la place de l'adjectif épithète, dont une à propos des participes passés :

> On place **après** le nom :
> ...
> - Les participes passés pris adjectivement et beaucoup d'adjectifs verbaux en *ant* : *Un homme estimé. Un monarque redouté. Une musique éclatante. Des sables mouvants.*

Etant parfois difficile d'arriver à un consensus parmi les opinions énoncées par différents linguistes, nous considérons que les exceptions (comme ***vénéré*** *maître*) peuvent toutefois être trouvées et seront traitées par des mécanismes spécifiques dans le système.

2 LES REGLES D'ACCORD

Parmi les rapports dont s'occupe la *syntaxe*, selon Grevisse [GREVISSE 69], ceux qui concernent l'accord ont, du point de vue de la correction et des règles orthographiques, une importance particulière.
On entend par *accord* la concordance établie, quant à une ou plusieurs des catégories morphologiques de genre, de nombre, de personne, entre deux ou plusieurs termes se rapportant à un même énoncé ou à une même chose. De ces termes, l'un, plus fort (par exemple, le nom), impose à un ou plusieurs autres, plus faibles (par exemple, l'épithète), situés dans sa sphère d'influence, la nécessité de prendre la même position que lui dans les catégories morphologiques.

2.1 Règles d'accord pour la langue française

Nous considérons comme les principales règles d'accord du français les règles décrites dans le volume 3 de "Le Nouveau Bescherelle" [BESCHERELLE 84], et listées dans le texte qui suit :

I. ACCORD DU VERBE AVEC LE GROUPE NOMINAL SUJET

A Il n'y a qu'un groupe nominal sujet :

A 1 *le verbe se met à la même personne que le groupe nominal sujet ;*
Exemple :
Nous dev**ons** vraiment partir.

A 2 *le verbe se met au singulier si le groupe nominal sujet est au singulier ; il se met au pluriel si le groupe nominal sujet est au pluriel ;*
Exemples :
La souris mange le fromage.
Les petits animaux dormai**ent**.

A 3 *lorsque le groupe nominal sujet comporte un adverbe de quantité comme* ***beaucoup de, peu de, combien de, que de****, etc, le verbe se met au pluriel ;*
Exemple :
Beaucoup de ces enfants chant**ent** dans la chorale.

Remarque :
Lorsque l'adverbe de quantité a un sens partitif, le verbe se met au singulier.

Exemple :
Peu de neige **est** tombé**e** cet hiver.

A 4 *lorsque le groupe nominal sujet représente un ensemble d'éléments (sujet collectif), le verbe se met soit au singulier soit au pluriel.*
Exemple :
Une foule de visiteurs se précipit**a** (ou se précipit**èrent**).

B Il y a plusieurs groupes nominaux sujets :

B 1 *lorsqu'il y a plusieurs groupes nominaux sujets, le verbe se met au pluriel ;*
Exemple :
Jacques et Pierre déci**dèrent** d'aller au cinéma.

B 2 *lorsque les groupes nominaux sujets sont de personnes différentes, plusieurs cas d'accord se présentent :*
a. 2e personne + 3e personne : le verbe se met à la 2e personne du pluriel ;
Exemple :
Marie et toi marche**rez** derrière.
b. 1re personne + 2e ou 3e personne : le verbe se met à la 1re personne du pluriel ;
Exemple :
Mes amis et moi voul**ions** faire ce cadeau.

B 3 *lorsque les deux groupes nominaux sujets sont réunis par* ***comme, ou, ainsi que, avec, ni,*** *le verbe se met soit au singulier soit au pluriel.*
Exemple :
La bière comme le vin cont**ient** (ou cont**iennent**) de l'alcool.

II. ACCORD DE L'ADJECTIF QUALIFICATIF

A Accord de l'adjectif qualificatif épithète :

A 1 *l'adjectif qualificatif épithète s'accorde en genre et nombre avec le nom qu'il qualifie ;*
Exemple :
L'enfant **ravi** monta sur son vélo **neuf**.

A 2 *lorsque l'adjectif qualifie plusieurs noms il se met au pluriel ;*
Exemple :
Les étrangers aiment la cuisine et la littérature **françaises**.

A 3 *lorsque l'adjectif qualifie plusieurs noms de genres différents, il se met au masculin pluriel ;*
Exemple :
L'homme portait une chemise et un pantalon **blancs**.

B Accord de l'adjectif qualificatif attribut :

B 1 *l'adjectif qualificatif attribut s'accorde en genre et en nombre avec le groupe nominal sujet ;*
Exemple :
Les enfants étaient **heureux**.

B 2 *s'il y a deux groupes nominaux sujets, l'adjectif attribut se met au pluriel ; si ces deux groupes nominaux sujets sont de genre différent, l'adjectif attribut se met au masculin pluriel ;*
Exemple :
La soupe et le poisson sont **froids**.

B 3 *lorsque l'adjectif attribut est construit avec* ***avoir l'air****, on peut soit le mettre au masculin singulier en l'accordant avec le nom* ***air*** *(masculin singulier), soit l'accorder avec le sujet ;*
Exemple :
Elle a l'air bien **sérieux**.
Elle a l'air bien **sérieuse**.

III. ACCORD DU PARTICIPE PASSE

A Participe passé employé comme épithète :

Le participe passé en fonction d'épithète s'accorde en genre et en nombre avec le nom qu'il qualifie .
Exemple :
Un homme averti en vaut deux.

B Participe passé employé avec l'auxiliaire avoir :

B 1 *lorsqu'il n'y a pas de complément d'objet direct, le participe passé reste invariable ;*
Exemple :
Ils avaient couru comme des fous.

B 2 *lorsque le complément d'objet direct se trouve après le verbe, le participe passé reste aussi invariable ;*
Exemple :
Les enfants ont dévoré tous les gâteaux.

B3 *lorsque le complément d'objet direct se trouve placé avant le verbe, le participe passé s'accorde en genre et en nombre avec lui.*
Exemple :
Tu n'as même pas regardé les fleurs que je t'ai offert**es**.

C Le participe passé employé avec l'auxiliaire être :

Il s'accorde en genre et en nombre avec le groupe nominal sujet.
Exemple :
Les feuilles des arbres étaient tomb**ées**.

C1 *le participe passé du verbe pronominal présente plusieurs cas d'accord :*
a. *lorsque le pronom (me, te, se, ...) est le complément d'objet direct du verbe (**se** rencontrer, **se** baigner, **se** vendre, **se** sauver, ...), le participe passé s'accorde en genre et en nombre avec le sujet ;*
Exemple :
Elles se sont baign**ées** dans la rivière.
b. *lorsque le pronom est le complément d'objet indirect du verbe (s'acheter, se faire mal, se dire, etc), le participe passé ne s'accorde pas ni en genre ni en nombre avec le sujet ;*
Exemple :
Elle s'est dit qu'il ne viendrait pas.
Elles se sont lav**é** les mains.

En revanche, le participe passé s'accordera avec le complément d'objet direct s'il est placé avant le verbe.
Exemple :
Tu ne peux imaginer les choses que je me suis di**tes**.

2.2 Le participe passé suivi d'un infinitif

L'accord du participe passé conjugué avec *avoir* et suivi d'un infinitif pur ou prépositionnel est réglé en tenant compte du rapport entre ce participe et le pronom qui le précède (règle générale extraite de [GREVISSE 69]).
Exemple :

*Les chanteuses que j'ai entend**ues** chanter.*
→ Le participe passé conjugué avec *avoir* et suivi d'un infinitif pur ou prépositionnel s'accorde lorsque le pronom objet direct qui précède se rapporte à ce participe.

Question : J'ai entendu qui?
Réponse : ***que***, c'est-à-dire, les chanteuses, qui chantaient.

Les chansons que j'ai entendu chanter...
→ Le participe reste invariable lorsque le pronom objet direct qui précède se rapporte à l'infinitif.

Question : J'ai entendu quoi?
Réponse : ***chanter que***, c'est-à-dire, chanter les chansons.

Cette règle peut être énoncée différemment (comme le font Richard et Lapalme dans [RICHARD 86]) :

> *Nous devons laisser le participe passé invariable chaque fois que le nom qui précède ne peut pas faire l'action. Autrement, il n'est pas certain qu'il y ait accord : nous devons alors procéder à d'autres vérifications.*

Toutefois, l'analyse de ce type de situation n'est possible qu'en présence des traits sémantiques et parfois même des informations contextuelles permettant de connaître les capacités du sujet par rapport à la réalisation de l'action signalée par le verbe et l'ensemble des circonstances qui entoure la phrase.
Exemple :
Nous ne pouvons pas décider l'accord du participe passé dans une phrase comme : *Les fillettes que Marie a vu(es) vendre* (exemple extrait de [RICHARD 86]). Il est impossible de dire sans ambiguïté le sens réel de cette phrase à moins de disposer d'informations supplémentaires, de nature contextuelle mais pas seulement sémantique.
Par exemple, supposons que Marie a vu les fillettes lorsqu'elles vendaient des fraises au marché. Dans ce cas, les fillettes ont pratiqué l'action de vendre. Le participe passé *vu* s'accorde alors avec *les fillettes*.
Supposons que nous sommes dans une autre situation, complètement différente de la première. Il s'agit maintenant d'un cas de vente d'enfants. Dans ce cas, Marie a vu quelqu'un d'autre vendre les fillettes. Le participe passé *vu* ne s'accorde donc pas.

Ce type de problème, aussi bien que plusieurs autres problèmes d'accord, a été l'objet d'analyse par le Ministère de l'Instruction Publique et des Beaux-Arts au début du siècle, étant à l'origine d'un arrêté relatif à la simplification de l'enseignement de la syntaxe française, signé à Paris le 26 février 1901 par le Ministre Georges LEYGUES. Des parties de cet arrêté qui se réfèrent au participe passé sont transcrites de [GREVISSE 69] :

Participe passé. - Il n'y a rien à changer à la règle d'après laquelle le participe passé construit comme épithète doit s'accorder avec le mot qualifié, et construit comme attribut avec le verbe *être* ou un verbe intransitif doit s'accorder avec le sujet.
Ex. : *des fruits gâtés* ; - *ils sont tombés* ; - *elles sont tombées.*

Pour le participe passé construit avec l'auxiliaire *avoir*, lorsque le participe passé est suivi, soit d'un infinitif, soit d'un participe présent ou passé, on tolérera qu'il reste invariable, quels que soient le genre et le nombre des compléments qui précèdent. Ex. : *les fruits que je me suis laissé* ou *laissés prendre* ; - *les sauvages qu'on a trouvé* ou *trouvés errant dans les bois.* Dans le cas où le participe passé est précédé d'une expression collective, on pourra à volonté le faire accorder avec le collectif ou avec son complément. Ex. : *la foule d'hommes que j'ai vue* ou *vus.*

3 MODELISATION DES REGLES D'ACCORD

Pour analyser l'accord entre les composants d'une paire de noeuds de l'arbre syntaxique, nous avons suivi les règles d'accord qui figurent dans le volume 3 du Nouveau Bescherelle [BESCHERELLE 84], reproduites dans le paragraphe 2 de ce chapitre.

Nous avons modélisé les accords entre deux catégories lexico-syntaxiques en tenant compte des catégories et des informations lexico-syntaxiques disponibles dans le cadre de l'analyse morphologique et syntaxique de PILAF.
En conséquence, nous avons parfois trouvé des règles d'accord difficiles ou même impossibles à modéliser, avec les seules informations dont nous disposons au niveau lexical et syntaxique.
Exemple :

Dans les catégories *adverbe* (**advo**) ou *adjectif* (**adjo**), nous n'avons pas de sous-catégories, ce qui nous empêche de connaître la nature d'un adjectif ou le type d'un adverbe (nous donnons, à la fin de ce chapitre, un ensemble de commentaires à propos des informations nécessaires à une modélisation exhaustive des règles d'accord).

Pour cette raison, certaines règles d'accord citées dans [BESCHERELLE 84] n'ont pas été modélisées dans la version actuelle du système de vérification/correction. Mais il est tout à fait possible d'ajouter ce genre d'informations à nos dictionnaires en traitant convenablement le problème, comme nous l'exposons dans la conclusion de ce chapitre.

3.1 Les règles modélisées

Dans le système, nous avons modélisé les règles suivantes :

a. *Accord du verbe avec le groupe nominal sujet, en nombre et en personne, quand il n'y a qu'un groupe nominal sujet* : règles I.A1 et I.A2 ;

b. *Accord du verbe avec le groupe nominal sujet, quand il y a plusieurs groupes nominaux sujets* : règle I.B1;

c. *Accord de l'adjectif qualificatif épithète* : règles II.A1, II.A2, II.A3 ;

d. *Accord de l'adjectif qualificatif attribut* : règles II.B1, II.B2 ;

e. *Accord du participe passé employé comme épithète* : règle III.A ;

f. *Accord du participe passé employé avec l'auxiliaire avoir* : règles III.B1, III.B2, III.B3 ;

g. *Accord du participe passé employé avec l'auxiliaire être* : règles III.C, III.C1 ;

h. *Accord du participe passé suivi d'un infinitif* : simplification selon l'arrêté de 1901.

3.2 Les situations omises dans la modélisation

3.2.1 L'accord du verbe avec le groupe nominal sujet

Nous n'avons pas modélisé, dans la présente version du système, le cas d'accord du verbe avec le groupe nominal sujet où il n'y a qu'un groupe nominal sujet et il comporte un adverbe de quantité (comme dans ***Beaucoup de*** *ces enfants chantent dans la chorale*).
Nous ne possédons pas des catégories distinctes d'adverbes. Par conséquent, nous ne pouvons pas distinguer les différents types d'adverbe (par exemple, nous ne distinguons pas un adverbe de quantité d'un adverbe de mode).

Une autre restriction importante est celle imposée par l'analyseur syntaxique en ce qui concerne les groupes nominaux sujets liés par une conjonction de coordination. L'analyseur syntaxique, tel qu'il se présente actuellement, n'accepte pas les situations de liaison, par conjonction de coordination, entre deux éléments qui n'ont pas la même catégorie lexico-syntaxique : aucun arbre n'est construit pour ce genre de proposition.

Par exemple, cette restriction temporaire nous empêche de modéliser des situations comme *Marie et toi marcherez derrière.*
A cause d'une telle restriction, nous n'avons pas défini des règles pour modéliser les situations dont les groupes nominaux sujets appartiennent à des catégories lexicales différentes.

3.2.2 L'accord de l'adjectif

Nous avons omis, dans cette version du système, le traitement des adjectifs composés.
Nous réunissons les différentes classes d'adjectifs dans une seule catégorie : la catégorie **adjo**.
Nous ne faisons pas de distinction entre les adjectifs qualificatifs simples et composés ni entre différents groupes tels que couleurs, taille, forme, etc. Par conséquent, nous n'avons qu'un seule modèle d'accord pour un adjectif : l'accord en genre et en nombre avec le nom qu'il qualifie.

3.2.3 Les accords du participe passé

En vue de la simplification contenue dans l'arrêté de 1901, nous avons pris l'option de ne pas considérer les règles traditionnelles d'accord appliquées à un participe passé suivi d'un infinitif, et de traiter cette situation selon la simplification proposée dans l'arrêté.
Cependant deux problèmes persistent :

Problème 1: la multiplicité d'options correctes.
Exemple :
*La pianiste que j'ai entendu**e** jouer* et *La pianiste que j'ai entendu jouer* sont donc des propositions également correctes.

Problème 2: la possibilité de considérer comme faute d'accord une situation correcte ou, au contraire, de considérer comme correcte une situation de faute d'accord, à cause de l'absence d'informations sémantiques pour permettre de décider quelle règle d'accord à appliquer.

Exemple :
Si le système détecte une faute d'accord dans une proposition comme *Les fillettes que Marie a **vues** laver...* nous tomberons dans une incohérence, car cette phrase peut être correcte. Par contre, nous ne pouvons pas assurer, avec les informations disponibles au niveau lexical et syntaxique, que cette phrase est correcte : il nous faut connaître des informations supplémentaires pour décider à propos de

l'objet direct associé à *laver* (*les fillettes* sont-elles vraiment l'objet direct associé à *laver* ?).

Dans la mise en oeuvre de cette première version de vérification syntaxique, nous avons décidé de considérer comme correctes seulement les phrases du type

la pianiste que j'ai entendu jouer

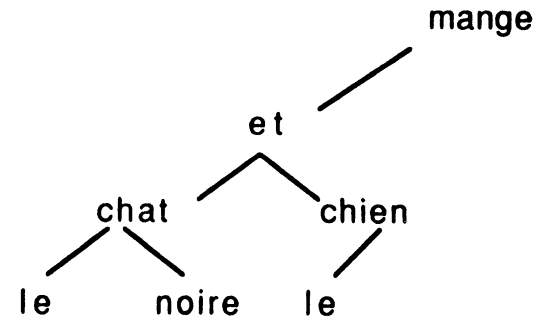
où le participe passé est invariable (au masculin singulier), et nous réalisons la correction selon ce critère.
Lorsque nous disposerons de traits syntaxico-sémantiques pour décider entre sujet et objet direct, nous pourrons avoir des règles pour modéliser les deux situations.
Les cas ayant besoin d'informations contextuelles, eux, sont encore un sujet d'étude à approfondir.

3.3 Un exemple de vérification/correction syntaxique

Considérons la phrase :

Le chat noire et le chien mange.

L'arbre syntaxique produit pour cette phrase a la forme :



Nous passons à la vérification syntaxique de l'arbre en ordre *postfixé* et nous analysons alors les paires :

1 le * chat → chat
2 chat * noire → *faute d'accord*
Option de correction : **noir**

Supposons que l'utilisateur accepte l'option de correction. Le mot *noire* est corrigé en *noir*, et la vérification syntaxique recommence :

1	le	*	chat	→	chat
2	chat	*	noir	→	chat
3	chat	*	et	→	et
4	le	*	chien	→	chien
5	et	*	chien	→	chien (nombre : pluriel)
6	chien(pluriel)	*	mange(singulier)	→	*faute d'accord*

Option de correction : **mangent**

De même, si l'utilisateur accepte l'option de correction, le mot *mange* est corrigé en *mangent*. La vérification syntaxique recommence et nous avons finalement une phrase considérée correcte.

Face aux changements introduits par une suite de corrections, nous avons pris l'option d'effectuer une nouvelle vérification syntaxique sur les phrases corrigées.

Une autre approche de vérification syntaxique peut être celle de la correction semi-automatique.
Par exemple, considérons le traitement de la phrase *le souris vert.*
L'analyse des noeuds *le* → *souris* détecte une faute d'accord. Supposons que la correction adoptée soit ***la*** *souris.*
La suite de l'analyse détecte une deuxième faute d'accord (*souris* → *vert*). Nous pouvons alors utiliser les mêmes valeurs adoptées pour la correction de l'article (sin fem) effectuant automatiquement la correction de *vert* en *verte.*
Ceci présente comme avantage une demande moins intense de l'utilisateur : même dans un système interactif, le fait d'être sollicité à des fréquentes intervensions peut s'avérer fastidieux voire décourageant à l'utilisation du système.

Le choix des variables et des valeurs à prendre en compte à l'occasion de la correction d'une faute d'accord est aussi un sujet à considérer. Nous effectuons des remarques à ce propos dans le paragraphe 4.1.

4 CORRECTION DE FAUTES D'ACCORD

4.1 Approches face à la correction d'une faute d'accord

La correction d'une faute d'accord peut être effectuée selon des différents critères. Par exemple, *le souris verte* peut être corrigé en *la souris verte* ou en *les souris vertes.*

Conformément à Veronis [VERONIS 88c], deux approches classiques peuvent être trouvées en ce qui concerne les critères à adopter face à la correction d'une faute d'accord :

a) certaines catégories morpho-syntaxiques sont privilégiées par rapport à d'autres et fournissent les traits morphologiques pour la correction.
Exemple : nous pouvons favoriser systématiquement le déterminant par rapport au nom. Dans ce cas, *les souris verte* serait corrigé en *les souris vertes.*

b) le choix d'une hypothèse de correction est basé sur le nombre d'erreurs entraînées. On compte le nombre d'erreur dans chacune des hypothèses de correction et on prend celle qui suppose le moins d'erreurs.
Exemple : pour *le petits chiens* on a les hypothèses *les petits chiens* et *le petit chien.* L'hypothèse choisie est *les petits chiens.*

Veronis essaie de modéliser cette situation en vue des phénomènes linguistiques. Il observe que :

- les marques morpho-syntaxiques muettes (comme le *s* du pluriel) sont très souvent oubliées, alors qu'elles sont rarement ajoutées là où elles doivent être absentes. Par conséquent, l'ajout d'un *s* est modélisé comme une opération plus coûteuse que son oubli.
Exemple : la correction adoptée pour *le chiens* devrait être *les chiens.*

- la prononciation joue un rôle important étant improbable que l'utilisateur se trompe en écrivant une forme prononcée de manière différente de la forme correcte.
Exemple : *le paysanne* serait corrigée en *la paysanne* puisque l'option *le paysan* présenterait un changement de prononciation.

- finalement, il suppose que les locuteurs francophones ne produisent normalement pas d'erreurs concernant la connaissance de traits morphologiques. Alors *chienne dressé* doit être corrigé en *chienne dressée* et non pas en *chien dressé.*

Ces observations amènent Veronis à caractériser trois classes d'erreurs de coût croissant :

1) la ***suppression*** d'une marque morphologique sans changement de prononciation (exemple : *belle* → *belles*) ;
2) l'***addition*** d'une marque sans changement de prononciation (exemple : *belles* → *belle*) ;
3) les erreurs qui modifient la prononciation (exemple : *vert* → *verte*).

Associant des vecteurs de coûts à chacune des hypothèses de correction, Veronis propose que, à la fin de l'analyse, l'option de correction la moins coûteuse soit choisie.

4.2 Paramètres adoptés dans le système

Dans le système développé, nous avons opté pour privilégier certaines catégories lexico-syntaxiques par rapport à d'autres, obtenant de ces catégories les traits morphologiques nécessaires à la génération d'une correction.

Cette méthode est aussi adoptée par Richard et Lapalme dans le système décrit dans le paragraphe 3.2.2 du chapitre 1, pour la correction de certains types d'accords.
Exemple : l'accord entre le nom et son déterminant est corrigé en prenant les traits morphologiques du déterminant. Si le genre du déterminant est variable (comme *les*), le genre du nom dominera s'il est connu, sinon le genre par défaut sera le masculin [RICHARD 86].

Nous décrivons ensuite la méthode utilisée dans le système développé.

La détection d'une faute d'accord amène à la préparation d'une correction : les paramètres de correction sont enregistrés dans un fichier de communication qui est utilisé pour engendrer les corrections.
Comme paramètres de correction nous retenons :
- la *position* du mot à corriger, dans la phrase ;
- le *mot* à corriger ;
- les *traits morphologiques* voulus pour la forme corrigée.

Par exemple, pour la correction de *noire* dans *le chat* ***noire*** nous retiendrons :

```
3/noire
mas
sin
```

Les traits morphologiques à retenir pour une correction dépendent des catégories lexico-syntaxiques des mots en cours d'analyse.
Exemple :
Pour engendrer un verbe nous retenons :
- le nombre et la personne associés à son sujet ;
- son temps et son mode d'origine dans la phrase.

De même, pour un adjectif, nous utilisons le genre et le nombre du nom qu'il qualifie.

Ayant comme objectif une formalisation du calcul des paramètres de génération, nous avons défini 6 types de traitement, qui aboutissent à 6 formes différentes d'obtention des traits morphologiques à engendrer. Les types proposés sont les suivants :

1 Correction d'un **verbo**, **verso**, **xeto** ou **xavo** :
Traits pris à gauche : personne, nombre
Traits pris du mot à corriger : mode, temps

2 Correction d'un **deto** ou **detpo** :
Traits pris à droite : genre, nombre

3 Correction d'un **adjo** à gauche d'un **gnomo** :
Traits pris à droite : genre, nombre

4 Correction d'un **adjo** ou **ppaso** à droite d'un **gnomo** ou correction d'un **gnomo** ou **adjo** à droite d'un **xeto** :
Traits pris à gauche : genre, nombre

5 Correction d'un participe passé (**ppaso**) invariant :
Constantes : ***mas sin***

6 Correction de deux adjectifs (**adjo**) liés par *et* :
Traits pris à gauche : genre, nombre
Traits pris à droite : genre

Remarque :
Dans ce cas, les formes correspondant aux deux adjectifs seront engendrées, pour chaque adjectif, avec les traits morphologiques de l'autre, pour présenter à l'utilisateur la globalité des options de correction.

4.3 Une analyse post-syntaxique en vue de la correction

A mesure que nous écrivons des règles de syntaxe, nous trouvons des situations exceptionnelles qui se rapportent, parfois, aux niveaux supérieurs de l'analyse syntaxique voire morphologique. Ce genre de problème est tout à fait naturel, en considérant que, après avoir préparé un niveau de définition, nous travaillons maintenant dans un niveau d'application.

La forme de traitement choisie pour les situations d'exception a été réalisée par une *unité de traitement* ***post-syntaxique.***
Cette unité permet de préparer, pour la correction, les arbres résultant de l'analyse syntaxique.

Evidemment, la solution adoptée à travers un traitement post-syntaxique n'est pas une solution définitive. Il sera intéressant de faire introduire les mesures de traitement post-syntaxique dans les étapes concernées de l'analyse syntaxique.
Toutefois, nous envisageons la détection/correction des erreurs, tandis que l'analyseur syntaxique de PILAF peut être utilisé dans d'autres domaines du traitement de la langue naturelle. Au lieu de procéder à une inclusion définitive des mesures de traitement post-syntaxique dans le système, nous préférons donc le faire postérieurement.

Nous incluons ici une description des problèmes les plus importants envisagés dans l'unité de traitement post-syntaxique avec une évaluation des solutions proposées.

4.3.1 La correction du verbe et de l'attribut dans une relative

En étudiant les arbres produits par l'analyseur syntaxique, nous avons observé des situations où il est parfois impossible de vérifier/corriger un verbe ou un auxiliaire par rapport à son groupe nominal sujet, de même qu'un attribut par rapport à son groupe nominal sujet.
Par exemple, il ne nous était pas pratique de corriger des phrases comme :

Il mange la soupe qui es chaud

où l'attribut *chaud* et l'auxiliaire être présentent des problèmes d'accord (*chaud* devrait être au féminin, l'auxiliaire devrait être à la troisième personne).

Nous illustrons cette situation avec l'arbre de la figure 5.1, correspondant à la phrase : *Le chat qui sont noir mange le poisson.*

Comme nous pouvons l'observer, le noeud associé au pronom relatif *qui* est le fils droit du nom auquel il se réfère (*chat*). Le verbe de la relative (*sont*) est son dépendant droit.

En parcourant l'arbre syntaxique en ordre *postfixé*, nous avons, pour l'exemple ci-dessus, la séquence suivante d'opérations :

1	le	*	chat	→	chat
2	sont	*	noir	→	sont
3	qui	*	sont	→	qui
4	chat	*	qui	→	chat
5	chat	*	mange	→	mange
6	le	*	poisson	→	poisson
7	mange	*	poisson	→	mange

Ainsi, la vérification de l'auxiliaire *être* avec son attribut (étape 2) et la vérification du pronom relatif avec l'auxiliaire *être* (étape 3) sont faites avant de connaître les traits morphologiques associés au gouverneur *chat* (étape 4).
Cette situation évoque les problèmes trouvés pendant la vérification/correction de l'auxiliaire *être* et de son attribut dans une relative, quand le sujet de la relative est placé avant le pronom relatif.

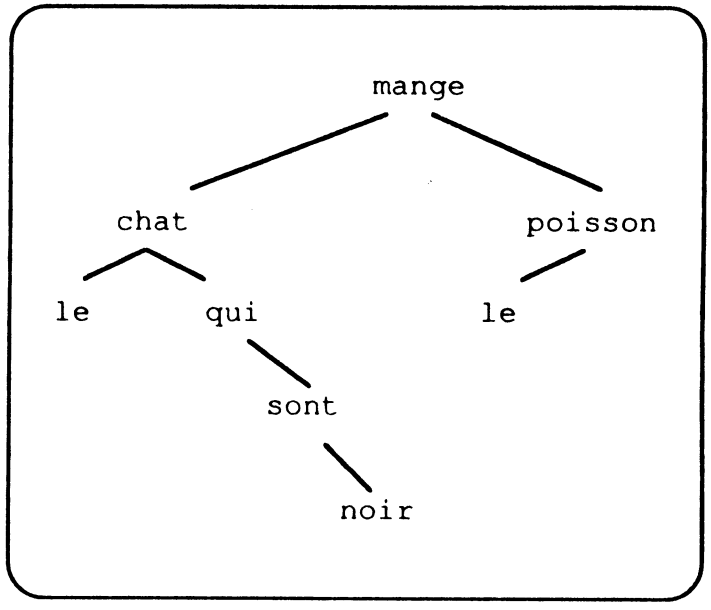


Figure 5.1

Le verbe et son attribut dans une relative

Bien sûr, au moment de l'analyse de la paire **(sont * noir)** dans l'étape 2, nous pouvons détecter une faute d'accord en nombre (*sont* est au pluriel et *noir* est au singulier), mais nous ne connaissons pas le genre du sujet de la relative, et une tentative de correction pourrait amener à une suggestion de correction incohérente.
Ce type de problème peut rester masqué, par exemple, dans l'analyse de la paire **(est * noir)** dans la phrase *Les chats qui est noir mangent le poisson*. Aucune faute ne serait détectée à ce niveau de l'analyse.

Pour résoudre ce genre de problème, il est nécessaire de connaître, avant l'analyse du verbe de la relative, les traits morphologiques associés à son sujet.

Nous avons envisagé différentes méthodes pour résoudre ce problème, et nous avons opté pour le rattachement au sous-arbre correspondant à la relative d'une copie du noeud qui la gouverne : nous avons incorporé

une copie du gouverneur de la relative comme dépendant gauche de son verbe.
Dans l'exemple, nous aurons donc comme fils gauche de *sont* une copie de son sujet *chat*.
La figure 5.2 présente l'arbre de l'exemple précédent adapté à la correction.

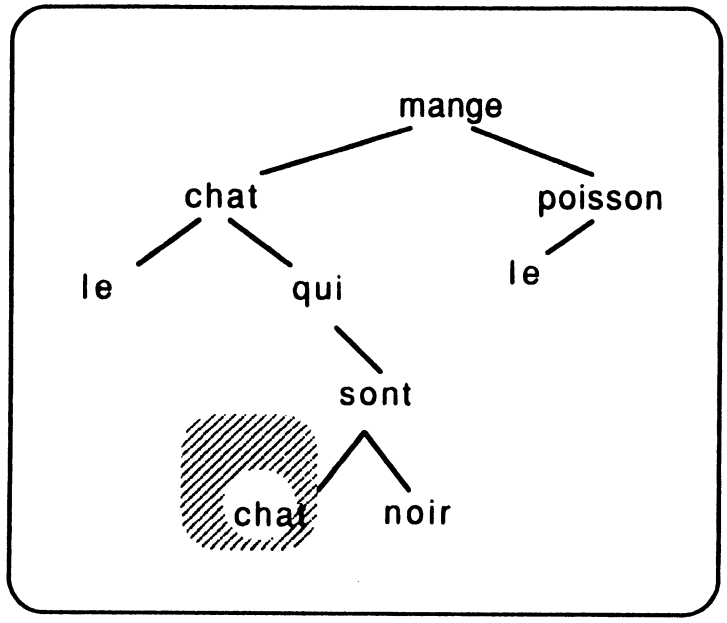


Figure 5.2

Arbre décoré adapté à la correction

Notre objectif, en rattachant cette copie de noeud à l'arbre de dépendances, est de permettre la connaissance du genre et du nombre du gouverneur de la relative avant d'analyser ses dépendants, sans modifier le parcours de l'arbre de dépendances ni les règles de dépendances utilisées par l'analyseur syntaxique.
Comme on peut l'observer sur le schéma d'arbre de la figure 5.2, la copie de noeud rattachée permet de :

1 Vérifier la paire de noeuds **(chat * sont)** et corriger immédiatement l'auxiliaire *être* en engendrant **est** ;
2 Garder les valeurs morphologiques associées à **chat** pour vérifier son attribut **noir**.

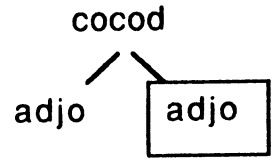
Le choix de cette option a permis de conserver le jeu de règles syntaxiques modélisées.

4.3.2 Les sous-arbres d'adjectifs gouvernés par la catégorie cocod

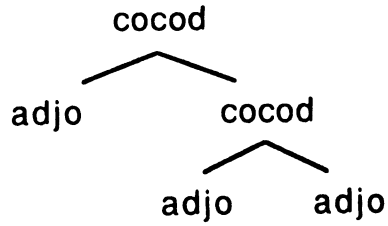
Dans une phrase, nous pouvons trouver une séquence d'adjectifs unis soit par la conjonction de coordination *et* soit par la virgule, se référant à un même nom.
Exemples :

Elle a choisi les poissons ***rouges, verts et gris****.*
La soupe est ***bonne et chaude****.*

L'analyse syntaxique de ce genre de constructions produit un sous-arbre binaire d'adjectifs gouverné par une conjonction de coordination, chaque noeud ayant exactement zéro ou deux fils :



Dans ce type de sous-arbre, au lieu d'un fils droit (signalé par le rectangle dans le dessin ci-dessus), nous pouvons trouver un sous-arbre dont la racine est une *cocod* (la virgule étant transformée en une conjonction de coordination).
Exemple :
Pour **rouges, verts et gris** dans la phrase *Elle a choisi les poissons rouges, verts et gris* nous aurons le sous-arbre :



La vérification syntaxique d'une telle arborescence présente des problèmes liés à l'accord.

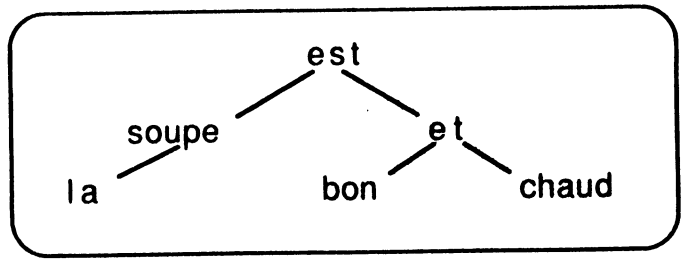


Figure 5.3

Sous-arbre d'adjectifs gouverné par un cocod

Analysons, par exemple, la phrase *La soupe est bon et chaud* (figure 5.3).

L'ordre des opérations de vérification est le suivant :

1	la	*	soupe	→	soupe	
2	soupe	*	est	→	est	
3	bon	*	et	→	et	
4	et	*	chaud	→	chaud	
5	est	*	et	→	est	→→→→ faute d'accord

Les dépendants de la conjonction de coordination *et* ne sont pas directement connectés à son gouverneur (l'auxiliaire *être*), ils seront donc analysés séparément (étapes 3 et 4).
Au moment de l'analyse du sous-arbre gouverné par la conjonction de coordination, nous ne connaissons pas les valeurs morphologiques associées à son gouverneur (l'auxiliaire *être*) et son sujet (*la soupe*), alors il est impossible de réaliser une détection/correction complète des fautes d'accord.
Nous pouvons, sans doute, détecter les cas où les dépendants gauche et droit de la conjonction de coordination n'ont pas le même genre ou nombre, comme ***bon*** et ***chaude*** dans la phrase *La soupe est bon et chaude*. Mais il nous est impossible de détecter complètement les erreurs d'accord dans *La soupe est* ***bon et chaud***, au moment de l'analyse du sous-arbre gouverné par *et*. C'est seulement au moment de l'analyse de la paire *auxiliaire être → son fils droit* que nous trouverons une faute d'accord.
Ainsi nous ne corrigerons pas le mot ***chaud***, car il sera considéré correct au moment de l'analyse du sous-arbre gouverné par *et*.

Pour résoudre ce problème, nous avons examiné deux possibilités :

1 modifier l'ordre de la vérification des noeuds de l'arbre de dépendances (ce qui obligerait à modifier les règles de vérification syntaxique) ;

2 modifier l'arbre produit par l'analyse syntaxique.

Changement de l'ordre d'application

Pour régler le problème des adjectifs, nous avons envisagé, premièrement, la possibilité de modifier l'ordre de vérification syntaxique. C'est-à-dire que nous pourrions analyser les sous-arbres

d'adjectifs gouvernés par un *cocod* en stockant tous les adjectifs pour une vérification future qui serait alors réalisée en série.

Prenons l'exemple *La soupe est bon et chaude.*
La vérification d'accord serait effectuée seulement au moment de l'analyse de la paire **est → (bon, chaude)**.

Cette possibilité a été jugée impropre pour deux raisons :

a. elle augmenterait considérablement le code des règles syntaxiques, en opposition aux restrictions de taille du code source ;

b. elle obligerait à modifier l'algorithme de vérification de l'arbre syntaxique, en introduisant une structure de données parallèle aux listes existantes : une pile de résolution des sous-arbres d'adjectifs unis par des *cocod.*

La restructuration de l'arbre syntaxique

La deuxième alternative proposée pour résoudre ce problème consiste à effectuer une transformation de l'arbre syntaxique en déplaçant tous les adjectifs composant un sous-arbre dont le gouverneur est la conjonction de coordination (*cocod*).
Le déplacement, fait à travers un algorithme récursif, a pour objectif de transformer les adjectifs en fils droits de leur gouverneur réel (le nom, le verbe, etc).

Pour la phrase *La soupe est bon et chaude*, le nouvel arbre syntaxique aurait la forme présentée dans la figure 5.4.

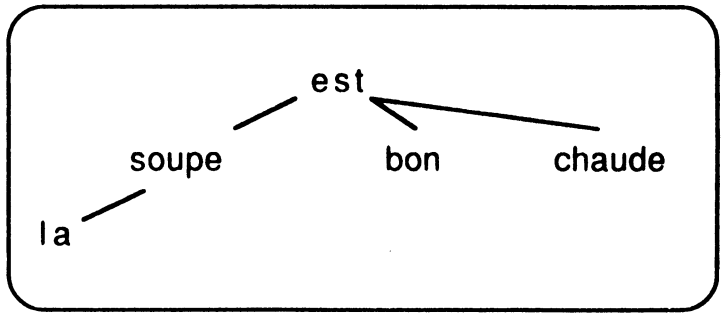


Figure 5.4

Sous-arbre d'adjectifs : nouvelle configuration

Par cette méthode, nous transformons la vérification syntaxique d'un sous-arbre d'adjectifs liés par une conjonction de coordination (*cocod*) en une série de vérifications du type **x * adjo**, où **x** est la catégorie lexico-syntaxique du gouverneur de la *cocod*.

Les règles syntaxiques concernant les liaisons du type **x * adjo** sont déjà présentes dans le système : aucune règle additionnelle n'est exigée.

Nous observons que l'élimination de la conjonction de coordination, dans ce cas, ne présente pas de problèmes pour la correction des fautes d'accord.
Le traitement de la conjonction de coordination effectué par l'analyseur syntaxique est suffisant pour garantir qu'un sous-arbre complet a été trouvé et attaché à son gouverneur.

La transformation proposée, évidemment, s'applique bien à la détection/correction des fautes d'accord, mais elle n'est pas valable pour toutes les autres applications et analyses possibles à partir d'un arbre syntaxique, qui doivent conserver l'arbre syntaxique original.

L'alternative choisie

Nous avons choisi cette dernière alternative - la restructuration de l'arbre syntaxique - pour résoudre le problème de la détection/correction des fautes d'accord dans un sous-arbre d'adjectifs gouverné par une conjonction de coordination du type *et* ou virgule.
Nous l'avons mise en oeuvre au niveau du traitement post-syntaxique, dans un premier temps. Une étude approfondie pourra permettre de décider si ce choix de modification de l'analyseur syntaxique couvre la plupart des applications.

5 LA SUITE D'UNE CORRECTION

Nous corrigeons un mot en obtenant sa base morphologique et en engendrant les formes dérivées de cette base qui se caractérisent par un certain ensemble de traits : les paramètres de génération.
Exemple : pour corriger *noire* dans *le chat noire* nous mettons la base **noir** au masculin singulier et engendrons *noir* .

A l'affichage d'un ensemble de corrections, l'utilisateur peut :
- soit choisir une option de correction pour son erreur ;
- soit invalider cette tentative de correction et passer à l'étude de l'arbre suivant de la même phrase.

Si une option de correction est choisie :

1 la phrase est modifiée ;
2 si nécessaire, l'arbre en cours d'analyse est modifié ;
3 tous les arbres produits pour la phrase corrigée sont éliminés, à l'exception de l'arbre en cours d'analyse (nous supposons que, si l'utilisateur a manifesté son choix de correction sur un certain arbre, les autres arbres peuvent être écartés) ;
4 une nouvelle vérification syntaxique valide la correction effectuée.

Ainsi, le système ne refait ni l'analyse morphologique ni l'analyse syntaxique d'une phrase corrigée : il utilise les structures de données existantes et les modifie.

6 INFORMATIONS NECESSAIRES A UNE MODELISATION EXHAUSTIVE DES REGLES D'ACCORD

Les problèmes rencontrés au cours de la préparation d'un vérificateur syntaxique ont mené à une série de réflexions en ce qui concerne les informations nécessaires au niveau lexical et syntaxique de vérification.

Premièrement, les catégories disponibles au niveau syntaxique doivent permettre la définition d'une grammaire de dépendances suffisamment étendue pour permettre de réaliser une détection/vérification syntaxique.
Deuxièmement, la vérification syntaxique même pour un sous-ensemble restreint de la langue exige la connaissance d'informations au niveau lexical et syntaxique, comme celles qui concernent le groupe nominal, le verbe et les compléments. Si cette vérification doit comprendre les règles d'accord, il est nécessaire de disposer, en plus, d'informations au niveau sémantique (les traits sémantiques).

Certaines catégories lexicales présentent une diversification beaucoup plus accentuée que celle que l'on considère normalement, ce qui rend plus difficile le travail de détermination des groupes de propriétés syntaxiques. Par exemple, nous pouvons citer les verbes. Il est nécessaire d'associer les verbes à des catégories spécifiques en vue d'une vérification syntaxique. Les catégories traditionnelles "transitif direct", "transitif indirect", "intransitif" ne sont pas suffisantes pour une vérification syntaxique automatique. Il devient obligatoire de proposer de nouvelles classifications en tenant compte des catégories qui peuvent suivre le verbe et de la possibilité d'accepter une complétive. Le système de Lapalme et Lacouture [LACOUTURE 86,

LACOUTURE 88] utilise, par exemple, l'attribut *compl* pour signaler qu'un verbe accepte une complétive.

6.1 Quelques propositions

En tenant compte des considérations concernant la vérification syntaxique des règles d'accord, nous proposons une particularisation des informations qui se réfèrent aux *substantifs*, aux *verbes* et aux *adverbes*. Nous proposons aussi une dissociation dans les catégories *pronom relatif* et *conjonction de coordination*.

Les substantifs doivent être analysés en vue de la capacité d'assumer le rôle de sujet ou d'objet direct associé à un certain verbe. Cette capacité représente un trait sémantique important, dont une analyse superficielle ne peut pas résoudre le problème posé avec la correction des fautes d'accord.
L'utilisation d'un grand dictionnaire exige un travail détaillé de classification, qui est aussi inséré dans une étude en vue de l'analyse sémantique, pour une vérification/correction sémantique, dans le projet PILAF.

I Informations associées aux verbes

La classification actuellement utilisée dans PILAF associe aux verbes des modèles de conjugaison. Si on envisage la vérification syntaxique et la correction de fautes d'accord, il est nécessaire de connaître, en plus :

a) *la transitivité du verbe* - nous devons savoir si le verbe est intransitif (comme *dormir* dans *Elle **dort** difficilement*), transitif direct (comme *manger* dans *Il a **mangé la pomme***), transitif indirect (comme *succéder* dans *Ce jeune homme **succédera à son père***) ou transitif direct et indirect en même temps (comme *offrir* dans *Il **offre des fleurs à sa mère***), pouvant admettre un complément d'objet direct, un complément d'objet indirect ou les deux.

b) *la possibilité d'accepter une complétive* - certains verbes peuvent accepter une proposition subordonnée conjonctive (introduite par une conjonction de subordination). Certaines conjonctives sont complément d'objet direct (les complétives, comme dans *Je pense **qu'il viendra.***). Il nous faut alors savoir si le verbe accepte ou n'accepte pas une conjonctive comme complément d'objet direct.

c) *la possibilité d'accepter un infinitif comme complément d'objet direct* - certains verbes peuvent accepter un infinitif comme complément d'objet direct (comme *aimer* dans *Il **aime lire*** équivalent à *Il aime la lecture*).

d) *la possibilité de fonctionner comme auxiliaire* - certains verbes, appelés parfois *semi-auxiliaires*, construits avec un infinitif peuvent fonctionner comme des auxiliaires. Parmi les principaux nous avons : *aller*, *devoir*, *faillir*, *manquer*, *laisser*, *paraître*, *sembler*, *pouvoir*, etc (exemple : *Elle **peut lire***).

e) *la situation "verbe d'état"* - certains verbes auxiliaires sont appelés verbes d'état : ils n'évoquent pas une action, même si le sujet est animé (exemple : *être*, *sembler*, *devenir*, etc, comme dans *Elle **devient de plus en plus belle***). Ces verbes acceptent un attribut qui se réfère au sujet et qui suit des règles d'accord. Alors, il est nécessaire de savoir si le verbe est ou non un verbe d'état.

II Nouvelles classifications concernant les adverbes, les conjonctions de coordination et les pronoms relatifs

Pour la vérification des règles d'accord, il faut connaître la nature de certains adverbes. Les adverbes de quantité comme, par exemple, ***beaucoup***, y mériteront un traitement spécifique .

Comme conjonctions de coordination nous groupons actuellement *et*, *ni*, *ou*, *mais*, *donc*, *car*, *or*, la virgule, etc. Mais il est évident que le groupement actuel n'est pas adapté à une vérification/correction de l'accord. Pour résoudre ce problème, nous pouvons avoir une nouvelle classification comme par exemple : conjonction de coordination accordée, conjonction de coordination simple.

Les pronoms relatifs présentent, eux aussi, un problème concernant la vérification d'accord : certains pronoms relatifs (exemple : *dont* dans *la fille dont je me souviens*) ne réalisent pas d'accord, tandis que d'autres (comme *que* dans *la réunion que j'ai provoquée*) exigent l'accord. Concernant ce problème, il serait intéressant de classifier les pronoms relatifs conformément à une vérification d'accord.

CHAPITRE 6

CONSIDERATIONS A PROPOS D'UN SYSTEME DE TRAITEMENT D'ERREURS AU NIVEAU LEXICO-SYNTAXIQUE

Après la présentation de la maquette de correction lexicale et syntaxique réalisée avec les outils de PILAF, nous discutons, ensuite, de considérations concernant le développement d'un système de traitement d'erreurs au niveau lexico-syntaxique, en ce qui concerne la méthode d'analyse lexico-syntaxique employée et les mécanismes de traitement d'erreurs qui y sont incorporés.

La maquette proposée accepte comme entrée une phrase écrite en français et effectue une analyse morphologique des mots qui la composent.
Les mots dont l'analyse morphologique ne produit pas de résultat sont à l'origine d'un processus de correction.
La phrase morphologiquement correcte est analysée au niveau syntaxique. Nous construisons les structures de dépendances associées à cette phrase et nous opérons ensuite une vérification syntaxique des structures de dépendances engendrées. Si des fautes d'accord sont détectées, nous les corrigeons par la méthode de génération morphologique.

Finalement, nous présentons comme suggestion l'emploi du concept de "Tree Adjoining Grammars" pour l'analyse et la vérification syntaxique d'une phrase.

1 *CORSYNT* : UNE MAQUETTE DE DETECTION/CORRECTION LEXICO-SYNTAXIQUE

Nous décrivons ici la maquette de détection/correction d'erreurs au niveau lexico-syntaxique développée avec les outils de PILAF.
Cette maquette se sert des analyseurs morphologique et syntaxique de PILAF, aussi bien que des modules de correction par clé squelette et par phonétique (au niveau lexical), de vérification syntaxique et de correction de fautes d'accord (au niveau syntaxique).
La figure I-1, présentée dans l'introduction de ce rapport, montre l'organisation adoptée pour les composants de la maquette CORSYNT.

1.1 L'organisation fonctionnelle

Nous présentons une phrase à l'analyseur morphologique de PILAF. A chaque mot considéré correct, nous signalons une entrée dans une structure de données morphologiques qui servira à l'étape d'analyse syntaxique.

Un mot dont l'analyse morphologique ne donne pas de résultats est considéré faux et conduit à une correction au niveau lexical.

L'unité de correction lexicale (CORRLEX) permet à l'utilisateur le choix parmi trois formes de correction :

A - la correction par une *nouvelle saisie du mot* (l'utilisateur peut substituer au mot erroné son correspondant correct, s'il connaît la correction) ;
Cette modalité de correction peut paraître d'une simplicité extrême mais, en fait, elle est assez souvent utilisée. Une fois que l'erreur est signalée, l'utilisateur peut lui-même remplacer le mot faux par le mot correct.

B - la correction par *clé squelette* (CORRCLE) ;
Le module de correction d'erreurs par clé squelette de PILAF est employé dans CORSYNT pour proposer des corrections aux mots faux.
La méthode de correction par clé squelette est spécialement applicable aux erreurs d'édition, mais sa rapidité de réponse nous a amené à l'adopter comme la première méthode de correction automatique appelée dans CORSYNT, indépendamment de la nature de l'erreur à corriger.
L'ensemble des suggestions engendrées est proposé à l'utilisateur, qui manifeste son choix. Le mot choisi remplace le mot faux et nous continuons l'analyse morphologique.

C - La correction par *phonétique* (CORPHONE).
Si aucune des propositions de correction présentées par la méthode de la clé squelette n'a été choisie et que l'utilisateur demande une deuxième correction automatique, nous y utilisons le module de correction par la phonétique.

La correction adoptée par l'utilisateur est analysée, ce qui peut déclencher un nouveau processus de correction en cas de faute (bien entendu, la correction par clé squelette ou par phonétique ne propose pas de mots faux, mais l'utilisateur lui-même peut éventuellement le faire).

Une fois que l'on dispose des renseignements morphologiques à propos de chacun des mots de la phrase, on passe à l'analyse syntaxique.

Cette analyse peut produire comme résultat aucune, une ou plusieurs structures de dépendances associées à la phrase.

Si l'analyse syntaxique produit des structures de dépendances, on passe à une vérification de ces structures. Sinon, on prend la prochaine phrase à analyser.

La vérification syntaxique opère sur une structure de dépendances à la fois. Si cette structure est correcte, on passe à la structure suivante, jusqu'à la vérification syntaxique de toutes les structures de dépendances engendrées.

Une structure de dépendances est considérée incorrecte par la vérification syntaxique si elle présente des erreurs de "style" ou des erreurs d'accord.
Les erreurs de "style", comme l'inversion de pronoms dans une phrase ou l'omission du déterminant associé à un mot, sont signalées à l'utilisateur. Aucune modification de la phrase n'est proposée, dans ce cas, par le système.
Les fautes d'accord détectées (y compris les fautes d'accord du participe passé) sont corrigées par le module de correction de fautes d'accord, qui tient compte des traits morphologiques envisagés pour le mot faux et engendre par génération des correspondants corrects.

Une correction au niveau syntaxique peut altérer les valeurs morphologiques associées à un mot voire sa catégorie lexicale. Elle peut alors produire des changements dans l'ensemble des structures de dépendances.
Par exemple, soit la correction de ***fermes*** dans *la belle* ***fermes.*** Parmi les options de correction, nous avons :

a	ferme	substantif	féminin
b	ferme	verbe	troisième personne, singulier, présent de l'indicatif
c	ferme	verbe	troisième personne, singulier, présent du subjonctif
d	ferme	adjectif	féminin, masculin, singulier

Si l'utilisateur manifeste son choix par une de ces options, toutes les structures de dépendances créées avec les autres trois options peuvent être éliminées.
De cette manière, dans le cas où l'utilisateur adopte une correction pour une certaine structure, toutes les autres structures de dépendances concernées sont supprimées, et on poursuit le travail avec la seule structure résultante de la correction.

Une correction syntaxique amène à une nouvelle vérification syntaxique. Ce processus se répète jusqu'à ce que la phrase soit considérée correcte ou que l'utilisateur détermine son interruption.
L'utilisateur peut toujours interrompre la vérification/correction syntaxique, par le moyen d'une option parmi les propositions de correction.

1.2 L'ordre d'application des méthodes de correction

Comme nous l'avons vu (paragraphe précédent), notre maquette utilise deux des modules de correction d'erreurs lexicales de PILAF : le module de correction par clé squelette et le module de correction par phonétique.

En considérant la rapidité de réponse de la méthode de correction par clé squelette, nous avons adopté l'ordre d'application :

utilisateur → clé squelette → phonétique *(A)*

parmi les méthodes utilisées.

Cette forme d'organisation s'applique effectivement aux systèmes censés traiter des textes contenant plutôt des erreurs de performance (exemple : des erreurs d'édition) que d'autres types d'erreurs.
Supposons que le système soit censé traiter des textes contenant un grand nombre d'erreurs de compétence et peu d'erreurs de performance.
L'ordre d'application des méthodes de correction doit alors tenir compte de cette situation et nous devons privilégier la correction par la phonétique en obtenant :

utilisateur → phonétique → clé squelette *(B)*

Une autre forme d'organisation possible serait celle tenant compte, dynamiquement, de la nature de l'erreur à corriger. C'est-à-dire que l'organisation du système serait adaptée aux performances et aux compétences de l'utilisateur [COURTIN 88d].
A l'analyse d'un sous-ensemble représentatif des mots erronés trouvés dans le texte, le système peut déterminer la nature des erreurs à corriger et configurer automatiquement son organisation : soit l'organisation du type A (privilégiant les erreurs de performance) soit l'organisation du type B (privilégiant les erreurs de compétence).

Une dernière considération à propos de l'ordre d'application des méthodes de correction concerne la conception d'un système multi-processeur de traitement d'erreurs.
Ayant disponibles plusieurs processeurs, nous pouvons déclencher l'exécution parallèle de nos algorithmes de correction, présentant à

l'utilisateur les résultats à la mesure que ces résultats sont obtenus par les processus parallèles de correction.
La suite de l'analyse morphologique pourrait également être réalisée, dans la plupart des cas, en parallèle avec les corrections lexicales.

1.3 Le système

Notre maquette a été construite sur un ordinateur compatible PC AT. Nous avons repris le programme d'analyse morpho-syntaxique de PILAF (programme PILMS, en TURBOPASCAL) et, à partir de ce programme, nous avons construit CORSYNT.
Pour l'analyse morpho-syntaxique et le traitement d'erreurs, nous avons utilisé le dictionnaire expérimental de PILAF (688 bases morphologiques).

Une erreur pendant l'analyse morphologique est à l'origine d'un processus de correction au niveau lexical. Les modules de correction utilisés sont programmés en TURBOPASCAL.

Après l'analyse syntaxique, notre programme appelle directement le module de vérification syntaxique (écrit en TURBOPROLOG).
La vérification syntaxique terminée, la communication avec le programme CORSYNT se fait au moyen d'un fichier contenant les paramètres associés à une correction. C'est dans CORSYNT que nous déclenchons, alors, la correction d'une faute d'accord.
Le module de correction de fautes d'accord par génération est écrit en TURBOPASCAL. Il contient des procédures extraites du module de génération de PILAF.

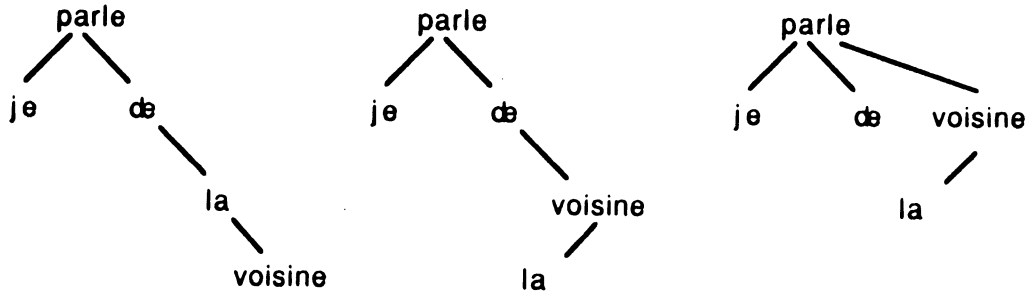
2 UN SYSTEME DE TRAITEMENT D'ERREURS AU NIVEAU LEXICO-SYNTAXIQUE

2.1 Observations concernant l'étape d'analyse syntaxique

Les formalismes d'une grammaire de dépendances comme celle employée par l'analyseur syntaxique de PILAF ont comme avantage la simplicité, la mise en oeuvre facile et une grande efficacité. Par contre, le seul contrôle contextuel des relations permis est celui des positions relatives des dépendants sous un gouverneur donné [COURTIN 86, COURTIN 89c].

La présence d'une règle de dépendance autorise la construction d'un arbre, mais il n'y a aucun moyen d'exprimer qu'un dépendant est obligatoire. On permet la construction de structures incomplètes (ce qui

peut être favorable, dès qu'il est possible d'attribuer une interprétation à une telle structure), mais on produit aussi un nombre de structures incohérentes. Par exemple, la phrase *je parle de la voisine* produit les structures de dépendances :



Les structures produites par l'analyseur de PILAF ne sont pas toujours adaptées à l'application de règles de vérification syntaxique. Comme nous l'avons observé dans le chapitre 4, paragraphe 4.2, quelques modifications ont été nécessaires pour permettre d'appliquer des règles de vérification syntaxique dans des arbres contenant des adjectifs liés par une conjonction de coordination *et*, etc.

2.2 Une proposition d'application des "Tree Adjoining Grammars" à l'analyse syntaxique en vue d'une vérification syntaxique

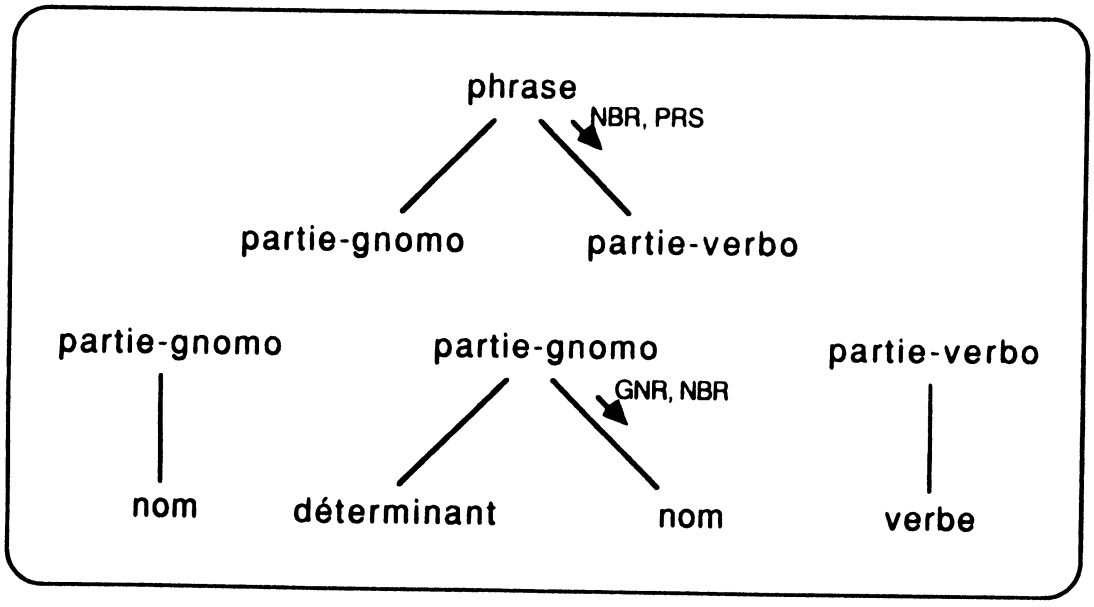


Figure 6.1

Une TAG : exemple simplifié

Le formalisme des "Tree Adjoining Grammars" (TAGs) a été introduit par A. Joshi, L. Levy et M. Takahashy en 1975. Ce formalisme a pour composant fondamental un ensemble fini d'arbres élémentaires dont chacun peut être considéré comme une structure linguistique minime.
La figure 6.1 présente comme exemple une TAG très simplifiée.

A. Abeillé, Y. Schabes, K. Vijay-Shanker et A. Joshi [ABEILLE 88, SCHABES 88, VIJAY-SHANKER 88] ont publié des travaux récents réalisés avec les TAGs, parmi lesquels une proposition d'analyse syntaxique du français, que nous reprenons ici.

2.2.1 Généralités à propos des TAGs

Les "Tree Adjoining Grammars", au contraire des autres procédés grammaticaux utilisés dans le traitement d'une langue naturelle, sont un système de réécriture basé sur les arbres [VIJAY-SHANKER 88].
Une TAG factorise récursivité et dépendances en produisant un ensemble fini d'arbres élémentaires. Ces arbres élémentaires correspondent à des structures linguistiques minimes qui localisent des dépendances comme l'accord (voir figure 6.1).

Une TAG comprend deux types d'arbres élémentaires, qui sont des unités syntaxiques :

- les arbres complets α, avec une racine et des noeuds pré-terminaux sur toutes les feuilles, correspondant à des "frontières" (ces arbres correspondent, en gros, à des phrases simples) et

- les arbres auxiliaires β, avec un et un seul noeud feuille étiqueté par un symbole non-terminal de la même catégorie de sa racine (ces arbres correspondent aux constructions récursives de base).

Les phrases sont engendrées par la composition d'un arbre complet, avec un nombre quelconque d'arbres auxiliaires, par le moyen d'une opération qu'on appelera *association*.

L'opération d'association insère un arbre auxiliaire dans le noeud correspondant d'un arbre élémentaire ou d'un arbre dérivé. On peut y associer récursivement les arbres auxiliaires.
Cette opération permet l'insertion d'une structure complète dans un noeud à l'intérieur d'une autre structure complète.

Avec une phrase comme unité linguistique, Abeillé [ABEILLE 88] a défini 40 structures de base pour le français, parmi lesquelles 12 se rapportent aux verbes prenant des compléments essentiels du type

nominal et 28 se rapportent aux verbes prenant des phrases comme compléments.
La taille de la grammaire entière estimée pour le français est d'environ 1200 arbres.

Ceci simplifie beaucoup l'organisation générale de la grammaire. Comme le cite Abeillé [ABEILLE 88], toutes les structures sont établies en termes de structures de surface, et il y a une relation directe entre les informations lexicales et les arbres.

2.2.2 TAGs et traitement d'erreurs syntaxiques

Au point de vue du traitement d'erreurs syntaxiques, nous pouvons effectuer deux remarques en ce qui concerne l'applicabilité des TAGs :

a) Même si la quantité d'informations exigée au niveau lexical est beaucoup plus significative que celle normalement manipulée par une analyse morphologique [SCHABES 88], ceci ne représente pas une perte. Comme nous l'avons déjà observé dans le paragraphe 6.1 du chapitre 5, une détection/correction syntaxique puissante exigera des informations beaucoup plus détaillées au niveau lexical.

b) Nous pouvons connaître d'avance les structures concernant des vérifications d'accord. Alors, nous pouvons signaler les fautes d'accord et en effectuer un traitement pendant l'analyse syntaxique, sans être obligé de réaliser la vérification syntaxique en une étape suivante, isolée.

Evidemment, l'utilisation d'une TAG associée à l'analyseur syntaxique de PILAF (à la place d'une grammaire de dépendances) doit être subordonnée une évaluation globale des applications envisagées avec cet analyseur.

Ce que nous pouvons remarquer c'est que le traitement d'erreurs au niveau syntaxique, comme on l'envisage dans CORSYNT, paraît une application pour laquelle on peut se servir du concept de TAG pour mettre en oeuvre un mécanisme permettant l'analyse, la détection et la correction syntaxique.

CONCLUSION

Nous avons étudié le problème du traitement d'erreurs dans un texte écrit en langue naturelle, spécifiquement la langue française.
Nous nous sommes intéressés aux erreurs aux niveaux lexical et syntaxique, et nous avons présenté une analyse concernant les erreurs pouvant apparaître, à ces deux niveaux, dans un texte.

Dans l'environnement PILAF de traitement de la langue naturelle, nous avons proposé et développé des outils de traitement d'erreurs : un module de correction d'erreurs lexicales par la phonétique et des prototypes de vérification syntaxique et de correction de fautes d'accord (réunis dans la maquette *CORSYNT*).

Le module PHONE de correction par la phonétique a été mis au point et testé avec un dictionnaire de taille réelle (environ 220000 formes).
Pour effectuer la transduction phonétique d'un mot, nous avons utilisé :
- un dictionnaire phonétique avec plus de mille p-graphèmes ;
- 235 modèles phonétiques ;
- 213 codes dérivation ;
- 5 règles.

Considérons sous deux aspects différents la conclusion de ce travail.
Nous pouvons examiner les possibilités d'une continuation, vers l'application de l'étude réalisée, en poursuivant deux lignes d'actions différentes.

La première, qui intéresse personnellement l'auteur, se rapporte à l'étude de la langue portugaise employée au Brésil. Il existe très peu de travaux concernant le traitement d'erreurs dans un texte écrit en portugais, alors que cette langue est parlée actuellement par plus de 150 millions de personnes [ALMANAQUE ABRIL 89].
Nous avons déjà réalisé une expérience (exposée dans le chapitre 2, paragraphe 4) en nous servant des analyseurs de PILAF pour l'analyse morphologique et syntaxique de la langue portugaise. Cette réalisation a été suivie d'une étude en vue de l'application au portugais des méthodes de traitement d'erreurs mises en oeuvre pour la langue française dans l'environnement PILAF.
L'applicabilité au portugais des méthodes de traitement d'erreurs disponibles permet d'envisager une continuation des travaux réalisés : la modélisation de la langue portugaise, qui permettrait d'utiliser les modules développés dans PILAF.

La deuxième ligne d'étude se rapporte à l'utilisation à grande échelle des modules de traitement d'erreurs proposés.
Analysons d'abord la mise en oeuvre d'une méthode de correction lexicale par la phonétique.

En observant les temps de réponse de l'étape de vérification des graphies engendrées, nous avons pu remarquer le rapport entre ces temps et les accès disque effectués. Ceci nous permet d'envisager une étude en vue de la réorganisation des fichiers du dictionnaire de formes, pour accélérer la vérification.

Nous avons également considéré le problème de la transcription phonétique des mots d'origine étrangère (exemple : *rewriter*). Dans la version mise en oeuvre, nous n'avons pas concentré nos efforts sur cette catégorie de mots, même s'ils représentent un nombre significatif de cas dans nos dictionnaires. Mais il peut se montrer souhaitable d'approfondir leur étude dans le cadre de la correction par phonétique.

Comme dernière proposition, nous pourrions reprendre ici le concept d'une "clé phonétique" déjà abordé dans [VERONIS 88c]. Le caractère multi-algorithmique d'application envisagé pour les outils de correction d'erreurs de PILAF permet de penser à une clé mixte "squelette-phonétique" qui pourrait autoriser l'accès combiné (soit du type squelette soit du type phonétique) au dictionnaire de formes de PILAF.

Considérons ensuite les modules de vérification et de correction syntaxique de la maquette *CORSYNT*.
Nous avons utilisé, pour la construction de ces modules, le dictionnaire expérimental de PILAF (688 bases). Mais il est évident que, en utilisant le dictionnaire complet de PILAF (environ 26000 bases morphologiques), nous serons obligés d'amplifier la couverture de la grammaire employée et, en conséquence, d'effectuer des modifications concernant les règles de vérification syntaxique.
Les conjonctions de coordination constituent un exemple de catégorie modélisée très strictement dans la présente version (nous n'avons considéré que la conjonction *et*).

L'amplification de couverture de la grammaire doit être insérée dans une étude à approfondir, basée sur les limites d'un ensemble fondamental de la langue (voir chapitre 4). Disposant de cette grammaire étendue, nous pourrons alors construire un ensemble de règles de vérification syntaxique approprié au dictionnaire complet de PILAF.

Il est aussi envisageable de disposer de quelques renseignements sémantiques pour corriger des situations de faute d'accord sur le participe passé suivi d'un infinitif. Les études menés par D. Genthial dans l'équipe TRILAN visent à produire un analyseur syntaxique capable de tenir compte de quelques informations sémantiques, parmi lesquelles les traits nécessaires à la correction de ce genre d'erreur.

Finalement, comme nous l'avons déjà proposé dans le chapitre 6, il est possible d'employer d'autres conceptions telles que celle des "Tree Adjoining Grammars" pour réaliser dans un seul module l'analyse et la vérification syntaxique.

La possibilité d'insérer le processus de vérification syntaxique à l'intérieur de l'analyseur syntaxique de PILAF tel qu'il est défini actuellement n'a pas été envisagée, puisque les sous-arbres produits par cet analyseur ne sont pas définitifs avant la fin de l'analyse complète de la phrase. Cet analyseur fonctionne récursivement avec retour arrière, et les sous-arbres produits à un niveau pourront être invalidés par un niveau supérieur.

Récemment, des accords de coopération ont été établis entre l'équipe TRILAN et le Groupe d'Etudes pour la Traduction Automatique (GETA). Ces accords visent à la construction d'un lémmatiseur pouvant s'intégrer comme accessoire de bureau sur un Macintosh.
A cette occasion, ils devront étudier une nouvelle structure pour le dictionnaire qui soit à la fois plus performante et plus compacte.
Cette nouvelle structure sera reprise dans nos outils.

REFERENCES

BIBLIOGRAPHIQUES

[ABEILLE 88]
ABEILLE A. *Parsing French with Tree Adjoining Grammar : some linguistic accounts.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 7-12.

[ADAMSON 74]
ADAMSON G., BOREHAM J. *The use of an association measure based on character structure to identify semantically related pairs of words and document titles.* Information Storage and Retrieval Vol. 10 nº7, 1974. p. 253-260.

[AHO 76]
AHO A. V. *Bounds on the complexity of the longest common subsequence problem.* Journal of the ACM, Vol. 23 nº1, Janvier 1976. p. 1-12.

[ALMANAQUE ABRIL 89]
ALMANAQUE ABRIL 1989. Editions ABRIL, Brésil, 1989. 834 p.

[ANGELL 83]
ANGELL R., FREUND G., WILLET P. *Automatic spelling correction using a trigram similarity measure.* Information Processing & Management Vol. 19 nº 4, 1983. p. 255-261.

[ANTONIADIS 88]
ANTONIADIS G., LALLICH-BOIDIN G., POLITY Y., ROUAULT J. *A french text recognition mode for information retrieval system.* 11th International Conference on Research & Development in Information Retrieval, Grenoble, Juin 1988.

[BAINIER 89]
BAINIER F. *Détection et correction d'erreurs orthographiques.* Mémoire de stage. I.U.T. Informatique de Grenoble, Juin 1989. 72 p.

[BEAR 86]
BEAR J. *A morphological recognizer with syntactic and phonological rules.* 11th International Conference on Computational Linguistics, Bonn, Juillet 1986. p. 272-276.

[BLAIR 60]
BLAIR C. *A program for correcting spelling errors.* Information and Control nº 3, 1960. p. 60-67.

[BESCHERELLE 84]
LE NOUVEAU BESCHERELLE - 3. La grammaire pour tous. Librairie Hatier, Paris, Mai 1984. 318 p.

[BOITET 88]
BOITET C., ZAHARIN Y. *Representation trees and string-tree correspondences.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 59-64.

[CARBONELL 83]
CARBONELL J., HAYES P. *Recovery strategies for parsing extragrammatical language*, American Journal of Computational Linguistics, Vol. 9 nº 3-4, Juillet-Déc. 1983. p. 123-146.

[CARBONELL 84]
CARBONELL J., HAYES P. *Coping with extragrammaticality.* 10th International Conference on Computational Linguistics, Juillet 1984. p. 437-443.

[CATACH 73]
CATACH N. *L'Orthographe*, Langue Française, nº 20, Déc. 1973.

[CATACH 79]
CATACH N., MEISSONNIER V. *Pour une meilleure formalisation de la conversion automatique graphème-phonème.* Dixièmes Journées d'Etude sur la Parole, Grenoble, Juin 1979.

[CATACH 80a]
CATACH N. *L'orthographe française - traité théorique et pratique.* Ed. Nathan, Paris, 1980. 334 p.

[CATACH 80b]
CATACH N., MEISSONNIER V. *Formalisation et conversion automatique graphème-phonème en français.* Deuxième Conférence Internationale sur les Bases de Données dans les Humanités et les Sciences Sociales. Madrid, Juin 1980.

[CATACH 80c]
CATACH N., DUPREZ D., LEGRIS M. *L'enseignement de l'orthographe.* Dossiers Didactiques Nathan, Luçon, 1980. 96 p.

[COHARD 88]
COHARD B. *Logiciel de détection et de correction des erreurs lexicales.* Mémoire Ingénieur C.N.A.M. C.N.A.M., Centre régional associé de Grenoble, USTMG, Mars 1988.

[COURTIN 76]
COURTIN J., DUJARDIN D. *Paramètres linguistiques de la Morphologie Française dans le système PIAF.* Université Scientifique et Médicale de Grenoble, Laboratoire d'Informatique, Déc. 1976. 134 p.

[COURTIN 77]
COURTIN, J. *Algorithmes pour le traitement interactif des langues naturelles.* Thèse de Doctorat d'Etat, USMG-INPG, Grenoble, Oct. 1977. 213 p.

[COURTIN 86]
COURTIN J., DUJARDIN D., KOWARSKI I. *Le système PILAF.* Table Ronde Franco-Allemande sur les langues naturelles, Déc. 1986.

[COURTIN 87a]
COURTIN J., DUJARDIN D., KOWARSKI I. *Détection et correction des erreurs.* Rapport réunions PRC du 17/09/87 et du 02/12/87.

[COURTIN 88]
COURTIN J., DUJARDIN D., KOWARSKI I., GENTHIAL D., STRUBE DE LIMA V. *Correção de erros de ortografia através da fonética em textos escritos em francês.* XIV Conferencia Latinoamericana de Informática, 17avas Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Sep. 1988. p. 873-891.

[COURTIN 89a]
COURTIN J., DUJARDIN D., KOWARSKI I., GENTHIAL D., STRUBE DE LIMA V. *DECOR - detecção e correção léxico-sintática num texto escrito.* XV Conferencia Latinoamericana de Informática, IX Conferencia Internacional de la Sociedad Chilena de Ciência de la Computación, Santiago du Chili, Juillet 1989. p. 67-76.

[COURTIN 89b]
COURTIN J., DUJARDIN D., KOWARSKI I., GENTHIAL D., STRUBE DE LIMA V. *Análise de textos escritos em português com PILAF - uma experiência e seus resultados.* 18avas Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Août 1989. p. 9.29-9.46.

[COURTIN 89c]
COURTIN J., DUJARDIN D., KOWARSKI I., GENTHIAL D., STRUBE DE LIMA V. *Vers un système complet de détection/correction d'erreurs en français.* Rapport interne, Equipe TRILAN, LGI-IMAG, Grenoble, Mars 1989.

[COURTIN 89d]
COURTIN J., DUJARDIN D., KOWARSKI I., GENTHIAL D., STRUBE DE LIMA V. *Interactive multi-level systems for correction of ill-formed french texts.* Second Scandinavian Conference on Artificial Intelligence - 1989, Tempere - Finlande, Juin 1989. p. 912-920.

[DAELEMANS 88]
DAELEMANS W. *GRAFON : a grapheme-to-phoneme conversion system for Dutch.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 133-138.

[DAMERAU 64]
DAMERAU F. J. *A technique for computer detection and correction of spelling errors*, Communications of the ACM, Vol. 7 nº 3, Mars 1964. p. 171-176.

[DANLOS 85]
DANLOS L. *Génération automatique de textes en langues naturelles.* Ed. Masson, Paris, 1985. 239 p.

[DEBILI 86]
DEBILI F. *Le traitement des entrées non prévues dans les systèmes de compréhension du langage naturel - détection et correction des graphies fautives.* Rapport d'activités PRC Communication Homme-Machine Pôle Langage Naturel, Orsay, 1986.

[DELGADO 86]
DELGADO A., LALLIAS J. *L'orthographe à l'heure de l'informatique.* Ed. Cedic/Nathan, Paris, 1986. 158 p.

[DEMEYERE 85]
DEMEYERE J. *Dictionnaire orthographique de base pour écrire tout seul.* Ed. Nathan, Luçon, 1985. 96 p.

[DEROUAULT 86]
DEROUAULT A.-M., MERIALDO B. *Natural language modeling for Phoneme-to-text transcription.* IEEE Transactions on Pattern

Analysis and Machine Intelligence, Vol. PAMI-8 nº 6, Nov. 1986. p. 742-749.

[DURHAM 83]
DURHAM I., LAMB D., SAXE J. *Spelling correction in user interfaces.* Comm. of the ACM, Oct. 1983, Vol. 26 nº 10. p. 764-773.

[EL WAKIL 69]
EL WAKIL, H. *L'orthographe d'usage - Contribution expérimentale à la rationalisation de son enseignement.* Thèse nº 40. Université de Genève.

[EMIRKANIAN 88a]
EMIRKANIAN L., BOUCHARD L. *Towards a knowledge-based tool for correcting french text.*, IFIP European Conference on Computer in Education, Lausanne, Juillet 1988. p. 583-588.

[EMIRKANIAN 88b]
EMIRKANIAN L., BOUCHARD L. *Knowledge integration in a robust and efficient morpho-syntactic analyser for French.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 166-171.

[FINDLER 79]
FINDLER N., VAN LEEUWEN J. *A family of similarity measures between two strings.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1 nº 1, Janvier 1979. p. 116-118.

[FOURNIER 84]
FOURNIER J.P. *Sauvetage de raisonnement en langue naturelle.* Thèse de troisième cycle, Paris-Orsay, Juin 1984.

[FOUQUERE 88]
FOUQUERE C. *Systèmes d'analyse tolérante du langage naturel.* Thèse de Doctorat, Paris-Nord, Janvier 1988.

[FRISCH 88]
FRISCH R., ZAMORA A. *Spelling assistance for compound words.* IBM Journal of Research and Development, Vol. 32 nº 2, Mars 1988.

[GENTHIAL 86]
GENTHIAL D. *Interrogation de bases de données en langue naturelle.* Rapport de D.E.A., Juin 1986, Grenoble.

[GIANNESINI 85]
GIANNESINI F. et al. *PROLOG.* InterEditions, Paris, 1985. 318 p.

[GRAMMONT 63]
GRAMMONT M. *La prononciation française - traité pratique.* Ed. Delagrave, Paris, 1963. 241 p.

[GRANGER 83]
GRANGER R. *The NOMAD system : expectation-base detection and correction of errors during understanding of syntactically and semantically Ill-Formed text*, American Journal of Computational Linguistics, Vol. 9 nº 3-4, Juillet-Déc. 1983. p. 188-196.

[GREANIAS 84]
GREANIAS E.C. *Computer aids for spelling correction and word selection.* IBM Europe Institute 1984, Davos-Suisse, Juillet-Août 1984.

[GREVISSE 69]
GREVISSE M. *Le bon usage.* Librairie Hatier, Paris, 1969. 1228 p.

[GROSS 75]
GROSS M. *Méthodes en syntaxe.* Ed. Hermann, Paris, 1975. 414 p.

[GROSS 86]
GROSS M. *Lexicon-grammar and the syntactic analysis of french.* 11th International Conference on Computational Linguistics, Bonn, Juillet 1986. p. 275-282.

[HALL 80]
HALL P., DOWLING G. *Approximate string matching.* Computing Surveys, Vol. 12 nº 4, Déc. 1980. p. 381-402.

[HEIDORN 82]
HEIDORN G. E. et al. *The EPISTLE text-critiquing system.* IBM Systems Journal, Vol. 21 nº 3, 1982.

[ICHBIAH 89]
ICHBIAH D. *Vérificateurs orthographiques : encore un effort.* Décision informatique, 24 Avril 1989.

[JELINEK 75]
JELINEK F., BAHL L., MERCER R. *Design of a linguistic statistical decoder for the recognition of continuous speech*, IEEE Transactions on Information Theory, Vol. IT-21 nº 3, Mai 1975. p. 250-256.

[JENSEN 83]
JENSEN K. et al. *Parse fitting and prose fixing : getting a hold on Ill-formedness.* American Journal of Computational Linguistics, Vol. 9 nº 3-4, Juillet-Déc. 1983.

[JENSEN 84]
Jensen K. *The EPISTLE / NLP grammar : computational evidence for an autonomous syntax.* IBM Europe Institute 1984, Davos-Suisse, Juillet-Août 1984.

[LACOUTURE 86]
LACOUTURE, R. *Une implantation informatique du français fondamental.* Mémoire présenté en vue de l'obtention du grade de Maître en Sciences (M. Sc.), Département d'Informatique et de Recherche Opérationnelle - Université de Montréal, Mai 1986.

[LACOUTURE 88]
LACOUTURE R., LAPALME G. *Une implantation informatique du français fondamental.* T.S.I. - Techniques et Sciences Informatiques, Vol. 7 nº 5, 1988.

[LAHENS 86]
LAHENS F. *Un modèle stochastique pour la vérification et la correction automatique de textes : le système VORTEX.* Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1986. 166 p.

[LALLICH-BOIDIN 86]
LALLICH-BOIDIN G. *Analyse syntaxique automatique du français - application à l'indexation automatique.* Thèse de Doctorat, Université de Sciences Sociales, Grenoble, Oct. 1986.

[LAPORTE 88]
LAPORTE E. *Méthodes algorithmiques et lexicales de phonétisation de textes - application au français.* Thèse de Doctorat, Université Paris 7, Centre d'études et de recherches en informatique linguistique, Paris, Mai 1988. 162 p.

[LAPORTE 89a]
LAPORTE E., SILBERZTEIN M. *Vérification et correction orthographiques assistées par ordinateur.* Actes de la Convention IA 1989. p. 283-298.

[LAPORTE 89b]
LAPORTE E. *La phonétisation automatique de textes français.* Rapport du Centre d'études et de recherches en Informatique linguistique, Evry, 1989.

[LAROUSSE 72]
Dictionnaire de mots communs en couleurs LAROUSSE France Loisirs. Librairie LAROUSSE, Paris, 1972. 992 p.

[LESMO 86]
LESMO L., TORASSO P. *Interpreting syntactically ill-formed sentences.* 11th International Conference on Computational Linguistics, Bonn, Juillet 1986. p. 534-539.

[LOPES 86]
LOPES G. *Transforming english interfaces to other natural languages : an experiment with portuguese.* 11th International Conference on Computational Linguistics, Bonn, Juillet 1986. p. 8-10.

[LOWRANCE 75]
LOWRANCE R., WAGNER R. *An extension of the string-to-string correction problem.* Journal of the ACM, Vol. 22 nº 2, Avr. 1975. p. 177-183.

[LU 86]
LU C. *Correction automatique des erreurs d'orthographe du français.* Rapport de D.E.A., INPG-USMG, Grenoble, Sep. 1986. 68 p.

[MARET 87]
MARET D. *Comparaisons de chaînes de caractères, accès lexicaux tolérants et applications.* Thèse de Doctorat, Univ. Sciences Sociales, Grenoble, Mai 1987. 249 p.

[MASSSIAN 79]
MASSIAN M. *L'orthographe en 10 leçons pour ne plus faire de fautes.* Ed. Hachette, Paris, 1979. 272 p.

[OKUDA 76]
OKUDA T., TANAKA E., KASAI T. *A method for the correction of garbled words based on the Levenshtein Metric*, IEEE Transactions on Computers, Vol. C-25 nº 2, Fév. 1976. p. 172-177.

[OWOLABI 88]
OWOLABI O., MCGREGOR D. *Fast approximate string matching.* SOFTWARE - Practice and experience, Vol. 18 nº 4, Avr. 1988. p. 387-393.

[PEQUEGNAT 87]
PEQUEGNAT C. *Compréhension automatique de requêtes en langue naturelle : aspects linguistiques dans le contexte d'un système de recherche d'informations.* Rapport de D.E.A., INPG-USTMG, Grenoble.

[PERENNOU 86]
PERENNOU G., DAUBEZE P., LAHENS F. *La vérification et la correction automatique de textes: le système VORTEX.* TSI - Technique et Sciences Informatiques, Vol. 5 nº 4, 1986. p. 285-304.

[PETERSON 80]
PETERSON J. *Computer programs for detecting and correcting spelling errors*, Communications of the ACM, Vol. 23, 1980. p. 676-687.

[POLLOCK 84]
POLLOCK J., ZAMORA A. *Automatic spelling correction in scientific and scholarly text.* Communications of the ACM, Vol. 27 nº 4, Avr. 1984.

[RICHARD 86]
RICHARD D., LAPALME G. *Un système de correction automatique des accords des participes passés.* TSI - Technique et Sciences Informatiques, Vol. 5 nº 4, 1986. p. 307-319.

[RISEMAN 71]
RISEMAN E., EHRICH R. *Contextual word recognition using binary digrams.* IEEE Transactions on Computers, Vol. C-20 nº 4, Avr. 1971, p. 397-403.

[RISEMAN 74]
RISEMAN E., HANSON A. *A contextual postprocessing system for error correction using binary n-grams.* IEEE Transactions on Computers, Vol. C-23 nº 5, Mai 1974. p. 480-493.

[ROBERT 78]
ROBERT P. *Dictionnaire alphabétique et analogique de la langue française.* Ed. Société du Nouveau Littré, Paris, 1978. 2173 p.

[SAIDI 86]
SAIDI S. *Pré-étude pour la vérification syntaxique d'une phrase dans l'environnement PILAF.* Rapport interne 01/1986, Equipe Bases de Données Textuelles et Traitement des Langues Naturelles, LGI-IMAG, Grenoble, Oct. 1986.

[SCHABES 88]
SCHABES Y., ABEILLE A., JOSHI A. *Parsing strategies with "lexicalized" grammars : application to Tree Adjoining Grammars.* 12th International Conference on Computational Linguistics, Budapest, Aôut 1988. p. 578-583.

[ST-DIZIER 84]
ST-DIZIER P. *Etude et réalisation de ESOPE : un système paramétrable pour la traduction de sous-ensembles du français en logique.* Thèse de 3ème cycle, Université de Rennes I, Rennes, 1984. 96 p.

[TURBO 86]
TURBO PROLOG - le langage naturel de l'intelligence artificielle. BORLAND International Guide de l'utilisateur. Sèvres, 1986. 269 p.

[VERGNE 86]
VERGNE J., PAGES P. *Synergy of syntax and morphology in automatic parsing of french language with a minimum of data.* 11th International Conference on Computational Linguistics, Bonn, Juillet 1986. p. 269-271.

[VERONIS 86]
VERONIS J. *Etude quantitative sur le système graphique et phonographique du français.* Cahiers de Psychologie Cognitive, Vol. 6 nº 5, Oct. 1986. p. 501-531.

[VERONIS 87]
VERONIS J. *Phonographic and typographical correction in Natural Language Interfaces.* 5th International Symposium in Applied Informatics, Grindelwald-Suisse, Fév. 1987.

[VERONIS 88a]
VERONIS J. *Correction of phonographic errors in natural language interfaces*, 11th International Conference on Research & Development in Information Retrieval, Juin 1988, Grenoble.

[VERONIS 88b]
VERONIS J. *Morphosyntactic correction in natural language interfaces.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 708-713.

[VERONIS 88c]
VERONIS J. *Contribution à l'étude de l'erreur dans le dialogue homme-machine en langage naturel.* Thèse de Doctorat en Informatique, Aix-Marseille III. Oct. 1988. 325 p.

[VIJAY-SHANKER 88]
VIJAY-SHANKER K., JOSHI A. *Feature structures based Tree Adjoining Grammars.* 12th International Conference on Computational Linguistics, Budapest, Août 1988. p. 714-719.

[WAGNER 74]
WAGNER C., FISCHER M. *The string-to-string correction problem.* Journal of the A.C.M., Vol. 21 nº 1, 1974. p. 168-173.

[WEISCHEDEL 83]
WEISCHEDEL R. *Meta-rules as a basis for processing ill-formed input.* American Journal of Computational Linguistics Vol. 9 nº 3-4, Juillet-Déc. 1983. p. 161-177.

[WONG 76]
WONG C., CHANDRA A. *Bounds for the string editing problem.* Journal of the A.C.M., Vol. 23 nº 1, Janvier 1976. p. 13-16.

ANNEXE 1

MODELES PHONETIQUES

*** MODELES ***

```
/ /:          REG:=  2-fin pb  .
/a.m/:        REG:= 21-pib sui ;  CD: a.m  ;  VAL:= 22-fin sui
/a/:          REG:= 21-pib sui ;  CD: a    ;  VAL:= 22-fin sui
/agefin/:     REG:=  3-sui ;      CD: a.j ;   VAL:= 17-fin
/ail%/:       REG:=  5-pif ;      CD: a.y
/ailfin/:     REG:=  3-sui ;      CD: a.y ;   VAL:= 17-fin
/aill/:       REG:= 21-pib sui ;  CD: a.y ;   VAL:= 22-fin sui
/aisonfin/:   REG:=  3-sui ;      CD: e<.z.on;VAL:= 17-fin
/al/:         REG:= 21-pib sui ;  CD: a.l;    VAL:= 22-fin sui
/ama/:        REG:= 21-pib sui ;  CD: a.m.a ; VAL:= 22-fin sui
/amb/:        REG:= 21-pib sui ;  CD: an.b ;  VAL:=  3-sui
/ame</:       REG:= 21-pib sui ;  CD: a.m.e<; VAL:= 22-fin sui
/ami/:        REG:= 21-pib sui ;  CD: a.m.i ; VAL:= 22-fin sui
/amo/:        REG:= 21-pib sui ;  CD: a.m.o ; VAL:= 22-fin sui
/amp/:        REG:= 21-pib sui ;  CD: an.p  ; VAL:=  3-sui
/amu/:        REG:= 21-pib sui ;  CD: a.m.u ; VAL:= 22-fin sui
/an/:         REG:= 21-pib sui ;  CD: an  ;   VAL:= 22-fin sui
/ana/:        REG:= 21-pib sui ;  CD: a.n.a ; VAL:= 22-fin sui
/ane</:       REG:= 21-pib sui ;  CD: a.n.e<; VAL:= 22-fin sui
/ani/:        REG:= 21-pib sui ;  CD: a.n.i ; VAL:= 22-fin sui
/ano/:        REG:= 21-pib sui ;  CD: a.n.o ; VAL:= 22-fin sui
/anti/:       REG:=  8-pib ;      CD: an.t.i; VAL:=  3-sui
/anu/:        REG:= 21-pib sui ;  CD: a.n.u ; VAL:= 22-fin sui
/ar/:         REG:= 21-pib sui ;  CD: a.r ;   VAL:= 22-fin sui
/asefin/:     REG:=  3-sui ;      CD: a.z  ;  VAL:= 17-fin .
/ationfin/:   REG:=  3-sui ;  CD: a.s.y.on;   VAL:= 17-fin .
/ay/:         REG:= 21-pib sui ; CD: e<.y  ;  VAL:= 22-fin sui
/b/:          REG:= 21-pib sui ; CD: b ;      VAL:= 22-fin sui
/car/:        REG:= 21-pib sui ; CD: k.a.r ;  VAL:= 22-fin sui
/ce<r/:       REG:= 21-pib sui ; CD: s.e<.r ; VAL:=  3-sui
/ce>r/:       REG:= 21-pib sui ; CD: s.e>.r ; VAL:=  3-sui
/ch/:         REG:= 21-pib sui ; CD: ch  ;    VAL:= 22-fin sui
/cher/:       REG:= 21-pib sui ; CD: ch.e<.r; VAL:=  3-sui
```

```
/cor/:       REG:= 21-pib sui ; CD: k.o.r ;   VAL:= 22-fin sui
/cur/:       REG:= 21-pib sui ; CD: k.u.r ;   VAL:= 22-fin sui
/cye<nfin/:  REG:= 21-pib sui ; CD: s.y.e<.n ; VAL:= 17-fin
/cyinfin/:   REG:= 21-pib sui ; CD: s.y.in ;  VAL:= 17-fin
/d%/:        REG:=  5-pif ; CD: d.e>
/d/:         REG:= 21-pib sui ; CD: d ;       VAL:= 22-fin sui
/dent/:      REG:=  3-sui ;                   VAL:= 17-fin
/dez/:       REG:=  3-sui ;   CD: e< ;        VAL:= 17-fin
/diez/:      REG:=  3-sui ;   CD: y.e< ;      VAL:= 17-fin
/dions/:     REG:=  3-sui ;   CD: y.on ;      VAL:= 17-fin
/e</:        REG:= 21-pib sui ; CD: e< ;      VAL:= 22-fin sui
/e<ks/:      REG:= 21-pib sui ; CD: e<.k.s;   VAL:= 3-sui
/e<man/:     REG:= 21-pib sui ; CD: e<.m.an;  VAL:= 17-fin
/e<ne</:     REG:= 21-pib sui ; CD: e<.n.e< ;VAL:= 22-fin sui
/e<nou/:     REG:= 21-pib sui ; CD: e<.n.ou ;VAL:= 22-fin sui
/e<r/:       REG:= 21-pib sui ; CD: e<.r ;    VAL:= 22-fin sui
/e>/:        REG:= 21-pib sui ; CD: e> ;      VAL:= 22-fin sui
/e>nou/:     REG:= 21-pib sui ; CD: e>.n.ou ;VAL:= 22-fin sui
/eil/:       REG:=  3-sui ; CD: e<.y  ;       VAL:= 22-fin sui
/einefin/:   REG:= 21-pib sui ; CD: e<.n   ;  VAL:= 17-fin
/el/:        REG:= 21-pib sui ; CD: e<.l  ;   VAL:= 22-fin sui
/ema/:       REG:= 21-pib sui ; CD: e>.m.a ;  VAL:= 22-fin sui
/eman/:      REG:=  3-sui ;  CD: e>.m.an  ;   VAL:= 17-fin
/eme</:      REG:= 21-pib sui ; CD: e>.m.e< ;VAL:= 22-fin sui
/emi/:       REG:= 21-pib sui ; CD: e>.m.i ;  VAL:= 22-fin sui
/emo/:       REG:= 21-pib sui ; CD: e>.m.o ;  VAL:= 22-fin sui
/emu/:       REG:= 21-pib sui ; CD: e>.m.u ;  VAL:= 22-fin sui
/ena/:       REG:= 21-pib sui ; CD: e>.n.a ;  VAL:= 22-fin sui
/enan/:      REG:= 21-pib sui ; CD: e>.n.an;VAL:= 22-fin sui
/ene/:       REG:= 21-pib sui ; CD: e>.n.e> ;VAL:=  3-sui
/ene</:      REG:= 21-pib sui ; CD: e>.n.e< ;VAL:= 22-fin sui
/enefin/:    REG:= 21-pib sui ; CD: e<.n  ;   VAL:= 17-fin
/eni/:       REG:= 21-pib sui ; CD: e>.n.i ;  VAL:= 22-fin sui
/eno/:       REG:= 21-pib sui ; CD: e>.n.o ;  VAL:= 22-fin sui
/ensefin/:   REG:=  3-sui ;  CD: an.s  ;      VAL:= 17-fin
/ension/:    REG:=  3-sui ; CD: an.s.y.on ;   VAL:= 17-fin
/ent/:       REG:= 21-pib sui ; CD: an.t ;    VAL:= 22-fin sui
```

```
/enti/:      REG:=  3-sui ; CD: an.t.i  ;     VAL:= 22-fin sui
/entions/:   REG:=  3-sui ; CD: an.t.y.on ;   VAL:= 17-fin
/entir/:     REG:=  3-sui ; CD: an.t.i.r ;    VAL:= 22-fin sui
/entis/:     REG:=  3-sui ; CD: an.t.i.s ;    VAL:=  3-sui
/enu/:       REG:= 21-pib sui ; CD: e>.n.u ;  VAL:= 22-fin sui
/erbefin/:   REG:=  3-sui ; CD: e<.r.b ;      VAL:= 17-fin
/ettefin/:   REG:=  3-sui ; CD: e<.t ;        VAL:= 17-fin
/eurfin/:    REG:=  3-sui ; CD: oe.r ;        VAL:= 17-fin
/eusefin/:   REG:=  3-sui ; CD: oe.z ;        VAL:= 17-fin
/ey/:        REG:= 21-pib sui ; CD: e<.y ;    VAL:= 22-fin sui
/f/:         REG:= 21-pib sui ; CD: f ;       VAL:= 22-fin sui
/g/:         REG:= 21-pib sui ; CD: g ;       VAL:= 22-fin sui
/gdeb/:      REG:=  8-pib ; CD: g ;           VAL:=  3-sui
/gl/:        REG:= 21-pib sui ; CD: g.l ;     VAL:=  3-sui
/gn/:        REG:= 21-pib sui ; CD: gn  ;     VAL:= 22-fin sui
/gr/:        REG:= 21-pib sui ; CD: g.r ;     VAL:= 22-fin sui
/gufin/:     REG:=  3-sui ; CD: g.u  ;        VAL:= 17-fin
/gw/:        REG:= 21-pib sui ; CD: g.u ;     VAL:=  3-sui
/gza/:       REG:=  3-sui ; CD: g.z.a ;       VAL:=  3-sui
/gzan/:      REG:= 21-pib sui ; CD: g.z.an ;  VAL:=  3-sui
/gze>/:      REG:=  3-sui ; CD: g.z.e>  ;     VAL:=  3-sui
/hdeb/:      REG:=  8-pib ;  VAL:=  3-sui
/hmil/:      REG:=  3-sui ;  VAL:=  3-sui
/i.m/:       REG:= 21-pib sui ; CD: i.m ;     VAL:= 22-fin sui
/i/:         REG:= 21-pib sui ; CD: i ;       VAL:= 22-fin sui
/ica/:       REG:= 21-pib sui ; CD: i.k.a  ;  VAL:= 22-fin sui
/ilfin/:     REG:=  3-sui ; CD: i.l ;         VAL:= 17-fin
/ilfiny/:    REG:=  3-sui ; CD: y  ;          VAL:= 17-fin
/illefin/:   REG:=  3-sui ; CD: i.y   ;       VAL:= 17-fin
/ima/:       REG:= 21-pib sui ; CD: i.m.a ;   VAL:= 22-fin sui
/imb/:       REG:= 21-pib sui ; CD: in.b ;    VAL:=  3-sui
/ime</:      REG:= 21-pib sui ; CD: i.m.e< ;  VAL:= 22-fin sui
/imi/:       REG:= 21-pib sui ; CD: i.m.i ;   VAL:= 22-fin sui
/imo/:       REG:= 21-pib sui ; CD: i.m.o ;   VAL:= 22-fin sui
/imp/:       REG:= 21-pib sui ; CD: in.p ;    VAL:=  3-sui
/imu/:       REG:= 21-pib sui ; CD: i.m.u  ;  VAL:= 22-fin sui
/in/:        REG:= 21-pib sui ; CD: in ;      VAL:= 22-fin sui
```

```
/ina/:      REG:= 21-pib sui ; CD: i.n.a ;  VAL:= 22-fin sui
/ine</:     REG:= 21-pib sui ; CD: i.n.e< ; VAL:= 22-fin sui
/inefin/:   REG:=  3-sui ; CD: i.n ;        VAL:= 17-fin
/ingfin/:   REG:=  3-sui ; CD: ing ;        VAL:= 17-fin
/ini/:      REG:= 21-pib sui ; CD: i.n.i ;  VAL:= 22-fin sui
/ino/:      REG:= 21-pib sui ; CD: i.n.o ;  VAL:= 22-fin sui
/inst/:     REG:= 21-pib sui ; CD: in.s.t ; VAL:=  3-sui
/inu/:      REG:= 21-pib sui ; CD: i.n.u ;  VAL:= 22-fin sui
/j/:        REG:= 21-pib sui ; CD: j ;      VAL:= 22-fin sui
/jan/:      REG:= 21-pib sui ; CD: j.an ;   VAL:= 22-fin sui
/je%/:      REG:=  5-pif ;      CD: j.e>
/je</:      REG:= 21-pib sui ; CD: j.e< ;   VAL:= 22-fin sui
/je<n/:     REG:= 21-pib sui ; CD: j.e<.n ; VAL:=  3-sui
/je<o/:     REG:= 21-pib sui ; CD: j.e<.o ; VAL:=  3-sui
/je>/:      REG:= 21-pib sui ; CD: j.e> ;   VAL:=  3-sui
/je>n/:     REG:= 21-pib sui ; CD: j.e>.n ; VAL:=  3-sui
/ji.n/:     REG:= 21-pib sui ; CD: j.i.n ;  VAL:=  3-sui
/ji/:       REG:= 21-pib sui ; CD: j.i ;    VAL:= 22-fin sui
/jin/:      REG:= 21-pib sui ; CD: j.in ;   VAL:= 22-fin sui
/jo/:       REG:= 21-pib sui ; CD: j.o ;    VAL:=  3-sui
/joi/:      REG:= 21-pib sui ; CD: j.w.a ;  VAL:= 22-fin sui
/k/:        REG:= 21-pib sui ; CD: k ;      VAL:= 22-fin sui
/ka/:       REG:= 21-pib sui ; CD: k.a ;    VAL:=  3-sui
/ke>/:      REG:= 21-pib sui ; CD: k.e> ;   VAL:=  3-sui
/ks/:       REG:= 21-pib sui ; CD: k.s ;    VAL:= 22-fin sui
/ksan/:     REG:=  3-sui ; CD: k.s.an ;     VAL:= 22-fin sui
/kse</:     REG:= 21-pib sui ; CD: k.s.e< ; VAL:= 22-fin sui
/kse>/:     REG:= 21-pib sui ; CD: k.s.e> ; VAL:=  3-sui
/ksefin/:   REG:=  3-sui ; CD: k.s ;        VAL:= 17-fin
/ksi/:      REG:=  3-sui ; CD: k.s.i ;      VAL:=  3-sui
/ksio/:     REG:=  3-sui ; CD: k.s.i.o ;    VAL:=  3-sui
/ksyon/:    REG:=  3-sui ; CD: k.s.y.on ;   VAL:= 22-fin sui
/kti/:      REG:=  3-sui ; CD: k.t.i ;      VAL:=  3-sui
/ku/:       REG:= 21-pib sui ; CD: k.u ;    VAL:= 22-fin sui
/kw/:       REG:= 21-pib sui ; CD: k.w ;    VAL:=  3-sui
/l/:        REG:= 21-pib sui ; CD: l ;      VAL:= 22-fin sui
/le%/:      REG:=  5-pif ; CD: l.e>
```

```
/leman/:     REG:=  3-sui ; CD: l.m.an  ;     VAL:= 17-fin
/les%/:      REG:=  5-pif ; CD: l.e<
/m/:         REG:= 21-pib sui ; CD: m ;       VAL:= 22-fin sui
/mentfin/:   REG:=  3-sui ; CD: m.an ;        VAL:= 17-fin
/mes%/:      REG:=  5-pif ; CD: m.e>
/muet%/:     REG:=  5-pif ; CD: muet
/muet/:      REG:=  3-sui ;   VAL:= 17-fin
/n/:         REG:= 21-pib sui ; CD: n ;       VAL:= 22-fin sui
/ne%/:       REG:=  5-pif ; CD: n.e>
/nou/:       REG:= 21-pib sui ; CD: n.ou ;    VAL:= 22-fin sui
/o/:         REG:= 21-pib sui ; CD: o  ;      VAL:= 22-fin sui
/oe/:        REG:= 21-pib sui ; CD: oe ;      VAL:= 22-fin sui
/oeil/:      REG:= 21-pib sui ; CD: oe.y ;    VAL:= 22-fin sui
/oi/:        REG:= 21-pib sui ; CD: w.a ;     VAL:= 22-fin sui
/oin/:       REG:= 21-pib sui ; CD: w.in  ;   VAL:= 22-fin sui
/oisefin/:   REG:=  3-sui ; CD: w.a.z ;       VAL:= 17-fin
/oma/:       REG:= 21-pib sui ; CD: o.m.a  ;  VAL:= 22-fin sui
/omb/:       REG:= 21-pib sui ; CD: on.b  ;   VAL:=  3-sui
/ome</:      REG:= 21-pib sui ; CD: o.m.e< ;  VAL:= 22-fin sui
/omfin/:     REG:=  3-sui ; CD: o.m  ;        VAL:= 17-fin
/omi/:       REG:= 21-pib sui ; CD: o.m.i  ;  VAL:= 22-fin sui
/omo/:       REG:= 21-pib sui ; CD: o.m.o ;   VAL:= 22-fin sui
/omp/:       REG:= 21-pib sui ; CD: on.p ;    VAL:=  3-sui
/omu/:       REG:= 21-pib sui ; CD: o.m.u ;   VAL:= 22-fin sui
/on/:        REG:= 21-pib sui ; CD: on ;      VAL:= 22-fin sui
/ona/:       REG:= 21-pib sui ; CD: o.n.a ;   VAL:= 22-fin sui
/one</:      REG:= 21-pib sui ; CD: o.n.e< ;  VAL:= 22-fin sui
/onefin/:    REG:= 21-pib sui ; CD: o.n ;     VAL:= 17-fin
/oni/:       REG:= 21-pib sui ; CD: o.n.i ;   VAL:= 22-fin sui
/ono/:       REG:= 21-pib sui ; CD: o.n.o ;   VAL:= 22-fin sui
/ont/:       REG:= 21-pib sui ; CD: on.t  ;   VAL:=  3-sui
/onu/:       REG:= 21-pib sui ; CD: o.n.u ;   VAL:= 22-fin sui
/ou/:        REG:= 21-pib sui ; CD: ou ;      VAL:= 22-fin sui
/ouill/:     REG:= 21-pib sui ; CD: ou.y  ;   VAL:= 22-fin sui
/ousefin/:   REG:=  3-sui ; CD: ou.z  ;       VAL:= 17-fin
/oy/:        REG:= 21-pib sui ; CD: w.a.y ;   VAL:= 22-fin sui
/p/:         REG:= 21-pib sui ; CD: p  ;      VAL:= 22-fin sui
```

```
/pt/:       REG:= 21-pib sui ; CD: p.t  ;   VAL:= 22-fin sui
/que%/:     REG:=  5-pif ;      CD: k.e>
/qw/:       REG:= 21-pib sui ; CD: k.w ;    VAL:=  3-sui
/qwa/:      REG:= 21-pib sui ; CD: k.w.a ;  VAL:=  3-sui
/r/:        REG:= 21-pib sui ; CD: r ;      VAL:= 22-fin sui
/rs/:       REG:=  3-sui ; CD: r.s  ;       VAL:= 22-fin sui
/rtial/:    REG:=  3-sui ; CD: r.s.i.a.l ;  VAL:= 22-fin sui
/rtiel/:    REG:=  3-sui ; CD: r.s.i.e.l ;  VAL:= 22-fin sui
/s1/:       REG:=  8-pib ; CD: s  ;         VAL:=  3-sui
/s2/:       REG:=  3-sui ; CD: s  ;         VAL:= 22-fin sui
/san/:      REG:=  8-pib ; CD: s.an   ;     VAL:= 22-fin sui
/se%/:      REG:=  5-pif ; CD: s.e>
/se/:       REG:=  8-pib ; CD: s.e>  ;      VAL:= 22-fin sui
/se</:      REG:= 21-pib sui ; CD: s.e< ;   VAL:= 22-fin sui
/se>n/:     REG:=  8-pib ; CD: s.e>.n ;     VAL:= 22-fin sui
/sens%/:    REG:=  5-pif ; CD: s.an.s
/sf/:       REG:= 21-pib sui ; CD: s.f  ;   VAL:=  3-sui
/si.n/:     REG:=  8-pib ; CD: s.i.n   ;    VAL:= 22-fin sui
/si/:       REG:= 21-pib sui ; CD: s.i  ;   VAL:= 22-fin sui
/sin/:      REG:=  8-pib ; CD: s.in  ;      VAL:= 22-fin sui
/sio.n/:    REG:=  3-sui ; CD: s.i.o.n ;    VAL:=  3-sui
/sk/:       REG:= 21-pib sui ; CD: s.k ;    VAL:= 22-fin sui
/sqwa/:     REG:= 21-pib sui ; CD: s.k.w.a ; VAL:=  3-sui
/st/:       REG:= 21-pib sui ; CD: s.t ;    VAL:= 22-fin sui
/stion/:    REG:=  3-sui ; CD: s.t.y.on  ;  VAL:= 22-fin sui
/sz/:       REG:=  3-sui ; CD: z ;          VAL:=  3-sui
/t/:        REG:= 21-pib sui ; CD: t   ;    VAL:= 22-fin sui
/tes%/:     REG:=  5-pif ; CD: t.e>
/ti1/:      REG:= 21-pib sui ; CD: t.i  ;   VAL:= 22-fin sui
/ti2/:      REG:=  3-sui ; CD: s.i ;        VAL:= 22-fin sui
/tin/:      REG:= 21-pib sui ; CD: t.in  ;  VAL:= 22-fin sui
/tiret/:    REG:=  3-sui ; CD: -   ;        VAL:=  3-sui
/u/:        REG:= 21-pib sui ; CD: u   ;    VAL:= 22-fin sui
/uma/:      REG:= 21-pib sui ; CD: u.m.a  ; VAL:= 22-fin sui
/ume</:     REG:= 21-pib sui ; CD: u.m.e< ; VAL:= 22-fin sui
/umi/:      REG:= 21-pib sui ; CD: u.m.i ;  VAL:= 22-fin sui
```

```
/umo/:      REG:= 21-pib sui ; CD: u.m.o ;  VAL:= 22-fin sui
/umu/:      REG:= 21-pib sui ; CD: u.m.u  ; VAL:= 22-fin sui
/un/:       REG:= 21-pib sui ; CD: un  ;    VAL:= 22-fin sui
/una/:      REG:= 21-pib sui ; CD: u.n.a ;  VAL:= 22-fin sui
/une</:     REG:= 21-pib sui ; CD: u.n.e< ; VAL:= 22-fin sui
/unefin/:   REG:= 21-pib sui ; CD: u.n  ;   VAL:= 17-fin
/uni/:      REG:= 21-pib sui ; CD: u.n.i ;  VAL:= 22-fin sui
/uno/:      REG:= 21-pib sui ; CD: u.n.o ;  VAL:= 22-fin sui
/unu/:      REG:= 21-pib sui ; CD: u.n.u ;  VAL:= 22-fin sui
/usfin/:    REG:=  3-sui ; CD: u.s  ;       VAL:= 17-fin
/uze</:     REG:= 21-pib sui ; CD: u.z.e< ; VAL:= 22-fin sui
/v/:        REG:= 21-pib sui ; CD: v ;      VAL:= 22-fin sui
/w/:        REG:= 21-pib sui ; CD: w ;      VAL:= 22-fin sui
/y/:        REG:= 21-pib sui ; CD: y ;      VAL:= 22-fin sui
/ye<nfin/:  REG:= 21-pib sui ; CD: y.e<.n ; VAL:= 17-fin
/yinfin/:   REG:= 21-pib sui ; CD: y.in  ;  VAL:= 17-fin
/yon/:      REG:= 21-pib sui ; CD: y.on ;   VAL:= 22-fin sui
/z/:        REG:= 21-pib sui ; CD: z  ;     VAL:= 22-fin sui
```

ANNEXE 2

DICTIONNAIRE DE P-GRAPHEMES

*** DICTIONNAIRE ***

```
(  1) 1/û(*)(1)/u/
(  2) 1/ô(*)(1)/o/
(  3) 1/î(*)(1)/i/
(  4) 1/ï(*)(1)/i/
(  5) 1/ï(*)(1)/y/
(  6) 3/èrr(*)(1)/e<r/
(  7) 2/èr(*)(1)/e<r/
(  8) 1/è(1)/e</
(  9) 1/ê(*)(1)/e>/
( 10) 1/ç(2)/s1/
( 11) 1/ç(1)/s2/
( 12) 2/à (*)(1)/a/
( 13) 1/â(*)(1)/a/
( 14) 3/érr(*)(1)/e<r/
( 15) 2/ér(*)(1)/e<r/
( 16) 5/énés (*)(1)/e<ne</
( 17) 6/énées (*)(1)/e<ne</
( 18) 5/énée (*)(1)/e<ne</
( 19) 3/éné(*)(1)/e<ne</
( 20) 4/énou(*)(1)/e<nou/
( 21) 5/énnou(*)(1)/e<nou/
( 22) 6/éments(*)(1)/e<man/
( 23) 5/ément(*)(1)/e<man/
( 24) 4/émen(*)(1)/e<man/
( 25) 6/émants(*)(1)/e<man/
( 26) 5/émant(*)(1)/e<man/
( 27) 4/éman(*)(1)/e<man/
( 28) 4/ées (*)(1)/e</
( 29) 3/ée (*)(1)/e</
( 30) 2/é (*)(1)/e</
( 31) 1/é(1)/e</
( 32) 4/zes (*)(1)/z/
( 33) 3/ze (*)(1)/z/
( 34) 2/z (*)(1)/z/
( 35) 1/z(1)/z/
( 36) 2/yn(*)(1)/in/
( 37) 2/ym(*)(1)/in/
( 38) 1/y(1)/y/
( 39) 4/xés (*)(1)/ksefin/
( 40) 5/xées (*)(1)/kse</
( 41) 4/xée (*)(1)/kse</
( 42) 3/xé (*)(1)/kse</
( 43) 2/xé(*)(1)/kse</
( 44) 3/xti(*)(1)/ksi/
( 45) 3/xsi(*)(1)/ksi/
( 46) 3/xsa(*)(1)/gza/
( 47) 4/xes (*)(1)/ksefin/
( 48) 3/xem(*)(1)/gzan/
( 49) 3/xe (*)(1)/ksefin/
( 50) 3/xcè(*)(1)/kse</
( 51) 3/xce(*)(1)/kse</
( 52) 3/xan(*)(1)/gzan/
( 53) 2/xa(*)(1)/gza/
( 54) 2/x (*)(1)/muet/
```

```
( 55) 1/x(*)(1)/ks/
( 56) 1/w(2)/v/
( 57) 1/w(1)/w/
( 58) 4/ves (*)(1)/v/
( 59) 3/ve (*)(1)/v/
( 60) 2/v (*)(1)/v/
( 61) 1/v(1)/v/
( 62) 5/usés (*)(1)/uze</
( 63) 6/usées (*)(1)/uze</
( 64) 5/usée (*)(1)/uze</
( 65) 4/usé (*)(1)/uze</
( 66) 3/usé(*)(1)/uze</
( 67) 3/us (*)(1)/usfin/
( 68) 4/unu_(*)(1)/unu/
( 69) 6/unues (*)(1)/unu/
( 70) 5/unue (*)(1)/unu/
( 71) 4/unu (*)(1)/unu/
( 72) 4/uns (*)(1)/un/
( 73) 5/unos (*)(1)/uno/
( 74) 4/uno (*)(1)/uno/
( 75) 3/uno(*)(1)/uno/
( 76) 7/unnues (*)(1)/unu/
( 77) 6/unnue (*)(1)/unu/
( 78) 5/unnu (*)(1)/unu/

( 79) 4/unnu(*)(1)/unu/
( 80) 6/unnos (*)(1)/uno/
( 81) 5/unno (*)(1)/uno/
( 82) 4/unno(*)(1)/uno/
( 83) 7/unnies (*)(1)/uni/
( 84) 6/unnie (*)(1)/uni/
( 85) 5/unni (*)(1)/uni/
( 86) 4/unni(*)(1)/uni/
( 87) 8/unneaux (*)(1)/uno/
( 88) 6/unneau(*)(1)/uno/
( 89) 5/unne (*)(1)/unefin/
( 90) 5/unna (*)(1)/una/
( 91) 7/unnaux (*)(1)/uno/
( 92) 5/unnau(*)(1)/uno/
( 93) 6/unnas (*)(1)/una/
( 94) 7/unnais (*)(1)/une</
( 95) 6/unnai (*)(1)/une</
( 96) 5/unnai(*)(1)/une</
( 97) 5/unna (*)(1)/una/
( 98) 6/unies (*)(1)/uni/
( 99) 5/unie (*)(1)/uni/
(100) 4/uni (*)(1)/uni/
(101) 3/uni(*)(1)/uni/
(102) 6/uneau (*)(1)/uno/
(103) 7/uneaux (*)(1)/uno/
(104) 6/uneau (*)(1)/uno/
(105) 4/une (*)(1)/unefin/
(106) 6/unaux (*)(1)/uno/
(107) 5/unau (*)(1)/uno/
(108) 4/unau(*)(1)/uno/
(109) 5/unas (*)(1)/una/
```

(110) 6/unais (*)(1)/une</
(111) 5/unai (*)(1)/une</
(112) 4/unai(*)(1)/une</
(113) 4/una (*)(1)/una/
(114) 3/una(*)(1)/una/
(115) 3/un (*)(1)/un/
(116) 2/un(*)(1)/un/
(117) 6/umues (*)(1)/umu/
(118) 5/umue (*)(1)/umu/
(119) 4/umu (*)(1)/umu/
(120) 3/umu(*)(1)/umu/
(121) 5/umos (*)(1)/umo/
(122) 3/umo(*)(1)/umo/
(123) 7/ummues (*)(1)/umu/
(124) 6/ummue (*)(1)/umu/
(125) 5/ummu (*)(1)/umu/
(126) 4/ummu(*)(1)/umu/
(127) 4/ummo(*)(1)/umo/
(128) 6/ummis (*)(1)/umi/
(129) 7/ummies (*)(1)/umi/
(130) 6/ummie (*)(1)/umi/
(131) 5/ummi (*)(1)/umi/
(132) 4/ummi(*)(1)/umi/
(133) 8/ummeaux (*)(1)/umo/
(134) 7/ummeau (*)(1)/umo/
(135) 6/ummeau(*)(1)/umo/
(136) 7/ummaux (*)(1)/umo/
(137) 6/ummau (*)(1)/umo/
(138) 5/ummau(*)(1)/umo/
(139) 6/ummas (*)(1)/uma/
(140) 7/ummais (*)(1)/ume</
(141) 6/ummai (*)(1)/ume</
(142) 5/ummai(*)(1)/ume</
(143) 5/umma (*)(1)/uma/
(144) 4/umma(*)(1)/uma/
(145) 5/umis (*)(1)/umi/
(146) 6/umies (*)(1)/umi/
(147) 5/umie (*)(1)/umi/
(148) 4/umi (*)(1)/umi/
(149) 3/umi(*)(1)/umi/
(150) 7/umeaux (*)(1)/umo/
(151) 6/umeau (*)(1)/umo/
(152) 5/umeau(*)(1)/umo/
(153) 6/umaux (*)(1)/umo/
(154) 5/umau (*)(1)/umo/
(155) 4/umau(*)(1)/umo/
(156) 5/umas (*)(1)/uma/
(157) 6/umais (*)(1)/ume</
(158) 5/umai (*)(1)/ume</
(159) 4/umai(*)(1)/ume</
(160) 4/uma (*)(1)/uma/
(161) 3/uma(*)(1)/uma/
(162) 3/um (2)/omfin/
(163) 3/um (*)(1)/un/
(164) 4/ues (*)(1)/u/
(165) 5/ueill(*)(1)/oeil/

```
(166) 5/ueil (*)(1)/oeil/
(167) 3/ue (*)(1)/u/
(168) 2/u (*)(1)/u/
(169) 1/u(1)/u/
(170) 5/ttes (*)(1)/t/

(171) 4/tte (*)(1)/t/
(172) 2/tt(*)(1)/t/
(173) 3/ts (*)(1)/muet/
(174) 3/tin(*)(1)/tin/
(175) 2/ti(2)/ti2/
(176) 2/ti(*)(1)/ti1/
(177) 2/th(*)(1)/t/
(178) 4/tes (2)/t/
(179) 4/tes (*)(1)/tes%/
(180) 3/te (*)(1)/t/
(181) 2/t-(*)(1)/hmil/
(182) 2/t (1)/t/
(183) 2/t (*)(2)/muet/
(184) 1/t(1)/t/
(185) 7/stions (*)(1)/stion/
(186) 6/stion (*)(1)/stion/
(187) 2/st(*)(1)/st/
(188) 5/sses (*)(1)/s2/
(189) 4/sse (*)(1)/s2/
(190) 3/ss (*)(1)/s2/
(191) 2/ss(*)(1)/s2/
(192) 6/sques (*)(1)/sk/
(193) 5/sque (*)(1)/sk/
(194) 4/sque(*)(1)/sk/
(195) 4/squa(*)(1)/sqwa/
(196) 3/squ(*)(1)/sk/
(197) 3/sph(*)(1)/sf/
(198) 2/sk(*)(1)/sk/
(199) 3/sin(2)/si.n/
(200) 3/sin(*)(1)/sin/
(201) 6/siens (*)(1)/cyinfin/
(202) 5/sien (*)(1)/cyinfin/
(203) 2/sh(*)(1)/ch/
(204) 2/sf(*)(1)/sf/
(205) 4/ses (2)/s2/
(206) 4/ses (*)(1)/z/
(207) 5/sens (*)(1)/sens%/
(208) 4/senn(*)(1)/se>n/
(209) 3/sen(2)/se>n/
(210) 3/sen(*)(1)/san/
(211) 5/sein (*)(1)/sin/
(212) 4/sein(*)(1)/si.n/
(213) 3/se (2)/s2/
(214) 3/se (*)(1)/z/
(215) 7/sciène (*)(1)/cye<nfin/
(216) 3/sci(*)(1)/si/
(217) 5/sces (*)(1)/s2/
(218) 7/scents (*)(1)/san/
(219) 6/scent (*)(1)/san/
(220) 5/scen (*)(1)/san/
```

(221) 4/sce (*)(1)/s2/
(222) 2/sc(3)/s1/
(223) 2/sc(2)/s2/
(224) 2/sc(*)(1)/sk/
(225) 5/sans (*)(1)/san/
(226) 3/san(1)/san/
(227) 6/saint (*)(1)/sin/
(228) 5/sain (*)(1)/sin/
(229) 2/s (2)/s2/
(230) 2/s (*)(1)/muet/
(231) 1/s(3)/s1/
(232) 1/s(2)/s2/
(233) 1/s(1)/sz/
(234) 9/rtielles (*)(1)/rtiel/
(235) 8/rtielle (*)(1)/rtiel/
(236) 6/rtiell(*)(1)/rtiel/
(237) 6/rtiel (*)(1)/rtiel/
(238) 5/rtiel(*)(1)/rtiel/
(239) 8/rtiales (*)(1)/rtial/
(240) 7/rtiale (*)(1)/rtial/
(241) 6/rtial (*)(1)/rtial/
(242) 5/rtial(*)(1)/rtial/
(243) 5/rses (*)(1)/rs/
(244) 4/rse (*)(1)/rs/
(245) 5/rrhe (*)(1)/r/
(246) 3/rrh(*)(1)/r/
(247) 5/rres (*)(1)/r/
(248) 4/rre (*)(1)/r/

(249) 2/rr(*)(1)/r/
(250) 2/rh(*)(1)/r/
(251) 4/res (*)(1)/r/
(252) 3/re (*)(1)/r/
(253) 3/rd (*)(1)/r/
(254) 5/rces (*)(1)/rs/
(255) 4/rce (2)/s2/
(256) 4/rce (*)(1)/rs/
(257) 2/r (*)(1)/r/
(258) 1/r(1)/r/
(259) 5/ques (*)(1)/k/
(260) 4/que (2)/que%/
(261) 4/que (*)(1)/k/
(262) 3/qua(2)/qwa/
(263) 3/qua(*)(1)/ka/
(264) 2/qu(2)/qw/
(265) 2/qu(*)(1)/k/
(266) 1/q(1)/k/
(267) 2/pt(2)/pt/
(268) 2/pt(*)(1)/t/
(269) 5/ppes (*)(1)/p/
(270) 4/ppe (*)(1)/p/
(271) 2/pp(*)(1)/p/
(272) 5/phes (*)(1)/f/
(273) 4/phe (*)(1)/f/
(274) 2/ph(*)(1)/f/
(275) 4/pes (*)(1)/p/

(276) 3/pe (*)(1)/p/
(277) 2/p (2)/muet/
(278) 2/p (*)(1)/p/
(279) 1/p(1)/p/
(280) 2/oy(*)(1)/oy/
(281) 4/oux (*)(1)/ou/
(282) 6/ouses (*)(1)/ousefin/
(283) 5/ouse (*)(1)/ousefin/
(284) 5/ouill(*)(1)/ouill/
(285) 5/ouil (*)(1)/ouill/
(286) 3/ou (*)(1)/ou/
(287) 2/ou(*)(1)/ou/
(288) 6/onues (*)(1)/onu/
(289) 5/onue (*)(1)/onu/
(290) 4/onu (*)(1)/onu/
(291) 3/onu(*)(1)/onu/
(292) 4/ont (*)(1)/on/
(293) 3/ont(*)(1)/ont/
(294) 3/ono(*)(1)/ono/
(295) 7/onnues (*)(1)/onu/
(296) 6/onnue (*)(1)/onu/
(297) 5/onnu (*)(1)/onu/
(298) 4/onnu(*)(1)/onu/
(299) 4/onno(*)(1)/ono/
(300) 6/onnis (*)(1)/oni/
(301) 7/onnies (*)(1)/oni/
(302) 6/onnie (*)(1)/oni/
(303) 5/onni (*)(1)/oni/
(304) 4/onni(*)(1)/oni/
(305) 6/onnes (*)(1)/onefin/
(306) 8/onneaux (*)(1)/ono/
(307) 7/onneau (*)(1)/ono/
(308) 6/onneau(*)(1)/ono/
(309) 5/onne (*)(1)/onefin/
(310) 4/onne(*)(1)/one</
(311) 6/onnau (*)(1)/ono/

(312) 5/onnau(*)(1)/ono/
(313) 6/onnas (*)(1)/ona/
(314) 7/onnaie (*)(1)/one</
(315) 5/onnai(*)(1)/one</
(316) 4/onna(*)(1)/ona/
(317) 5/onis (*)(1)/oni/
(318) 6/onies (*)(1)/oni/
(319) 5/onie (*)(1)/oni/
(320) 4/oni (*)(1)/oni/
(321) 3/oni(*)(1)/oni/
(322) 5/ones (*)(1)/onefin/
(323) 7/oneaux (*)(1)/ono/
(324) 6/oneau (*)(1)/ono/
(325) 5/oneau(*)(1)/ono/
(326) 4/one (*)(1)/onefin/
(327) 3/one(*)(1)/one</
(328) 5/onau (*)(1)/ono/
(329) 4/onau(*)(1)/ono/
(330) 5/onas (*)(1)/ona/

(331) 5/onai (*)(1)/one</
(332) 4/onai(*)(1)/one</
(333) 3/ona(*)(1)/ona/
(334) 3/on (*)(1)/on/
(335) 2/on(*)(1)/on/
(336) 6/omues (*)(1)/omu/
(337) 5/omue (*)(1)/omu/
(338) 4/omu (*)(1)/omu/
(339) 3/omu(*)(1)/omu/
(340) 3/omt(*)(1)/ont/
(341) 4/ompt(*)(1)/ont/
(342) 3/omp(*)(1)/omp/
(343) 5/omos (*)(1)/omo/
(344) 4/omo (*)(1)/omo/
(345) 3/omo(*)(1)/omo/
(346) 7/ommues (*)(1)/omu/
(347) 6/ommue (*)(1)/omu/
(348) 5/ommu (*)(1)/omu/
(349) 4/ommu(*)(1)/omu/
(350) 6/ommos (*)(1)/omo/
(351) 5/ommo (*)(1)/omo/
(352) 4/ommo(*)(1)/omo/
(353) 6/ommis (*)(1)/omi/
(354) 7/ommies (*)(1)/omi/
(355) 6/ommie (*)(1)/omi/
(356) 5/ommi (*)(1)/omi/
(357) 4/ommi(*)(1)/omi/
(358) 6/ommes (*)(1)/omfin/
(359) 8/ommeaux (*)(1)/omo/
(360) 7/ommeau (*)(1)/omo/
(361) 6/ommeau(*)(1)/omo/
(362) 5/omme (*)(1)/omfin/
(363) 7/ommaux (*)(1)/omo/
(364) 6/ommau (*)(1)/omo/
(365) 5/ommau(*)(1)/omo/
(366) 6/ommas (*)(1)/oma/
(367) 7/ommais (*)(1)/ome</
(368) 6/ommai (*)(1)/ome</
(369) 5/ommai(*)(1)/ome</
(370) 5/omma (*)(1)/oma/
(371) 4/omma(*)(1)/oma/
(372) 5/omis (*)(1)/omi/
(373) 6/omies (*)(1)/omi/
(374) 5/omie (*)(1)/omi/
(375) 4/omi (*)(1)/omi/
(376) 3/omi(*)(1)/omi/
(377) 5/omes (*)(1)/omfin/
(378) 8/omeaux (*)(1)/omo/
(379) 6/omeau (*)(1)/omo/
(380) 5/omeau(*)(1)/omo/
(381) 4/ome (*)(1)/omfin/
(382) 3/omb(*)(1)/omb/
(383) 6/omaux (*)(1)/omo/
(384) 5/omau (*)(1)/omo/
(385) 4/omau(*)(1)/omo/
(386) 5/omas (*)(1)/oma/

```
(387) 6/omais (*)(1)/ome</
(388) 5/omai (*)(1)/ome</
(389) 4/omai(*)(1)/ome</
(390) 4/oma (*)(1)/oma/
(391) 3/oma(*)(1)/oma/
(392) 3/om (*)(1)/on/
(393) 6/oises (*)(1)/oisefin/
(394) 5/oise (*)(1)/oisefin/

(395) 4/oin (*)(1)/oin/
(396) 3/oin(*)(1)/oin/
(397) 5/oids (*)(1)/oi/
(398) 2/oi(*)(1)/oi/
(399) 3/oeu(*)(1)/oe/
(400) 5/oeill(*)(1)/oeil/
(401) 5/oeil (*)(1)/oeil/
(402) 3/où (*)(1)/ou/
(403) 2/où(*)(1)/ou/
(404) 2/oû(*)(1)/ou/
(405) 1/o(1)/o/
(406) 4/noux(*)(1)/nou/
(407) 4/nous(*)(1)/nou/
(408) 3/nou(*)(1)/nou/
(409) 5/nnoux(*)(1)/nou/
(410) 5/nnous(*)(1)/nou/
(411) 4/nnou(*)(1)/nou/
(412) 5/nnes (*)(1)/n/
(413) 4/nne (*)(1)/n/
(414) 2/nn(*)(1)/n/
(415) 4/nes (*)(1)/n/
(416) 3/ne (2)/ne%/
(417) 3/ne (*)(1)/n/
(418) 2/n (*)(1)/n/
(419) 1/n(1)/n/
(420) 4/mne (*)(1)/n/
(421) 2/mn(*)(1)/n/
(422) 5/mmes (*)(1)/m/
(423) 4/mme (*)(1)/m/
(424) 2/mm(*)(1)/m/
(425) 4/mes (2)/m/
(426) 4/mes (*)(1)/mes%/
(427) 3/me (2)/m/
(428) 3/me (*)(1)/mes%/
(429) 2/m (*)(1)/m/
(430) 1/m(1)/m/
(431) 6/lment (*)(1)/leman/
(432) 5/lmen (*)(1)/leman/
(433) 6/lmant (*)(1)/leman/
(434) 5/lman (*)(1)/leman/
(435) 2/ll(2)/y/
(436) 2/ll(*)(1)/l/
(437) 4/les (2)/l/
(438) 4/les (*)(1)/les%/
(439) 7/lement (*)(1)/leman/
(440) 3/le (2)/l/
(441) 3/le (*)(1)/le%/
```

(442) 3/l' (*)(1)/l/
(443) 2/l'(*)(1)/l/
(444) 1/l(1)/l/
(445) 4/kurr(*)(1)/cur/
(446) 4/kur (*)(1)/cur/
(447) 3/kur(*)(1)/cur/
(448) 3/ku (*)(1)/ku/
(449) 2/ku(*)(1)/ku/
(450) 2/ks(*)(1)/ks/
(451) 4/korr(*)(1)/cor/
(452) 4/kor (*)(1)/cor/
(453) 3/kor(*)(1)/cor/
(454) 2/kh(*)(1)/k/
(455) 4/karr(*)(1)/car/
(456) 4/kar (*)(1)/car/
(457) 3/kar(*)(1)/car/
(458) 2/ka(*)(1)/ka/
(459) 2/k (2)/muet/
(460) 2/k (*)(1)/k/
(461) 1/k(1)/k/
(462) 5/jois (*)(1)/joi/
(463) 4/joi (*)(1)/joi/
(464) 4/jes (*)(1)/j/
(465) 6/jeois (*)(1)/joi/
(466) 5/jeoi (*)(1)/joi/
(467) 3/jeo(2)/jo/
(468) 3/jeo(*)(1)/je<o/

(469) 3/je (2)/j/
(470) 3/je (*)(1)/je%/
(471) 3/j' (*)(1)/j/
(472) 2/j'(*)(1)/j/
(473) 1/j(*)(1)/j/
(474) 5/ions (*)(1)/dions/
(475) 4/ion (*)(1)/dions/
(476) 3/ion(*)(1)/yon/
(477) 6/inues (*)(1)/inu/
(478) 5/inue (*)(1)/inu/
(479) 4/inu (*)(1)/inu/
(480) 3/inu(*)(1)/inu/
(481) 4/inst(*)(1)/inst/
(482) 4/ino (*)(1)/ino/
(483) 3/ino(*)(1)/ino/
(484) 7/innues (*)(1)/inu/
(485) 6/innue (*)(1)/inu/
(486) 5/innu (*)(1)/inu/
(487) 4/innu(*)(1)/inu/
(488) 5/inno (*)(1)/ino/
(489) 4/inno(*)(1)/ino/
(490) 6/innis (*)(1)/ini/
(491) 7/innies (*)(1)/ini/
(492) 6/innie (*)(1)/ini/
(493) 5/inni (*)(1)/ini/
(494) 4/inni(*)(1)/ini/
(495) 8/inneaux (*)(1)/ino/
(496) 7/inneau (*)(1)/ino/

(497) 6/inneau(*)(1)/ino/
(498) 7/innaux (*)(1)/ino/
(499) 6/innau (*)(1)/ino/
(500) 5/innau(*)(1)/ino/
(501) 6/innas (*)(1)/ina/
(502) 7/innais (*)(1)/ine</
(503) 6/innai (*)(1)/ine</
(504) 5/innai(*)(1)/ine</
(505) 5/inna (*)(1)/ina/
(506) 4/inna(*)(1)/ina/
(507) 5/inis (*)(1)/ini/
(508) 6/inies (*)(1)/ini/
(509) 5/inie (*)(1)/ini/
(510) 4/ini (*)(1)/ini/
(511) 3/ini(*)(1)/ini/
(512) 5/ines (*)(1)/inefin/
(513) 7/ineaux (*)(1)/ino/
(514) 6/ineau (*)(1)/ino/
(515) 5/ineau(*)(1)/ino/
(516) 4/ine (*)(1)/inefin/
(517) 6/inaux (*)(1)/ino/
(518) 5/inau (*)(1)/ino/
(519) 4/inau(*)(1)/ino/
(520) 5/inas (*)(1)/ina/
(521) 6/inais (*)(1)/ine</
(522) 5/inai (*)(1)/ine</
(523) 4/inai(*)(1)/ine</
(524) 4/ina (*)(1)/ina/
(525) 3/ina(*)(1)/ina/
(526) 3/in (*)(1)/in/
(527) 2/in(*)(1)/in/
(528) 6/imues (*)(1)/imu/
(529) 5/imue (*)(1)/imu/
(530) 4/imu (*)(1)/imu/
(531) 3/imu(*)(1)/imu/
(532) 3/imp(*)(1)/imp/
(533) 5/imos (*)(1)/imo/
(534) 4/imo (*)(1)/imo/
(535) 3/imo(*)(1)/imo/
(536) 7/immues (*)(1)/imu/
(537) 6/immue (*)(1)/imu/
(538) 5/immu (*)(1)/imu/
(539) 4/immu(*)(1)/imu/
(540) 6/immos (*)(1)/imo/
(541) 5/immo (*)(1)/imo/
(542) 4/immo(*)(1)/imo/
(543) 6/immis (*)(1)/imi/
(544) 7/immies (*)(1)/imi/
(545) 6/immie (*)(1)/imi/
(546) 5/immi (*)(1)/imi/
(547) 4/immi(*)(1)/imi/
(548) 8/immeaux (*)(1)/imo/
(549) 6/immeau(*)(1)/imo/
(550) 7/immaux (*)(1)/imo/
(551) 6/immau (*)(1)/imo/
(552) 5/immau(*)(1)/imo/

(553) 7/immais (*)(1)/ime</
(554) 6/immai (*)(1)/ime</
(555) 5/immai(*)(1)/ime</
(556) 5/imma (*)(1)/ima/
(557) 4/imma(*)(1)/ima/
(558) 3/imm(*)(1)/i.m/
(559) 5/imis (*)(1)/imi/
(560) 6/imies (*)(1)/imi/
(561) 5/imie (*)(1)/imi/

(562) 4/imi (*)(1)/imi/
(563) 3/imi(*)(1)/imi/
(564) 7/imeaux (*)(1)/imo/
(565) 5/imeau(*)(1)/imo/
(566) 3/imb(*)(1)/imb/
(567) 6/imaux (*)(1)/imo/
(568) 5/imau (*)(1)/imo/
(569) 4/imau(*)(1)/imo/
(570) 6/imais (*)(1)/ime</
(571) 5/imai (*)(1)/ime</
(572) 4/imai(*)(1)/ime</
(573) 4/ima (*)(1)/ima/
(574) 3/ima(*)(1)/ima/
(575) 3/im (*)(1)/in/
(576) 2/im(*)(1)/i.m/
(577) 4/ils (*)(1)/ilfiny/
(578) 6/illes (2)/ilfin/
(579) 6/illes (*)(1)/illefin/
(580) 5/ille (2)/ilfin/
(581) 5/ille (*)(1)/illefin/
(582) 3/il (2)/ilfin/
(583) 3/il (*)(1)/ilfiny/
(584) 3/ika(*)(1)/ica/
(585) 4/iez (*)(1)/diez/
(586) 4/ies (*)(1)/i/
(587) 5/iens (*)(1)/yinfin/
(588) 7/iennes (*)(1)/ye<nfin/
(589) 6/ienne (*)(1)/ye<nfin/
(590) 4/ien (*)(1)/yinfin/
(591) 3/ie (*)(1)/i/
(592) 5/icas (*)(1)/ica/
(593) 4/ica (*)(1)/ica/
(594) 3/ica(*)(1)/ica/
(595) 2/i (*)(1)/i/
(596) 1/i(1)/i/
(597) 6/halle (*)(1)/al/
(598) 2/h (*)(1)/muet/
(599) 1/h(2)/hdeb/
(600) 1/h(1)/hmil/
(601) 2/gè(*)(1)/je</
(602) 3/gé (*)(1)/je</
(603) 2/gé(*)(1)/je</
(604) 5/gues (*)(1)/g/
(605) 4/gue (*)(1)/g/
(606) 5/gues (*)(1)/gufin/
(607) 4/gue (*)(1)/gufin/

(608) 3/gu (*)(1)/gufin/
(609) 2/gu(1)/gw/
(610) 2/gu(*)(2)/g/
(611) 3/grr(*)(1)/gr/
(612) 2/gr(*)(1)/gr/
(613) 5/gnes (*)(1)/gn/
(614) 4/gne (*)(1)/gn/
(615) 2/gn(*)(1)/gn/
(616) 2/gl(*)(1)/gl/
(617) 4/gin (*)(1)/jin/
(618) 3/gin(2)/jin/
(619) 3/gin(*)(1)/ji.n/
(620) 2/gi(*)(1)/ji/
(621) 5/gges (*)(1)/j/
(622) 4/gge (*)(1)/j/
(623) 2/gg(*)(1)/g/
(624) 4/ges (*)(1)/j/
(625) 4/ger (*)(1)/je</
(626) 6/geois (*)(1)/joi/

(627) 4/geoi(*)(1)/joi/
(628) 3/geo(2)/jo/
(629) 3/geo(*)(1)/je<o/
(630) 4/gen (*)(1)/jan/
(631) 3/gen(2)/jan/
(632) 3/gen(*)(1)/je<n/
(633) 3/ge (*)(1)/j/
(634) 2/ge(*)(1)/je>/
(635) 2/g (2)/g/
(636) 2/g (*)(1)/muet/
(637) 1/g(1)/g/
(638) 5/ffes (*)(1)/f/
(639) 4/ffe (*)(1)/f/
(640) 2/ff(*)(1)/f/
(641) 4/fes (*)(1)/f/
(642) 3/fe (*)(1)/f/
(643) 2/f (2)/f/
(644) 2/f (*)(1)/muet/
(645) 1/f(1)/f/
(646) 3/eûn(*)(1)/un/
(647) 3/eû (*)(1)/u/
(648) 3/ez (*)(1)/dez/
(649) 2/ey(*)(1)/ey/
(650) 2/ex(*)(1)/e<ks/
(651) 4/eux (*)(1)/oe/
(652) 6/euses (*)(1)/eusefin/
(653) 5/euse (*)(1)/eusefin/
(654) 4/eus (*)(1)/u/
(655) 5/eurs (*)(1)/eurfin/
(656) 6/eures (*)(1)/eurfin/
(657) 5/eure (*)(1)/eurfin/
(658) 4/eur (*)(1)/eurfin/
(659) 4/eun (*)(1)/un/
(660) 6/euill (*)(1)/oeil/
(661) 5/euill(*)(1)/oeil/
(662) 5/euil (*)(1)/oeil/

(663) 5/eues (*)(1)/u/
(664) 4/eue (2)/oe/
(665) 4/eue (*)(1)/u/
(666) 3/eu (2)/oe/
(667) 3/eu (*)(1)/u/
(668) 2/eu(*)(1)/oe/
(669) 6/ettes (*)(1)/ettefin/
(670) 5/ette (*)(1)/ettefin/
(671) 3/et (*)(1)/e</
(672) 4/est (*)(1)/e</
(673) 3/es (2)/muet/
(674) 3/es (*)(1)/e</
(675) 4/ert (*)(1)/e<r/
(676) 6/erres (*)(1)/e<r/
(677) 5/erre (*)(1)/e<r/
(678) 3/err(*)(1)/e<r/
(679) 6/erbes (*)(1)/erbefin/
(680) 4/erbe(*)(1)/erbefin/
(681) 3/er (2)/e</
(682) 3/er (*)(1)/e<r/
(683) 2/er(*)(1)/e<r/

(684) 5/eois (*)(1)/oi/
(685) 4/eoi (*)(1)/oi/
(686) 3/eoi(*)(1)/oi/
(687) 5/enus (*)(1)/enu/
(688) 6/enues (*)(1)/enu/
(689) 5/enue (*)(1)/enu/
(690) 4/enu (*)(1)/enu/
(691) 3/enu(*)(1)/enu/
(692) 4/enti(*)(1)/anti/
(693) 6/entes (*)(1)/ent/
(694) 5/ente (*)(1)/ent/
(695) 4/ent (2)/an/
(696) 4/ent (*)(1)/dent/
(697) 3/ent(*)(1)/ent/
(698) 8/ensions (*)(1)/ension/
(699) 7/ension (*)(1)/ension/
(700) 6/enses (*)(1)/ensefin/
(701) 5/ense (*)(1)/ensefin/
(702) 4/enou(*)(1)/e>nou/
(703) 5/enos (*)(1)/eno/
(704) 4/eno (*)(1)/eno/
(705) 3/eno(*)(1)/eno/
(706) 6/ennus (*)(1)/enu/
(707) 7/ennues (*)(1)/enu/
(708) 6/ennue (*)(1)/enu/
(709) 5/ennu (*)(1)/enu/
(710) 4/ennu(*)(1)/enu/
(711) 5/ennou(*)(1)/e>nou/
(712) 6/ennos (*)(1)/eno/
(713) 5/enno (*)(1)/eno/
(714) 4/enno(*)(1)/eno/
(715) 6/ennis (*)(1)/eni/
(716) 7/ennies (*)(1)/eni/
(717) 6/ennie (*)(1)/eni/

(718) 5/enni (*)(1)/eni/
(719) 4/enni(*)(1)/eni/
(720) 8/enneaux (*)(1)/eno/
(721) 7/enneau (*)(1)/eno/
(722) 6/enneau(*)(1)/eno/
(723) 5/enne (*)(1)/enefin/
(724) 7/ennaux (*)(1)/eno/
(725) 6/ennau (*)(1)/eno/
(726) 5/ennau(*)(1)/eno/
(727) 6/ennas (*)(1)/ena/
(728) 7/ennait (*)(1)/ene</
(729) 7/ennais (*)(1)/ene</
(730) 6/ennai (*)(1)/ene</
(731) 5/ennai(*)(1)/ene</
(732) 5/enna (*)(1)/ena/
(733) 4/enna(*)(1)/ena/
(734) 5/enis (*)(1)/eni/
(735) 6/enies (*)(1)/eni/
(736) 5/enie (*)(1)/eni/

(737) 4/eni (*)(1)/eni/
(738) 3/eni(*)(1)/eni/
(739) 7/enents (*)(1)/enan/
(740) 6/enent (*)(1)/enan/
(741) 4/enen(*)(1)/enan/
(742) 7/eneaux (*)(1)/eno/
(743) 6/eneau (*)(1)/eno/
(744) 5/eneau(*)(1)/eno/
(745) 4/ene (*)(1)/enefin/
(746) 3/ene(*)(1)/ene/
(747) 6/enaux (*)(1)/eno/
(748) 5/enau (*)(1)/eno/
(749) 4/enau(*)(1)/eno/
(750) 5/enas (*)(1)/ena/
(751) 7/enants (*)(1)/enan/
(752) 6/enant (*)(1)/enan/
(753) 5/enan (*)(1)/enan/
(754) 6/enait (*)(1)/ene</
(755) 6/enais (*)(1)/ene</
(756) 5/enai (*)(1)/ene</
(757) 4/enai(*)(1)/ene</
(758) 4/ena (*)(1)/ena/
(759) 3/ena(*)(1)/ena/
(760) 2/en(*)(1)/an/
(761) 5/emue (*)(1)/emu/
(762) 4/emu (*)(1)/emu/
(763) 3/emu(*)(1)/emu/
(764) 5/emps (*)(1)/an/
(765) 3/emp(*)(1)/amp/
(766) 4/emo (*)(1)/emo/
(767) 3/emo(*)(1)/emo/
(768) 6/emmue (*)(1)/emu/
(769) 5/emmu (*)(1)/emu/
(770) 5/emmo (*)(1)/emo/
(771) 6/emmis (*)(1)/emi/
(772) 7/emmies (*)(1)/emi/

(773) 6/emmie (*)(1)/emi/
(774) 5/emmi (*)(1)/emi/
(775) 4/emmi(*)(1)/emi/
(776) 6/emmai (*)(1)/eme</
(777) 5/emmai(*)(1)/eme</
(778) 5/emma (*)(1)/ema/
(779) 4/emma(*)(1)/ema/
(780) 5/emis (*)(1)/emi/
(781) 6/emies (*)(1)/emi/
(782) 5/emie (*)(1)/emi/
(783) 4/emi (*)(1)/emi/
(784) 3/emi(*)(1)/emi/
(785) 6/ements(*)(1)/eman/
(786) 5/ement(2)/mentfin/
(787) 5/ement(*)(1)/eman/
(788) 4/emen(*)(1)/eman/
(789) 7/emeaux (*)(1)/emo/
(790) 6/emeau (*)(1)/emo/
(791) 5/emeau(*)(1)/emo/
(792) 3/emb(*)(1)/amb/
(793) 6/emaux (*)(1)/emo/
(794) 5/emau (*)(1)/emo/
(795) 4/emau(*)(1)/emo/
(796) 6/emants(*)(1)/eman/
(797) 5/emant(*)(1)/eman/
(798) 4/eman(*)(1)/eman/
(799) 5/emai (*)(1)/eme</

(800) 4/emai(*)(1)/eme</
(801) 4/ema (*)(1)/ema/
(802) 3/ema(*)(1)/ema/
(803) 4/els (*)(1)/el/
(804) 6/elles (*)(1)/el/
(805) 5/elle (*)(1)/el/
(806) 3/ell(*)(1)/el/
(807) 3/el (*)(1)/el/
(808) 4/eis (*)(1)/e</
(809) 6/eines (*)(1)/einefin/
(810) 5/eine (*)(1)/einefin/
(811) 4/ein (*)(1)/in/
(812) 3/ein(*)(1)/in/
(813) 4/eill(*)(1)/eil/
(814) 4/eil (*)(1)/eil/
(815) 3/ei (*)(1)/e</
(816) 3/ecs(*)(1)/e<ks/
(817) 3/ec (*)(1)/e</
(818) 4/eaux(*)(1)/o/
(819) 7/eaunue (*)(1)/onu/
(820) 6/eaunu (*)(1)/onu/
(821) 5/eaunu(*)(1)/onu/
(822) 7/eaunos (*)(1)/ono/
(823) 6/eauno (*)(1)/ono/
(824) 5/eauno(*)(1)/ono/
(825) 9/eaunnues (*)(1)/onu/
(826) 8/eaunnue (*)(1)/onu/
(827) 7/eaunnu (*)(1)/onu/

(828) 6/eaunnu(*)(1)/onu/
(829) 6/eaunno(*)(1)/ono/
(830) 8/eaunnis (*)(1)/oni/
(831) 7/eaunni (*)(1)/oni/
(832) 6/eaunni(*)(1)/oni/
(833) 8/eaunnes (*)(1)/onefin/
(834)10/eaunneaux (*)(1)/ono/
(835) 8/eaunneau(*)(1)/ono/
(836) 7/eaunne (*)(1)/onefin/
(837) 9/eaunnaux (*)(1)/ono/
(838) 8/eaunnau (*)(1)/ono/
(839) 7/eaunnau(*)(1)/ono/
(840) 8/eaunnas (*)(1)/ona/
(841) 8/eaunnai (*)(1)/one</
(842) 7/eaunnai(*)(1)/one</
(843) 7/eaunna (*)(1)/ona/
(844) 6/eaunna(*)(1)/ona/
(845) 7/eaunis (*)(1)/oni/
(846) 6/eauni (*)(1)/oni/
(847) 5/eauni(*)(1)/oni/
(848) 7/eaunes (*)(1)/onefin/
(849) 9/eauneaux (*)(1)/ono/
(850) 7/eauneau(*)(1)/ono/
(851) 6/eaune (*)(1)/onefin/
(852) 8/eaunaux (*)(1)/ono/
(853) 7/eaunau (*)(1)/ono/
(854) 6/eaunau(*)(1)/ono/
(855) 7/eaunas (*)(1)/ona/
(856) 7/eaunai (*)(1)/one</
(857) 6/eaunai(*)(1)/one</
(858) 6/eauna (*)(1)/ona/
(859) 5/eauna(*)(1)/ona/
(860) 8/eaumues (*)(1)/omu/
(861) 7/eaumue (*)(1)/omu/

(862) 6/eaumu (*)(1)/omu/
(863) 5/eaumu(*)(1)/omu/
(864) 7/eaumos (*)(1)/omo/
(865) 6/eaumo (*)(1)/omo/
(866) 5/eaumo(*)(1)/omo/
(867) 9/eaummues (*)(1)/omu/
(868) 8/eaummue (*)(1)/omu/
(869) 7/eaummu (*)(1)/omu/
(870) 6/eaummu(*)(1)/omu/
(871) 8/eaummos (*)(1)/omo/
(872) 6/eaummo(*)(1)/omo/
(873) 8/eaummis (*)(1)/omi/
(874) 9/eaummies (*)(1)/omi/
(875) 8/eaummie (*)(1)/omi/
(876) 7/eaummi (*)(1)/omi/
(877) 6/eaummi(*)(1)/omi/
(878)10/eaummeaux (*)(1)/omo/
(879) 9/eaummeau (*)(1)/omo/
(880) 8/eaummeau(*)(1)/omo/
(881) 9/eaummais (*)(1)/ome</
(882) 8/eaummai (*)(1)/ome</

(883) 7/eaummai(*)(1)/ome</
(884) 7/eaumis (*)(1)/omi/
(885) 8/eaumies (*)(1)/omi/
(886) 7/eaumie (*)(1)/omi/
(887) 6/eaumi (*)(1)/omi/
(888) 5/eaumi(*)(1)/omi/
(889) 7/eaumeau(*)(1)/omo/
(890) 8/eaumais (*)(1)/ome</
(891) 7/eaumai (*)(1)/ome</
(892) 6/eaumai(*)(1)/ome</
(893) 3/eau(*)(1)/o/
(894) 2/e (*)(1)/muet/
(895) 1/e(2)/e>/
(896) 1/e(1)/e</
(897) 4/des (2)/d/
(898) 4/des (*)(1)/d%/
(899) 3/de (2)/d/
(900) 3/de (*)(1)/d%/
(901) 5/ddes (*)(1)/d/
(902) 4/dde (*)(1)/d/
(903) 2/dd(*)(1)/d/
(904) 3/d' (*)(1)/d/
(905) 2/d'(*)(1)/d/
(906) 2/d (2)/d/
(907) 2/d (*)(1)/muet/
(908) 1/d(1)/d/
(909) 2/cè(*)(1)/se</
(910) 3/cér(*)(1)/ce<r/
(911) 2/cé(*)(1)/se</
(912) 3/czé(*)(1)/gze>/
(913) 4/cus (*)(1)/ku/
(914) 4/curr(*)(1)/cur/
(915) 4/cur (*)(1)/cur/
(916) 3/cur(*)(1)/cur/
(917) 5/cues (*)(1)/ku/
(918) 4/cue (*)(1)/ku/
(919) 3/cu (*)(1)/ku/
(920) 7/ctions (*)(1)/ksyon/
(921) 6/ction (*)(1)/ksyon/
(922) 4/ctio(*)(1)/ksio/
(923) 3/cti(*)(1)/kti/
(924) 3/ct (*)(1)/muet/
(925) 2/cs(*)(1)/ks/
(926) 3/cqu(2)/kw/
(927) 3/cqu(*)(1)/k/
(928) 4/corr(*)(1)/cor/
(929) 4/cor (*)(1)/cor/
(930) 3/cor(*)(1)/cor/
(931) 3/cku(*)(1)/ku/
(932) 3/cin(2)/si.n/
(933) 3/cin(1)/sin/
(934) 6/ciens (*)(1)/cyinfin/
(935) 8/ciennes (*)(1)/cye<nfin/
(936) 7/cienne (*)(1)/cye<nfin/
(937) 5/cien (*)(1)/cyinfin/
(938) 2/ci(*)(1)/si/

(939) 4/chsi(*)(1)/ksi/

(940) 3/chs(*)(1)/ks/
(941) 5/ches (*)(1)/ch/
(942) 5/cherr(*)(1)/cher/
(943) 4/cher(*)(1)/cher/
(944) 4/che (*)(1)/ch/
(945) 2/ch(2)/k/
(946) 2/ch(*)(1)/ch/
(947) 4/ces (2)/se%/
(948) 4/ces (*)(1)/s2/
(949) 4/cerr(2)/ce>r/
(950) 4/cerr(*)(1)/ce<r/
(951) 3/cer(2)/ce>r/
(952) 3/cer(*)(1)/ce<r/
(953) 4/cenn(*)(1)/se>n/
(954) 3/cen(2)/se>n/
(955) 3/cen(*)(1)/san/
(956) 4/cein(1)/sin/
(957) 3/ce (2)/se%/
(958) 3/ce (*)(1)/s2/
(959) 2/ce(*)(1)/se/
(960) 3/ccé(*)(1)/kse</
(961) 5/ccur (*)(1)/cur/
(962) 4/ccur(*)(1)/cur/
(963) 5/ccor (*)(1)/cor/
(964) 4/ccor(*)(1)/cor/
(965) 3/cci(*)(1)/ksi/
(966) 6/ccent (*)(1)/ksan/
(967) 3/cce(2)/kse>/
(968) 3/cce(*)(1)/kse</
(969) 5/ccar (*)(1)/car/
(970) 4/ccar(*)(1)/car/
(971) 3/cca(*)(1)/ka/
(972) 2/cc(*)(1)/k/
(973) 4/carr(*)(1)/car/
(974) 4/car (*)(1)/car/
(975) 3/car(*)(1)/car/
(976) 2/c (2)/muet/
(977) 1/c(1)/k/
(978) 2/c (*)(1)/k/
(979) 4/bes (*)(1)/b/
(980) 3/be (*)(1)/b/
(981) 5/bbes (*)(1)/b/
(982) 4/bbe (*)(1)/b/
(983) 2/bb(*)(1)/b/
(984) 2/b (2)/muet/
(985) 1/b(1)/b/
(986) 2/b (*)(1)/b/
(987) 4/aît (*)(1)/e</
(988) 2/aî(*)(1)/e</
(989) 2/ay(*)(1)/ay/
(990) 4/aux (*)(1)/o/
(991) 5/aunu_(*)(1)/onu/
(992) 7/aunues (*)(1)/onu/
(993) 6/aunue (*)(1)/onu/

(994) 5/aunu (*)(1)/onu/
(995) 6/aunos (*)(1)/ono/
(996) 5/auno (*)(1)/ono/
(997) 4/auno(*)(1)/ono/
(998) 8/aunnues (*)(1)/onu/
(999) 7/aunnue (*)(1)/onu/
(1000) 5/aunnu(*)(1)/onu/
(1001) 5/aunno(*)(1)/ono/
(1002) 5/aunni(*)(1)/oni/
(1003) 7/aunnes (*)(1)/onefin/
(1004) 9/aunneaux (*)(1)/ono/
(1005) 8/aunneau (*)(1)/ono/
(1006) 6/aunne (*)(1)/onefin/
(1007) 7/aunnau (*)(1)/ono/
(1008) 6/aunnau(*)(1)/ono/
(1009) 7/aunnas (*)(1)/ona/
(1010) 7/aunnai (*)(1)/one</
(1011) 6/aunnai(*)(1)/one</
(1012) 5/aunna(*)(1)/ona/
(1013) 4/auni(*)(1)/oni/
(1014) 6/aunes (*)(1)/onefin/
(1015) 8/auneaux (*)(1)/ono/
(1016) 7/auneau (*)(1)/ono/

(1017) 6/auneau(*)(1)/ono/
(1018) 5/aune (*)(1)/onefin/
(1019) 6/aunau (*)(1)/ono/
(1020) 5/aunau(*)(1)/ono/
(1021) 6/aunas (*)(1)/ona/
(1022) 6/aunai (*)(1)/one</
(1023) 5/aunai(*)(1)/one</
(1024) 4/auna(*)(1)/ona/
(1025) 7/aumues (*)(1)/omu/
(1026) 6/aumue (*)(1)/omu/
(1027) 5/aumu (*)(1)/omu/
(1028) 4/aumu(*)(1)/omu/
(1029) 6/aumos (*)(1)/omo/
(1030) 5/aumo (*)(1)/omo/
(1031) 8/aummues (*)(1)/omu/
(1032) 7/aummue (*)(1)/omu/
(1033) 6/aummu (*)(1)/omu/
(1034) 5/aummu(*)(1)/omu/
(1035) 7/aummos (*)(1)/omo/
(1036) 6/aummo (*)(1)/omo/
(1037) 5/aummo(*)(1)/omo/
(1038) 7/aummis (*)(1)/omi/
(1039) 8/aummies (*)(1)/omi/
(1040) 7/aummie (*)(1)/omi/
(1041) 6/aummi (*)(1)/omi/
(1042) 5/aummi(*)(1)/omi/
(1043) 8/aummaux (*)(1)/omo/
(1044) 7/aummau (*)(1)/omo/
(1045) 6/aummau(*)(1)/omo/
(1046) 7/aummas (*)(1)/oma/
(1047) 8/aummais (*)(1)/ome</
(1048) 6/aummai(*)(1)/ome</

(1049) 6/aumma (*)(1)/oma/
(1050) 5/aumma(*)(1)/oma/
(1051) 6/aumis (*)(1)/omi/
(1052) 7/aumies (*)(1)/omi/
(1053) 6/aumie (*)(1)/omi/
(1054) 5/aumi (*)(1)/omi/
(1055) 4/aumi(*)(1)/omi/
(1056) 5/aumau(*)(1)/omo/
(1057) 6/aumas (*)(1)/oma/
(1058) 7/aumais (*)(1)/ome</
(1059) 6/aumai (*)(1)/ome</
(1060) 5/aumai(*)(1)/ome</
(1061) 5/auma (*)(1)/oma/
(1062) 4/auma(*)(1)/oma/
(1063) 2/au(*)(1)/o/
(1064) 7/ations (*)(1)/ationfin/
(1065) 6/ation (*)(1)/ationfin/
(1066) 5/ases (*)(1)/asefin/
(1067) 4/ase (*)(1)/asefin/
(1068) 4/art (*)(1)/ar/
(1069) 6/arres (*)(1)/ar/
(1070) 5/arre (*)(1)/ar/
(1071) 3/arr(*)(1)/ar/
(1072) 4/ard (*)(1)/ar/
(1073) 5/août (*)(1)/ou/
(1074) 6/anues (*)(1)/anu/

(1075) 5/anue (*)(1)/anu/
(1076) 4/anu (*)(1)/anu/
(1077) 3/anu(*)(1)/anu/
(1078) 5/ants (*)(1)/an/
(1079) 4/anti(*)(1)/anti/
(1080) 4/ant (*)(1)/an/
(1081) 8/ansions (*)(1)/ension/
(1082) 7/ansion (*)(1)/ension/
(1083) 6/anses (*)(1)/ensefin/
(1084) 5/anse (*)(1)/ensefin/
(1085) 4/ans (*)(1)/an/
(1086) 5/anos (*)(1)/ano/
(1087) 4/ano (*)(1)/ano/
(1088) 3/ano(*)(1)/ano/
(1089) 7/annues (*)(1)/anu/
(1090) 6/annue (*)(1)/anu/
(1091) 5/annu (*)(1)/anu/
(1092) 4/annu(*)(1)/anu/
(1093) 6/annos (*)(1)/ano/
(1094) 5/anno (*)(1)/ano/
(1095) 4/anno(*)(1)/ano/
(1096) 6/annis (*)(1)/ani/
(1097) 7/annies (*)(1)/ani/
(1098) 6/annie (*)(1)/ani/
(1099) 5/anni (*)(1)/ani/
(1100) 4/anni(*)(1)/ani/
(1101) 8/anneaux (*)(1)/ano/
(1102) 7/anneau (*)(1)/ano/
(1103) 7/annaux (*)(1)/ano/

(1104) 6/annau (*)(1)/ano/
(1105) 5/annau(*)(1)/ano/
(1106) 6/annai (*)(1)/ane</
(1107) 5/annai(*)(1)/ane</
(1108) 5/anna (*)(1)/ana/
(1109) 4/anna(*)(1)/ana/
(1110) 5/anis (*)(1)/ani/
(1111) 6/anies (*)(1)/ani/
(1112) 5/anie (*)(1)/ani/
(1113) 4/ani (*)(1)/ani/
(1114) 3/ani(*)(1)/ani/
(1115) 7/aneaux (*)(1)/ano/
(1116) 6/aneau (*)(1)/ano/
(1117) 4/and (*)(1)/an/
(1118) 5/ancs (*)(1)/an/
(1119) 4/anc (*)(1)/an/
(1120) 6/anaux (*)(1)/ano/
(1121) 4/anau(*)(1)/ano/
(1122) 5/anai (*)(1)/ane</
(1123) 4/anai(*)(1)/ane</
(1124) 4/ana (*)(1)/ana/
(1125) 3/ana(*)(1)/ana/
(1126) 3/an (*)(1)/an/
(1127) 2/an(*)(1)/an/
(1128) 6/amues (*)(1)/amu/
(1129) 5/amue (*)(1)/amu/
(1130) 4/amu (*)(1)/amu/
(1131) 3/amu(*)(1)/amu/
(1132) 4/amp (*)(1)/an/
(1133) 3/amp(*)(1)/amp/
(1134) 3/amo(*)(1)/amo/
(1135) 4/amo (*)(1)/amo/

(1136) 5/ammu (*)(1)/amu/
(1137) 4/ammu(*)(1)/amu/
(1138) 5/ammo (*)(1)/amo/
(1139) 4/ammo(*)(1)/amo/
(1140) 6/ammis (*)(1)/ami/
(1141) 5/ammi (*)(1)/ami/
(1142) 4/ammi(*)(1)/ami/
(1143) 8/ammeaux (*)(1)/amo/
(1144) 7/ammeau (*)(1)/amo/
(1145) 6/ammeau(*)(1)/amo/
(1146) 7/ammaux (*)(1)/amo/
(1147) 6/ammau (*)(1)/amo/
(1148) 5/ammau(*)(1)/amo/
(1149) 5/ammai(*)(1)/ame</
(1150) 5/amma (*)(1)/ama/
(1151) 4/amma(*)(1)/ama/
(1152) 3/amm(*)(1)/a.m/
(1153) 5/amis (*)(1)/ami/
(1154) 6/amies (*)(1)/ami/
(1155) 5/amie (*)(1)/ami/
(1156) 4/ami (*)(1)/ami/
(1157) 3/ami(*)(1)/ami/
(1158) 7/ameaux (*)(1)/amo/

```
(1159) 6/ameau (*)(1)/amo/
(1160) 5/ameau(*)(1)/amo/
(1161) 3/amb(*)(1)/amb/
(1162) 6/amaux (*)(1)/amo/
(1163) 5/amau (*)(1)/amo/
(1164) 4/amau(*)(1)/amo/
(1165) 4/amai(*)(1)/ame</
(1166) 4/ama (*)(1)/ama/
(1167) 3/ama(*)(1)/ama/
(1168) 3/am (2)/a.m/
(1169) 3/am (*)(1)/an/
(1170) 2/am(*)(1)/a.m/
(1171) 3/al (*)(1)/al/
(1172) 4/ait (*)(1)/e</
(1173) 7/aisons (*)(1)/aisonfin/
(1174) 6/aison (*)(1)/aisonfin/
(1175) 4/ais (*)(1)/e</
(1176) 4/air (*)(1)/e<r/
(1177) 6/ainés (*)(1)/e<ne</
(1178) 7/ainées (*)(1)/e<ne</
(1179) 6/ainée (*)(1)/e<ne</
(1180) 4/ainé(*)(1)/e<ne</
(1181) 6/aints (*)(1)/in/
(1182) 5/aint (*)(1)/in/
(1183) 5/ains (*)(1)/in/
(1184) 6/ainne (*)(1)/enefin/
(1185) 6/aines (*)(1)/enefin/
(1186) 5/aine (*)(1)/enefin/
(1187) 5/ainc (*)(1)/in/
(1188) 4/ain (*)(1)/in/
(1189) 3/ain(*)(1)/in/
(1190) 5/aims (*)(1)/in/
(1191) 4/aim (*)(1)/in/
(1192) 4/aill(*)(1)/aill/
(1193) 4/ail (2)/ailfin/
(1194) 4/ail (*)(1)/aill/
(1195) 6/aient (*)(1)/e</
(1196) 4/aie (*)(1)/e</
(1197) 2/ai(*)(1)/e</
(1198) 3/ai (*)(1)/e</
(1199) 6/agges (*)(1)/agefin/
(1200) 5/agge (*)(1)/agefin/
(1201) 5/ages (*)(1)/agefin/
(1202) 4/age (*)(1)/agefin/
(1203) 4/aen (*)(1)/an/
(1204) 2/ae(*)(1)/e</
(1205) 1/a(1)/a/
(1206) 1/_(1)/tiret/
(1207) 2/: (*)(1)/muet%/
(1208) 1/.(1)/muet%/
(1209) 2/. (*)(1)/muet%/
(1210) 1/-(1)/tiret/
(1211) 2/' (1)/muet%/
(1212) 1/'(1)/muet%/
(1213) 1/ (1)/ /
```

ANNEXE 3

REGLES DE COMPOSITION GRAPHIQUE

```
z
 1s010
 2se001
 1z111
 2ze001
 3***
y
 2hi110
 1i011
 2il001
 1ï010
 2ll010
 3lle001
 1y111
 3***
w.in
 3oin111
 4oint001
 4ouin111
 3***
w.a.y
 2oy010
 3***
w.a
 3eoi011
 4eois001
 2oi111
 3oit001
 3***
w
 1w111
 3***
v
 1v111
 2ve001
 1w110
 3***
un
 3eun010
 3eûn011
 2um001
 2un011
 3***
u
 1u111
 1û010
 2hu111
 3***
t
 1t111
 2te001
 2té001
 2th110
 3the001
 2tt010
 3tte001
```

```
  3***
 s
  1c110
  2ce001
  1ç010
  1s111
  2sc110
  3sce001
  2se001
  2ss011
  3sse001
  1t010
  3***
 r
  1r111
  2rd001
  2re001
  2rh110
  2rr010
  3rre001
  2rt001
  3***
 p
  1p111
  2pe001
  2pp010
  3ppe001
  3***
 ou
  2ou111
  3ous001
  3out001
  3oux001
  2oû010
  3hou100
  3***
on.t
  4hont100
  5honte001
  4ompt010
  5ompte001
  3omt010
  4omte001
  3ont110
  4onte001
  3***
on.p
  3omp010
  4ompe001
  3***
on.b
  3omb110
  4ombe001
  3***
on
  2om001
```

```
  2on111
  3ont001
  3***
 oe
  2eu111
  3eux001
  2oe110
  3oeu111
  3***
 o
  2au111
  3aud001
  3aux011
  3eau011
  4eaux001
  3hau100
  2ho100
  2hô100
  1o111
  2ot001
  1ô110
  3***
n
 2mn010
 3mne001
 1n111
 2ne001
 2nh010
 2nn010
 3nne001
 3***
m
 1m111
 2me001
 2mm010
 3mme001
 3***
l.m.an
 5leman011
 5lemen011
 6lement001
 4lman011
 5lmant001
 4lmen011
 5lment001
 3***
l.i.t.e<
 5liter001
 4lité111
 5littél11
 5llité011
 3***
l
 11111
 2le001
 2ll010
```

```
 3lle001
 3***
k.w.in
 4coin011
 5coing001
 3***
k.w
 3cqu010
 2qu110
 3***
k.s
 2cc010
 2cs010
 2ct010
 2ks010
 1x111
 2xe001
 3***
k
 1c111
 2cc010
 2ch111
 3cqu011
 1k111
 2kh110
 2qu110
 3que001
 3***
j
 1g110
 2ge001
 2gg010
 3gge001
 1j111
 3***
ing
 3ing001
 3***
in.p
 3imp110
 3***
in.b
 3imb110
 3***
in
 3aim001
 3ain011
 4ainc001
 4ains001
 4aint001
 3ein011
 2im001
 2in111
 2ym010
 2yn010
 3***
```

```
i
 2hi111
 2hy100
 1i111
 2it001
 2iz001
 1ï011
 1î010
 1y010
 3***
gn
 2gn110
 3gne001
 3***
g.z
 2cz010
 1x110
 2xs010
 3***
g
 1g111
 2gg010
 2gu110
 3gue001
 3***
f
 1f111
 2fe001
 2ff011
 3ffe001
 2ph111
 3phe001
 3***
e>
 1e110
 1é110
 3***
e<.y
 2ay110
 3eil001
 4eill011
 2ey011
 3***
e<
 2ae001
 2ai111
 3aie001
 5aient001
 3ais001
 3ait001
 2aî110
 3aît001
 1e110
 2ei010
 2er001
 3est001
```

```
 2et001
 2ez001
 3hai111
 2he110
 2hé100
 1é111
 1è010
 1ê110
 3***
d
 1d111
 2dd010
 3dde001
 2de001
 3***
ch
 2ch111
 3che001
 2sh111
 3***
b.l
 2bl110
 3ble001
 3***
b
 1b111
 2bb010
 3bbe001
 2be001
 2bh011
 3***
an.p
 3amp110
 4ampe001
 3emp110
 3***
an.b
 3amb110
 4ambe001
 3emb110
 3***
an
 3aen001
 2am001
 3amp001
 2an111
 3anc001
 3and001
 3ans001
 3ant001
 4emps001
 2en111
 3ent001
 3han111
 3hen111
 3***
```

```
a
 1a111
 2ha111
 2hâ100
 1â110
 3***
-
 1-010
 3***
```

ANNEXE 4

META-REGLES
DE VERIFICATION SYNTAXIQUE

```
/*    FICHIER VDMETA.PRO    - contient les meta-règles de vérification */
project "verifier.prj"
include "verifier.glo"  /* déclarations globales de domaines et prédicats */
/*_____________________________________________________________
 OBS. - Les meta-règles éffectuent un groupement entre des règles ayant le
      même texte ou un texte similaire. Pour certains cas, les meta-règles
      font directement l'appel aux propres règles */
 CLAUSES
 mregle(Gauche,Droit,ms(MotRes,ClsRes,R),Rep):-
    prep_apl_reg(Gauche,Droit,NouvGauche,NouvDroit,MotRes,ClsRes)
   , regle(NouvGauche,NouvDroit,R,Rep).
/*-------------------------------------------------------------------*/
/* Préparation pour l'application des règles:                        */

prep_apl_reg(ms(MotGauche,deto,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,deto,Gauche),ms(MotDroit,gnomo,Droit),
        MotDroit,gnomo):-!.
prep_apl_reg(ms(MotGauche,adjo,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotDroit,gnomo,Droit),ms(MotGauche,adjo,Gauche),
        MotDroit,gnomo):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,adjo,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,adjo,Droit),
        MotGauche,gnomo):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,verbo,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,verbo):-!.
prep_apl_reg(ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,verbo):-!.
prep_apl_reg(ms(MotGauche,verbo,Gauche),ms(MotDroit,prepo,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,prepo,Droit),
        MotGauche,verbo):-!.
prep_apl_reg(ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
        ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,verbo):-!.
prep_apl_reg(ms(MotGauche,detpo,Gauche),ms(MotDroit,verbo,Droit),
        ms(MotGauche,detpo,Gauche),ms(MotDroit,verbo,Droit),
```

```
        MotDroit,verbo):-!.
prep_apl_reg(ms(MotGauche,detpo,Gauche),ms(MotDroit,xavo,Droit),
        ms(MotGauche,detpo,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xavo):-!.
prep_apl_reg(ms(MotGauche,popel,Gauche),ms(MotDroit,xavso,Droit),
        ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xavso):-!.
prep_apl_reg(ms(MotGauche,prepo,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,prepo,Gauche),ms(MotDroit,gnomo,Droit),
        MotDroit,gnomo):-!.
prep_apl_reg(ms(MotGauche,detpo,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,deto,Gauche),ms(MotDroit,gnomo,Droit),
        MotDroit,gnomo):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,prepo,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,prepo,Droit),
        MotGauche,gnomo):-!.
prep_apl_reg(ms(MotGauche,popel,Gauche),ms(MotDroit,xeto,Droit),
        ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xeto):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,xeto,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xeto):-!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,xeto):-!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
        MotGauche,xeto):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,cocod,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,cocod,Droit),
        MotDroit,cocod):-!.
prep_apl_reg(ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
        MotDroit,gnomo):-!.
prep_apl_reg(ms(MotGauche,popel,Gauche),ms(MotDroit,xavo,Droit),
        ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xavo):-!.
```

```
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,xavo,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xavo):-!.
prep_apl_reg(ms(MotGauche,xavo,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,xavo):-!.
prep_apl_reg(ms(MotGauche,xavo,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xavo,Gauche),ms(MotDroit,ppaso,Droit),
        MotDroit,ppaso):-!.
prep_apl_reg(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        MotGauche,xavso):-
        not(verif("SET",Gauche)).
prep_apl_reg(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        MotGauche,xavso):-
        verif("SET",Gauche)
       , verif("inf",Droit).
prep_apl_reg(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
        MotDroit,ppaso):-
        verif("SET",Gauche)
       , not(verif("inf",Droit)) , ! .
prep_apl_reg(ms(MotGauche,xetso,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
        MotGauche,xetso):-!.
prep_apl_reg(ms(MotGauche,xetso,Gauche),ms(MotDroit,adjo,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
        MotGauche,xetso):-!.
prep_apl_reg(ms(MotGauche,ppaso,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,ppaso,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,ppaso):-!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
        MotDroit,ppaso):-not(verif("OBJ",Droit)),!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,xavo,Gauche),ms(MotDroit,ppaso,Droit),
```

```
        MotDroit,ppaso):-verif("OBJ",Droit),!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,adjo,Droit),
        MotGauche,gnomo):-!.
prep_apl_reg(ms(MotGauche,ppaso,Gauche),ms(MotDroit,prepo,Droit),
        ms(MotGauche,ppaso,Gauche),ms(MotDroit,prepo,Droit),
        MotGauche,ppaso):-!.
prep_apl_reg(ms(MotGauche,verbo,Gauche),ms(MotDroit,cocod,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,cocod,Droit),
        MotDroit,cocod):-!.
prep_apl_reg(ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
        ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,verbo):-!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,cocod,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,cocod,Droit),
        MotDroit,cocod):-!.
prep_apl_reg(ms(MotGauche,cocod,Gauche),ms(MotDroit,xeto,Droit),
        ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xeto):-!.
prep_apl_reg(ms(MotGauche,adjo,Gauche),ms(MotDroit,cocod,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,cocod,Droit),
        MotGauche,cocod):-!.   /* prend l'adjo comme mot gauche */
prep_apl_reg(ms(MotGauche,cocod,Gauche),ms(MotDroit,adjo,Droit),
        ms(MotGauche,cocod,Gauche),ms(MotDroit,adjo,Droit),
        MotDroit,adjo):-!.
prep_apl_reg(ms(MotGauche,xavo,Gauche),ms(MotDroit,popel,Droit),
        ms(MotDroit,popel,Droit),ms(MotGauche,verbo,Gauche),
        MotGauche,xavo):-!.
prep_apl_reg(ms(MotGauche,xavo,Gauche),ms(MotDroit,detpo,Droit),
        ms(MotDroit,detpo,Droit),ms(MotGauche,verbo,Gauche),
        MotGauche,xavo):-!.
prep_apl_reg(ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Droit),
        MotDroit,ppaso):-!.
prep_apl_reg(ms(MotGauche,relco,Gauche),ms(MotDroit,xavso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Droit),
        MotGauche,xavso):-!.
```

```
prep_apl_reg(ms(MotGauche,porel,Gauche),ms(MotDroit,ppaso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Droit),
        MotDroit,ppaso):-!.
prep_apl_reg(ms(MotGauche,porel,Gauche),ms(MotDroit,verso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Ndroit),
        MotDroit,porel):-
        difference(Droit,["set"],Ndroit),!.
prep_apl_reg(ms(MotGauche,porel,Gauche),ms(MotDroit,xetso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Ndroit),
        MotDroit,porel):-
        difference(Droit,["set"],Ndroit),!.
prep_apl_reg(ms(MotGauche,porel,Gauche),ms(MotDroit,xavso,Droit),
        ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Ndroit),
        MotDroit,porel):-
        difference(Droit,["set"],Ndroit),!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,porel,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,porel,Droit),
        MotGauche,gnomo):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,verso,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verso,Droit),
        MotDroit,verso):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,xetso,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verso,Droit),
        MotDroit,xetso):-!.
prep_apl_reg(ms(MotGauche,gnomo,Gauche),ms(MotDroit,xavso,Droit),
        ms(MotGauche,gnomo,Gauche),ms(MotDroit,verso,Droit),
        MotDroit,xavso):-!.
prep_apl_reg(ms(MotGauche,verso,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,verso):-!.
prep_apl_reg(ms(MotGauche,xavso,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,xavso):-!.
prep_apl_reg(ms(MotGauche,xetso,Gauche),ms(MotDroit,gnomo,Droit),
        ms(MotGauche,xeto,Gauche),ms(MotDroit,gnomo,Droit),
        MotGauche,xetso):-!.
prep_apl_reg(ms(MotGauche,neg1,Gauche),ms(MotDroit,verbo,Droit),
```

```
        ms(MotGauche,neg1,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,verbo):-!.
prep_apl_reg(ms(MotGauche,neg1,Gauche),ms(MotDroit,xavo,Droit),
        ms(MotGauche,neg1,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xavo):-!.
prep_apl_reg(ms(MotGauche,neg1,Gauche),ms(MotDroit,xeto,Droit),
        ms(MotGauche,neg1,Gauche),ms(MotDroit,verbo,Droit),
        MotDroit,xeto):-!.
prep_apl_reg(ms(MotGauche,verbo,Gauche),ms(MotDroit,neg2,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,neg2,Droit),
        MotGauche,verbo):-!.
prep_apl_reg(ms(MotGauche,xavo,Gauche),ms(MotDroit,neg2,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,neg2,Droit),
        MotGauche,xavo):-!.
prep_apl_reg(ms(MotGauche,xeto,Gauche),ms(MotDroit,neg2,Droit),
        ms(MotGauche,verbo,Gauche),ms(MotDroit,neg2,Droit),
        MotGauche,xeto):-!.
prep_apl_reg(ms(MotGauche,ppaso,Gauche),ms(MotDroit,infio,Droit),
        ms(MotGauche,ppaso,Gauche),ms(MotDroit,infio,Droit),
        MotGauche,ppaso):-!.
```

ANNEXE 5

REGLES DE VERIFICATION SYNTAXIQUE

```
/*-----------------------------------------------------------------
 Fichier VDREGLES.PRO, contient les règles de vérification
 -----------------------------------------------------------------*/
code=2048
project "verifier.prj"
include "verifier.glo"    /* Déclarations globales de domaines et prédicats */
/*-----------------------------------------------------------------*/
CLAUSES
/*--- REGLE_1 ----------------------------------------------------------*/
regle(ms(MotGauche,deto,Gauche),ms(MotDroit,gnomo,Droit),
    R,Rep) :-
 /* SI */  coraccord(ms(MotGauche,deto,Gauche),
                ms(MotDroit,gnomo,Droit),
                 ["GNR","NBR"],R1,Rep)
       , ecr_res(1,ms(MotGauche,deto,Gauche),
              ms(MotDroit,gnomo,Droit))
 /*ALORS*/
       , somme([R1,["art","tre"]],R).
/*--- REGLE_2 ----------------------------------------------------------*/
regle(ms(MotGauche,gnomo,Gauche),ms(MotDroit,adjo,Droit),
    R,Rep) :-
 /* SI */ verif("ART",Gauche)
       , coraccord(ms(MotGauche,gnomo,Gauche),
                ms(MotDroit,adjo,Droit),
                 ["GNR","NBR"],R1,Rep)
       , ecr_res(2,ms(MotGauche,gnomo,Gauche),
              ms(MotDroit,adjo,Droit))
/* ALORS */, affect("GNO","NUL",Gauche,[],R6)
       , somme([R1,["tre"],R6],R).
/*--- REGLE_3 ----------------------------------------------------------*/
regle(ms(MotGauche,gnomo,Gauche),ms(MotDroit,verbo,Droit),
    R,Rep) :-
/* SI */  not(verif("SET",Droit))
       , not(verif("IMP",Droit))
       , verif("ART",Gauche)
       , coraccord(ms(MotGauche,gnomo,Gauche),
              ms(MotDroit,verbo,Droit),
```

```
                ["NBR","PRS"],R1,Rep)
      , ecr_res(3,ms(MotGauche,gnomo,Gauche),
                ms(MotDroit,verbo,Droit))
/* ALORS */, affect("TPS","NUL",Droit,[],R5)
       , affect("MOD","NUL",Droit,[],R6)
       , affect("GNR","NUL",Gauche,[],R7)
       , somme([R1,R5,R6,R7,["set"]],R).
/*--- REGLE_4A ----------------------------------------------------------*/
/* Prend l'objet direct                                          */
/* Deux versions : soit verif("SET",Gauche), soit verif("IMP",Gauche)    */
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI  */ verif("SET",Gauche)
       , verif("ART",Droit)
       , not(verif("OBJ",Gauche))
       , ecr_res(4,ms(MotGauche,verbo,Gauche),
                ms(MotDroit,gnomo,Droit))
/*ALORS*/ , affect("NBR","NUL",Gauche,[],R1)
        , affect("PRS","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Gauche,[],R3)
        , affect("MOD","NUL",Gauche,[],R4)
        , affect("VBO","UNI",Gauche,["obj"],R5)
        , somme([R1,R2,R3,R4,R5],R).
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI  */ verif("IMP",Gauche)
       , verif("ART",Droit)
       , not(verif("OBJ",Gauche))
       , ecr_res(4,ms(MotGauche,verbo,Gauche),
                ms(MotDroit,gnomo,Droit))
 /*ALORS*/ , affect("NBR","NUL",Gauche,[],R1)
        , affect("PRS","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Gauche,[],R3)
        , affect("MOD","NUL",Gauche,[],R4)
        , affect("VBO","UNI",Gauche,["obj"],R5)
        , somme([R1,R2,R3,R4,R5],R).
```

```
/*--- REGLE_4B ----------------------------------------------------------*/
/* Deux versions : soit verif("SET",Gauche), soit verif("IMP",Gauche)   */
/* Prend l'objet indirect ?                                             */
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ verif("SET",Gauche)
      , verif("PRP",Droit)
      , ecr_res(4,ms(MotGauche,verbo,Gauche),
            ms(MotDroit,gnomo,Droit))
/*ALORS*/ , affect("PRS","NUL",Gauche,[],R1)
       , affect("TPS","NUL",Gauche,[],R2)
       , affect("MOD","NUL",Gauche,[],R3)
       , affect("NBR","NUL",Gauche,[],R4)
       , affect("VBO","NUL",Gauche,[],R5)
       , somme([R1,R2,R3,R4,R5],R).
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ verif("IMP",Gauche)
      , verif("PRP",Droit)
      , ecr_res(4,ms(MotGauche,verbo,Gauche),
            ms(MotDroit,gnomo,Droit))
/*ALORS*/ , affect("PRS","NUL",Gauche,[],R1)
       , affect("TPS","NUL",Gauche,[],R2)
       , affect("MOD","NUL",Gauche,[],R3)
       , affect("NBR","NUL",Gauche,[],R4)
       , affect("VBO","NUL",Gauche,[],R5)
       , somme([R1,R2,R3,R4,R5],R).
/*--- REGLE_5 -----------------------------------------------------------*/
/* Deux versions : soit verif("SET",Gauche), soit verif("IMP",Gauche)   */
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,prepo,Droit),
    R,ac) :-
/* SI */ verif("SET",Gauche)
      , ecr_res(5,ms(MotGauche,verbo,Gauche),
            ms(MotDroit,prepo,Droit))
/*ALORS*/ , affect("PRS","NUL",Gauche,[],R1)
       , affect("TPS","NUL",Gauche,[],R2)
       , affect("MOD","NUL",Gauche,[],R3)
```

```
        , affect("NBR","NUL",Gauche,[],R4)
        , affect("VBO","NUL",Gauche,[],R5)
        , affect("GNR","NUL",Gauche,[],R6)
        , somme([R1,R2,R3,R4,R5,R6],R).
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,prepo,Droit),
    R,ac) :-
/* SI */ verif("IMP",Gauche)
        , ecr_res(5,ms(MotGauche,verbo,Gauche),
                ms(MotDroit,prepo,Droit))
/*ALORS*/ , affect("PRS","NUL",Gauche,[],R1)
        , affect("TPS","NUL",Gauche,[],R2)
        , affect("MOD","NUL",Gauche,[],R3)
        , affect("NBR","NUL",Gauche,[],R4)
        , affect("VBO","NUL",Gauche,[],R5)
        , somme([R1,R2,R3,R4,R5],R).
/*--- REGLE_6 ----------------------------------------------------------*/
regle(ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
    R,Rep) :-
/* SI */  verif("SUJ",Gauche)
        , not(verif("SET",Droit))
        , not(verif("IMP",Droit))
        , coraccord(ms(MotGauche,popel,Gauche),
                ms(MotDroit,verbo,Droit),
                ["NBR","PRS"],R1,Rep)
        , ecr_res(6,ms(MotGauche,popel,Gauche),
                ms(MotDroit,verbo,Droit))
/* ALORS */, affect("GNR","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Droit,[],R5)
        , affect("MOD","NUL",Droit,[],R6)
        , somme([R1,R2,R5,R6,["set"]],R).
/*--- REGLE_7 ----------------------------------------------------------*/
regle(ms(MotGauche,detpo,Gauche),ms(MotDroit,verbo,Droit),
    R,ac) :-
/* SI */  verif("COD",Gauche)
        , not(verif("DAT",Gauche))
        , verif("SET",Droit)
        , not(verif("COD",Droit))
```

```
      , not(verif("CPR",Droit))
      , ecr_res(7,ms(MotGauche,detpo,Gauche),
            ms(MotDroit,verbo,Droit))
/*ALORS*/ , affect("PRS","NUL",Droit,[],R1)
      , affect("TPS","NUL",Droit,[],R2)
      , affect("MOD","NUL",Droit,[],R3)
      , affect("NBR","NUL",Droit,[],R4)
      , affect("CAS","UNI",Droit,["cod"],R5)
      , affect("VBO","NUL",Droit,[],R6)
       , somme([R1,R2,R3,R4,R5,R6],R).
/*--- REGLE_8 ----------------------------------------------------------*/
  /*  Deux versions: - troisième personne - les autres personnes     */
regle(ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
    R,ac) :-
       /*     Pour la troisième personne!
            exemple: je le lui donne
                ( ... .. lui ____) */
/* SI  */ verif("TRE",Gauche)
      , verif("DAT",Gauche)
      , verif("SET",Droit)
      , not(verif("DAT",Droit))
      , not(verif("CPR",Droit))
      , ecr_res(8,ms(MotGauche,popel,Gauche),
            ms(MotDroit,verbo,Droit))
 /*ALORS*/ , affect("PRS","NUL",Droit,[],R1)
      , affect("TPS","NUL",Droit,[],R2)
      , affect("MOD","NUL",Droit,[],R3)
      , affect("NBR","NUL",Droit,[],R4)
      , affect("CAS","UNI",Droit,["dat"],R5)
      , affect("VBO","NUL",Droit,[],R7)
      , affect("GNR","NUL",Droit,[],R8)
       , somme([R1,R2,R3,R4,R5,R7,R8],R).
regle(ms(MotGauche,popel,Gauche),ms(MotDroit,verbo,Droit),
    R,ac) :-
         /* toutes les personnes à l'exception de la troisième */
    /* SI */ not(verif("TRE",Gauche))
        , verif("DAT",Gauche)
```

```
        , verif("SET",Droit)
        , not(verif("DAT",Droit))
        , not(verif("CPR",Droit))
        , ecr_res(8,ms(MotGauche,popel,Gauche),
                ms(MotDroit,verbo,Droit))
    /* ALORS */ , affect("PRS","NUL",Droit,[],R1)
            , affect("TPS","NUL",Droit,[],R2)
            , affect("MOD","NUL",Droit,[],R3)
            , affect("NBR","NUL",Droit,[],R4)
            , affect("CAS","UNI",Droit,["dat"],R5)
            , affect("VBO","NUL",Droit,[],R7)
            , affect("GNR","NUL",Droit,[],R8)
            , somme([R1,R2,R3,R4,R5,R7,R8],R).
/*--- REGLE_9 ----------------------------------------------------------*/
regle(ms(MotGauche,prepo,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ ecr_res(9,ms(MotGauche,prepo,Gauche),
            ms(MotDroit,gnomo,Droit))
/*ALORS*/ , affect("GNR","NUL",Droit,[],R1)
        , affect("NBR","NUL",Droit,[],R2)
        , somme([R1,R2,["prp"]],R).
/*--- REGLE_10 ---------------------------------------------------------*/
regle(ms(MotGauche,gnomo,Gauche),ms(MotDroit,prepo,Droit),
    R,ac) :-
/* SI */ ecr_res(10,ms(MotGauche,gnomo,Gauche),
                ms(MotDroit,prepo,Droit))
/*ALORS*/ , affect("GNR","NUL",Gauche,[],R1)
        , affect("NBR","NUL",Gauche,[],R2)
        , affect("GNO","NUL",Gauche,[],R3)
        , affect("PRS","NUL",Gauche,[],R4)
        , somme([R1,R2,R3,R4],R).
/*------REGLE_11-----------------------------------------------------------*/
/*      Deux versions:  - être avec sujet - être dans l'impératif       */
regle(ms(MotGauche,xeto,Gauche), ms(MotDroit,gnomo,Droit),
    R,Rep) :-
/* SI */ verif("SET",Gauche)
    , coraccord(ms(MotGauche,xeto,Gauche),
```

```
            ms(MotDroit,gnomo,Droit),
             ["NBR","GNR"],R1,Rep)
     , ecr_res(11,ms(MotGauche,xeto,Gauche),
             ms(MotDroit,gnomo,Droit))
/* ALORS */ , affect("PRS","NUL",Gauche,[],R3)
        , affect("TPS","NUL",Gauche,[],R4)
        , affect("MOD","NUL",Gauche,[],R5)
        , affect("VBO","NUL",Gauche,[],R6)
        , somme([R1,R3,R4,R5,R6,["obj"]],R).
regle(ms(MotGauche,xeto,Gauche),ms(MotDroit,gnomo,Droit),
   R,Rep) :-
/* SI */ verif("IMP",Gauche)
     , not(verif("SET",Gauche))
     , coraccord(ms(MotGauche,xeto,Gauche),
            ms(MotDroit,gnomo,Droit),
             ["NBR"],R1,Rep)
     , ecr_res(11,ms(MotGauche,xeto,Gauche),
             ms(MotDroit,gnomo,Droit))
/* ALORS */ , affect("PRS","NUL",Gauche,[],R3)
        , affect("TPS","NUL",Gauche,[],R4)
        , affect("MOD","NUL",Gauche,[],R5)
        , affect("VBO","NUL",Gauche,[],R6)
        , somme([R1,R3,R4,R5,R6,["obj"]],R).
/*-------REGLE_12-----------------------------------------------------*/
/* 2 formes: être dans l'impératif ou pas                          */
regle(ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
   R,Rep) :-
/* SI */ verif("IMP",Gauche)
     , coraccord(ms(MotGauche,xeto,Gauche),
            ms(MotDroit,adjo,Droit),
             ["NBR"],R1,Rep)
     , ecr_res(12,ms(MotGauche,xeto,Gauche)
             ,ms(MotDroit,adjo,Droit))
/* ALORS */, affect("PRS","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Gauche,[],R3)
        , affect("MOD","NUL",Gauche,[],R4)
        , affect("VBO","NUL",Gauche,[],R5)
```

```
        , somme([R1,R2,R3,R4,R5],R).
regle(ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
   R,Rep) :-
/* SI */ not(verif("IMP",Gauche))
     , verif("SET",Gauche)
     , coraccord(ms(MotGauche,xeto,Gauche),
            ms(MotDroit,adjo,Droit),
             ["NBR","GNR"],R1,Rep)
     , ecr_res(12,ms(MotGauche,xeto,Gauche)
            ,ms(MotDroit,adjo,Droit))
/* ALORS */, affect("PRS","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Gauche,[],R3)
        , affect("MOD","NUL",Gauche,[],R4)
        , affect("VBO","NUL",Gauche,[],R5)
        , somme([R1,R2,R3,R4,R5],R).
/* cette troisième version accepte et corrige les phrases comme:
   ELLES SE SONT LAVE LES MAINS.                    */
regle(ms(MotGauche,xeto,Gauche),ms(MotDroit,adjo,Droit),
    R,Rep) :-
/* SI */ not(verif("IMP",Gauche))
     , verif("SET",Gauche)
     , verif("OBJ",Droit)
     , coraccord(ms(MotGauche,xeto,Gauche),
            ms(MotDroit,adjo,Droit),
             ["INV"],R1,Rep)
     , ecr_res(12,ms(MotGauche,xeto,Gauche)
            ,ms(MotDroit,adjo,Droit))
/* ALORS */, affect("PRS","NUL",Gauche,[],R2)
        , affect("TPS","NUL",Gauche,[],R3)
        , affect("MOD","NUL",Gauche,[],R4)
        , affect("VBO","NUL",Gauche,[],R5)
        , somme([R1,R2,R3,R4,R5],R).
/*_____REGLE_13_____________________________________________*/
regle(ms(MotGauche,gnomo,Gauche),ms(MotDroit,cocod,Droit),
    Gauche,ac) :-
       /* le 'et' (coco) prend son gnomo gauche */
/* SI*/ ecr_res(13,ms(MotGauche,gnomo,Gauche)
```

```
            ,ms(MotDroit,cocod,Droit))
/* ALORS */.
/*__________REGLE_14_____________________________________________*/
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ verif("art",Droit)
     , verif("art",Gauche)
     , verif("fem",Gauche)
     , verif("fem",Droit)
     , ecr_res(14,ms(MotGauche,cocod,Gauche)
            ,ms(MotDroit,gnomo,Droit))
/* ALORS */ , somme([["fem"],["plu"],["tre"],["art"]],R).
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */  verif("art",Droit)
      , verif("art",Gauche)
      , ecr_res(14,ms(MotGauche,cocod,Gauche)
            ,ms(MotDroit,gnomo,Droit))
/* ALORS */ , somme([["tre"],["mas"],["plu"],["art"]],R).
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ not(verif("art",Droit))
     , not(verif("art",Gauche))
     , verif("fem",Gauche)
     , verif("fem",Droit)
     , ecr_res(14,ms(MotGauche,cocod,Gauche)
            ,ms(MotDroit,gnomo,Droit))
/* ALORS */ , somme([["fem"],["plu"],["tre"],["art"]],R).
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */  not(verif("art",Droit))
      , not(verif("art",Gauche))
      , ecr_res(14,ms(MotGauche,cocod,Gauche)
            ,ms(MotDroit,gnomo,Droit))
/* ALORS */ , somme([["tre"],["mas"],["plu"],["art"]],R).
/*_______REGLE_15_____________________________________________*/
regle(ms(MotGauche,xavo,Gauche),ms(MotDroit,ppaso,Droit),
```

```
   R,Rep) :-
   /*      auxiliaire avoir xavo prend un ppaso comme attribut.
          Les subordinations comme :
             LES FLEURS QUE J'AI T'OFFERTES.
          ... sont corrigées par le règles que traitent xavso et xetso! */
/* SI */ not(verif("IMP",Gauche))
     , verif("SET",Gauche)
     , coraccord(ms(MotGauche,xavo,Gauche),
              ms(MotDroit,ppaso,Droit),
              ["INV"],R1,Rep)  /*  INV = Invariant ( toujours masc sin ) */
     , ecr_res(15,ms(MotGauche,xavo,Gauche)
           ,ms(MotDroit,ppaso,Droit))
/* ALORS */ , affect("VBO","NUL",Droit,[],R5)
          , affect("CAS","NUL",Droit,[],R6)
          , somme([R1,R5,R6],R).
/*_________REGLE_16_____________________________________________________*/
regle(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
     R,Rep):-
    /*  Relative introduite par le pronom relatif "qui" où le nom précedent
        est le sujet réel du verbe de la relative, comme par exemple:
        LA FILLE QUI A VU UN CHAT.                              */
/* SI */ not(verif("SET",Gauche))
      , coraccord(ms(MotGauche,xavo,Gauche),
              ms(MotDroit,ppaso,Droit),
              ["INV"],R1,Rep)  /*  INV = Invariant ( toujours masc sin ) */
      , ecr_res(16,ms(MotGauche,xavo,Gauche)
            ,ms(MotDroit,ppaso,Droit))
/* ALORS */ , affect("PRS","NUL",Gauche,[],R21)
          , affect("TPS","NUL",Gauche,[],R22)
          , affect("MOD","NUL",Gauche,[],R23)
          , affect("NBR","NUL",Gauche,[],R24)
          , affect("VBO","NUL",Gauche,[],R5)
          , affect("CAS","NUL",Gauche,[],R6)
           , somme([R1,R21,R22,R23,R24,R5,R6],R).
regle(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
     R,ac) :-
     /*      auxiliaire avoir xavso prend un ppaso comme attribut.
```

```
        Traite les subordinations comme :
          LES FLEURS QUE J'AI T'OFFERTES.                    */
/* SI */ not(verif("IMP",Gauche))
     , verif("SET",Gauche)
     , not(verif("INF",Droit))
     , ecr_res(16,ms(MotGauche,xavo,Gauche)
          ,ms(MotDroit,ppaso,Droit))
/* ALORS */ , affect("VBO","NUL",Droit,[],R5)
         , affect("CAS","NUL",Droit,[],R6)
         , affect("GNR","NUL",Droit,[],R1)
         , affect("NBR","NUL",Droit,[],R11)
         , somme([R1,R11,R5,R6],R).
regle(ms(MotGauche,xavso,Gauche),ms(MotDroit,ppaso,Droit),
    R,Rep):-
    /*  Part. Passé suivi d'un infinitif dans une relative:
      LE CHAT QUI NOUS AVONS VU MANGER.                      */
/* SI */ verif("SET",Gauche)
     , verif("INF",Droit)
     , coraccord(ms(MotGauche,xavo,Gauche),
          ms(MotDroit,ppaso,Droit),
          ["INV"],R1,Rep) /*  INV = Invariant ( toujours masc sin ) */
     , ecr_res(16,ms(MotGauche,xavo,Gauche)
          ,ms(MotDroit,ppaso,Droit))
/* ALORS */ , affect("PRS","NUL",Gauche,[],R21)
         , affect("TPS","NUL",Gauche,[],R22)
         , affect("MOD","NUL",Gauche,[],R23)
         , affect("NBR","NUL",Gauche,[],R24)
         , affect("VBO","NUL",Gauche,[],R5)
         , affect("CAS","NUL",Gauche,[],R6)
          , somme([R1,R21,R22,R23,R24,R5,R6],R).
/*_____________REGLE_17_____________________________________*/
regle(ms(MotGauche,ppaso,Gauche),ms(MotDroit,gnomo,Droit),
    R,ac) :-
/* SI */ verif("ART",Droit)
     , ecr_res(17,ms(MotGauche,ppaso,Gauche),
          ms(MotDroit,gnomo,Droit))
/* ALORS */ , affect("GNR","NUL",Gauche,[],R1)
```

```
          , affect("NBR","NUL",Gauche,[],R2)
          , somme([R1,R2,["obj"]],R).
/*_____REGLE_18_____*/
regle(ms(MotGauche,ppaso,Gauche),ms(MotDroit,prepo,Droit),
    R,ac) :-
/* SI */ ecr_res(18,ms(MotGauche,ppaso,Gauche)
              ,ms(MotDroit,prepo,Droit))
/* ALORS */ , affect("NBR","NUL",Gauche,[],R1)
          , affect("GNR","NUL",Gauche,[],R2)
          , somme([R1,R2,["dat"]],R).
/*_____REGLE_19_____*/
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,cocod,Droit),
    Gauche,ac) :-
/* deux cas: - le verbe à l'impératif ou le verbe avec sujet;        */
/* SI */    verif("IMP",Gauche)
         , ecr_res(19,ms(MotGauche,verbo,Gauche)
                ,ms(MotDroit,cocod,Droit))
/* ALORS */.
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,cocod,Droit),
    Gauche,ac) :-
/* SI */ verif("SET",Gauche)
     , ecr_res(19,ms(MotGauche,verbo,Gauche)
            ,ms(MotDroit,cocod,Droit))
/* ALORS */.
/*_____REGLE_20_____*/
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
    Droit,ac) :-
/* Trois cas: - le verbe à droite a déjà pris son sujet; le verbe à droite
 est à l'impératif; le verbe à droite doit prendre le même sujet du verbe
 à gauche de la coco */
/* SI */ verif("SET",Droit)
     , ecr_res(20,ms(MotGauche,cocod,Gauche)
          ,ms(MotDroit,verbo,Droit))
/* ALORS */.
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
    Droit,ac) :-
/* SI */  verif("IMP",Droit)
```

```
        , ecr_res(20,ms(MotGauche,cocod,Gauche),
              ms(MotDroit,verbo,Droit))
 /* ALORS */.
 regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,verbo,Droit),
     R,Rep) :-
 /* SI */ not(verif("SET",Droit))
      , coraccord(ms(MotGauche,cocod,Gauche),
              ms(MotDroit,verbo,Droit),
               ["NBR","PRS"],R1,Rep)
      , ecr_res(20,ms(MotGauche,cocod,Gauche),
              ms(MotDroit,verbo,Droit))
 /* ALORS */ , affect("GNR","NUL",Gauche,[],R2)
          , affect("TPS","NUL",Droit,[],R3)
          , affect("MOD","NUL",Droit,[],R4)
          , affect("CAS","NUL",Droit,[],R5)
          , somme([R1,R2,R3,R4,R5],R).
/*_____REGLE_21_____________________________________________________*/
regle(ms(MotGauche,cocod,Gauche),ms(MotDroit,adjo,Droit),
     R,Rep) :-
/* SI */ coraccord(ms(MotGauche,cocod,Gauche),
              ms(MotDroit,adjo,Droit),
               ["NBR","GNR"],R,Rep)
      , ecr_res(21,ms(MotGauche,cocod,Gauche),
              ms(MotDroit,adjo,Droit)).
/* ALORS */ /* résultat déjà calculé par l'accord */
/*_____REGLE_22_____________________________________________________*/
regle(ms(MotGauche,relco,Gauche),ms(MotDroit,ppaso,Droit),
     Droit,ac) :-
/* SI */ ecr_res(22,ms(MotGauche,relco,Gauche),
              ms(MotDroit,ppaso,Droit)).
/* ALORS */ /* le résultat sont toutes les informations de la liste droite */
/*_____REGLE_23_____________________________________________________*/
regle(ms(MotGauche,neg1,Gauche),ms(MotDroit,verbo,Droit),
     R,ac) :-
/* SI */ ecr_res(23,ms(MotGauche,neg1,Gauche),
              ms(MotDroit,verbo,Droit))
/* ALORS */ , union(Droit,["NE1"],R).
```

```
/*_____REGLE_24_____*/
regle(ms(MotGauche,verbo,Gauche),ms(MotDroit,neg2,Droit),
    Gauche,ac) :-
/* SI */ verif("NE1",Gauche)
     , ecr_res(24,ms(MotGauche,verbo,Gauche),
          ms(MotDroit,neg2,Droit))
/* ALORS */ .
/*_____REGLE_25_____*/
regle(ms(MotGauche,gnomo,Gauche),ms(MotDroit,verso,Droit),
    Res,Rep) :-
/* SI */  not(verif("SET",Droit))
      , not(verif("IMP",Droit))
     /*  , verif("ART",Gauche) enlevée pour permettre accepter le gnomo
                    copié à gauche d'un verso, xetso ou xavso */
      , union(Gauche,["tre"],Rgauche)  /* pour permettre de verifier l'accord
                                  de "NBR" */
      , coraccord(ms(MotGauche,gnomo,Rgauche),
            ms(MotDroit,verso,Droit),
             ["NBR","PRS"],R1,Rep)
      , ecr_res(25,ms(MotGauche,gnomo,Gauche),
            ms(MotDroit,verso,Droit))
/* ALORS */, affect("TPS","NUL",Droit,[],R2)
        , affect("MOD","NUL",Droit,[],R3)
        , affect("GNR","NUL",Gauche,[],R4)
        , somme([R1,R2,R3,R4,["set"]],Res).
/*_____REGLE_26_____*/
regle(ms(MotGauche,ppaso,Gauche),ms(MotDroit,infio,Droit),
    R,ac) :-
/* SI */ ecr_res(26,ms(MotGauche,ppaso,Gauche),
            ms(MotDroit,infio,Droit))
/* ALORS */ , union(Gauche,["inf"],R).
/*_____REGLE_27_____*/
regle(ms(_,gnomo,Gauche),ms(_,porel,_),
    Gauche,ac). /* Fin de l'analyse de la relative */
```

ANNEXE 6

LE SYSTEME *PILAF*

INTRODUCTION

L'environnement **PILAF** (**P**rocédures **I**nteractives **L**inguistiques **A**ppliquées au **F**rançais) est une version entièrement nouvelle du système PIAF [COURTIN 77]. PILAF dispose d'un analyseur morpho-syntaxique qui calcule la ou les structures de dépendances associées à une phrase et dispose également d'autres outils de traitement des langues naturelles, comme la génération morphologique ou des mécanismes de détection/correction d'erreurs. Le but est de construire des outils de traitement :

- adaptables et extensibles : pour pouvoir les utiliser dans de nombreuses applications ;
- transportables et efficaces : pour éviter que le choix du matériel ne décide d'une implantation complètement nouvelle ;
- interfaçables : avec d'autres langages comme LISP ou PROLOG en vue d'applications plus larges.

L'analyseur morpho-syntaxique de PILAF est construit modulairement en deux parties, morphologie et syntaxe, qui sont relativement indépendantes. L'analyse morphologique de PILAF est réalisée par un transducteur d'états finis lequel peut être utilisé aussi dans d'autres applications. L'analyse syntaxique de PILAF est réalisé par un analyseur de dépendances.
Nous consacrons la première partie de cette annexe à l'analyse morphologique et la deuxième à l'analyse syntaxique.

Les renseignements contenus dans cette annexe sont obtenus de [COURTIN 77, COURTIN 86, COURTIN 89d].

1 L'ANALYSE MORPHOLOGIQUE

1.1 Le transducteur général d'états finis

Nous présentons dans ce paragraphe le transducteur d'états finis utilisé dans l'analyseur morphologique de PILAF, décrit en détail dans [COURTIN 77].

Si on envisage l'analyse morphologique, le transducteur de PILAF a pour buts :

- de segmenter une chaîne de caractères (phrase) pour obtenir ses chaînes composantes ;
- de déterminer un certain nombre de renseignements linguistiques sur les chaînes obtenues par la segmentation.

Ce transducteur est composé de deux dictionnaires, d'une grammaire et d'une liste de modèles.

Nous donnons ici quelques définitions concernant les chaînes de caractères, la segmentation et la transduction dans l'analyse morphologique. Ensuite, nous exposons le fonctionnement du transducteur de PILAF.

1.1.1 Définitions

Vocabulaire

Soit un ensemble fini V, appelé vocabulaire, contenant tous les caractères :

$$V = \{' ', A, B, ..., Z, ., ;, ..., 0, 1, ..., 9\}$$

où ' ' représente l'espace.

Chaîne

On appelle chaîne sur V une suite finie de symboles de V :

$$x_1 x_2 x_3 ..... x_n$$

où $x_i \in V$ et $i \in [1,n]$.

V* et V+

On désigne par V* l'ensemble de toutes les chaînes que l'on peut construire sur V, y compris la chaîne vide notée L.

On désigne par V^+ l'ensemble V* - {L}.

On définit une relation d'ordre sur les symboles de V notée <. On prend la convention :

$$... < , < ... < ' ' < A < B < ... < Z < ... < 0 < 1 < ... < 9$$

Longueur

La longueur d'une chaîne α, notée LONGUEUR(α), est une application de V* dans N (ensemble des entiers naturels).

Si $\alpha = L$ alors LONGUEUR(α) = 0
sinon si $\alpha = \alpha_1 \lambda$ alors LONGUEUR(α) = 1 + LONGUEUR (λ)
$\forall \alpha_1 \in V$ et $\forall \lambda \in V^*$.

Préfixe

Le préfixe d'une chaîne est une application de $V^* \times V^*$ dans V*.
Soient $\alpha , \beta \in V^*$.

Soient $a, b \in V$, $\lambda, \gamma, \upsilon \in V^*$.

Si $\alpha = a\lambda$ et $\beta = b\lambda$ alors PREFIXE(α,β) = L
Si $\alpha = \gamma\lambda$ et $\beta = \gamma\upsilon$ alors PREFIXE(α,β) = γ PREFIXE(λ,υ)

Relation inférieur

On définit une relation inférieur < entre deux chaînes non-vides.
Soient α et β deux éléments de V^+, avec $\alpha \neq \beta$.
Soient $a_1, a_2 \in V$, $\lambda, \gamma, \upsilon \in V^*$.

Si $\alpha = \gamma a_1 \lambda$ et $\beta = \gamma a_2 \upsilon$ avec $a_1 \neq a_2$,
$\alpha < \beta$ si et seulement si $a_1 < a_2$, sinon $\beta < \alpha$.
Si $\beta = \alpha a_2 \lambda$ alors $\alpha < \beta$.
Si $\alpha = \beta a_1 \lambda$ alors $\beta < \alpha$.

Cette relation est transitive :

si $\alpha < \beta$ et $\beta < \delta$,
alors $\alpha < \delta \quad \forall \alpha, \beta, \delta \in V^+$.

1.1.2 Segmentation et transduction

Etant donnée une phrase

$$a_1 a_2 \ldots a_n \quad (\forall i \in [1,n], a_i \in V)$$

on doit déterminer des mots ou groupes de mots m_i ($m_i \in V^+$) tels que :

$$m_1 m_2 \ldots m_k = a_1 a_2 \ldots a_n \quad \text{avec } k \leq n.$$

Le principe de la segmentation employée dans le transducteur de PILAF comprend la détermination de m_1 et contrôle par application de règles de grammaire, puis itération du processus jusqu'au point de fin de phrase.
La décomposition d'un m_i n'est pas unique. Exemple : *livre* peut être décomposé comme le substantif *livre* ou comme *livr* + *e* (verbe régulier *livrer*).

La transduction donne les interprétations de toutes les décompositions d'un m_i.
Par exemple, les décompositions associées à la chaîne *livre* ont pour interprétation :

a) substantif commun, masculin, singulier ;
b) substantif commun, féminin, singulier ;
c) verbe *livrer* à la troisième personne du singulier du présent de l'indicatif ou à la troisième personne du singulier du présent du subjonctif.

Cette transduction est réalisée par la grammaire, qui vérifie la segmentation d'un m_i étant donné que la plupart des éléments constituant un m_i sont porteurs d'information.

1.1.3 Grammaires à validations et saturations

Nous allons reprendre ici les concepts présentés dans [COURTIN 77].

Une grammaire à validations et saturations est définie par :

$$GVS = (V_{TVS}, S_{VS}, R_{VS}, E)$$

V_{TVS} : vocabulaire terminal {a,b,c,...,z}
S_{VS} : symbole initial, élément de P(E) x P(E)
Exemple : S=[1,2][ ]
P_{VS} : ensemble des productions (règles)
E : ensemble des noms de règles, E = {1,2,3,....,n}

P_{VS} est une application de E dans V_{TVS} x P(E) x P(E).
Un élément de P_{VS} a la forme : $\mathbf{e} \rightarrow \mathbf{v}\ [\mathbf{e}_1...\mathbf{e}_i...\mathbf{e}_p]\ [\mathbf{f}_1...\mathbf{f}_j...\mathbf{f}_q]$
où $\mathbf{e}, \mathbf{e}_i, \mathbf{f}_j \in E$, $1 \leq i \leq p$, $1 \leq j \leq q$ et $\mathbf{v} \in V_{TVS}$.

Les e_i sont les validations et définissent les règles applicables après application de la règle **e**. Les f_j sont les saturations et définissent les règles interdites après application de la règle **e**.
Exemple (extrait de [COURTIN 77]) :

V_{TVS} = {a, b, c, d, f, y, z, t, u}
E = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
S_{VS} = [1,2] [∅]
P_{VS} :

1	→	f[2][6,10]	2	→	a[3][∅]
3	→	b[3,4][∅]	4	→	c[5,6][∅]
5	→	d[7][∅]	6	→	e[5][9]
7	→	y[7,8][∅]	8	→	z[9,10][∅]
9	→	t[∅][∅]	10	→	u[∅][∅]

Un **état** d'une dérivation est un objet de la forme :

A[liste de validations][liste de saturations]

où $\mathbf{A} \in V^*$ (suite de symboles de V, éventuellement vide).

Pour passer d'un état **i** à un état **i+1**, on choisit dans la liste des validations de **i** une règle e_j qu'on applique à **i** de la manière suivante :

- $A_{i+1} = A_i$ + le symbole de V en partie droite de e_j ;
- saturations de i+1 = union des saturations de A_i et des saturations de la partie droite de e_j ;
- validations de i+1 = intersection des validations de la partie droite de e_j et du complémentaire des saturations de i+1, c'est-à-dire les validations de la partie droite de e_j sauf celles qui sont des saturations de i+1.

La dérivation s'arrête dès que la liste des validations de l'état courant est vide. Voici un exemple de dérivation extrait de [COURTIN 86] :

GVS = ({a,b,c,d},[1,2][],P,{1,2,3,4})
avec P :

1 → a[1,2,3][]
2 → b[2,3][1]
3 → c[1,3,4][2]
4 → d[][]

Etat initial : [1,2][]
On choisit une validation :

1 → a[1,2,3][]
1 → aa[1,2,3][]
2 → aab[2,3][1] union des saturations ([] U [1])
3 → aabc[3,4][1,2] 1 est une saturation : elle est éliminée
4 → aabcd[][1,2]

Après application de 2, on ne peut plus appliquer 1, ce qui se traduit par la suppression de 1 de la liste des validations associée à la règle 3.

Le transducteur de PILAF utilise les règles de la même façon qu'une GVS mais en tenant compte du modèle et du code morphologique. Le code morphologique est une liste de règles. Cette liste est associée à un système de désinences qui pourra être activé dans certaines conditions. Pour passer d'un état i à un état i+1 en appliquant la règle *r*, et pour le modèle *m*, on utilise les formules :

$$SAT_{i+1} = \text{union} \quad (SAT_i, SAT_r, SAT_m)$$
$$VAL_{i+1} = \text{union} \quad (VAL_r, VAL_m) - SAT_{i+1}$$

où

SAT_i : les saturations associées à l'état i ;
SAT_{i+1} : les saturations associées à l'état i+1 ;
SAT_r : les saturations associées à la règle *r* ;

SAT_m : les saturations associées au modèle m ;
VAL_{i+1} : les validations associées à l'état i ;
VAL_r : les validations associées à la règle r ;
VAL_m : les validations associées au modèle m ;

1.1.4 Utilisation du transducteur

Des compilateurs et des éditeurs de règles et de modèles autorisent une mise au point interactive des paramètres linguistiques du transducteur de PILAF.
Ce transducteur est entièrement paramétré, ce qui a permis de l'utiliser dans de nombreuses applications, parmi lesquelles :
- traduction d'un texte en braille abrégé,
- traduction d'un texte en morse abrégé,
- traduction de programmes écrits en PL360 en programmes équivalents écrits en LP80,
- traduction d'une chaîne en un codage pseudo-phonétique,
- analyse morphologique d'une autre langue.

Ce transducteur est réversible. Il permet ainsi d'engendrer toutes les formes associées à une même base morphologique. La génération morphologique est utilisée pour corriger certains types d'erreurs dans DECOR (paragraphe 3.2 du chapitre 1).

1.2 Fonctionnement de l'analyse morphologique

L'analyse morphologique est réalisée par le transducteur général d'états finis, composé d'un dictionnaire, d'une grammaire et d'une liste de modèles. Les buts de l'analyse morphologique sont les suivants :
- segmenter une suite de caractères en une suite de chaînes ;
- déterminer les renseignements linguistiques associés à chaque chaîne, c'est à dire effectuer une transduction.

Les segments (unités lexicales) obtenus ne peuvent sont "insegmentables". Pour une phrase donnée, on obtient un nombre unique de segments. Cette restriction est imposée par l'analyseur syntaxique qui doit connaître le nombre de segments de la phrase.

On ne peut envisager de calculer toutes les segmentations et de les fournir à l'analyseur syntaxique sans porter atteinte à l'efficacité globale de l'analyse. On ne calcule donc qu'une segmentation en essayant de la rendre la plus cohérente possible avec le modèle syntaxique.
Exemple :

Il y a à ce propos une controverse.

se décompose en *Il + y + a + à ce propos + une + controverse + .*
On notera que le blanc n'est pas un séparateur, ce qui permet de considérer *à ce propos* comme une unité lexicale. Ceci est cohérent puisqu'elle n'apparaît que sous cette forme (les trois mots sont inséparables).

En français, chaque segment s peut être composé de : base ou racine, suffixe(s) et désinence.
Exemple :
retrouvera se décompose en :
préfixe : **re**
base : **trouv**
suffixe : **er**
désinence : **a**
Les parties "suffixe" et "désinence" peuvent être vides.

La détermination d'un segment s consiste donc à trouver sa décomposition.
On dispose à cet effet :
- d'un dictionnaire de bases et d'un dictionnaire de suffixes et désinences qui permettent par identification la décomposition d'une partie contiguë de la chaîne d'entrée ;
- d'une grammaire chargée d'infirmer ou de confirmer la concaténation des divers éléments ;
- de modèles permettant de réaliser l'interface entre la grammaire et les dictionnaires.

L'algorithme de l'analyseur morphologique est alors le suivant :
1 Déterminer un segment s (décomposition) à l'aide des dictionnaires ;
2 Valider ce segment avec la grammaire :
- s'il est correct passer au segment suivant ;
- sinon retourner en 1.

La transduction est une interprétation du segment s obtenu : à chacune des parties (base, suffixe, désinence) sont associées des informations linguistiques qui pourront être utilisées par les modules ultérieurs.
Exemple :

"COUVENT " -->	base "COUVENT" : substantif commun masculin	
	désinence " " : singulier	
"COUVENT " -->	base "COUV" : verbe 1er groupe	
	désinence "ENT " :	3ème pers. pluriel présent indicatif
	ou	3ème pers. pluriel présent subjonctif

On notera que la décomposition d'un segment n'est pas unique, les ambiguïtés seront traitées par les outils ultérieurs : syntaxique voire sémantique.
Cette transduction est réalisée par la grammaire qui vérifie la décomposition et rend les informations linguistiques sur les éléments constitutifs du segment.

1.3 Les dictionnaires

La structure du dictionnaire va refléter un trait particulier de l'analyse du langage naturel : on ne dispose pas d'un symbole séparateur comme en compilation. Pour déterminer la segmentation d'une phrase, on doit donc procéder par essais successifs et la structure du dictionnaire doit permettre d'en limiter le nombre.

1.3.1 Organisation des dictionnaires

Un des buts de l'organisation actuelle des dictionnaire de bases et de terminaisons de PILAF est de rendre l'analyseur morphologique adaptable et transportable sur le plus grand nombre de machines. Elle est inspirée de la technique des B-arbres. Le dictionnaire de bases, par exemple, est un ensemble de bases réparties en fichiers de la façon suivante : on part d'un fichier $\mathbf{F}_1$ vide dans lequel on insère des éléments ; dès que $\mathbf{F}_1$ est plein (sa taille est actuellement paramétrée à 600 éléments), on le partage en deux fichiers $\mathbf{F}_1$ et $\mathbf{F}_2$ de 300 éléments : les 300 premiers dans $\mathbf{F}_1$ et les 300 suivants dans $\mathbf{F}_2$ (premiers et suivants dans l'ordre alphabétique). Si à la suite d'autres insertions un de ces fichiers sature, on le découpe de nouveau en deux, etc. Les éléments dans les fichiers sont rangés en ordre alphabétique inverse ce qui permet pendant une recherche d'obtenir l'élément le plus long en premier.

L'accès aux fichiers $\mathbf{F}_i$ est assuré par une table appelée TABMAITRE qui contient pour chaque entrée :
- le nom $\mathbf{F}_i$ du fichier ;
- BINF et BSUP : deux chaînes correspondant aux bornes inférieure et supérieure de l'ensemble des éléments du fichier ;
- NBELE : nombre d'éléments du fichier.

La taille de TABMAITRE est paramétrée.

Un élément du dictionnaire est un enregistrement contenant 7 champs qui se rapportent à une chaîne (les trois derniers champs peuvent se rapporter à la chaîne suivante dans un chaînage circulaire) :

1 LETTRES : chaîne de caractères, de longueur variable ;
2 OCCLETTRES : occurrence de la chaîne dans le dictionnaire ;
3 MODELE : numéro du modèle correspondant ;
4 INDECOUPABLE : booléen signalant que l'élément est non-réductible.

Concernant l'élément suivant dans le chaînage circulaire :

5 LGCIRC : longueur de l'élément ;
6 CHCIRC : chaîne de caractères, de longueur variable ;

7 OCCLRC : numéro d'occurrence de l'élément dans le dictionnaire.

Remarques

a) Les champs OCCLETTRES distinguent deux éléments identiques sur LETTRES mais différents par le modèle ou les informations associées.

b) Les informations sur le chaînage circulaire sont utilisées actuellement par la génération morphologique pour engendrer, par exemple, toutes les formes d'un verbe.

Exemple : pour le verbe **ALLER,** le chaînage circulaire est le suivant :

ALL	=>	AILL	la 2ème partie désigne l'élément suivant dans la liste circulaire.
AILL	=>	IR	
IR	=>	VA	
VA	=>	VAS	
VAS	=>	VAIS	
VAIS	=>	VONT	
VONT	=>	ALL	

c) La présence de **vrai** dans l'indicateur INDECOUPABLE permet d'arrêter la recherche, c'est à dire de ne pas chercher d'autre décomposition de la chaîne d'entrée.

1.3.2 Algorithme de recherche dans le dictionnaire

En analyse morphologique, on ne dispose pas de séparateur de mot, l'entrée de l'algorithme est donc un simple pointeur sur le premier caractère de la chaîne restant à analyser.
Exemple :

Les allées et venues
↑

On a la possibilité, bien entendu, de lire tout ou partie de la chaîne restante (droit de regard en avant).
L'algorithme comprend deux étapes :
1 Déterminer le fichier F_i dans lequel on va effectuer la recherche ;
2 Essayer de retrouver un élément de F_i dans la chaîne d'entrée.

Détermination de F_i
Pour déterminer le bon fichier, on effectue une recherche dans TABMAITRE. On a fixé une longueur arbitraire N à laquelle on a ramené les BINF et BSUP (par troncature ou en rajoutant des blancs). On extrait de la chaîne d'entrée la

chaîne A de longueur N, il suffit ensuite de trouver l'entrée i de TABMAITRE telle que :

$$BINF_i \leq A \leq BSUP_i$$

qui permet de déterminer le nom du fichier F_i.

Exemple :

Les allées et venues
↑ ↑

On extrait la sous-chaîne */allées et /* composée de 10 caractères (N=10) permettant de déterminer le fichier dans lequel doit se trouver */allée/*.

Recherche dans F_i

Une fois déterminé F_i, la recherche consiste en un parcours avec comparaisons successives. Comme on ne sait pas a priori ou s'arrête le mot dans la chaîne d'entrée, on utilise la longueur des éléments du dictionnaire pour faire les comparaisons. L'ordre alphabétique inverse permet d'obtenir d'abord les éléments les plus longs.

Exemple (on n'a pas mentionné les informations sur le chaînage circulaire) :

Dictionnaire de terminaisons :

.....
2/s /1/s /vrai
4/ées /1/ées /vrai
.....

Dictionnaire de bases :

.....
5/allée/1/maison/faux
3/all/1/all/vrai
.....

(on n'a pas mentionné les informations sur le chaînage circulaire)

Le premier élément qui coincide avec la chaîne d'entrée est la chaîne /allée/.
Dès qu'un élément est reconnu on appelle la grammaire pour le valider et calculer les informations linguistiques. Le dictionnaire fournit le modèle associé à l'élément. Deux cas sont alors possibles :

1 la grammaire ne valide pas cet élément. La chaîne reconnue ne peut donc pas faire partie du segment en cours de construction. On lance alors la recherche d'une chaîne plus courte (RECHPETIT).

2 la grammaire valide cet élément. On a de nouveau deux cas :
- la concaténation des éléments déjà reconnus constitue un segment incomplet. On relance alors la recherche sur les caractères qui suivent la chaîne courante (RECHSUIV).

- la concaténation des éléments déjà reconnus constitue un segment complet. Ce résultat est alors ajouté à la liste des résultats. Ensuite, si l'indicateur INDECOUPABLE de l'entrée courante du dictionnaire est positionné à vrai, on arrête la recherche, sinon on lance la recherche d'une chaîne plus courte (RECHPETIT).

La base /allée/ n'est pas un segment complet, la grammaire relance une recherche avec le pointeur d'entrée positionné immédiatement après /allée/, sur le /s/. On obtient la désinence /s / et le segment est alors bien déterminé :

"allées " -->	base /allée/	substantif commun féminin
	désinence /s /	pluriel

Après analyse de "allées " comme ci-dessus, on regarde INDECOUPABLE de /s /, il est positionné à *vrai*, on arrête, on regarde alors INDECOUPABLE associé à /allée/, il est à *faux* : On appelle RECHPETIT. On obtient la base /all/ et on appelle la grammaire qui se charge de la reconnaissance de /ées /, on obtient alors :

"allées " -->	base /all/	verbe 1er groupe
	désinence /ées /	participe passé féminin pluriel

Remarques

a) L'indicateur INDECOUPABLE et l'ordre alphabétique inverse permettent de reconnaître des locutions telles que *à ce propos* comme des unités lexicales entières, il suffit d'avoir l'entrée correspondante dans le dictionnaire :
12/à ce propos /1/à ce propos /vrai

b) La sortie de la recherche est une liste de toutes les solutions possibles. Toutes ces solutions ont la même longueur. En effet dès qu'on a un segment possible on interdit tout segment plus court (préfixe du premier) comme solution. Ceci permet de n'obtenir, pour une phrase donnée, qu'un seul découpage.

c) On aura noté l'interaction très forte entre le dictionnaire et la grammaire : au niveau algorithmique, il y a récursivité croisée entre les deux.

1.4 La grammaire et les modèles

Dans le système d'analyse morphologique les règles de la grammaire sont nommées (RIB, PR5,...) et le vocabulaire est constitué d'éléments du dictionnaire. Les règles ont la forme :

nom : informations linguistiques ;
VAL := (liste de noms de règles) ou valeur prédéfinie ;
SAT := (liste de noms de règles) ou valeur prédéfinie ;
indicateurs.

VAL et SAT définissent respectivement les validations et les saturations. On a deux indicateurs possibles :
- FIM qui indique une règle terminale ;
- ND qui indique une non-désinence.

Le mot clé VALOO désigne la liste prédéfinie des règles initiales.

Pour assurer l'interface entre le dictionnaire et la grammaire on utilise des modèles. Ce sont des représentants d'une classe de mots ayant le même comportement linguistique (les verbes du 1er groupe par exemple). Ils sont de la forme suivante :

/nom/ : REG := (liste de noms de règles) ou valeur prédéfinie ;
informations linguistiques ;
VAL := (liste de noms de règles) ou valeur prédéfinie ;
SAT := (liste de noms de règles) ou valeur prédéfinie ;

On a les mêmes interprétations que ci-dessus avec en plus REG qui donne une liste des règles applicables à ce modèle.

1.4.1 Les informations linguistiques

Elles sont de deux sortes :
1 **type :** le CL, c'est-à-dire, la classe lexicale du mot.
Exemple : pour DIVISION on a CL = SUBSTANTIF
2 **variable :** VAR, c'est-à-dire, les valeurs morphologiques associées à ce mot.
Exemple : (FEMININ SINGULIER) pour DIVISION.

Chaque information CL ou VAR peut être considérée comme un registre auquel on peut affecter des valeurs, les expressions autorisées dépendent du registre.

Exemple pour un registre X :
X := XG : valeur de l'état précédent dans la dérivation
X := XD : valeur contenue dans le modèle associé à l'unité lexicale
X := valeur : si **X** est un type
VAR := liste de valeurs exemple : **VAR:=VARG-SIN+PLU** : on remplace SIN par PLU.

Une liste de valeurs est donnée soit explicitement (on écrit la liste), soit par une expression formée à partir de numéros de masques, de valeurs explicites et des opérateurs + et -. Un masque est un vecteur de booléens de longueur fixe, chaque booléen correspondant à une information élémentaire (MAS, FEM, PLU, ...) :

- vrai : l'information appartient à la liste donnée par ce vecteur ;
- faux : l'information n'appartient pas à la liste donnée par ce vecteur ;

Le nombre d'informations linguistiques élémentaires est actuellement paramétré à 32.

1.4.2 Règles applicables

Si le numéro du modèle sélectionné est **m+1**, la liste des règles applicables est donnée par :

intersection (REG_{m+1} , VAL_{i+1})

Si cette intersection est vide et que **r** ne contient pas l'indicateur **ND**, alors la liste des règles applicables est :

intersection (REG_{m+1} , CM_r) (code morphologique)

Pour l'état initial on a : **SAT_0 = [ ] et VAL_0 = VALOO.**

Pour représenter les listes de règles on utilise des masques de la même manière que pour les informations linguistiques : le masque permet de déterminer quels sont les noms des règles de la liste parmi un ensemble prédéfini de noms de règles (le nombre maximum de règles est actuellement paramétré à 256).

Nous illustrons les formules et le fonctionnement du transducteur sur un exemple.

VALOO := RIB ...
VAL1:= E1 E8

Les règles :

```
RIB :  CL:=CLD ; CM:=CMD ;
       VAR:=VARD ;
E1  :  CL:=CLG ;
       VAR:=VARG+SIN ; FIM .
PR4 :  CL:=CLG ;
       VAR:=VARD+PRE+IND ; FIM .
SU1 :  CL:=CLG ;
       VAR:=VARD+PRE+SUB ; FIM .
```

Les modèles :

```
/femme/ : REG:=RIB ; CL:=SUBC ;
          VAR:=FEM ; VAL:=VAL1 .
//    :  REG:=NF1 PR7 E1 ;
          VAR:=TRE+SING .
 /aim/  :  REG:=RIB ; CL:=VERB ;
          CM:=PR4 SU1 .
/ent / :  REG:=PR4 IS3 SU1 ;
          VAR:=TRE+PLU .
```

Dans la plupart des règles et des modèles ci-dessus, les listes de validation et saturation étant vides, les mots clés VAL et SAT n'apparaissent pas.

Le dictionnaire (on ne donne que la chaîne et le modèle associé) :
/maison/femme/
/chant/aim/
///
/ent /ent /

Analyse de /maison /
Le dictionnaire fournit la base /maison/ dont le modèle est /femme/ ; par intersection entre REG de /femme/ et VALOO on sélectionne la règle RIB qui donne :
CL:= CLD → CL := SUBC (substantif commun)
VAR := VARD → VAR := FEM (féminin)
CM := CMD → CM := ( ) puisque pas de CM dans le modèle.
Les validations sont égales à VAL1 et sont donc (E1,E8).
On identifie ensuite la désinence / / dont le modèle est / /, avec :
REG := NF1 PR7 E1
Par intersection avec les validations (E1,E8) on sélectionne E1 qui donne :
CL := CLG transmission de CL donc CL:=SUBC
VAR := VARG+SIN transmission de VAR:=FEM à laquelle on ajoute SIN (singulier)
De plus, la présence de l'indicateur FIM sur la règle E1 permet l'acceptation de l'unité lexicale /maison / avec les informations linguistiques *substantif commun féminin singulier.*

Analyse de /chantent /:
On procède de la même façon, le modèle de /chant/ est /aim/, ayant REG := RIB, on applique RIB et on obtient :
CL := VERB, CM := PR4 SU1
Les validations étant vides et l'indicateur ND étant absent, on utilise les règles du code morphologique.
La désinence /ent /, dont le modèle a pour REG (PR4, IS3, SU1), permet par intersection avec CM d'obtenir deux règles PR4 et SU1, elles assurent toutes les deux la transmission de CL et :
PR4 → VAR := VARD+PRE+IND → VAR := TRE+PLU+PRE+IND
(troisième personne du pluriel du présent de l'indicatif)
SU1 → VAR := VARD+PRE+SUB → VAR := TRE+PLU+PRE+SUB
(troisième personne du pluriel du présent du subjonctif)

1.4.3 Utilisation du transducteur

Des compilateurs de règles et de modèles, des éditeurs et des masques d'écran autorisent une mise au point interactive et très aisée des paramètres linguistiques.

Le transducteur étant entièrement paramétré peut être utilisé pour de nombreuses applications :
- traduction d'un texte en braille abrégé,
- traduction d'un texte en morse abrégé,
- traduction de programmes écrits en PL360 en programmes équivalents écrits en LP80,
- traduction d'une chaîne en un codage pseudo-phonétique,
- analyse morphologique d'une autre langue,
- etc...

On notera également que ce transducteur est réversible, et qu'il permet ainsi d'engendrer toutes les formes d'une même base. Cette réversibilité nous a été très utile pour vérifier les règles et modèles. D'autre part, ce module de génération est utilisé pour corriger certains types d'erreurs.

2 L'ANALYSE SYNTAXIQUE

Cette analyse est réalisée par un module qui construit la ou les structures de dépendances associées à une phrase, en fonction des catégories délivrées par l'analyse morphologique.

2.1 Le constructeur de dépendances et l'algorithme d'analyse

Selon Tesnières dans *Eléments de Syntaxe Structurale*, Klinshsieck 1959, la connexion est indispensable à l'expression de la pensée et elle donne à la phrase son caractère organique et vivant. Les connexions structurales établissent entre les mots des rapports de dépendances liant un terme inférieur, le subordonné et un terme supérieur, le régissant.

La méthode employée dans l'analyseur syntaxique de PILAF est une analyse directe. On transforme la chaîne d'entrée en une structure de dépendances sans utiliser de structure intermédiaire. L'algorithme est donc basé sur une grammaire de dépendances mais n'utilise pas directement cette grammaire comme le font les analyseurs basés sur des grammaires HC. En effet, comme on l'a vu précédemment, ces grammaires imposent la description de toutes les configurations possibles de dépendants pour un gouverneur donné. Pour pallier à cet inconvénient on introduit la notion de relation de dépendances entre deux catégories syntaxiques.
Exemple :
La relation SUBC * ADJQ indique que la catégorie SUBC gouverne la catégorie ADJQ.

2.1.1 Fonctionnement de l'algorithme

L'analyseur commence par construire la matrice suivante :

GOUV / DEP	x_0 phra	x_1	x_2		x_n
x_0 phra	$\varnothing$	P_{01}	P_{02}		P_{0n}
x_1	P_{10}	$\varnothing$	P_{12}		P_{1n}
x_2	P_{20}	P_{21}	$\varnothing$		P_{2n}
x_n	P_{n0}	P_{n1}	P_{n2}		$\varnothing$

avec
P_{ij} : ensemble des poids déterminés par les relations entre le dépendant X_i et le gouverneur X_j.
P_{ii} : $\varnothing$

On ne veut construire que des structures projectives, donc pour un gouverneur X_j donné, tous ses dépendants gauches devront avoir un indice i < j (l'ordre des indices correspond à l'ordre de la phrase initiale). On peut raisonner de même pour les dépendants droits, d'indice k > j. On élimine donc des ensembles de poids du triangle supérieur droit de la matrice tous les poids positifs, et du triangle inférieur gauche tous les poids négatifs. On a alors les deux propriétés :
- $1 \leq i, j \leq n,\ i > j,$ si $P_{ij} \neq \varnothing$ alors $\forall\ p \in P_{ij}$, on a $p > 0$.
- $1 \leq i, j \leq n,\ i < j,$ si $P_{ij} \neq \varnothing$ alors $\forall\ p \in P_{ij}$, on a $p < 0$.

L'algorithme construit ensuite toutes les structures de dépendances compatibles avec ce tableau. La construction est récursive et descendante : pour tous les gouverneurs possibles de la phrase (colonne PHRA), construire tous les sous-arbres gauches puis tous les sous-arbres droits (récursivement) et bâtir les structures finales avec les structures partielles obtenues.

On détaille les différentes étapes sur l'exemple suivant :
Le chien jaune mange la soupe de poisson.
DET SUBC ADJQ VERB DET SUBC PREP SUBC

Prenant comme base la grammaire de dépendances présentée dans la page 65, on a les relations :

PHRA*SUBC := (+1)
PHRA*VERB := (+1)
VERB*SUBC := (-20,+20)
VERB*PREP := (+30)
SUBC*DET := (-16)
SUBC*ADJQ := (-15,-14,+18)
SUBC*PREP := (+20)
PREP*SUBC: = (+7)

En tenant compte de la projectivité, on obtient le tableau suivant :

	PHRA	DET Le	SUBC chien	ADJQ jaune	VERB mange	DET la	SUBC soupe	PREP de	SUBC poisson
PHRA	O								
DET		0	-16				-16		-16
SUBC	+1		0		-20				
ADJQ			+18	0			-15 -14		-15 -14
VERB	+1				0				
DET						0	-16		-16
SUBC	+1				+20		0		
PREP			+20		+30		+20	0	
SUBC	+1				+20			+7	0

Exemple de matrice d'analyse

Etape a :
VERB est le seul gouverneur de phrase compatible avec ce tableau (les autres conduiraient à des échecs), en effet VERB ne peut dépendre que de PHRA (la ligne VERB ne contient qu'une seule valeur de poids).

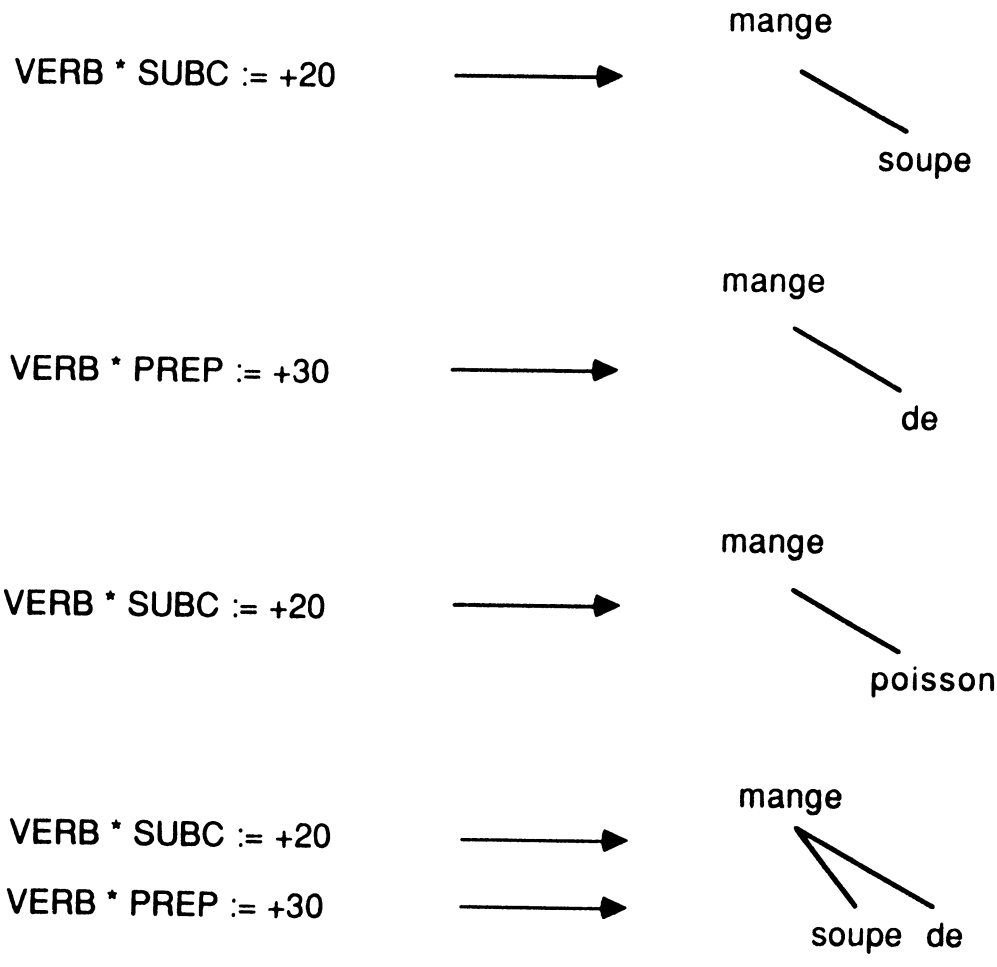
Etape b :
Comme on ne veut que des structures projectives, on peut calculer les sous arbres droits indépendamment des sous arbres gauches. Prenons le cas des dépendants droits $X_{j+1}...X_n$ d'un gouverneur X_n donné (ici *mange*).

On consulte la colonne X_j et on élimine d'abord les X_i tels que $P_{ij} = \emptyset$ (ce ne sont pas des dépendants directs). Avec les restants, notés $X_1...X_k$, on construit toutes les structures possibles correspondant à 1, puis 2, puis 3, ..., jusqu'à k dépendants directs de X_j ($1 \leq k \leq n-j$).
On a la contrainte :

si X_m et X_q sont deux dépendants directs de X_j, tels que $m < q$,
alors il existe $p_m \in P_{mj}$ et $p_q \in P_{qj}$ tels que $p_m < p_q$.

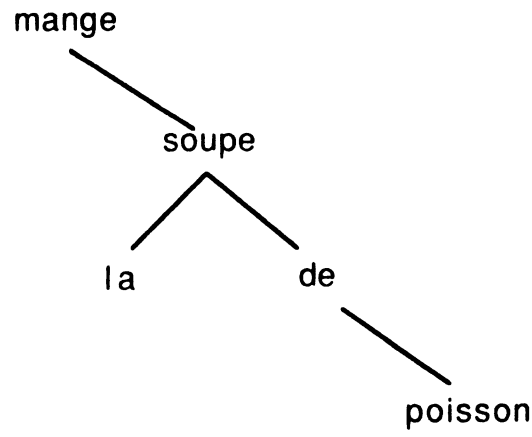
C'est-à-dire qu'on doit toujours pouvoir associer à une suite de dépendants directs une suite de poids strictement croissante.

Sur l'exemple on peut construire :

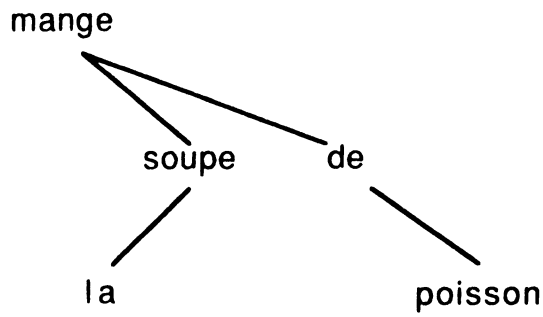


S'il reste des mots à gauche ou à droite d'un des dépendants, on considère ce dépendant comme un nouveau gouverneur et on appelle récursivement l'analyse.

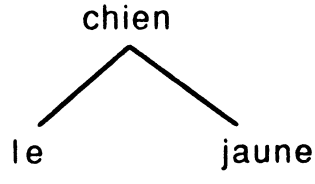
Les deuxième et troisième structures ci-dessus conduisent à des échecs : on ne parviendra pas à placer le SUBC *soupe*. La première permet d'obtenir :



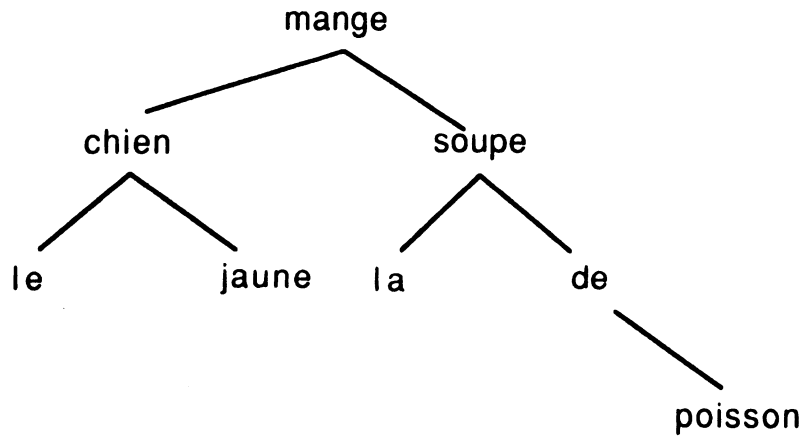
et la quatrième :



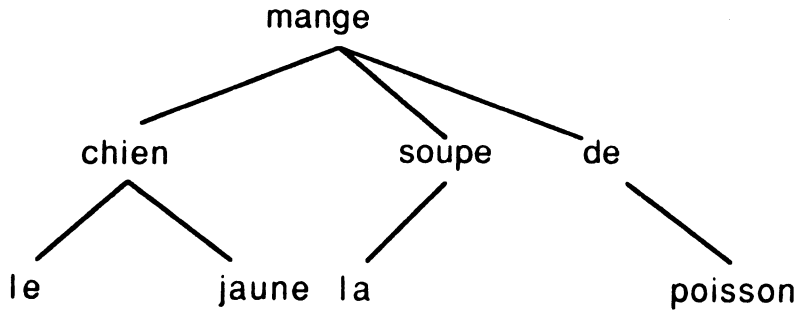
qui est syntaxiquement correcte mais sémantiquement fausse (on peut manger de bon appétit mais pas de poisson).
L'analyse des dépendants gauches de *mange* permet d'obtenir l'unique structure :



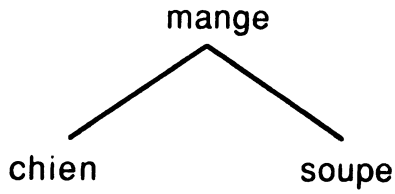
On construit toutes les combinaisons possibles avec les structures obtenues à l'étape **b**, on obtient alors :



et



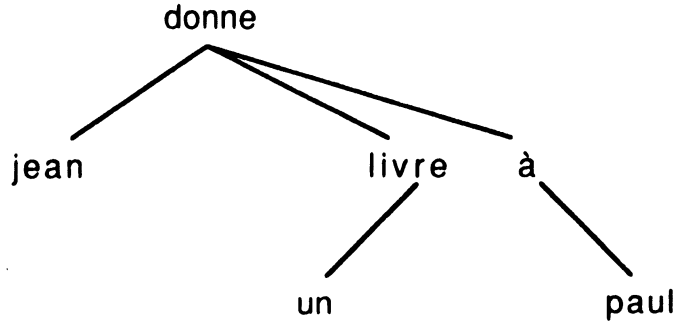
Remarque :
Un tel algorithme permet également la construction de structures incorrectes comme :



De telles structures peuvent suffire en entrée d'un modèle sémantique qui serait uniquement intéressé par l'information que véhicule la phrase.

Si les structures de dépendances ne constituent pas un modèle idéal pour l'analyse purement syntaxique du langage, elles sont par contre bien adaptées aux analyses de niveau supérieur (sémantique, logique).
Exemple :

Jean donne un livre à Paul.



On trouve directement sur la structure les arguments du prédicat *donner* : l'agent (Jean), l'objet (un livre) et le bénéficiaire (à Paul).