N° d'ordre : 248

THÈSE

présentée à

l'UNIVERSITÉ DES SCIENCES ET TECHNIQUES de LILLE I

pour obtenir le titre de

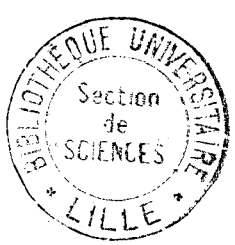
DOCTEUR DE SPÉCIALITÉ

(Mathématiques appliquées)

par

Jean-Michel MAESTRACCI

AIDE A L'INTERROGATION DANS UN SYSTÈME DE DOCUMENTATION AUTOMATIQUE



THÈSE soutenue le 21 Mai 1971

devant la Commission d'examen !

Monsieur P. BACCHUS, Président
Monsieur B. DRIEUX, examinateur
Monsieur J.-P. TRYSTRAM, examinateur
Monsieur C. CARREZ, rapporteur

UNIVERSITE DE LILLE

FACULTE DES SCIENCES

DOYENS HONORAIRES

MM. H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES

MM. ARNOULT, BROCHARD, CAU, CHAPPELON, CHAUDRON, CORDONNIER, DEHEUVELS, DEHORNE, DOLLE, FLEURY, P. GERMAIN, KAMPE DE FERIET, KOURGANOFF, LAMOTTE, LELONG, Mme LELONG, MM. MAZET, MICHEL, NORMANT, PARISELLE, PAUTHENIER, ROIG, ROSEAU, ROUBINE, ROUELLE, WIEMAN, ZAMANSKY.

PROFESSEURS TITULAIRES

M. BACCHUS Pierre	Astronomie et Calcul
M. BEAUFILS Jean-Pierre	Chimie Générale
M. BLOCH Vincent	Psychophysiologie
M. BONNEMAN Pierre	Chimie Industrielle
M. BONTE Antoine	Géologie Appliquée
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie Végétale
M. CELET Paul	Géologie Générale
M. CONSTANT Eugène	Electronique
M. CORSIN Pierre	Paléobotanique
M. DECUYPER Marcel	Mathématiques
M. DEDECKER Paul	Mathématiques
M. le Doyen DEFRETIN René	Directeur du Laboratoire de Biologie Maritime de Wimereux
M. DELATTRE Charles	Géologie Générale
M. DURCHON Maurice	Biologie Animale
M. FOURET René	Physique
M. GABILLARD Robert	Electronique
M. GLACET Charles	Chimie Organique
M. GONTIER Gérard	Mécanique des Fluides
M. GUILLAUME Jean	Biologie Végétale
M. HEUBEL Joseph	Chimie Minérale
Mme LENOBLE Jacqueline	Physique
M. MONTREUIL Jean	Chimie Biologique
Mme SCHWARTZ Marie Hélène	Mathématiques
M. TILLIEU Jacques	Physique
M. TRIDOT Gabriel	Chimie Minérale Appliquée E.N.S.C.L.
M. VIDAL Pierre	Automatique
M. VIVIER Emile	Biologie Animale
M. WATERLOT Gérard	Géologie et Minéralogie
M. WERTHEIMER Raymond	Physique

PROFESSEURS A TITRE PERSONNEL

M. BOUISSET Simon	Physiologie Animale
M. DELHAYE Michel	Chimie Physique et Minérale 1er Cycle
M. LINDER Robert	Biologie Végétale
M. LUCQUIN Michel	Chimie Physique
M. PARREAU Michel	Mathématiques
M. SAVARD Jean	Chimie Générale
M. SCHALLER François	Biologie Animale
M. SCHILTZ René	Physique

PROFESSEURS SANS CHAIRE

M. BELLET Jean	Physique
M. BODART Marcel	Biologie Végétale
M. BOILLET Pierre	Physique
M. DERCOURT Jean-Michel	Géologie et Minéralogie
M. DEVRAINNE Pierre	Chimie Minérale
Mle MARQUET Simone	Mathématiques
M. MONTARIOL Frédéric	Chimie Minérale Appliquée
M. PROUVOST Jean	Géologie et Minéralogie
M. VAILLANT Jean	Mathématiques

MAITRES DE CONFERENCE (et chargés des fonctions)

M. AUBIN Thierry	Mathématiques Pures
M. BEGUIN Paul	Mécanique des Fluides
M. BILLARD Jean	Physique
M. BKOUCHE Rudolphe	Mathématiques
M. BOILLY Bénoni	Biologie Animale
M. BONNOT Ernest	Biologie Végétale
M. CAPURON Alfred	Biologie Animale
M. CARREZ Christian	Calcul
M. CORDONNIER Vincent	Calcul
M. CORTOIS Jean	Physique
M. COULON Jean-Paul	Electrotechnique
M. DEBRABAN Pierre	Sciences Appliquées
M. ESCAIG Bertrand	Physique
M. FROLICH Daniel	Sciences Appliquées
M. GOBLOT Rémi	Mathématiques
M. GOUDMAND Pierre	Chimie Physique
M. GRUSON Laurent	Mathématiques
M. GUILBAULT Pierre	Physiologie Animale
M. HERMAN Maurice	Physique
M. HUARD de la MARRE Pierre	Calcul
M. JOURNEL Gérard	Sciences Appliquées
Mle KOSMANN Yvette	Mathématiques
M. LABLACHE COMBIER Alain	Chimie Générale
M. LACOSTE Louis	Biologie Végétale
M. LANDAIS Jean	Chimie Organique
M. LAURENT François	Automatique
M. LEHMANN Daniel	Mathématiques
Mme LEHMANN Josiane	Mathématiques
M. LOCQUENEUX Robert	Physique
M. LOUAGE Francis	Sciences Appliquées

M. LOUCHEUX Claude	Chimie Physique
M. MAES Serge	Physique
M. MAIZIERES Christian	Automatique
M. MESSELYN Jean	Physique
M. MIGEON Michel	Sciences Appliquées
M. MONTEL Marc	Physique
M. OUZIAUX Roger	Sciences Appliquées
M. PANET Marius	Electrotechnique
M. PAQUET Jacques	Sciences Appliquées
M. PARSY Fernand	Mécanique des Fluides
M. POVY Jean-Claude	Sciences Appliquées
M. RACZY	Radioélectrique
M. ROUSSEAU Jean-Paul	Biologie Animale
M. ROYNETTE Bernard	Mathématiques
M. SALMER Georges	Electronique
M. SMET Pierre	Physique
M. VANDORPE Bernard	Sciences Appliquées
M. WATERLOT Michel	Géologie Générale
Mme ZINN **JUSTIN** Nicole	Mathématiques

Je voudrais exprimer ma profonde reconnaissance à Monsieur le Professeur BACCHUS, Directeur de l'U.E.R. Informatique - Electronique - Electrotechnique et Automatique de l'Université de LILLE qui m'a fait l'honneur d'accepter la présidence du jury.

J'adresse aussi ma vive reconnaissance et mes remerciements à Monsieur le Professeur CARREZ qui a suivi de très près la réalisation de ce travail, et m'a guidé par de précieux conseils.

Je remercie également le Professeur DRIEUX qui a bien voulu m'encourager et accepter de faire partie du jury.

Je tiens à évoquer la bienveillante attention dont m'a entouré le Professeur J.P. TRYSTRAM dans la réalisation de ce travail, notamment à l'Institut de Recherche en Informatique et Automatique.

Enfin, je remercie les personnes du Laboratoire de Calcul de Lille qui ont aidé à la réalisation de cette thèse, Monsieur et Madame DEBOCK, et particulièrement Mademoiselle DRIESSENS qui a su donner à ce travail une présentation originale et agréable.

ooOoo

A mes parents,
A ma fiancée,

pour leur compréhension
et leurs encouragements.

T A B L E des M A T I E R E S

INTRODUCTION

Depuis quelques années, on assiste à un développement considérable du volume des informations produites et traitées dans les différents secteurs de l'activité humaine.

Dans le domaine des publications scientifiques, par exemple, un comité gouvernemental américain, le COSATI (Committee on Scientific and Technical Information) faisait état en 1965, d'une augmentation annuelle d'environ 6% du nombre des articles, ce qui implique un doublement tous les 15 ans.

A mesure que le volume des informations croît, rechercher, trouver, traiter cette information devient un problème de plus en plus long et délicat. Très rapidement, grâce au développement de la puissance des calculateurs, et des techniques de software, on a essayé d'automatiser le traitement de grosses masses d'informations. Un nombre assez important d'études a été mené, tant aux Etats-Unis qu'en Europe. De ces études assez disparates au début, on peut tirer deux classes de réalisations.

Les systèmes de documentation automatique,d'une part, qui permettent l'analyse, l'organisation, la mémorisation et la recherche de l'information portant sur ses documents, son contenu, et non par son emplacement, comme dans un système de gestion de fichier simple.

Les systèmes de banques de données, d'autre part, qui ont été élaborés plus tard, et dont la terminologie n'est pas encore très précise. Ces systèmes sont constitués par un ensemble de fichiers appelé base de données, un système de gestion de cette base, et un sous-système permettant l'exploitation de la base de données. Dans cet ensemble de fichiers, certains ne contiennent que des données, d'autres contiennent des données sur les données pour permettre de retrouver les informations par leur contenu syntactique ou sémantique.

En ce qui concerne les banques de données, certaines techniques de recherche d'informations par contenu sont très voisines de celles utilisées dans les systèmes documentaires. Un exemple de recherche documentaire, utilisé par une banque de données sera décrit à la fin de cet ouvrage.

Les systèmes qui permettent de retrouver l'information par son contenu utilisent assez souvent un codage pertinent des contenus syntactiques ou sémantiques des ensembles gérés par le système, afin de limiter le volume nécessaire pour traiter l'information et faciliter l'organisation des recherches.

Pour retrouver les documents qui répondent à une demande formulée au système, un des problèmes les plus importants est d'associer à l'ensemble de mots du langage d'entrée du système, l'ensemble des codes syntactiques et sémantiques nécessaires pour orienter une recherche performante. Cette association sera réalisée à l'aide d'un, ou de plusieurs thesaurus, ou dictionnaires.

La création, la mise à jour, l'amélioration des thesaurus et des fichiers qui contribuent à la recherche documentaire sont des opérations généralement très lourdes, car elles nécessitent des procédures manuelles assez longues, et qui augmentent, de plus, les causes d'erreur. Un des buts du travail présenté ici est d'automatiser ces opérations, pour les rendre plus fiables et plus rapides.

La première partie de l'exposé (chapitres I à V) est consacrée à l'étude et à la création d'un système de thesaurus qui va permettre d'associer à un mot de la langue française l'ensemble des codes syntactiques et sémantiques utilisable par le système, et qui correspond à ce mot.

Pour que le système soit plus performant, on a cherché à associer automatiquement les mêmes codes aux différentes formes orthographiques du même mot, sans qu'il soit nécessaire de donner toutes les orthographes possibles. Une étude de la formation par suffixe des mots de la langue française, résumée dans le chapitre I débouche sur la notion de classe de suffixes. Afin de ne pas surcharger le thesaurus, la méthode des stems de mots a été utilisée. Le chapitre II justifie la méthode du choix du stem de mot et donne plusieurs méthodes pour déterminer automatiquement le stem associé aux différentes orthographes possibles du même mot.

Dans un contexte de multiprogrammation, le volume de mémoire demandé à l'allocateur de ressources est un paramètre très important du traitement. On

a donc cherché à minimiser le volume de mémoire nécessaire pour représenter le thesaurus, tout en obtenant des temps de recherche assez rapides. La structure retenue est une représentation sous forme arborescente. Le chapitre III contient la description de cette méthode. Le chapitre IV décrit les algorithmes nécessaires pour la création, la mise à jour, l'interrogation d'un thesaurus représenté sous forme d'arborescence. Le chapitre V est consacré à une étude comparative des performances, qui justifie le choix de cette méthode de représentation.

La fin de l'exposé (chapitres VI et VII) est consacrée à un exemple de système documentaire, inséré dans une banque de données, et qui utilise un langage interactif pour gérer les demandes formulées par un utilisateur, en français courant.

CHAPITRE I

DEFINITIONS

CLASSES de SUFFIXES

I.1 INTRODUCTION

Ce chapitre va donner un certain nombre de définitions qui préciseront le vocabulaire employé dans la suite de cet exposé.

Par un bref rappel de l'étymologie des mots français, on mettra en relief le rôle des suffixes dans la formation des mots, et on introduira, après une étude plus détaillée des suffixes, la notion, importante pour la suite de l'étude, de classe de suffixes.

En même temps, un des problèmes importants dans un système de recherche d'informations consiste, pour associer à un mot les indications internes au système qui lui correspondent, à faire cette association par l'intermédiaire du stem de ce mot. La notion fondamentale de stem de mot sera introduite dans ce chapitre.

I.2 QUELQUES DEFINITIONS

Ce paragraphe servira à préciser un certain nombre de définitions, couramment utilisées dans les ouvrages traitant de la documentation automatique, et qui seront employés au cours de cet exposé.

A- Il est nécessaire, d'abord, de préciser un certain nombre de définitions ayant trait à la sémantique.

* Phrase-Concept sémantique de phrase : une phrase est un ensemble de mots de la langue courante, qui permet de définir un concept sémantique de phrase.

En effet, la réunion des mots de la phrase détermine un ensemble qui spécifie une information, une idée représentable par l'esprit. Nous appellerons cette information ainsi déterminée un concept sémantique de phrase.

Par exemple, la phrase : "*les oiseaux volent*", évoque, pour l'esprit, l'idée de voler pour des oiseaux.

La phrase : "*les oiseaux qui volent*", bien que correspondant à un ensemble de mots différent, évoque la même idée que la phrase précédente. Elle sera donc associée au même concept sémantique de phrase.

La phrase : "*les oiseaux ne volent pas*", n'a pas le même sens qu'une des phrases précédentes, mais elle évoque cependant la même idée de voler, pour les oiseaux. Nous pourrons donc associer cette phrase au même concept sémantique de phrase que les deux phrases précédentes.

* Mot significatif ou mot-pivot, ou pivot : un mot significatif dans une phrase est un mot dont la suppression modifie le concept sémantique de phrase. Nous les appellerons également mots-pivots, ou pivots.

Dans l'exemple précédent, les mots significatifs pour les trois phrases sont : *oiseau* et *voler*.

* Mot non-significatif, ou mot-vide : un mot-vide est un mot dont la suppression dans la phrase ne mofifie pas le concept sémantique de phrase.

Par exemple, dans la phrase : "*les oiseaux qui ne volent pas*", les mots soulignés sont des mots-vides.

Donc, pour une phrase, le concept sémantique est déterminé par le sous-ensemble des mots-pivots. Le sens de la phrase est dû à l'ordonnancement des mots-pivots, et leur liaison par des mots-vides. Les deux phrases : "*les oiseaux volent*" et : "*l'oiseau vole*" correspondent à la même sémantique de phrase, les mots-pivots sont *oiseau*, et *voler* avec des orthographes différentes.

* Un concept : L'exemple précédent montre que le concept sémantique de phrase est déterminé par les mots-pivots de la phrase. L'idée, représentable par l'esprit, associée à la phrase est donc la combinaison, la réunion des idées élémentaires associées à chacun des mots-pivots. Nous appellerons ces idées élémentaires des concepts sémantiques simples, ou concepts.

Le concept sémantique de phrase résulte donc de l'association de concepts sémantiques simples. Un même concept sera donc associé aux diverses formes possibles d'un même mot. Par exemple, l'idée de chanter sera associée à toutes les formes conjuguées du verbe chanter.

Dans certains cas, un concept peut être associé à un ensemble déterminé de mots, par exemple : porte-à-faux, timbre-poste...

* Hiérarchie de concepts : un concept évoque, pour l'esprit, une idée. Cette idée peut être la généralisation, la somme d'idées plus élémentaires.

Par exemple, le concept : "*véhicule*" est la somme des concepts associés aux mots : *automobile, camion, fiacre, bateau, avion...*

Nous voyons donc qu'il est possible de définir une relation de dépendance de certains concepts par rapport à d'autres.

Nous appellerons l'ensemble de ces relations la hiérarchie des concepts.

* Voisinage sémantique strict : nous noterons ainsi la relation qui associe les différentes formes d'un même mot. Ex : *chanteur - chanteurs* : les deux

formes de ce mot correspondent strictement au même concept sémantique. Elles constituent un voisinage sémantique strict.

B- Nous allons donner maintenant un certain nombre de définitions, qui ont trait à la structure des mots, et à leur utilisation syntactique.

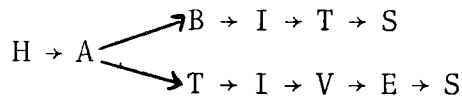
* Suffixe : un suffixe est une terminaison particulière qui s'ajoute à un radical, ou à un mot, pour former un mot nouveau.

Par exemple : *port - er* → *port - eur*

chant → *chant - eur.*

* Stem : ce mot anglais est très souvent employé dans les articles sur la documentation automatique. Il évoque la notion de partie commune, de tronc, et est associé à l'idée d'arborescence :

Considérons, par exemple, deux mots dont l'orthographe est voisine : *habits, hâtives*. Si nous représentons sous forme d'arborescence les deux chaînes de lettres qui correspondent à ces deux mots, nous obtiendrons :



Dans l'arbre ainsi créé, un mot est un chemin entre le noeud initial (H) et un des noeuds terminaux, ou feuille. Il existe une partie commune à ces deux chaînes, ou tronc : H → A.

Cet exemple nous montre qu'il n'est pas nécessaire, pour différencier les deux mots, de décrire les chemins correspondants en totalité, mais que les chaînes : H - A - B et H - A - T suffisent.

Nous appellerons les deux chaînes précédentes des stems de mots.

On constate donc que la notion de stem est associée à l'idée d'arbre, ou plus exactement, de tronc d'arbre, et que le stem évoque la suppression d'une partie située à la fin du mot, que nous appellerons improprement suffixe.

On voit donc qu'à un mot sera associé un stem ; nous allons montrer que la relation n'est pas biunivoque. En effet, dans le système de documentation automatique, à un stem sera associé un concept sémantique. Si on associe à plusieurs chaînes de mots le même stem, nous aurons ainsi réalisé entre ces chaînes un voisinage sémantique, par l'intermédiaire du stem. Nous verrons, dans la suite de cet exposé, qu'à chaque stem correspond au moins un voisinage sémantique strict.

Nous pouvons noter, également, que la notion de stem diffère de la notion de radical. Le terme radical recouvre, en plus, une notion étymologique. Nous verrons, dans la suite, que la chaîne représentant le stem ne correspond pas toujours à celle du radical.

* Mots ordinaires. Mots grammaticaux ou mots-outils

Le Français distingue deux espèces de mots, celle des mots ordinaires, et celle des mots grammaticaux. Nous verrons que ce partitionnement de l'ensemble des mots ne correspond pas à celui en mot-vide et pivots. Un mot ordinaire pourra être un mot vide, ou un mot-pivot.

- Les mots ordinaires. Ils ont un sens, et éveillent dans l'esprit une image. Par exemple, le mot *chanter*.

Ce sont :
- . les noms
- . les adjectifs
- . les verbes
- . les adverbes.

- Les mots grammaticaux ou mots-outils : n'ont pas de signification propre. Ils accompagnent, précisent, ou remplacent les mots ordinaires.

Ce sont :
- . les articles
- . les adjectifs et pronoms possessifs
- . les adjectifs et pronoms démonstratifs
- . les pronoms personnels
- . les numéraux
- . les prépositions
- . les conjonctions
- . les pronoms relatifs.

Nous allons étudier brièvement, dans le paragraphe qui suit, la formation des mots français, et mettre en relief le rôle des suffixes, que nous étudierons plus en détail.

I.3 RAPPELS SUR L'ETYMOLOGIE DES MOTS FRANCAIS

Le Français est une langue riche de plusieurs dizaines de milliers de mots, formés à partir d'un noyau de quelques milliers de mots, exprimant les notions les plus courantes.

Cette formation s'est faite soit par dérivation, soit par composition.

I.3.1 La composition

Un nom composé est formé de deux ou plusieurs mots. Soit il s'écrit en un seul mot, soit les mots qui le composent sont joints par des traits d'union.

Exemples : *porte-aiguilles, porte-crayon, portefeuille.*

Nous étudierons dans la description du dictionnaire, le pluriel de ces mots composés.

I.3.2 La dérivation

La dérivation s'est faite, soit par préfixes, soit par suffixes.

* La formation par préfixes

Le verbe *porter*, par exemple, a donné, par dérivation par préfixes les verbes : *apporter, rapporter, exporter, importer, reporter, déporter.*

La formation par préfixe modifie le sens du mot initial. On constate que le préfixe est moins lié au mot que le suffixe, et qu'il a, par lui-même, un sens qui modifie le sens du mot auquel il est accolé.

Par exemple, le préfixe *anti-* porte le sens de "*avant*" dans *anti-dater*, ou le sens de "*contre*" dans *anti-social.*

Nous pouvons voir qu'entre deux mots dérivés, par exemple : *importer - exporter*, il existe une relation sémantique, mais que les concepts ainsi définis ne sont pas voisins.

Dans un système de documentation automatique complet, les relations entre concepts sont représentées dans un ensemble différent du dictionnaire, où on fait figurer les relations et hiérarchies entre concepts. Pour le système de dictionnaire que nous étudions ici, nous chercherons simplement à reconnaître, et représenter les voisinages sémantiques. Nous n'étudierons pas, dans le cadre de ce dictionnaire, la dérivation par suffixes.

* La formation par suffixes

Cette formation par suffixes est assez riche dans la langue française.

Par exemple, le mot *poste* a donné les dérivés : *postal, postier, postière, postillon, postier.* Nous pouvons remarquer, à l'aide de cet exemple, que les mots dérivés par suffixes correspondent à des concepts sémantiques assez voisins.

Ce voisinage sémantique entre les mots dérivés par suffixes, beaucoup plus fort que dans le cas d'une dérivation par préfixes, nous a donc amené à étudier plus précisément les suffixes.

I.4 ETUDE DES SUFFIXES

Nous pouvons constater que la formation des mots par suffixes est souvent irrégulière.

× Souvent, à cause de la phonétique, il y a modification de l'orthographe du radical.

On peut noter,par exemple, que *bazarder (bazar + er)* a été fait sur le type de *hasarder*, de *hasard* ; *plafonneur (plafond + eur)* a comme modèle la série *raisonneur*, de *raison*. *Tabatière*, de *tabac* ; *marivaudage*, de *Marivaux* s'expliquent, eux aussi, par leur formation tardive, à une époque où le *c* de *tabac*, l'*x* de *Marivaux* ne se prononçaient pas. A remarquer que *tabac* a donné, plus tard le mot populaire : *tabasser*.

× On peut associer à un certain nombre de verbes, des dérivés qui ne présentent aucun suffixe : sur le modèle de *chanter : chant*, le français a formé, au masculin : *galop* de *galoper*, *oubli* de *oublier*... ; au féminin : *aide* de *aider* *visite* de *visiter*...

Nous allons montrer que les suffixes jouent deux rôles : un rôle sémantique et un rôle syntactique. Nous avons choisi de ne considérer que le rôle syntactique des suffixes, et nous justifierons ce choix.

I.4.1 Fonction sémantique de certains suffixes

Dans la langue française, certains suffixes modifient le sens du mot auquel ils sont associés. On peut donc distinguer deux catégories de suffixes : les suffixes ordinaires et les suffixes expressifs.

* Les suffixes ordinaires

Jouent uniquement un rôle grammatical. Ils servent à former des noms à partir de verbes, des verbes à partir des noms... Par exemple :

Quant le télégraphe a été inventé, il a fallu créer : *télégraphier*, *télégraphie*, *télégraphique*, *télégraphiste*.

Le verbe marque "*l'action*" : *porter*. Des suffixes servent à former le nom

. de celui qui fait l'action : un *porteur*

. de l'action elle-même : le *portage*.

* Les suffixes expressifs

Ont une valeur particulière. Ils modifient le sens des mots auxquels ils sont associés. On peut noter, en particulier :

- les diminutifs :

tourelle : une petite tour

garçonnet : un petit garçon

lionceau : un petit lion

aiglon : un petit aigle.

- Certains suffixes donnent aux noms ou aux verbes un sens défavorable

noiraud signifie "qui est d'un noir désagréable à voir" à comparer avec *noirâtre*, "qui tire sur le noir

marmaille désigne une bande de marmots malpropres et bavards

chauffard : un mauvais chauffeur

rêvasser c'est se laisser aller à de vagues rêveries.

* Remarque :

On cherche, dans un système de dictionnaire, seulement à noter un voisinage sémantique, et non à caractériser une relation sémantique entre deux mots. La caractérisation d'une relation relève, dans un système de documentation automatique complet, de l'ensemble représentant la hiérarchie et les relations entre concepts. Nous avons donc, dans le cadre de ce travail, considéré tous les suffixes comme des suffixes ordinaires, et nous avons étudié plus particulièrement la fonction syntaxique des suffixes.

I.4.2 Fonction syntactique des suffixes

Nous avons, dans le paragraphe précédent, noté la fonction syntactique des suffixes ordinaires. Certains suffixes sont donc caractéristiques d'une fonction grammaticale. Par exemple, *-ation* est un suffixe de nom ; *-er* est un suffixe de verbe ; *-uble* un suffixe d'adjectif.

Nous avons relevé, en annexe, une liste des suffixes caractéristiques de noms, et des suffixes caractéristiques d'adjectifs.

Nous allons maintenant exposer le principe des classes de suffixes, et montrer qu'il nous permet de définir un voisinage sémantique strict entre les différentes formes d'un même mot.

I.5 LES CLASSES DE SUFFIXES

Nous avons choisi de faire figurer dans le dictionnaire, comme nous le verrons par la suite, non pas les mots, mais les stems des mots.

Nous allons introduire dans ce paragraphe la notion d'ensemble de suffixes associés, ou classe de suffixes et montrer qu'elle permettra de relier un stem à un ensemble restreint de chaînes de mots, réalisant un voisinage sémantique strict, et de déterminer la fonction grammaticale de ces chaînes.

I.5.1 Définition d'une classe de suffixes. Fonction syntactique

Un exemple va nous permettre d'expliciter le principe des classes de suffixes. Soit le mot : *maladif*, et supposons qu'on ait choisi comme stem : *malad-*. La décomposition *malad - if* nous donnera donc le suffixe - *if*.

Le mot *maladif* est un adjectif, dont les diverses formes possibles sont : *malad-if, malad-ifs, malad-ive, malad-ives*. Dans ce cas, le stem *malad-* de l'adjectif *maladif* n'admettra comme suffixe que l'un des quatre suffixes suivants : *-if, -ifs, -ive, -ives*. Donc, l'ensemble (E), de quatre suffixes (s_i) défini précédemment est un ensemble de suffixes d'adjectifs tel que si S est la chaîne du stem : *malad-* , l'ensemble des chaînes $\{S + s_i \mid s_i \in E ; i = 1,4\}$ sera l'ensemble des chaînes des différentes formes possibles de l'adjectif *maladif*. L'ensemble E, ainsi déterminé sera, par définition, une classe de suffixes.

* Une classe de suffixes est un ensemble E de suffixes s_i ; i=1,n tel que si S est la chaîne associée au stem, l'ensemble des chaînes $\{S+s_i \mid s_i \in E, i=1,n\}$ contienne l'ensemble des chaînes de toutes les formes orthographiques possibles d'un même mot.

En caractérisant une classe de suffixes, nous avons ainsi créé un voisinage sémantique strict entre les formes possibles d'un même mot. L'exemple montre, de plus, que la fonction grammaticale de ces mots est déterminée ; la classe est une classe de suffixes d'adjectifs.

* Fonction syntaxique - Code syntaxique

Pour caractériser une classe de suffixes, il est possible, d'abord, d'indiquer la fonction syntactique associée : NOM pour un nom, ADJ pour un adjectif, VER pour un verbe, ADV pour un adverbe. L'ensemble des classes de suffixes est ainsi partitionné en quatre sous-ensembles. Une classe de suffixes sera déterminée complètement si on lui donne un numéro de séquence dans le sous-ensemble auquel elle appartient.

On aura ainsi créé un code syntactique. Par exemple, la classe

ADJ 18 sera : *-if, -ifs, -ive, -ives.*

Il suffira d'associer au stem *malad-* le code ADJ 18 pour indiquer que : *malad-if, malad-ifs, malad-ive, malad-ives* sont les formes possibles d'un même adjectif. Nous aurons ainsi noté un voisinage sémantique strict.

I.5.2 Eléments des classes de suffixes

Les classes de suffixes comprennent :

* Pour les noms. Les formes singulier et pluriel.

exemple : NOM 19 : *-ation, -ations.*

* Pour les adjectifs : les formes masculin singulier et pluriel, et féminin singulier et pluriel, si ces formes existent.

exemple : ADJ 31 : *-ateur, -ateurs, -atrice, -atrices.*

* Pour les verbes : nous avons limité les formes à celles du présent, infinitif, participes présent et passé.

exemple : VER 20 : *-e, -es, -ons, -ez, -ent, -ee, -es, -ees, -ant, -er.*

* Pour les adverbes : une classe ne comprend qu'un suffixe.

exemple : ADV 02 : *-amment.*

I.5.3 Création des classes de suffixes

L'ensemble des classes de suffixes figure en annexe. Nous allons exposer brièvement l'origine de ces classes.

* On peut, tout d'abord, remarquer que chaque suffixe caractéristique de nom ou d'adjectif a engendré une classe de suffixes. Par exemple, le suffixe de nom : *-ace* a engendré la classe NOM 02 : *-ace, -aces*.

* D'autres classes ont été introduites pour reconnaître les différentes formes d'un mot, en plus des cas précédemment déterminés.

  Par exemple, la classe ADJ 00 : -␣, *-e, -s, -es* *délicat* | -␣ / *-e* / *-s* / *-es*

* Certaines classes ont été créées pour permettre d'associer le même stem à deux mots différents, dont certaines formes peuvent avoir la même orthographe. Les cas ambigus pourront, ainsi être reconnus. Ces cas seront étudiés dans le chapitre suivant, dans l'exposé du choix des stems.

  exemple : *habit* | VER 20 → *er* / NOM 75 → *ant* / NOM 76 → *ante*

Ainsi, le choix du même stem permettra de trouver que la chaîne de lettres de *habitant* peut avoir comme fonction grammaticale celle d'un verbe (forme participe), ou celle d'un nom.

L'expérience montre que ces classes de suffixes permettent de déterminer le même stem pour l'ensemble des mots dérivés par suffixes les uns des autres, s'il n'y a pas modification de l'écriture du radical. Nous verrons, dans le chapitre suivant, l'utilisation de ce résultat dans le choix des stems.

Exemple :

port VER 20 → *er*
port ADJ 01 → *able*
port NOM 04 → *age*
port NOM 76 → *ante*
port ADJ 37 → *atif*
port NOM 81 → *eur*
port NOM 82 → *euse*

I.5.4 Ambiguïtés grammaticales

On constate, en français, que certaines chaînes de lettres peuvent avoir plusieurs fonctions grammaticales. Nous avons vu, dans le paragraphe précédent, l'exemple de *habitant*.

On rencontre, le plus souvent, les cas suivants :

* une forme verbale qui donne un nom ou un adjectif

exemple : *habiter* ⟶ *un habitant*

ignorer ⟶ *ignorant* : nom
ignorant : adjectif

Certaines classes de suffixes ont été introduites pour associer le même stem aux différentes fonctions grammaticales, dans un cas ambigu. Dans le chapitre suivant, on verra qu'il sera possible de déterminer que plusieurs fonctions grammaticales sont possibles pour la même chaîne de lettres.

* un adjectif est employé comme nom, ou inversément.

Par exemple, *futuriste* est à la fois un adjectif et un nom.

Si une des formes adjectives n'existe pas, nous trouverons des classes de suffixes identiques. Par exemple, les classes NOM 73 et ADJ 15 sont identiques : *-iste, -istes*.

Une liste de ces classes de suffixes identiques figure en annexe, et les problèmes posés par la reconnaissance de ces classes seront exposés dans le chapitre suivant.

Dans le chapitre qui suit, nous allons étudier le problème très important du choix du stem d'un mot. En effet, le chapitre III montrera que le dictionnaire sert à associer à un mot un ensemble d'indications sémantiques et syntactiques, utilisées par le système de documentation, pour rechercher les informations correspondant à la demande. Mais nous verrons que ces indications sémantiques et syntaxiques sont attachées à un stem, et que le problème de la reconnaissance d'un mot se ramène donc à celui de la détermination du stem correspondant à un mot, d'où l'importance du choix des stems de mot.

C H A P I T R E II

C R E A T I O N des S T E M S

II.1 INTRODUCTION

Pour réduire la taille du dictionnaire, nous verrons qu'il est préférable d'y introduire des stems de mot, et non des mots complets.

Le choix de ces stems est un problème important, car c'est par l'intermédiaire du stem que le système de documentation associera à un mot les fonctions sémantiques et syntaxiques qui lui correspondent.

Le chapitre précédent a permis d'introduire la notion de classe de suffixes, regroupement de plusieurs suffixes. Nous justifierons, ici, l'existence et l'utilisation de ces classes de suffixes.

Ce chapitre permettra de montrer, avec les bases choisies, qu'il est possible d'automatiser l'opération de création des stems de mot, en associant le même stem aux différentes formes du mot, et en déterminant, dans un des cas, la fonction grammaticale du mot.

II.2 INFLUENCE DES STEMS SUR LE VOLUME DU DICTIONNAIRE. OPTIMISATION

Nous allons montrer que la division d'un mot en stem+suffixe va permettre de réduire la taille du dictionnaire, que l'on optimisera le volume du dictionnaire en y introduisant les stems les plus courts possibles.

II.2.1 Structure arborescente

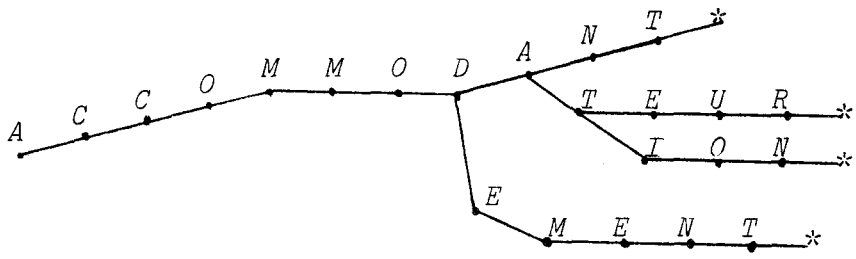
Pour simplifier la procédure de recherche, dans de très nombreux systèmes, les dictionnaires sont mémorisés sous forme d'une structure arborescente.

Il existe un certain nombre de noeuds initiaux (ou racines) correspondant aux caractères qui peuvent figurer en première position dans une chaîne.

Une chaîne de caractères est représentée par un chemin entre un noeud initial, et un noeud terminal (ou feuille) de l'arbre.

Par exemple :

Soit l'arbre constitué par les mots : *accommodant*
accommodateur
accommodation
accommodement



Dans ce mode de représentation, la partie commune à deux chaînes de caractères ne figure qu'une fois. Donc, plus ces chaînes sont proches l'une de l'autre, plus l'augmentation du volume de l'arbre est faible, lorsqu'on y introduit une nouvelle chaîne. Par exemple, si on introduit le mot *accommoder* à l'arbre précédent, on crée seulement deux noeuds supplémentaires : - *R* - *.

L'examen de ces chaînes montre que plus on se rapproche des noeuds terminaux de l'arbre, plus ces chaînes diffèrent. Le stockage est d'autant moins

performant que ces parties de chaînes non communes sont importantes.

Si, au lieu d'introduire dans l'arborescence une chaîne représentant le mot, on n'introduit qu'une partie de cette chaîne, celle correspondant au stem, en supprimant la partie suffixe, nous réduirons l'extrémité terminale de la chaîne, et donc la partie non commune à d'autres chaînes de l'arbre.

II.2.2 Optimisation du volume occupé par le chaînage

Nous avons vu, à l'aide d'un exemple, que la division d'un mot en stem et suffixe réduira le volume du dictionnaire, en permettant de représenter le mot à l'aide de deux chaînes plus courtes associées.

En effet, le suffixe ne figurera qu'une seule fois dans une liste de suffixes, soit incluse dans le dictionnaire des stems, soit dans un dictionnaire annexe des suffixes, et pourra être associé à un grand nombre de stems.

La chaîne du stem, plus courte que la chaîne du mot correspondant, aura plus de parties communes avec d'autres chaînes.

Il est donc aisé de constater que le volume occupé par la chaînage sera d'autant plus réduit, et le stockage plus performant, que les stems correspondant aux mots à introduire dans le dictionnaire seront courts.

Donc, pour optimiser la taille du dictionnaire, il sera préférable d'y introduire les stems les plus courts possibles. Nous allons, au cours de l'exposé de deux méthodes d'utilisation des stems, examiner les problèmes posés par la recherche d'un mot à l'aide d'un dictionnaire de stems de mot.

II.3 RECHERCHE DU STEM D'UN MOT

Le dictionnaire des stems permet d'associer à un stem déterminé des indications sémantiques, et parfois syntactiques, que le système de documentation utilisera pour la recherche de documents.

Il s'agit donc, par l'intermédiaire du stem, d'associer à un mot les fonctions sémantiques et syntactiques qui lui correspondent ; donc, d'associer à un mot le stem qui lui correspond.

Cette contrainte va influer, comme nous allons le montrer, sur la méthode de reconnaissance du stem du mot.

II.3.1 Méthode du plus long stem contenu

Cette méthode est très souvent utilisée dans les systèmes de documentation automatique [1]. Pour rechercher un mot, on recherche dans le dictionnaire le plus long stem possible contenu dans ce mot,et ce stem est associé au mot cherché.

Cela suppose donc que le stem soit caractéristique d'un mot, donc le plus long possible, et que les suffixes soient assez courts. Certains auteurs parlent d'ailleurs d'ensemble de terminaisons, et non de suffixes.

Dans cette méthode, l'introduction et le choix du stem d'un mot nouveau dépendent des stems voisins déjà existants dans le dictionnaire. Dans certains cas défavorables, on peut constater que, par suite du choix de stems trop courts, on peut être obligé de modifier les stems déjà existants.

Supposons qu'on ait voulu faire figurer dans le dictionnaire le mot *initiation*, en introduisant le stem *initi-*, et en supposant que *-ation* soit un suffixe. Si on veut introduire le mot initial, il faudra choisir un stem différent de *initi-*, bien que *-al* soit un suffixe possible. En effet, la méthode du plus long stem associerait le stem *initi-* aux mots *initial* et *initiation*. Il faudra donc, pour faire figurer le mot initial, modifier le stem de *initiation*, et choisir, par exemple, *initiat-ion*. Le plus long stem contenu dans *initial* sera *initi-*, et celui contenu dans *initiation* : *initiat-* ; les deux mots sont donc maintenant associés à des stems différents, mais il a été nécessaire de modifier un des stems existants pour en introduire un nouveau. Le problème risque d'ailleurs de se reposer pour introduire le mot *initiative*.

La création de mots à partir d'un mot initial, par dérivation par suffixes a donné naissance à de très nombreux mots français. La méthode du plus long stem contenu oblige donc à choisir des stems très courts, pour ne pas trouver trop de cas défavorables lors de la mise-à-jour du dictionnaire.

Cette méthode permet d'associer le même stem à plusieurs mots, et donc de créer un voisinage sémantique entre ces mots. Mais le problème est de limiter cette propriété aux voisinages recherchés, et de ne pas créer de voisinages

sémantiques faux.

Cette méthode permet seulement d'associer à un mot une fonction sémantique. Nous allons exposer une autre méthode, dérivée de celle du plus long stem, qui permettra de vérifier que le suffixe associé au stem trouvé est un suffixe possible, et de déterminer, en outre, la fonction syntactique du mot, pour permettre une analyse plus fine de la demande par le système de documentation.

II.3.2 Amélioration de la méthode. Système Smart : reconnaissance d'un mot et construction du code syntactique

Une amélioration de la méthode du plus long stem contenu consiste à vérifier que la partie de la chaîne du mot non commune avec celle du stem constitue un suffixe possible, en la comparant à une liste de suffixes. Si ceci n'est pas vérifié, une procédure spéciale recherche un autre stem pour le mot.

Le système Smart contient un exemple intéressant d'utilisation de cette méthode. Smart a été réalisé à Harvard, puis à Cornel University, sous la direction de G. Salton [2].

Dans ce système, les dictionnaires et thesaurus utilisent des stems de mots. Le "thésaurus de niveau zéro" est le dictionnaire le plus évolué ; c'est l'association d'un dictionnaire de stems, et d'un dictionnaire de suffixes.

La méthode de recherche d'un mot est basé sur la reconnaissance d'un stem, et la vérification de l'existence d'un suffixe, mais, pour permettre une analyse plus fine du mot dans son contexte, un code syntactique décrivant la fonction grammaticale et le genre du mot est construit sur les informations syntactiques du stem et du suffixe. [3], [2].

Dans le dictionnaire des suffixes, à chaque suffixe correspondent des codes syntactiques partiels qui se superposeront aux codes syntactiques complémentaires du stem. Les exemples sont tirés de la langue anglaise :

. Soit le suffixe : *-fully*. Le code associé sera ADV000, indiquant qu'il s'agit d'un suffixe d'adverbe, en anglais.

. Le suffixe : *-ies*. Un des codes associé sera VOOSO, indiquant que c'est un suffixe de verbe, à la troisième personne du singulier.

Dans le dictionnaire des stems, une partie des codes associés au stem donne une indication syntaxique partielle. Par exemple, pour le stem *multipl-*, le code syntactique indique qu'il s'agit d'un stem de verbe transitif : OT10.

L'association des codes syntactiques du stem *multipl-* et du suffixe *-ies* donnera : (VOOSO) ∪ (OT10) → VT1SO, indiquant qu'il s'agit d'un verbe transitif à la 3° personne du singulier.

Dans cette méthode, il faut que l'union des codes syntactiques partiels ne fasse pas apparaître d'ambiguïté, ce qui complique encore le choix du stem.

Les stems sont codés avec un code syntactique potentiel. Par exemple, le stem *recti-* est codé comme un verbe probable, car il peut donner *recti-fy*.

Mais ce codage potentiel peut introduire une ambiguïté. Par exemple, si le stem : *capital-* est codé comme un verbe potentiel, pour être associé au suffixe : *-ize*, le nom pluriel *capitals* sera reconnu comme un verbe à la troisième personne du singulier.

Pour tourner cette difficulté, dans le système Smart, on considère que la fin d'un mot peut être un suffixe composé lui-même de plusieurs suffixes. Par exemple : *-lessly = - less-ly*.

L'exemple ambigu pourra alors être modifié : on introduit le stem *capit-* avec un codage partiel de verbe. Le suffixe *-als* sera associé à un code syntactique complet de nom pluriel qui, lors de la réunion des codes syntactiques partiels, annulera le code partiel de verbe du stem. Le verbe *capitalize* sera retrouvé grâce à une double consultation de la liste des suffixes : *capit-al-ize*.

Dans cette méthode, encore, l'introduction d'un nouveau stem peut donc obliger à modifier les stems déjà existants.

De plus, la recherche de deux mots différents, qui ne sont pas deux formes différentes du même mot peut aboutir au même stem. Par exemple, le stem :

reduct- n'aura pas de code syntactique, et pourra être combiné avec des suffixes communs, dont le code est complet ; tels que :

- *ion* : suffixe de nom singulier
- *ible* : suffixe d'adjectif.

Le voisinage sémantique ainsi créé peut être valable dans certains cas, par exemple : *réduction - réductible*. Mais il existe d'autres cas où l'association de deux mots au même stem peut engendrer une erreur dans la recherche.

Un des rapports sur le système Smart [4] cite le cas des mots *compressors* et *compressible* qui sont regroupés par erreur.

Cette méthode est donc une extension de la méthode du plus long stem contenu. Elle n'utilise pas les codes syntactiques partiels, lors de la recherche d'un mot dans le dictionnaire pour déterminer si l'association stem-suffixe trouvée correspond bien au mot cherché, et ne crée pas de voisinage sémantique inexact. En effet, on vérifie seulement que l'association des chaînes du stem et du suffixe donne la chaîne du mot, mais il se peut que le stem ainsi déterminé ne soit pas le stem correspondant au mot.

Les stems obtenus par cette méthode sont plus courts que les stems correspondant à la première méthode, car la vérification de l'existence du suffixe aide à identifier plus précisément le stem du mot. Donc, le stockage correspondant sera plus performant.

Nous allons montrer que dans la méthode que nous avons choisie, nous évitons d'associer les mêmes codes syntactiques et sémantiques à deux mots qui n'ont pas de voisinage sémantique strict, même si la recherche dans le dictionnaire leur affecte le même stem, en liant un stem déterminé à un ensemble fini de suffixes, ou classe de suffixes, par l'intermédiaire de codes syntactiques. Nous montrerons que cette méthode permet de choisir des stems plus courts, et en particulier, si pour un mot à introduire, plusieurs stems sont possibles, le plus court sera choisi, afin de réduire le volume occupé par le dictionnaire.

II.4 JUSTIFICATION DU CHOIX DU PLUS COURT STEM POSSIBLE

Ce paragraphe va expliciter la méthode de reconnaissance d'un mot dans le dictionnaire, basée sur la concordance stem-suffixe par l'intermédiaire des codes syntactiques de la partie stem et de la partie suffixe de ce mot. Cette méthode peut permettre, lors de l'introduction d'un nouveau mot dans le dictionnaire, plusieurs choix stem-suffixe possibles. Le choix portera alors sur le stem le plus court possible, afin d'optimiser le volume occupé.

II.4.1 Utilisation des codes syntactiques pour la recherche d'un mot

Le dictionnaire ne contient pas des mots complets, mais des stems de mots et des suffixes.

A chaque stem est associé un, ou plusieurs ensembles de codes.

Chaque ensemble de code est divisé en deux sous-ensembles : les codes sémantiques et les codes syntactiques.

- Les codes sémantiques permettent au système de déterminer l'utilisation des informations sémantiques associées au stem, et, éventuellement, le concept associé au stem.

- Les codes "syntaxiques" permettent de définir la fonction grammaticale du mot complet, et un ensemble de suffixes, ou classe de suffixes, qui peuvent être associés à ce stem.

Les codes syntaxiques permettent de définir la classe des suffixes admis comme terminaison du stem, et donc, un ensemble de mots du langage courant.

Par exemple : soit le stem : *malad-* ; la partie syntactique du codage pourra être représentée par : ADJ09, indiquant qu'il s'agit d'un adjectif, et que la classe de suffixes admise est la 9ème des classes de suffixes d'adjectif. Le code syntaxique de cette classe sera représenté également par ADJ09.

Si la classe de suffixes ADJ09 est : *-if, -ifs, -ive, -ives* ; nous aurons associé les mots : *maladif, maladifs, maladive, maladives* au même stem : *malad-*. Un voisinage sémantique strict a été, ainsi, déterminé entre un ensemble de mots

qui sont les différentes formes possibles du même mot : *"maladif"*. La fonction grammaticale de ce mot est, en même temps, retrouvée : la classe de suffixes définie est une classe de suffixes d'adjectifs.

Donc,le code syntaxique associé à une classe de suffixes permet d'associer les différentes formes d'un mot et d'en déterminer la fonction grammaticale.

Cette méthode permet, en associant le même stem pour deux mots distincts, de différencier les codes sémantiques qui s'y rattachent.

Si, par exemple, le nom : *malade* a le même stem : *malad-* que l'adjectif *maladif*, un autre ensemble de codes que celui de l'adjectif maladif aura pour partie syntactique NOM02, par exemple, indiquant qu'il s'agit d'un nom, et que la classe de suffixes associée est la seconde des classes de noms, dont le code syntactique sera également NOM02.

Si la classe NOM02 est représentée par les suffixes : *-e, -es ;* *malad-* [NOM02] sera associé aux noms : *malade* et *malades*.

La recherche du mot, associant deux parties stem et suffixe ayant même code syntactique permettra de différencier *malad* $\xrightarrow{\text{[ADJ09]}}$ *if* de *malad* $\xrightarrow{\text{[NOM02]}}$ *e*. Les voisinages sémantiques abusifs seront ainsi évités, si nécessaire.

Le paragraphe suivant va permettre d'étudier les problèmes posés par l'introduction d'un nouveau stem dans le dictionnaire, et montrer que le choix du stem ne dépend pas des stems voisins déjà existants.

II.4.2 Problème du choix du stem d'un mot

Dans la méthode du plus long stem contenu, pour introduire un nouveau stem, il est nécessaire d'examiner les stems voisins déjà existants, et de les modifier si le cas est défavorable. Ceci alourdit la procédure de mise-à-jour du dictionnaire. La relation stem-classe de suffixes va permettre de retrouver un mot, même si le stem qui lui correspond n'est pas le plus long stem contenu.

Soit le mot *initiative*, qui a pour stem correspondant : *initiat-*, dont le code syntactique est : NOM89. La relation peut être notée sous la forme :

initiat $\xrightarrow{\text{[NOM89]}}$ *ive*.

Soit le mot *initiation*, qui correspond à la décomposition : *initi* $\xrightarrow{\text{[NOM19]}}$ *ation*.

Lors de la recherche du mot *initiation*, le plus long stem trouvé sera *initiat-*, mais le suffixe *-ion* n'appartient pas à la classe NOM89, et le stem *initiat-* sera rejeté. Il suffira alors de rechercher si un des stems plus court admet la fin du mot parmi les suffixes possibles. Ainsi, la décomposition *initi* $\xrightarrow{\text{[NOM19]}}$ *ation* sera acceptée.

Donc, dans ce cas défavorable pour la méthode du plus long stem, il n'a pas été nécessaire de modifier les stems déjà existants. La recherche du mot, dans ce cas, est un peu plus longue, mais le mot a bien été associé au stem qui lui correspond.

Cela permet donc, lors de l'introduction du stem d'un mot, de ne pas tenir compte pour ce stem du voisinage déjà existant dans le dictionnaire, et d'alléger ainsi la procédure de mise-à-jour. Il est, ainsi, également possible d'automatiser le choix des stems.

Cette méthode conduit donc, contrairement aux deux méthodes précédentes, à envisager des stems aussi voisins que possible, et, à la limite, le même stem pour plusieurs mots, car dans la recherche, la partie de vérification de la relation syntactique est plus rapide que la partie recherche d'une chaîne de caractères dans l'arborescence.

Le paragraphe qui suit va permettre de justifier le choix du plus court stem possible, et montrer que ce choix permet de diminuer la taille du dictionnaire, et de construire des stems voisins, sans créer de voisinages sémantiques abusifs.

I.4.3 Justification du plus court stem possible

Précédemment, nous avons vu qu'il n'est pas nécessaire qu'à un mot soit associé le plus long stem contenu dans ce mot, pour retrouver dans le dictionnaire le stem qui correspond à ce mot. D'autre part, dans une structure arborescente, plus les stems sont courts, plus le volume du dictionnaire sera réduit.

Il est possible que, pour un même mot, plusieurs décompositions soient possibles ; c'est-à-dire qu'il existe, dans la chaîne de ce mot, plusieurs suffixes possibles, appartenant à des classes de suffixes différentes, telles que le code syntactique corresponde à la fonction grammaticale de ce mot et telles qu'elles associent dans un voisinage sémantique strict les différentes formes de ce mot. La méthode permet, comme nous l'avons vu, de choisir une des classes de suffixes possible. Pour optimiser la taille du dictionnaire, c'est la classe, parmi celles possibles, qui détermine le stem le plus court qui sera choisie.

Par exemple, le mot *initiations* peut donner les décompositions possibles :

initi $\xrightarrow{\text{NOM19}}$*ations*, *initia* $\xrightarrow{\text{NOM64}}$*tions*, *initiati* $\xrightarrow{\text{NOM60}}$*ons*, *initiation* $\xrightarrow{\text{NOM1}}$*s*

Le stem le plus court sera choisi : *initi* $\xrightarrow{\text{NOM19}}$*ations*.

Nous allons montrer que cette méthode va permettre de déterminer des stems voisins, et d'áccélérer la reconnaissance d'un mot dans le dictionnaire.

II.4.4 Détermination de stems voisins

De nombreux mots de la langue française ont été créés, à partir d'autres mots, par dérivation par suffixe.

Un exemple va permettre de montrer que la méthode du plus court stem possible donne, le plus souvent, le même stem pour l'ensemble de ces mots, si la dérivation par suffixe ne modifie par l'orthographe du radical.

Par exemple, soient les mots : *initial ; initiation ; initiateur ; initiatrice ; initier ; initié* (nom) *; initié* (adjectif) et les décompositions possibles de ces mots en stem-suffixe compatibles

Initi $\xrightarrow{\text{ADJ04}}$ *al*			
Initi $\xrightarrow{\text{NOM19}}$ *ation*	*initia* $\xrightarrow{\text{NOM64}}$ *tion*	*initiati* $\xrightarrow{\text{NOM60}}$ *on*	*initiation* $\xrightarrow{\text{NOM01}}$
Initi $\xrightarrow{\text{NOM18}}$ *ateur*	*initia* $\xrightarrow{\text{NOM65}}$ *teur*	*initiat* $\xrightarrow{\text{NOM81}}$ *eur*	*initiateur* $\xrightarrow{\text{NOM01}}$
Initi $\xrightarrow{\text{NOM78}}$ *atrice*	*initia* $\xrightarrow{\text{NOM66}}$ *trice*	*initiatric* $\xrightarrow{\text{NOM72}}$ *e*	*initiatrice* $\xrightarrow{\text{NOM01}}$
Initi $\xrightarrow{\text{VER20}}$ *er*			
Initi $\xrightarrow{\text{ADJ25}}$ *é*	*initié* $\xrightarrow{\text{ADJ00}}$		
Initi $\xrightarrow{\text{NOM72}}$ *é*	*initié* $\xrightarrow{\text{NOM01}}$		

Ce tableau montre que la méthode du plus court stem possible associera aux sept mots le même stem : *initi-*. A ce stem seront associés sept ensembles de codes, dont la partie syntactique permettra de choisir, pour un des mots, l'ensemble de codes qui lui correspond.

L'exemple montre, de plus, que la méthode permet d'associer à une chaîne de lettres ambiguë tous les codes correspondant aux diverses fonctions grammaticales de la chaîne. Cela permet, pour la fonction sémantique du mot, de définir qu'elle est l'union des fonctions sémantiques des diverses interprétations de cette chaîne.

Ici, le mot *initié* sera reconnu comme un verbe (participe passé), un nom, ou un adjectif.

Remarque : pour une chaîne de lettres ambiguë, c'est le seul cas où le choix du stem possible, dans cette méthode, n'est pas indifférent. Comme dans l'exemple donné, le stem doit être le même pour les différents mots ambigus. Ceci est dû à la procédure de recherche qui s'arrête dès qu'un stem acceptable a été trouvé pour le mot, mais permet de trouver pour ce stem plusieurs fonctions grammaticales correspondant à la chaîne du mot. En effet, afin de ne pas augmenter les temps de recherche, la procédure ne détecte pas si un autre stem pourrait être également associé au mot, dans ce cas ambigu.

Ces ambiguïtés sont rares, en dehors des participes pris comme adjectif,

ou comme nom, et les classes de suffixes ont été étudiées pour permettre, dans ce cas, d'associer effectivement le même stem aux différentes chaînes.

Un problème important va être abordé maintenant: celui de la génération des stems à partir des mots, afin d'automatiser cette opération qui est très lourde, surtout lors de la création du dictionnaire.

I.5 GENERATION DES STEMS

Pour déterminer le stem d'un mot à introduire dans le dictionnaire, le problème est d'associer à ce mot une classe de suffixes telle que :

- son code syntactique corresponde à la fonction grammaticale du mot

- les suffixes de la classe permettent d'associer les différentes formes du mot au stem ainsi déterminé, dans un voisinage sémantique strict.

Donc, pour automatiser la création des stems, le problème se ramène à celui de caractériser une classe de suffixes parmi l'ensemble de ces classes. Deux méthodes vont être présentées, avec leur champ d'application.

I.5.1 Caractérisation d'une classe de suffixes

Pour caractériser une classe de suffixes parmi l'ensemble de ces classes, il est souhaitable d'utiliser le moins de paramètres possible. Deux solutions sont alors possibles, pour déterminer une classe :

- soit par la fonction grammaticale de la classe, et un des suffixes

- soit par deux suffixes appartenant à la même classe.

I.5.1.1 Caractérisation d'une classe de suffixes par sa fonction grammaticale et un des suffixes

La fonction grammaticale permet de diviser l'ensemble des classes de suffixes en quatre sous-ensembles ; les classes : de nom, d'adjectif, de verbe, et d'adverbe.

Il reste donc à différencier une classe parmi un de ces sous-ensembles. Il est intéressant de caractériser cette classe à l'aide du plus petit nombre d'éléments possibles. Examinons donc si, pour chacun des sous-ensembles, une forme de suffixe permet de caractériser complètement une classe.

a) Pour les noms. L'examen des classes de nom montre que la forme de suffixe singulier permet de différencier une classe dans le sous-ensemble des classes de suffixes de nom, à condition d'y inclure les différentes

formes du pluriel, régulières ou non, qui correspondent à la forme singulier.

Par exemple : classe NOM06 : *-ail, -ails, -aux*

pour les pluriels irréguliers : *un vitrail ; des vitraux.*

b) Pour les adjectifs. La forme de suffixe masculin permet de reconnaître une classe de suffixes d'adjectif, en étendant cette classe aux formes irrégulières du pluriel.

Par exemple : classe ADJ04 : *-al, -ale, -als, -aux, -ales*

pour les pluriels : *bancal → bancals ; banal → banaux.*

c) Pour les adverbes. Le suffixe permet de déterminer la classe de suffixes d'adverbe qui, dans ce cas particulier, est une classe à un seul élément.

d) Pour les verbes. Dans de nombreux cas, une seule forme de suffixe ne permet pas de déterminer une classe.

Par exemple, l'infinitif *-ir* peut donner, pour la troisième personne du pluriel de l'indicatif présent :

-issent : *haïssent*

-ent : *assaillent.*

Donc, cette méthode permet de déterminer sans ambiguïté les classes de suffixes de nom, d'adjectif ou d'adverbe. Mais les paramètres donnés sont insuffisants pour caractériser une classe de suffixes de verbe.

Il est donc nécessaire de rechercher si d'autres paramètres ne permettent pas de différencier une classe de suffixes dans tous les cas.

II.5.1.2 Caractérisation d'une classe de suffixes par deux suffixes appartenant à cette classe

Il n'est plus possible, comme dans le cas précédent, de séparer l'ensemble des classes de suffixes en sous-ensembles, fixés à priori. Ici, la première forme de suffixe détermine un sous-ensemble : celui des classes de suffixes où cette

première forme de suffixe figure. Le problème est donc, maintenant, de savoir si, dans ce sous-ensemble, une deuxième forme de suffixe suffit pour caractériser une seule classe.

Afin de standardiser les données, il est nécessaire d'introduire une contrainte supplémentaire : que les formes de suffixes soient les mêmes pour les classes ayant même fonction grammaticale.

L'examen des classes de suffixes permet de choisir les deux formes de suffixes suivantes :

- Pour un nom : les formes singulier et pluriel.
- Pour un adjectif : les formes masculin singulier et féminin pluriel.
- Pour un adverbe : deux fois le suffixe.
- Pour un verbe : les formes infinitif, et troisième personne du pluriel du présent de l'indicatif.

Les résultats montrent, en ce qui concerne le sous-ensemble des classes de suffixes de verbe, que les deux formes choisies permettent de caractériser l'une quelconque des classes appartenant à ce sous-ensemble.

Pour les noms et les adjectifs, deux formes de suffixes ne permettent pas, dans tous les cas, de déterminer une classe de suffixes parmi l'ensemble des classes de suffixes de nom ou d'adjectif.

En effet, la forme masculin singulier permet de déterminer soit une classe unique, soit deux classes correspondant à des fonctions grammaticales différentes. La méthode précédente a montré que la forme masculin singulier permet de caractériser une classe de suffixes, dans le sous-ensemble des classes de nom, ou celui des classes d'adjectif. Ici, la méthode ne pourra déterminer au maximum que deux classes : une correspondant à la fonction grammaticale de nom, et l'autre d'adjectif.

Par exemple, le suffixe : *-ateur* va déterminer deux classes : NOM18 et ADJ31, qui sont respectivement : NOM18 : *-ateur, -ateurs*

ADJ31 : *-ateur, -atrice, -ateurs, -atrices.*

La deuxième forme de suffixe va permettre de différencier la classe de nom de la classe d'adjectif, si la première forme de suffixe détermine un sous-ensemble de deux classes.

Considérons l'ensemble de suffixes constitué, pour les classes de nom, par les formes singulier et pluriel, et pour les classes d'adjectif, par les formes masculin singulier et féminin pluriel. On a ainsi constitué un ensemble de classes telles que chaque classe est contenue dans une des classes initiales. On appellera ces classes des classes réduites, et on leur donnera le même code syntactique que celui des classes initiales.

Dans l'exemple, le suffixe *-ateur* va déterminer les deux classes réduites :

NOM18 : *-ateur, -ateurs*

ADJ31 : *-ateur, -atrices.*

La deuxième forme de suffixe permet, dans le sous-ensemble défini par la première forme, de déterminer la classe réduite, et son code, et donc la classe initiale de même code.

L'exemple montre qu'il est, en général, possible de déterminer une classe unique. Il existe cependant un certain nombre de cas ambigus.

Ces cas ambigus peuvent être dûs à la similitude des formes masculin et féminin de certains adjectifs. Par exemple, pour les adjectifs en *-ible*, la forme masculin ressemble à la forme féminin, et il existera deux classes de suffixes identiques : la classe NOM73, et la classe ADJ15 : *-ible, -ibles.*

Certains mots de la langue française sont utilisés à la fois comme nom et comme adjectif, et il n'est pas toujours possible de différencier les classes qui correspondent à chacune des fonctions grammaticales. Par exemple : *fumiste* est à la fois un nom et un adjectif, et deux classes correspondent à ce mot : NOM51 et ADJ21 : *-iste, -istes.*

Ces cas ambigus ne concernent qu'un nombre restreint de classes, qui sont données en annexe.

Ces deux méthodes de caractérisation d'une classe de suffixes ont été

programmées. Les deux programmes sont basés sur le même principe, qui va être maintenant exposé : la recherche de l'ensemble des suffixes possibles dans une chaîne représentant une forme d'un mot.

II.5.2 Principe de la recherche des suffixes possibles dans une chaîne de caractères

Cette méthode est, comme nous allons le voir, une adaptation de la méthode de recherche de la plus longue chaîne contenue.

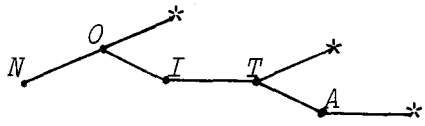
Soit un ensemble de suffixes. Considérons les chaînes de lettres obtenues en écrivant ces suffixes à l'envers. Ainsi, par exemple, la chaîne *"noita"* correspondra au suffixe : *-ation*. Représentons l'ensemble des chaînes ainsi obtenues sous forme d'arborescence.

Par exemple :

Soient les suffixes : *-ation, -tion, -on*.

On leur associera les chaînes : *noita, noit, no*.

La représentation sous forme arborescente donnera :



Si on recherche, à l'aide de l'arborescence ainsi créée, la plus longue chaîne de cette arborescence contenue dans la chaîne créée en écrivant un mot à l'envers. Si on note, dans cette procédure de recherche, les chaînes successives de l'arborescence qui sont trouvées, on pourra déterminer tous les suffixes possibles pour un même mot.

On appellera les chaînes de cette arborescence des chaînes inverses de suffixes.

Par exemple, le mot *initiation* donnera la chaîne : *noitaitini*. La procédure de recherche trouvera les chaînes contenues : *no, noit, noita* qui correspondent aux suffixes : *-on, -tion, -ation*.

On peut remarquer qu'il n'est pas nécessaire d'écrire le mot à l'envers, mais d'analyser la chaîne représentant ce mot de la droite vers la gauche.

Cette méthode a été utilisée dans les deux programmes de création des stems. On peut noter que, pour réaliser la programmation, les chaînes inverses de suffixe ont été rangées suivant la même structure arborescente que le dictionnaire, qui sera étudié au chapitre suivant. Ceci a permis de réutiliser, pour créer cette structure, le programme de création du dictionnaire lui-même. La procédure de recherche de la plus longue chaîne contenue a été réutilisée, en la complétant, pour la recherche d'un mot dans le dictionnaire.

I.6 PROGRAMMES DE CREATION DES STEMS

Les deux programmes de création des stems sont basés sur les deux méthodes vues précédemment, de caractérisation des classes de suffixes.

I.6.1 Creatstems - gramm + mot : détermination du stem d'un mot à partir de la fonction grammaticale et d'une des formes de ce mot

Cette méthode va chercher à déterminer une classe de suffixes par sa fonction grammaticale et une des formes de suffixe. Les résultats précédents montrent qu'il est possible de caractériser ainsi sans ambiguïté une classe de suffixes de nom, d'adjectif ou d'adverbe, mais non de verbe.

Le principe de cette méthode est le suivant :

a) arborescence associée aux formes de suffixes

La méthode de la recherche des suffixes possibles dans une chaîne de caractères utilisera ici les chaînes inverses des formes de suffixes suivantes :

- forme singulier pour les classes de nom
- forme masculin singulier pour les classes d'adjectif
- forme du suffixe d'adverbe.

A chacun des suffixes est associé le code syntaxique correspondant à la classe de suffixes à laquelle il appartient.

b) données

Les données nécessaires sont :

- le code de la fonction grammaticale du mot
- une des formes du mot :
  - singulier pour les noms
  - masculin singulier pour les adjectifs
  - l'adverbe
  - une forme quelconque du verbe.

c) principe de creatstems-gramm+mot

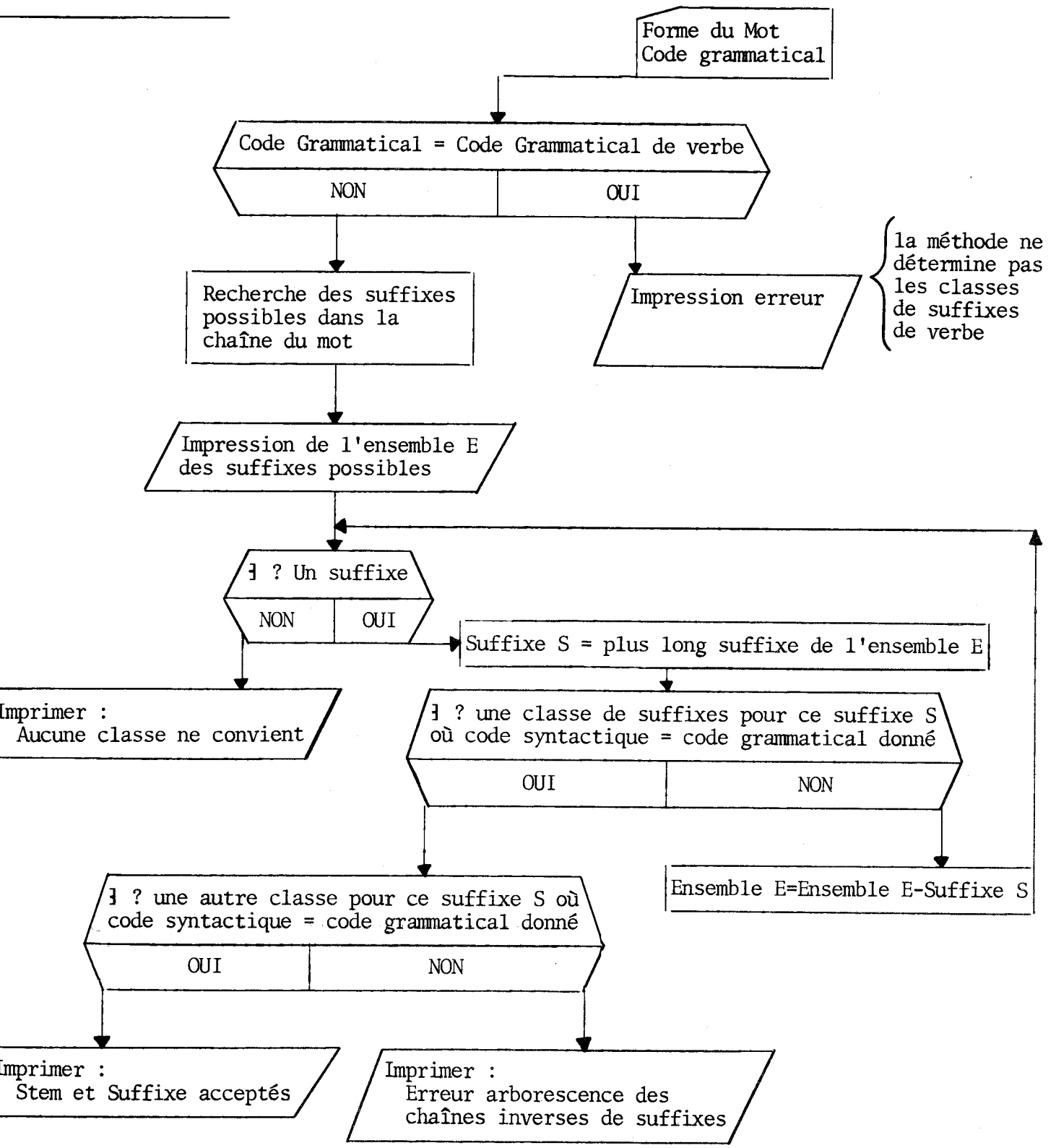
On recherche l'ensemble des suffixes possibles pour la forme du mot.

On cherche, dans l'ensemble ainsi créé, le suffixe le plus long possible, appartenant à une classe dont le code grammatical correspond à celui donné avec la forme du mot.

Si on a trouvé un suffixe acceptable, on recherche, de plus, si ce suffixe n'appartient pas à une autre classe de suffixes correspondant au code grammatical donné. Ce cas ambigu serait dû à une erreur dans l'arborescence associée aux formes de suffixes.

L'ensemble de cette procédure est résumé dans un organigramme simplifié. L'emploi et la portée de cette méthode seront exposés après le paragraphe suivant, qui traite de la seconde méthode : creatstems - double - forme.

CREATSTEMS-GRAMM+MOT



II.6.2 Creatstems-double-forme : détermination du stem d'un mot à partir de deux formes de ce mot

Dans cette méthode, on va chercher à déterminer une classe de suffixes par deux suffixes appartenant à cette classe. Il est possible de déterminer ainsi une classe de suffixes de verbe sans ambiguïté, mais comme on l'a vu précédemment, il y a, dans certains cas, ambiguïté entre une classe de nom et d'adjectif.

Le principe de cette méthode est le suivant :

a) arborescence associée aux formes de suffixes

La méthode de la recherche des suffixes possibles dans une chaîne de caractères utilisera ici les chaînes inverses des formes de suffixes suivantes :

- Formes singulier et pluriel pour les noms.
- Formes masculin singulier et féminin pluriel pour les adjectifs.
- Le suffixe de l'adverbe.
- Les formes infinitif et troisième personnes du pluriel de l'indicatif pour les verbes.

A chacun des suffixes est associé, comme dans la méthode précédente, le code syntaxique correspondant à la classe de suffixes à laquelle il appartient.

b) données

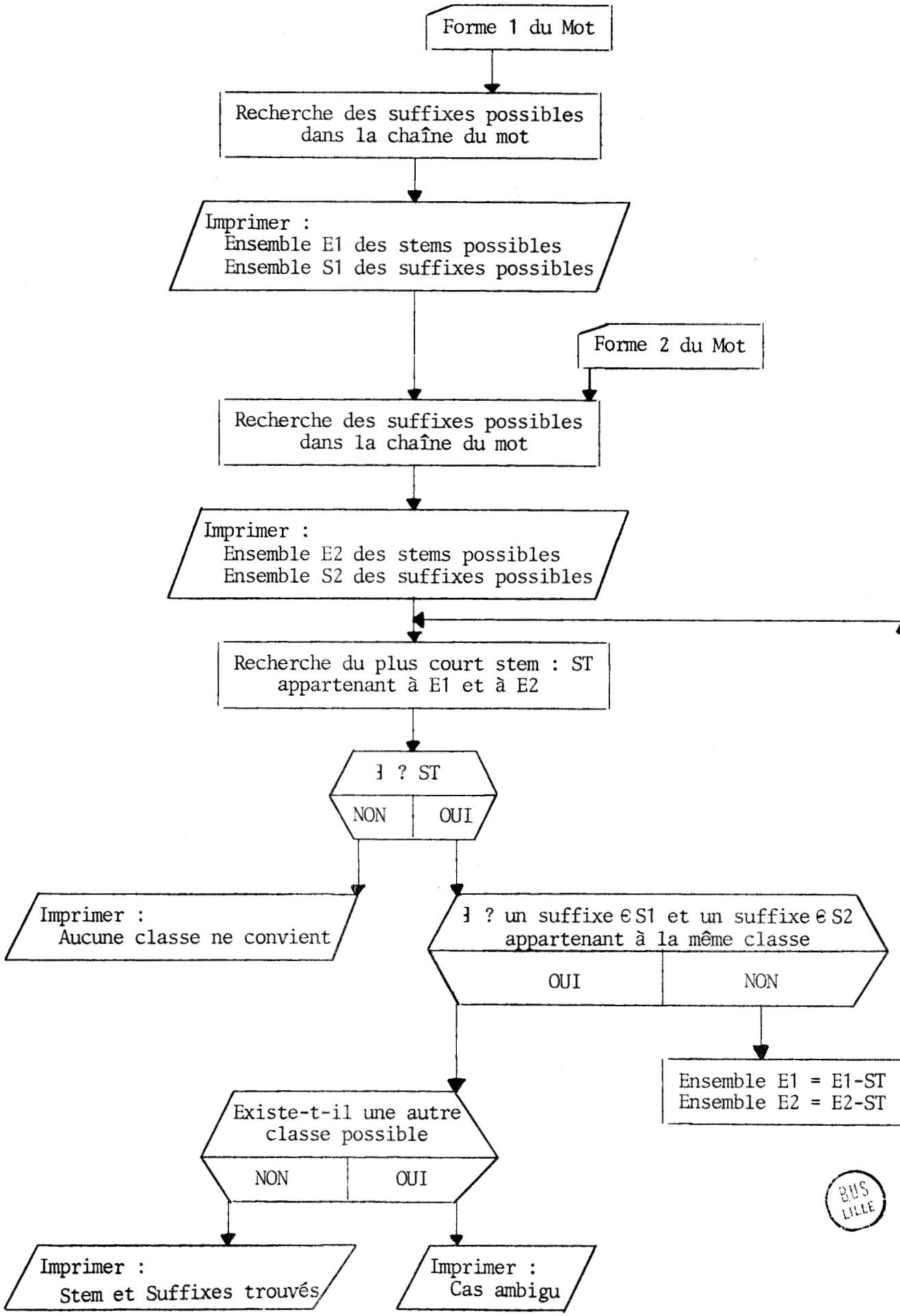
Deux formes du même mot seront nécessaires :

- Les formes singulier et pluriel pour un nom.
- Les formes masculin singulier et féminin pluriel pour un adjectif.
- Deux fois la forme d'un adverbe.
- Les formes infinitif et troisième personne du pluriel de l'indicatif pour un verbe.

c) principe de Creatstems-double-forme

On mémorise pour chacune des formes du mot l'ensemble des suffixes possibles. On analyse ensuite les deux ensembles de suffixes possibles obtenus, en recherchant le stem possible le plus court.

CREATSTEMS-DOUBLE-FORME



Une chaîne de lettres sera la chaîne d'un stem possible si, dans chacune des deux formes données du mot, à cette chaîne de lettres peut être associée une chaîne qui est celle d'un suffixe, et si les deux suffixes ainsi trouvés appartiennent à une seule et même classe de suffixes. Si pour un stem donné, il n'a pas été possible d'y associer une classe de suffixes, on recherche si un stem plus long est possible.

Une combinaison des deux méthodes de création des stems peut permettre de créer automatiquement les stems de mots, dans tous les cas ; c'est ce qui va être exposé dans le paragraphe qui suit.

I.7 CREATION DES STEMS

L'étude des deux méthodes de création des stems a montré que Creatstems-gramm+mot détermine le plus court stem possible, sans ambiguïté pour les noms, les adverbes, les adjectifs, tandis que Creatstems-double-forme détermine les suffixes de verbe sans ambiguïté.

Il est donc possible d'envisager une utilisation successive des deux méthodes pour déterminer l'ensemble des stems des mots à introduire, pour la création du dictionnaire ; donc pour un nombre assez important de mots.

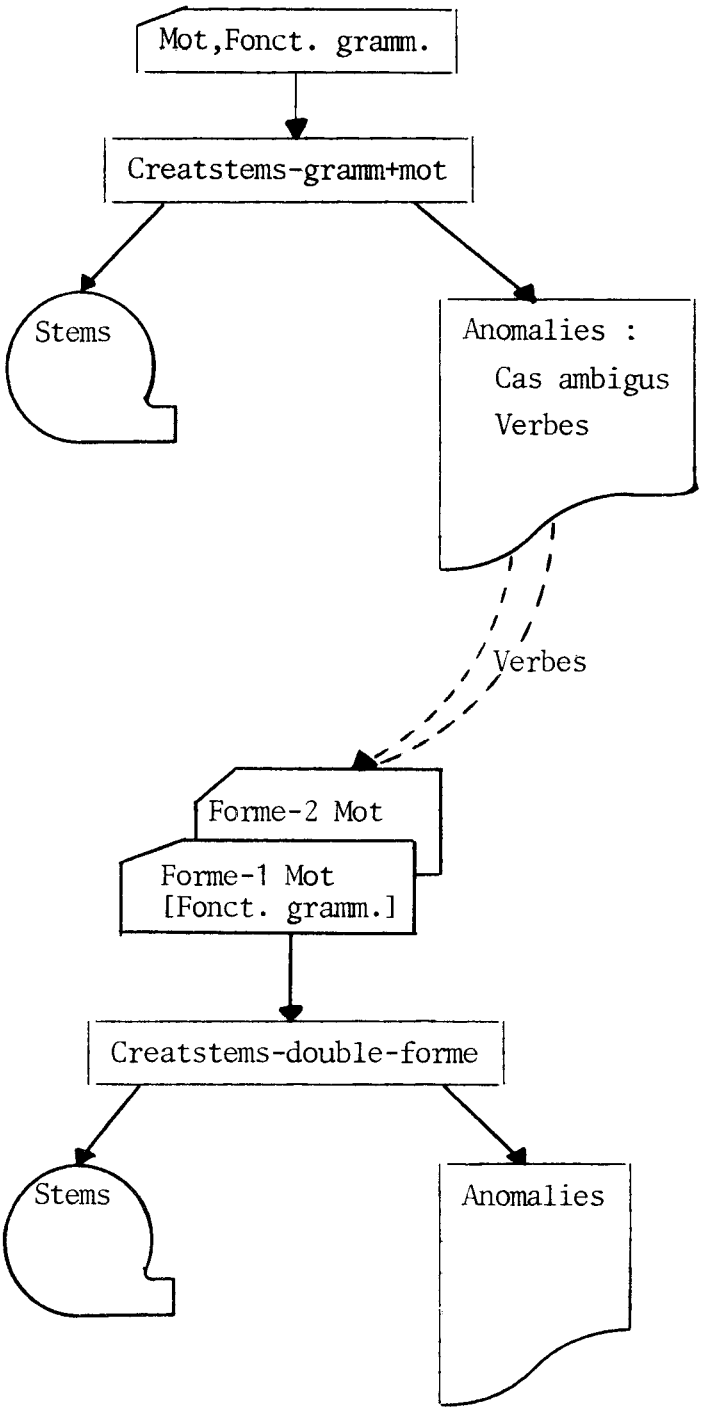
Dans l'ordinogramme : "création des stems", on peut prévoir un format des données tel que les données des verbes,utilisées pour creatstems-gramm+mot puissent être complétées, et réutilisées pour creatstems-double-forme (association de deux formes du mot pour créer le stem). On peut lister les anomalies et les cas ambigus, et faire un choix, pour les cas ambigus, parmi les solutions proposées.

Pour la mise à jour du dictionnaire, donc pour un ensemble de mots assez restreint, il semble préférable d'utiliser la méthode Creatstems-double-forme, d'imprimer les cas ambigus, et de faire un choix parmi les cas listés par cette méthode.

Pour créer le dictionnaire qui a servi dans le système de documentation décrit dans un des chapitres suivants, la méthode Creatstems-double-forme a été utilisée. Comme ce système de documentation ne comporte pas d'analyse syntactique de l'ensemble des mots d'une question posée, les codes syntaxiques n'ont été utilisés que pour associer à un stem une classe de suffixes qui crée un voisinage sémantique strict des différentes formes du mot. Le code de la fonction grammaticale du mot n'a donc pas été pris en compte, et les cas ambigus ont été éliminés en supprimant une des deux classes ambiguës. La classe qui correspond à l'utilisation grammaticale la plus fréquente a été seule conservée.

Ceci a été fait afin de limiter au maximum la vérification des données. Dans le programme écrit, il faut en effet entrer deux cartes correspondant aux deux formes du mot. Cela permet de détecter plus facilement les omissions de cartes, les erreurs de perforation, et certaines erreurs d'orthographe ; même si le volume des cartes données a été doublé. Dans la méthode Creatstems-gramm+mot, il est impossible de détecter une erreur sur le code grammatical, et cela

ORDINOGRAMME : "Création des stems"



oblige à vérifier très soigneusement les cartes, et l'erreur sur le code grammatical ne peut pas toujours être détectée par le programme. Dans la méthode Creatstems-double-forme, par contre, un code 1 ou 2 en première colonne indique qu'il s'agit de la première ou de la seconde forme du mot dont on recherche le stem. L'alternance des codes 1 et 2 permet de détecter facilement les cartes omises, ou les bordereaux de perforation incomplets, d'autre part, les erreurs sur ces codes entraînent un rejet des cartes de donnée correspondantes, et évite ainsi l'introduction de stems erronés.

Il est donc possible d'automatiser la création des stems de mots, qui est une opération très lourde dans un système de documentation. Les chapitres qui suivent vont expliciter la structure du dictionnaire, et comparer plusieurs dictionnaires qui ont été créés, afin de donner une idée des performances du stockage, et des temps de recherche.

C H A P I T R E III

P R I N C I P E de M E M O R I S A T I O N du D I C T I O N N A I R E

[I.1 I N T R O D U C T I O N

Ce chapitre, ainsi que les deux qui vont suivre sont consacrés à l'étude du dictionnaire, de sa structure, et de ses performances.

Le principe de stockage sous forme arborescente, et la méthode de mémorisation de cette arborescence sont exposés dans ce chapitre. Les algorithmes, utilisant ce mode de mémorisation, et qui permettent la création, la mise-à-jour, la recherche de mots dans le dictionnaire seront décrits dans le chapitre qui suit.

III.2 PRINCIPE DE LA MEMORISATION : METHODE PAR CHAINAGE

Les chaînes de caractères contenues dans le dictionnaire sont stockées suivant une structure arborescente. Le principe en a déjà été esquissé dans le chapitre précédent. Il va être repris maintenant plus en détail, ainsi que la représentation en machine du dictionnaire.

III.2.1 Structure arborescente

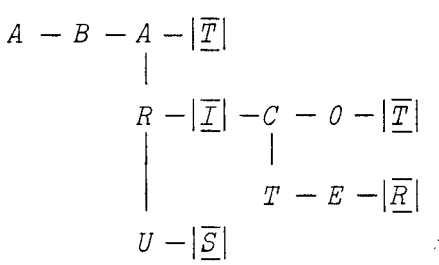
Dans cette structure, la partie commune à deux chaînes de caractères n'est stockée qu'une seule fois.

Dans l'arbre ainsi créé, il existe un certain nombre de noeuds initiaux (ou racines) qui correspondent aux caractères qui peuvent figurer en première position dans une chaîne, et des noeuds terminaux (ou feuilles) qui correspondent à la fin des chaînes. Un mot, représenté par une chaîne de caractères sera un chemin entre un noeud initial et un noeud terminal.

Soit un ensemble de mots, classés par ordre alphabétique :

abat, abri, abricot, abriter, abus.

On peut représenter l'arbre qui correspond à cet ensemble de chaînes de caractères : on relie les caractères par des arcs horizontaux signifiant la succession, et verticaux signifiant l'alternative. On repère le dernier caractère de chaque chaîne par un signe spécial : $|\overline{\_}|$



Dans l'exemple précédent, il existe des chaînes contenues dans d'autres, par exemple : *abri* - *abricot*. Ceci se concrètise sur l'arborescence par un caractère de fin de chaîne suivi d'autres caractères, reliés par des arcs horizontaux.

I.2.2 Caractères spéciaux de fin de chaîne

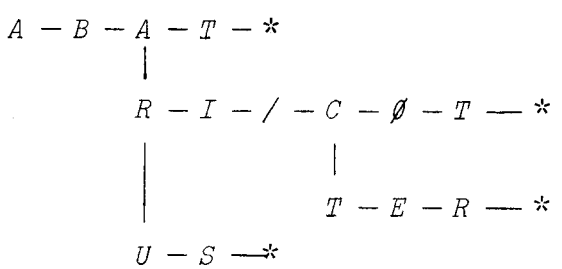
Il est possible, dans l'arbre créé précédemment, d'associer à chaque caractère de fin de chaîne un caractère spécial, qui sera un noeud supplémentaire de l'arbre.

On peut alors distinguer deux caractères spéciaux :

- * : indique que le caractère n'a pas de successeur possible (feuille de l'arbre)

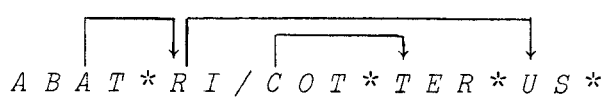
- / : indique que le caractère est la fin d'une chaîne, mais que cette chaîne est incluse dans une autre (noeud intermédiaire de l'arbre).

Avec ces conventions, l'arbre précédent devient :



I.2.3 Représentation en mémoire

On peut obtenir une autre représentation de l'arbre précédent en écrivant à la suite l'une de l'autre les portions de chaînes qui figurent les différentes branches de cet arbre, et indiquer les liaisons entre les noeuds de l'arbre :



Pour décrire entièrement cet arbre, il est donc suffisant d'indiquer pour un caractère déterminé, donc pour le noeud qui lui correspond, s'il existe un alternant possible, et de le repérer. On crée ainsi un chaînage entre les alternants possibles. Pour limiter la recherche, il est nécessaire que les caractères correspondant aux noeuds successifs du chaînage soient rangés suivant un ordre

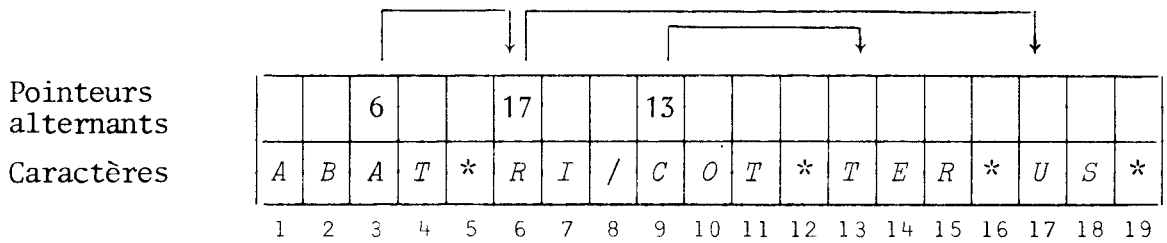
déterminé, alphabétique, par exemple pour les lettres. On a ainsi, le chaînage
A *R* *U* dans l'arbre précédent.

On peut donc représenter cet arbre en mémoire à l'aide de deux tableaux associés

- un tableau des chaînes de caractères

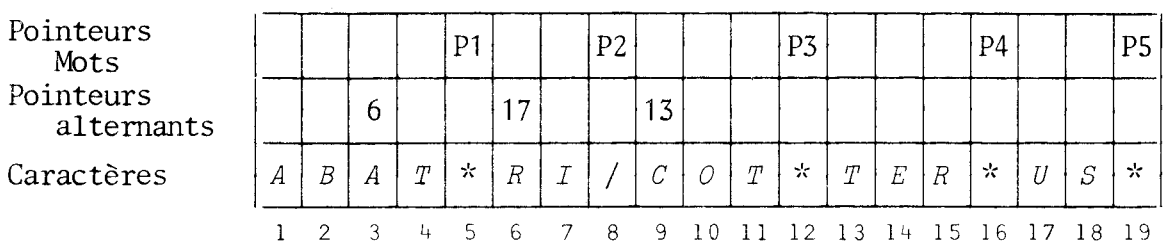
- un tableau des pointeurs d'alternants, contenant l'adresse de l'alternant, s'il existe, pour un caractère détermin

L'arbre précédent devient :



Les deux tableaux précédents représentent uniquement l'arbre. Mais le dictionnaire sert à reconnaître des chaînes de lettres correspondant aux mots du langage courant, pour leur associer les fonctions sémantiques ou syntactiques qui seront utilisées ensuite par le système. On pourra représenter, ou repérer ces fonctions à l'aide de pointeurs de mots, comme la suite de ce chapitre le montrera.

La représentation complète du dictionnaire devrait donc comporter un troisième tableau ; celui des pointeurs de mots : {Pi}. Ceux-ci seraient associés aux caractères de fin de chaîne :



On peut remarquer que le tableau des pointeurs de mots, et celui des pointeurs d'alternants ne contiennent jamais, pour un indice i donné, simultanément des valeurs significatives. En effet, les pointeurs de mots sont associés aux caractères de fin de chaîne, tandis que les pointeurs d'alternants sont associés aux autres caractères.

Il est donc possible de réunir dans un seul tableau tous les pointeurs. Les caractères auxquels ils sont associés permettront de différencier pointeurs de mots et pointeurs d'alternants.

Le représentation du dictionnaire sera donc :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Pointeurs		6			P1	17		P2	13			P3				P4			P5
Caractères	*A*	*B*	*A*	*T*	*	*R*	*I*	/	*C*	Ø	*T*	*	*T*	*E*	*R*	*	*U*	*S*	*

Dans le chapitre précédent, l'étude des stems a montré qu'une chaîne de caractères pouvait être associée à plusieurs fonctions syntactiques ou sémantiques différentes, donc à plusieurs pointeurs de mots différents. Pour représenter ceci, on peut associer à la chaîne autant de caractères de fin de chaîne qu'il y a, pour cette chaîne, de fonctions différentes.

Par exemple, si dans l'arbre précédent,

- le mot *abri* a deux pointeurs associés : P2 et P'2
- le mot *abriter* correspond également à deux pointeurs : P4, P'4

la représentation correspondante sera :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
			6		P1	19		P2	P'2	14			P3				P4	P'4			P5
	A	*B*	*A*	*T*	*	*R*	*I*	/	/	*C*	*O*	*T*	*	*T*	*E*	*R*	*	*	*U*	*S*	*

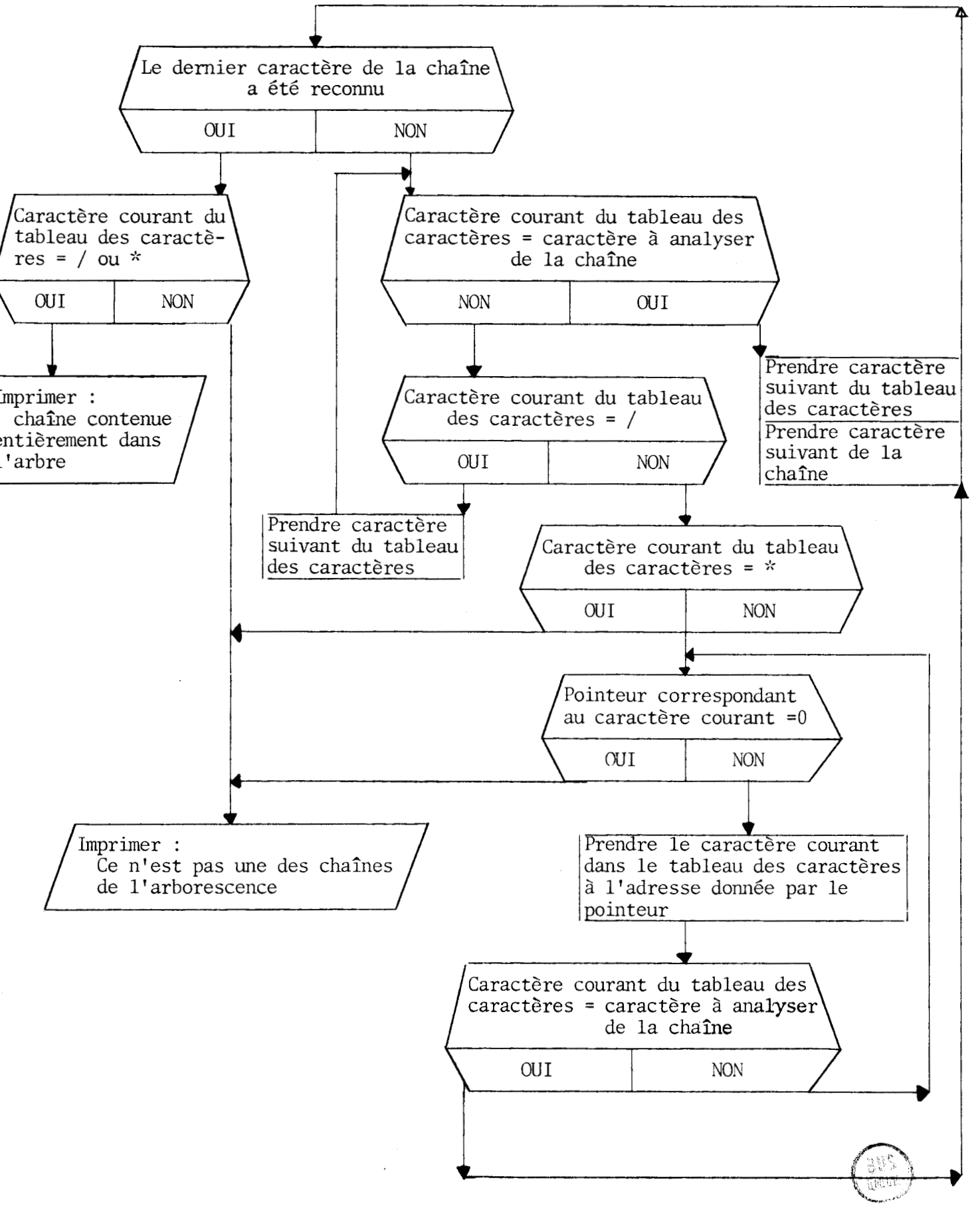
III.2.4 Reconnaissance d'une chaîne de caractères dans l'arborescence

Le dictionnaire sera représenté en mémoire suivant la dernière forme donnée.

Afin de mieux préciser ceci, il est souhaitable de représenter l'organigramme de l'algorithme de reconnaissance d'une chaîne de caractères dans l'arborescence. On a supposé ici que le dernier caractère de la chaîne était déterminé, et donc que la longueur de la chaîne à reconnaître était connue.

Cet algorithme sera réutilisé, et complété dans d'autres algorithmes, en particulier pour la mise-à-jour du dictionnaire, et la recherche d'un mot du langage courant, comme le montrera l'étude du dictionnaire. Il est nécessaire, auparavant, de donner plus de détails sur la notion de pointeur de mot, qui a été introduite dans ce paragraphe.

ORGANIGRAMME : RECONNAISSANCE D'UNE CHAINE DE CARACTERES DANS L'ARBORESCENCE



III.3 POINTEURS DE MOTS. CODAGE DES CONCEPTS

Le dictionnaire va servir à associer aux mots du langage courant les fonctions syntactiques et sémantiques qui lui correspondent, et que le système de documentation utilisera pour rechercher les documents qui répondent à la demande.

Comme le chapitre II l'a montré, la reconnaissance d'un mot est basée sur la caractérisation du stem de mot qui lui correspond. C'est à ce stem que seront associées les fonctions syntactiques et sémantiques du mot ; fonctions repérées, ou représentées par les pointeurs de mots cités au paragraphe précédent. Afin de réutiliser la même procédure de recherche, le dictionnaire va contenir à la fois des chaînes de stems et des chaînes de suffixes. Il faudra donc différencier ces chaînes, à l'aide d'un codage sémantique.

Donc, en ce qui concerne les chaînes de stems, en plus des codes syntactiqu utilisés pour reconnaître la fonction grammaticale du mot qui correspond à ce stem, et la classe de suffixes qui lui est associée, il est nécessaire d'adjoindre aux codes syntactiques des codes sémantiques précisant l'utilisation du mot associé au stem par le système documentaire.

Le code sémantique pourra se décomposer en :

- code utilisation documentaire : indiquant si la chaîne est associée :
  * à un suffixe
  * au stem d'un mot, et si, de plus ce mot est :
    . un mot-pivot
    . un mot-vide
    . un mot grammatical, dont la reconnaissance permettra d'affiner l'analyse syntactique de l'ensemble des mots de la demande.

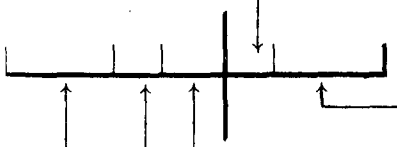
- code de concept : associant un mot-pivot à un concept déterminé, ou permettant de différencier un mot-outil parmi l'ensemble des mots grammaticaux.

- un code indiquant si le concept associé au mot-pivot est un concept élémentaire ou s'il fait partie d'un ensemble de concepts élémentaires

dont l'intersection est associée au mot : code de concept non-élémentaire. Par exemple : au "mot" : *S.N.C.F.* pourront être associés trois codes sémantiques correspondant respectivement aux concepts de *chemin*, *fer*, et *français*.

L'ensemble des codes contenus dans un pointeur de mots peut donc se résumer :

* Partie syntaxique :



- code de la fonction grammaticale
- numéro de séquence de la classe de suffixes dans le sous-ensemble des classes de suffixes correspondant à la fonction grammaticale.

* Partie sémantique :

- code utilisation documentaire
- code de concept
- code de concept non-élémentaire.

Ces codes vont être utilisés lors de la création, la mise-à-jour, et l'exploitation du dictionnaire.

C H A P I T R E IV

A L G O R I T H M E S du D I C T I O N N A I R E

-1- C H A I N A G E

V.1 INTRODUCTION

Le dictionnaire sera représenté à l'aide de deux tableaux associés : un tableau des caractères et un tableau des pointeurs, suivant le principe qui a été exposé précédemment.

Ce chapitre va réunir les descriptions des algorithmes de base pour la création, la mise-à-jour, et l'exploitation du dictionnaire. L'annexe IV complète la description de ce dictionnaire.

IV.2 INSERTION D'UNE CHAINE DANS L'ARBORESCENCE DU DICTIONNAIRE : "INSERTSTEM"

Soit une arborescence (A) représentée par les tableaux associés : tableau des caractères (TC) et tableau des pointeurs (TP).

Si on ajoute à cette arborescence une chaîne de caractères, on obtient une nouvelle arborescence (A'), représentée par les tableaux TC' et TP'.

L'insertion d'une chaîne dans l'arbre se ramène donc, pour les tableaux associés, à la relation (TC,TP) → (TC',TP'). Cette modification des tableaux diffère suivant la configuration relative de la chaîne à introduire par rapport aux chaînes déjà existantes.

IV.2.1 Présentation des différents cas d'insertion possibles

On peut utiliser, pour simplifier l'exposé, les notations suivantes :

x = chaîne à introduire

w = plus grande chaîne de l'arbre contenue dans x

p = plus longue partie de x commune avec d'autres chaînes de l'arbre

z = chaîne de l'arborescence qui a la plus longue partie commune avec x

α = premier caractère de x non commun à x et z

β = premier caractère de z non commun à x et z

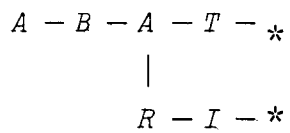
c = chaînage d'alternants de l'arbre, qui contient β.

Les modifications de l'arborescence dans les exemples donnés seront soulignées.

Les différents cas possibles sont les suivants :

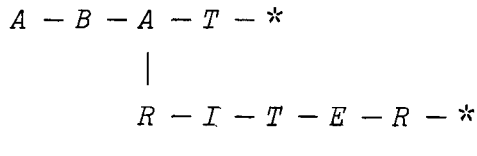
a) La chaîne à introduire figure déjà dans le dictionnaire.

Exemple : on veut introduire la chaîne : *abri*.

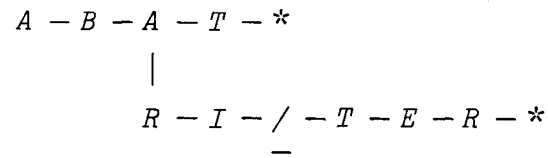


b) La chaîne à introduire est contenue dans une chaîne déjà existante de l'arbre.

Exemple :

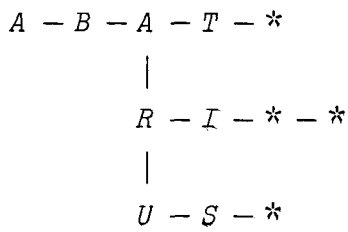


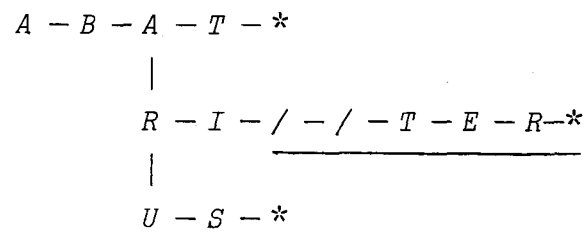
on veut ajouter la chaîne : *abri*



c) Pour la chaîne à introduire, on a p=w.

Exemple :



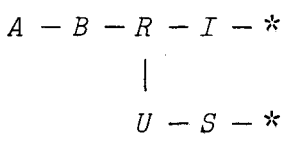


La plus longue partie commune de la chaîne à introduire avec les autres chaînes de l'arbre était, dans l'exemple, la chaîne : *abri*.

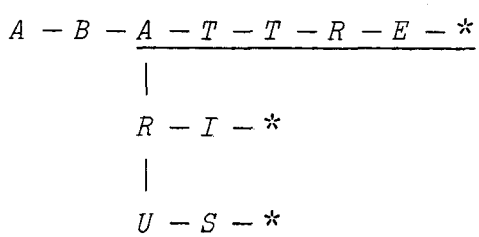
Lorsque p est plus longue que w, on va introduire le premier caractère non commun de x dans un chaînage. Trois cas sont possibles :

d) La chaîne à introduire, x , précède dans l'ordre alphabétique la chaîne z.

Par exemple :

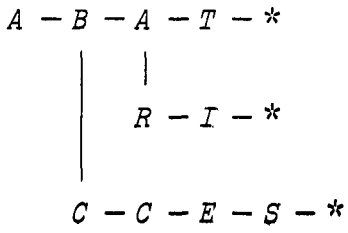


si on ajoute la chaîne : *abattre*

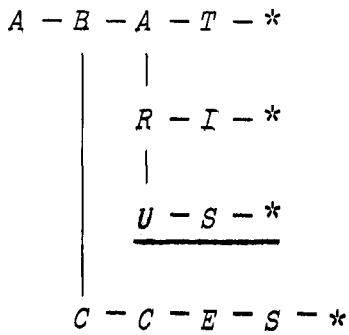


e) Le caractère α vient, dans l'ordre alphabétique, après le dernier caractère du chaînage c.

Par exemple :

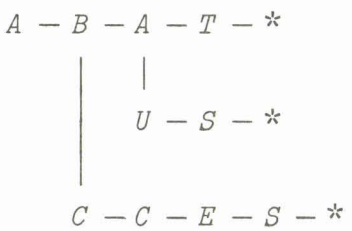


on ajoute la chaîne : *abus*

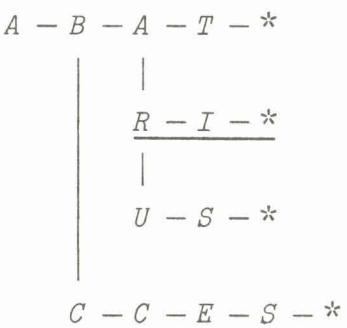


f) Le caractère α se place, suivant l'ordre alphabétique, entre deux caractères du chaînage c.

Par exemple :



on ajoute la chaîne : *abri*



Ces différents cas d'insertion sont associés à un libellé qui figurera dans l'organigramme, et dans les programmes qui correspondent. Ce libellé sera donné lors de l'exposé des différents cas d'insertion.

IV.2.2 Différents cas d'insertion d'une chaîne de caractères

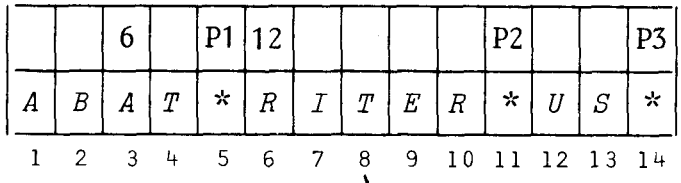
a) La chaîne à introduire figure déjà dans le dictionnaire.

Il faudra alors vérifier si les codes de la chaîne à introduire ne figurent pas dans le dictionnaire. Il s'agira alors de nouveaux codes à introduire pour cette chaîne. Si ces codes figurent déjà dans le dictionnaire, il peut s'agir d'une modification de ces codes (on résoud ainsi les cas de synonymes : on peut modifier les codes d'une chaîne pour la rendre synonyme d'une autre, en modifiar la partie sémantique des codes). Sinon, il s'agit d'une erreur :on a voulu introduire une chaîne et des codes associés qui figurent déjà dans le dictionnaire, et il s'agit donc d'une erreur.

b) La chaîne à introduire est contenue dans une chaîne déjà existante dans l'arbre. "Cas 1" de l'organigramme.

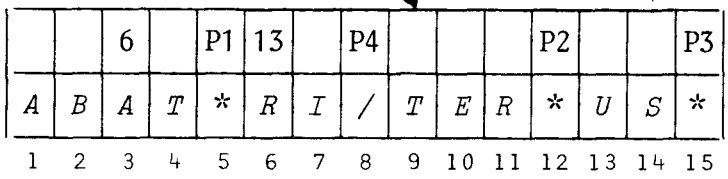
Par exemple :

Soit la mise en arbre des chaînes : *abat, abriter, abus :*



		6		P1	12					P2			P3
A	*B*	*A*	*T*	*	*R*	*I*	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Si on ajoute la chaîne : *abri :*



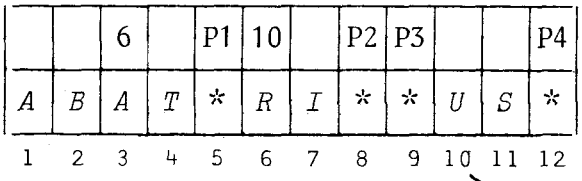
		6		P1	13		P4				P2			P3
A	*B*	*A*	*T*	*	*R*	*I*	/	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

On a donc inséré dans le tableau des caractères un caractère de fin de chaîne : /, et décalé la partie du tableau des pointeurs correspondante. Les pointeurs supérieurs ou égaux à l'adresse du nouveau caractère de fin de chaîne ont été augmentés de 1. ex : $\boxed{\overline{12}}\ \boxed{R}$ → $\boxed{\overline{13}}\ \boxed{R}$.

c) La partie de la chaîne à introduire figurant déjà dans l'arborescence est une des chaînes de cette arborescence. "Cas 2" de l'organigramme.

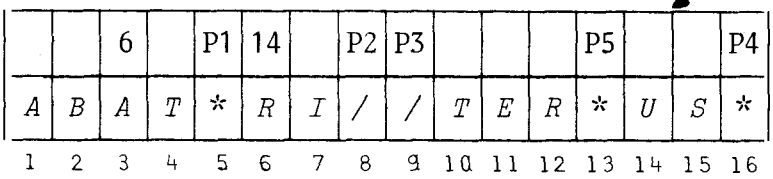
Exemple :

Soit l'arborescence associée à : *abat, abri, abus,* et on suppose que *abri* a deux pointeurs de mot :



		6		P1	10		P2	P3			P4
A	*B*	*A*	*T*	*	*R*	*I*	*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12

On ajoute le mot : *abriter.*



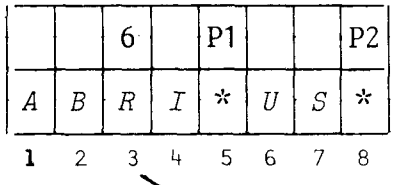
		6		P1	14		P2	P3				P5			P4
A	*B*	*A*	*T*	*	*R*	*I*	/	/	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

On change donc le (ou les) caractère de fin de chaîne : * en /, et on insère, à la suite, le reste de la chaîne avec le caractère de fin de chaîne : *. On translate la partie du tableau des pointeurs associés et on remet en ordre les pointeurs en ajoutant à ceux qui sont supérieurs ou égaux à l'adresse initiale d'insertion la longueur de la partie de la chaîne non commune, +1 (pour le caractère de fin de chaîne).

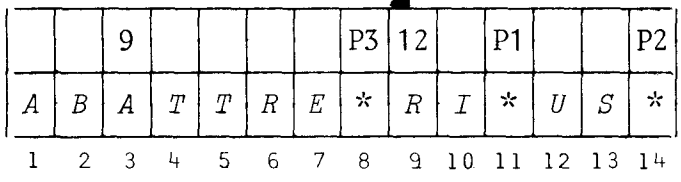
d) La chaîne à introduire précède dans l'ordre alphabétique la chaîne existante qui a la plus longue partie commune avec la chaîne à introduire.
"Cas Lchain < Lmot"

Exemple :

Soit l'arbre associé à *abri, abus.*



On ajoute *abattre :*



On translate le tableau des caractères, et les pointeurs associés à partir du premier caractère non commun, de façon à insérer le reste de la chaîne à introduire et le caractère. On remet en ordre les pointeurs comme précédemment. Ex : $\left|\frac{\overline{6}}{\underline{R}}\right| \rightarrow \left|\frac{\overline{12}}{\underline{R}}\right|$ la translation est égale à 6.

On complète le chaînage en créant, pour le premier caractère inséré un pointeur qui renvoit au caractère qui suit le groupe d'* de la fin de la chaîne introduite.

e) Le caractère α de la chaîne à introduire suit, dans l'ordre alphabétique le dernier caractère du chaînage c. "Cas 3.1".

Ex : Soit l'arbre associé à : *abat, abri, accès :*

	9	6		P1			P2					P3
A	*B*	*A*	*T*	*	*R*	*I*	*	*C*	*C*	*E*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13

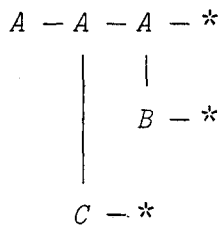
On ajoute : *abus :*

	12	6		P1	9		P2			P4					P3
A	*B*	*A*	*T*	*	*R*	*I*	*	*U*	*S*	*	*C*	*C*	*E*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

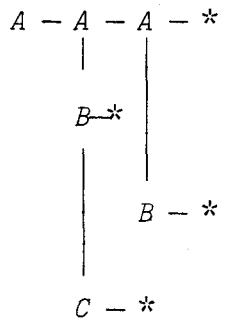
Dès qu'on a répété le premier caractère non commun avec la chaîne, le premier caractère à translater sera situé à la suite du premier groupe d'* rencontré en suivant le chaînage. On insère ensuite la partie non commune de la chaîne, et on remet en ordre les pointeurs.

Remarque : On ne crée pas un arbre régulier en insérant la chaîne après le premier groupe d'*. Cela revient à insérer une branche nouvelle immédiatement après celle qu'on vient d'explorer.

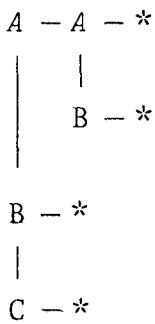
Par exemple, soit l'arbre :



Si on ajoute la chaîne *AB*, l'arbre obtenu sera :



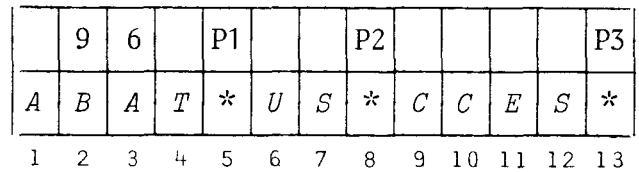
l'arbre régulier serait :



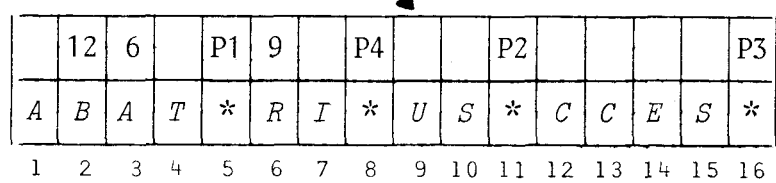
Ceci ne modifie pas la procédure de recherche car on suit toujours un chaînage où les alternants sont rangés suivant un ordre déterminé, en particulier, le temps de recherche est strictement le même que pour un arbre régulier. Par contre, la procédure d'insertion est plus rapide, car on n'a pas à rechercher la branche qui correspondrait à celle de l'arbre régulier.

f) Le caractère α se place, suivant l'ordre alphabétique, entre deux caractères du chaînage d'alternants c. "Cas 3.2".

Exemple : soit l'arbre associé à *abat, abus, accès.*



On ajoute *abris :*

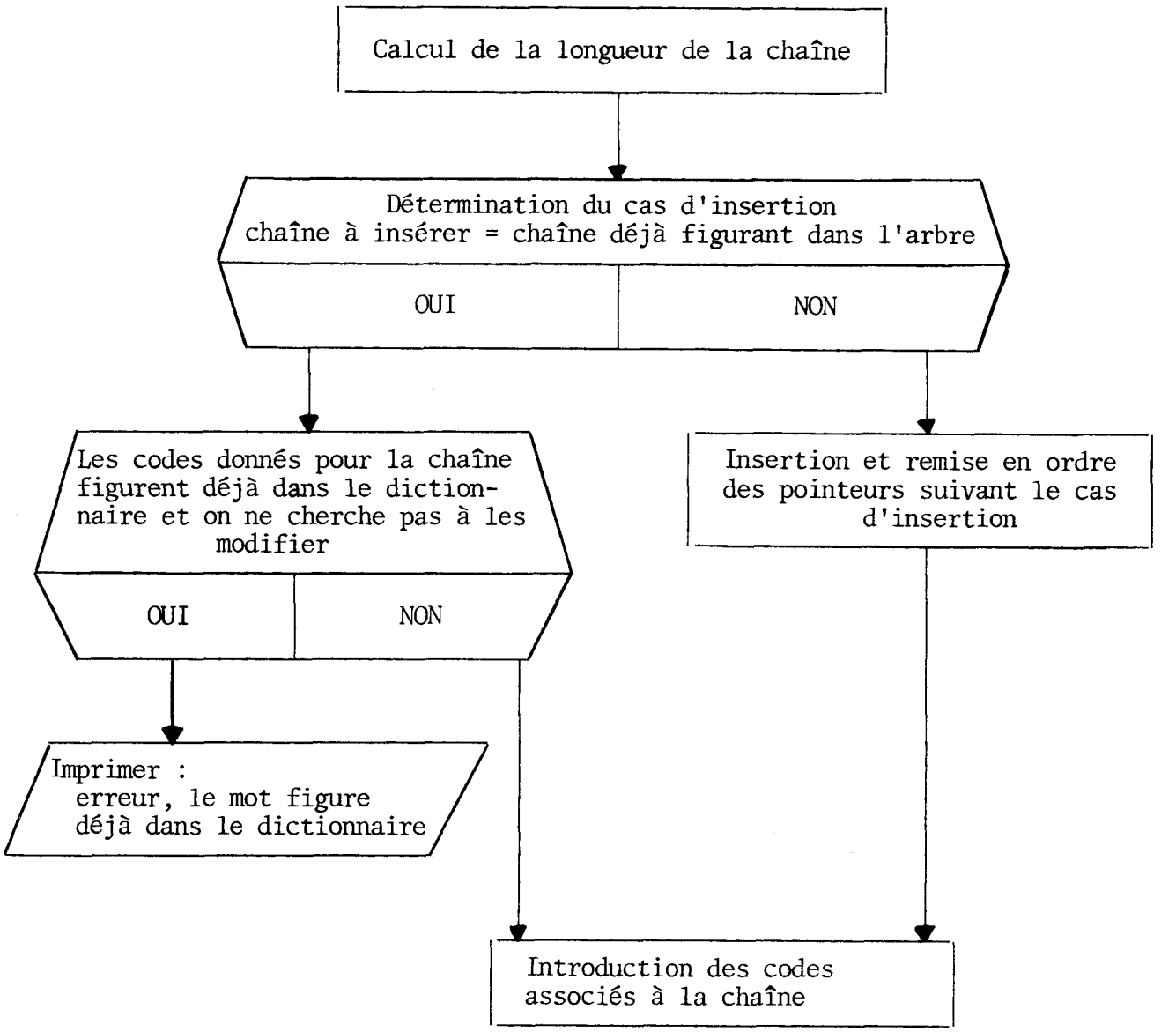


Le premier caractère à translater est le caractère du chaînage qui suit dans l'ordre alphabétique le premier caractère non commun rencontré, lors de la recherche de la chaîne dans le chaînage. On insère le reste de la chaîne et le caractère * et on remet en ordre les pointeurs.

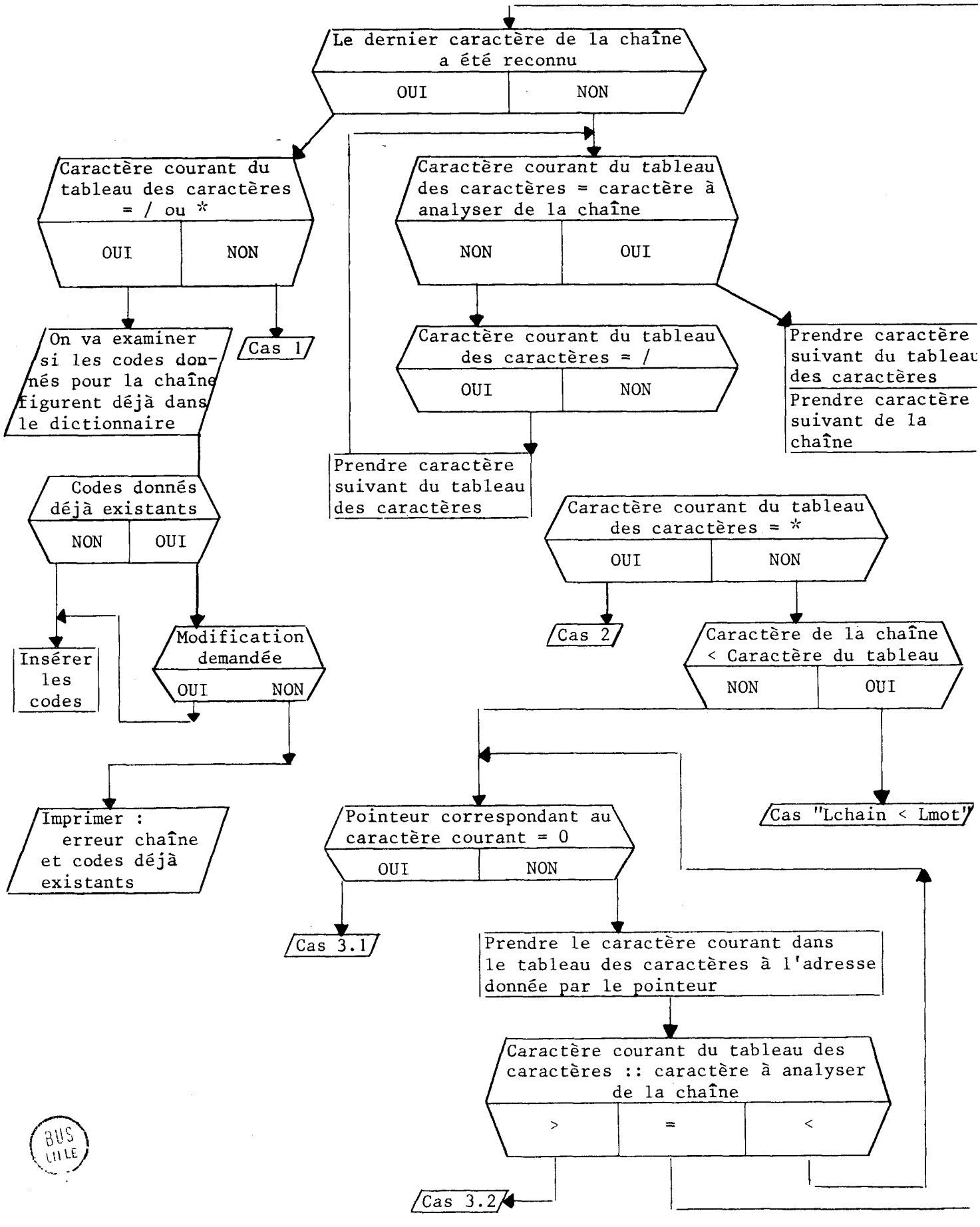
IV.2.3 Organigramme de l'insertion d'une chaîne

L'organigramme général de l'insertion d'une chaîne sera ici esquissé. Un organigramme partiel, plus détaillé explicitera la détermination des cas d'insertion, afin de montrer que c'est une extension de l'organigramme d'une chaîne de caractères dans l'arborescence.

ORGANIGRAMME GENERAL DE L'INSERTION D'UNE CHAINE



ORGANIGRAMME PARTIEL D'INSERTION D'UNE CHAINE : DETERMINATION DES CAS D'INSERTION



IV.3 SUPPRESSION D'UNE CHAINE DANS L'ARBORESCENCE : "CONCACSTEM"

L'insertion d'une chaîne se ramenait à la relation : (TC,TP) → (TC',TP'). La suppression d'une chaîne est la relation inverse : (TC',TP') → (TC,TP). L'algorithme de suppression d'une chaîne dans l'arborescence sera utilisé pour retirer du chaînage une chaîne erronée, introduite par erreur, sans avoir à recréer tout le chaînage.

Il existe différents cas de suppression d'une chaîne, suivant la configuration relative de la chaîne à retirer, par rapport aux chaînes qui figurent dans le dictionnaire. Ces cas sont les cas inverses de ceux de l'insertion. Ils ont reçu un libellé qui figurera sur l'organigramme, comme pour l'algorithme précédent.

On peut utiliser, pour simplifier l'exposé, les notations suivantes.

x = chaîne à retirer.

α = caractère du tableau des caractères dont le pointeur associé pointe vers le caractère qui suit le dernier caractère de fin de chaîne associé à x.

β = dernier caractère de x inclus dans un chaînage d'alternant.

y = plus grande chaîne de l'arborescence contenue dans la chaîne à supprimer.

Les parties de l'arborescence à supprimer seront soulignées dans les exemples.

V.3.1 Présentation des différents cas de suppression possibles

Différents cas sont possibles :

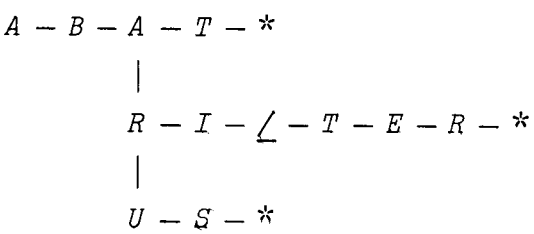
On a vu que dans le dictionnaire, à une même chaîne de caractères, on pouvait associer un, ou plusieurs des ensembles de codes, dont la structure a été décrite à la fin du chapitre III.

Dans le cas où plusieurs ensembles de codes sont associés à une même chaîne de caractères, la suppression d'un des ensembles de codes se ramène à celle d'un des caractères de fin de chaîne et du pointeur de mot correspondant, et à la remise en ordre des pointeurs. Ce cas se traite de la même façon que celui d'une chaîne, associée à un seul ensemble de codes, mais contenue dans une autre chaîne de l'arborescence ; comme précédemment, on supprimera le caractère : / de fin de chaîne et le pointeur de mot qui lui correspond.

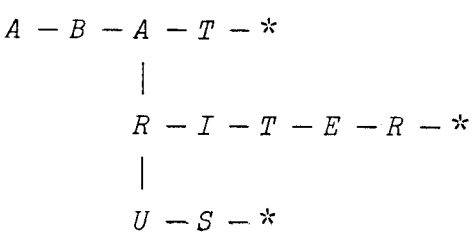
Les autres cas à envisager concernent donc des chaînes associées à un seul ensemble de codes :

a) La chaîne à retirer est contenue dans une chaîne déjà existante de l'arbre. Le caractère de fin de chaîne est : /.

Par exemple :

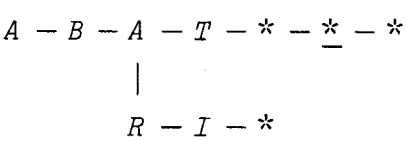


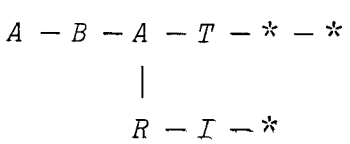
on retire la chaîne : *abri*



On a vu que le cas de plusieurs ensembles de codes associés à une même chaîne se traitait de la même façon.

Exemple :



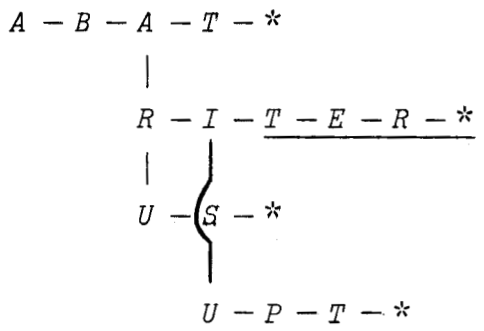


Pour l'ensemble des cas qui suivent, les chaînes ne sont pas contenues dans une autre chaîne de l'arbre. Le caractère de fin de chaîne est donc : *.

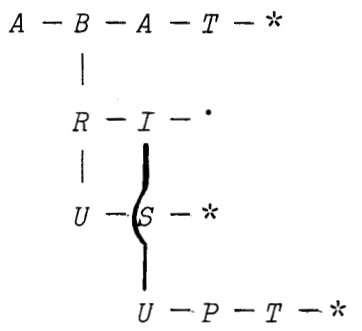
b) Dans la chaîne x, le caractère β suit le caractère α, et β n'appartient pas à la chaîne y.

Il s'agit d'un arbre non régulier. On ne peut enlever, dans la chaîne que l'on veut supprimer, que les caractères qui ne sont pas chaînés.

Exemple :

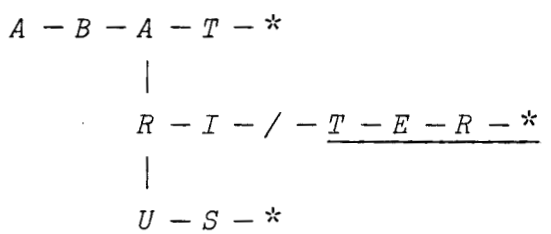


On supprime la chaîne : *abriter*

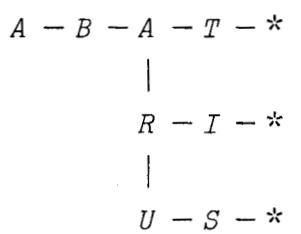


c) Le caractère β appartient à la chaîne y, contenue dans la chaîne à supprimer. Les caractères non communs à y et x n'appartiennent donc pas à des chaînages d'alternants.

Exemple :

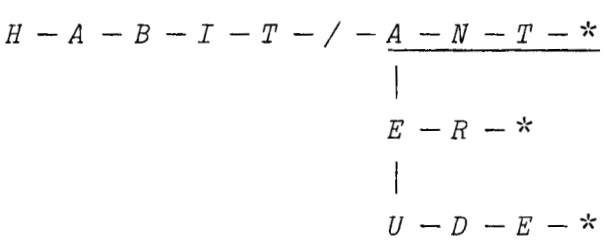


On supprime la chaîne : *abriter*

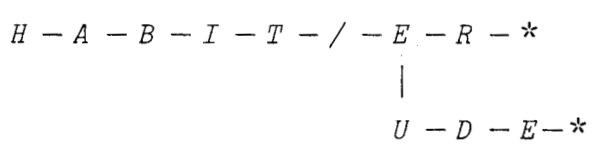


d) Le caractère α n'appartient pas à la chaîne y, et α ≡ β.

Exemple :



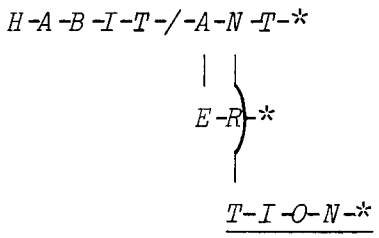
On retire la chaîne : *habitant*



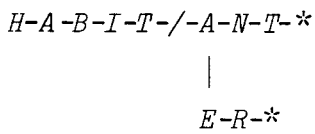
e) La chaîne à supprimer, x ne possède pas de caractère α. Le caractère β n'appartient pas à la chaîne y.

Il s'agit d'un arbre non régulier.

Exemple :



On retire la chaîne : *habitation*



Reprenons chacun de ces cas.

IV.3.2 Différents cas de suppression d'une chaîne

A chaque cas correspond un libellé qui figurera dans l'organigramme, et dans le programme qui lui correspond.

a) La chaîne à retirer est contenue dans une chaîne déjà existante de l'arbre "Cas 1" de l'organigramme.

Exemple : soit l'arborescence associée à : *abat, abri, abriter, abus.*

		6		P1	13		P2				P3			P4
A	*B*	*A*	*T*	*	*R*	*I*	/	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

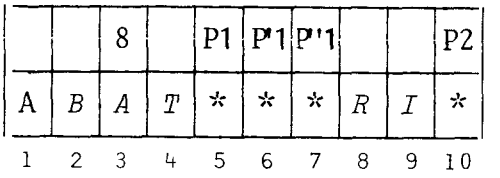
on retire la chaîne : *abri*

		6		P1	12					P3			P4
A	*B*	*A*	*T*	*	*R*	*I*	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cas particulier :

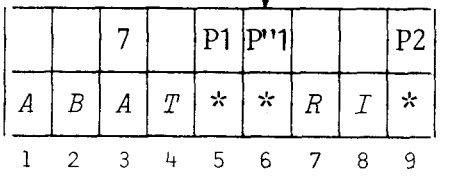
On a vu que si plusieurs codes {Pi / i = 1,n} sont associés à une chaîne, la suppression de la chaîne correspondant au code Pj | $1 \leq j \leq n$ se ramène à la suppression du caractère de fin de chaîne, et du pointeur Pj qui lui est associé C'est donc le même cas de suppression.

Exemple : soit l'arborescence associée à : *abat, abri.* Plusieurs pointeurs correspondent à la chaîne *abat :* P1, P'1, P''1.



		8		P1	P'1	P'1			P2
A	*B*	*A*	*T*	*	*	*	*R*	*I*	*
1	2	3	4	5	6	7	8	9	10

La suppression de la chaîne *abat* associée à P'1 donnera :

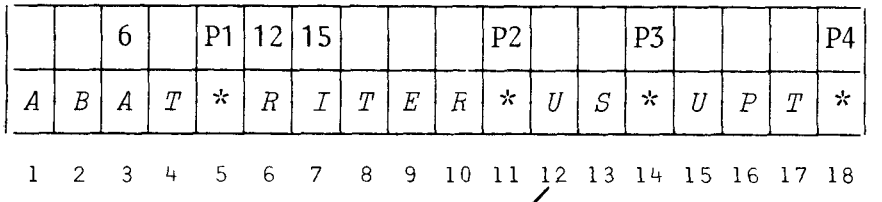


		7		P1	P''1			P2
A	*B*	*A*	*T*	*	*	*R*	*I*	*
1	2	3	4	5	6	7	8	9

On a donc retiré dans le tableau des caractères un caractère de fin de chaîne. Il y a eu décalage d'une partie du tableau des caractères, et des pointeurs associés, et remise en ordre des pointeurs.

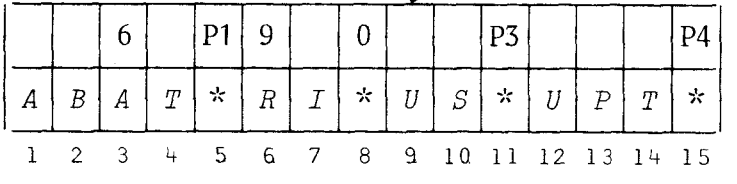
b) Le caractère β n'appartient pas à la chaîne y, et précède, dans la chaîne x le caractère α. "Cas 2.1.1" de l'organigramme.

Exemple : soit l'arbre associé à : *abat, abriter, abus, abrupt*



		6		P1	12	15				P2			P3				P4
A	*B*	*A*	*T*	*	*R*	*I*	*T*	*E*	*R*	*	*U*	*S*	*	*U*	*P*	*T*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Si on retire la chaîne : *abriter.*



		6		P1	9		0			P3				P4
A	*B*	*A*	*T*	*	*R*	*I*	*	*U*	*S*	*	*U*	*P*	*T*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Le caractère *I* de *abriter* est chaîné ; tandis que le caractère *R* à l'adresse 6 est chaîné avec le caractère *U* à l'adresse 12.

Il s'agit d'un cas où le chaînage ne représente pas un arbre régulier, et où on ne peut enlever dans la chaîne que l'on veut supprimer que les

caractères qui ne sont pas chaînés.

Dans ce cas, l'algorithme indiquera que l'arbre n'est pas régulier.

c) Le caractère β appartient à la chaîne y. Ce cas correspond à deux libellés dans l'organigramme : "Cas 2.1.2 ; 2" et "Cas 2.2.1".

Exemple : soit l'arborescence constituée par les chaînes : *abat*, *abri*, *abriter*, *abus*.

		6		P1	13		P2				P3			P4
A	*B*	*A*	*T*	*	*R*	*I*	/	*T*	*E*	*R*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

on retire la chaîne : *abriter*

		6		P1	9		P2			P4
A	*B*	*A*	*T*	*	*R*	*I*	*	*U*	*S*	*
1	2	3	4	5	6	7	8	9	10	11

Le caractère de fin de chaîne de *abri* : / est modifié, et devient égal au caractère de fin de chaîne de *abriter* : *. Il y a concaténation des deux tableaux, et remise en ordre des pointeurs.

d) Le caractère $\alpha \equiv \beta$ n'appartient pas à la chaîne y. "Cas 2.1;1".

Exemple : soit l'arbre associé aux chaînes : *habit*, *habitant*, *habiter*, *habitude*.

					P1	11			P2	14		P3				P4
H	*A*	*B*	*I*	*T*	/	*A*	*N*	*T*	*	*E*	*R*	*	*U*	*D*	*E*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

on retire la chaîne : *habitant*.

					P1	10		P3				P4
H	*A*	*B*	*I*	*T*	/	*E*	*R*	*	*U*	*D*	*E*	*
1	2	3	4	5	6	7	8	9	10	11	12	13

e) Le caractère β n'appartient pas à la chaîne y. La chaîne x ne possède pas de caractère α. "Cas 2.2.2".

Exemple : soit l'arbre constitué par : *habit, habitant, habitation, habiter :*

					P1	11	14		P2			P3					P4
H	*A*	*B*	*I*	*T*	/	*A*	*N*	*T*	*	*E*	*R*	*	*T*	*I*	*O*	*N*	*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

on retire la chaîne : *habitation*

					P1	11			P2			P3
H	*A*	*B*	*I*	*T*	/	*A*	*N*	*T*	*	*E*	*R*	*
1	2	3	4	5	6	7	8	9	10	11	12	13

Remarque : il s'agit encore ici d'un cas où l'arbre représenté n'est pas régulier.

.3.3 Organigramme de suppression d'une chaîne

Le principe de la suppression d'une chaîne dans l'arborescence est le suivant :

- On recherche dans l'arborescence la chaîne à supprimer. Pour chacun des caractères de la chaîne trouvé dans le chaînage, on mémorise son adresse dans le tableau des caractères, et le pointeur qui lui est associé. On mémorise, de plus, l'adresse dans le tableau des caractères du dernier caractère de fin de mot : / trouvé dans la recherche de la chaîne.

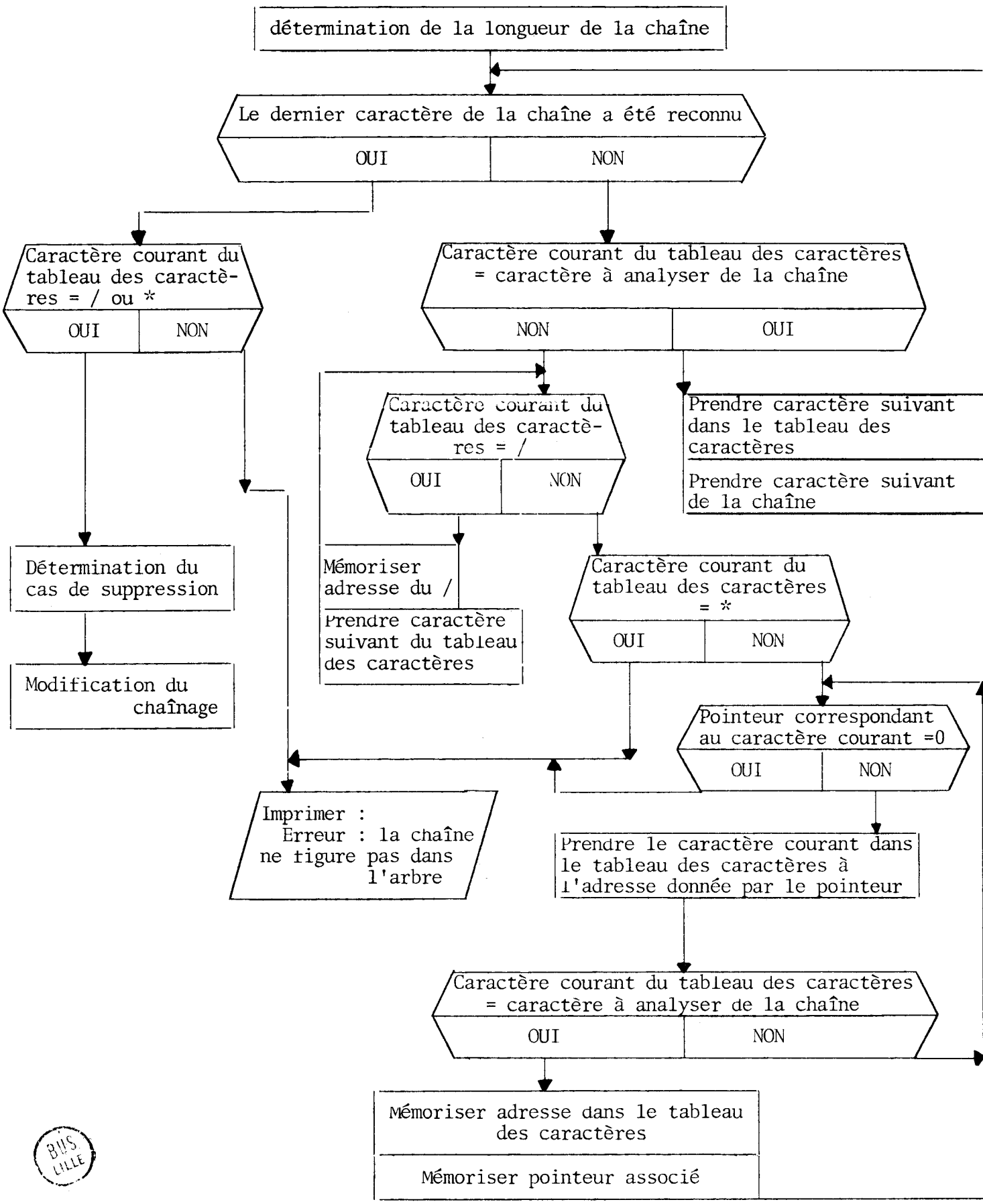
- Si la chaîne existe dans l'arbre, les renseignements mémorisés précédemment permettent de déterminer le cas de suppression.

- Suivant le cas déterminé, on modifie les tableaux des caractères et des pointeurs.

L'organigramme général résume ces trois points, en explicitant la recherche de la chaîne à supprimer.

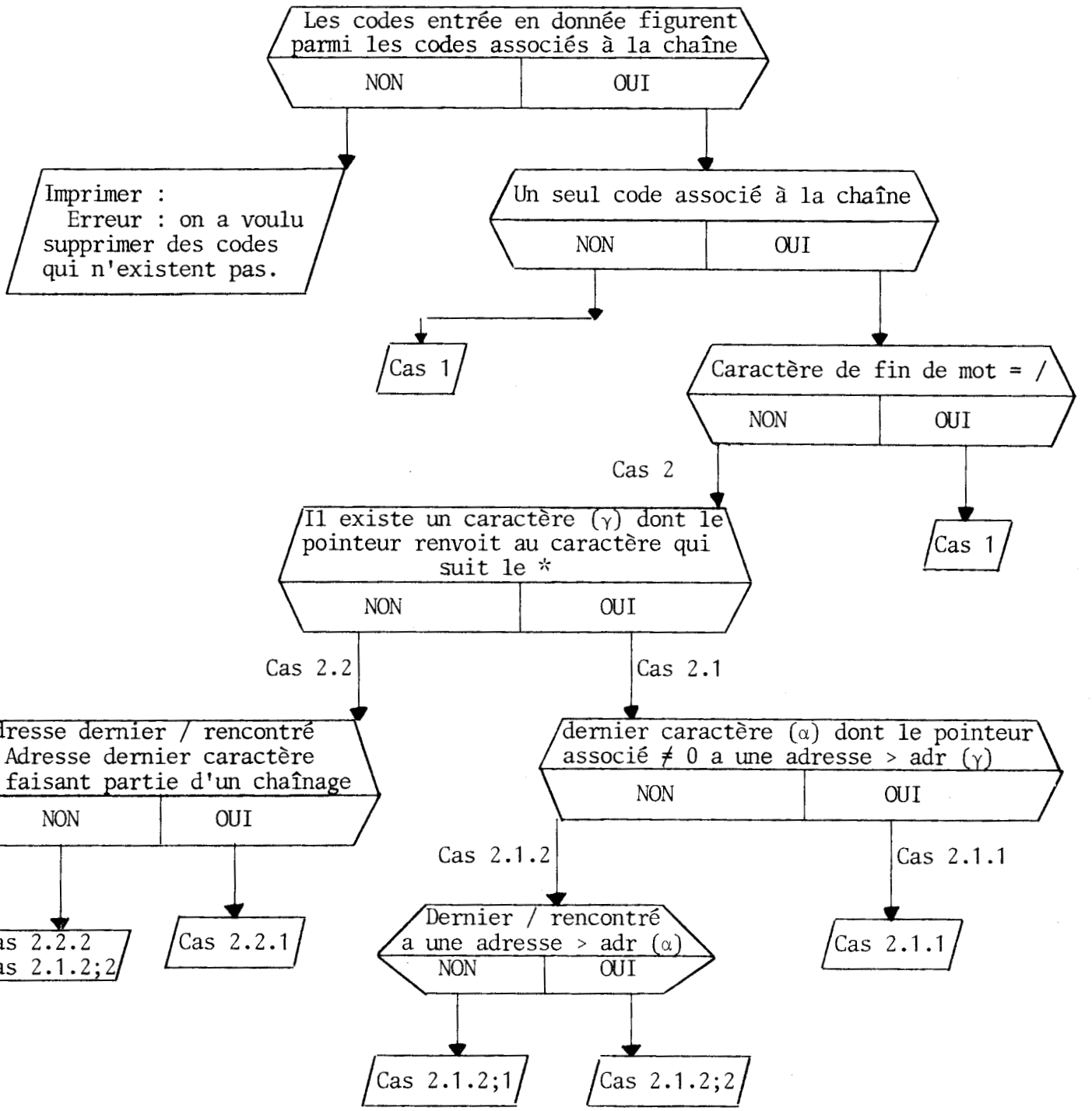
Un organigramme partiel montre la détermination du cas d'insertion.

ORGANIGRAMME GENERAL DE SUPPRESSION D'UNE CHAINE



ORGANIGRAMME PARTIEL DE SUPPRESSION D'UNE CHAINE :

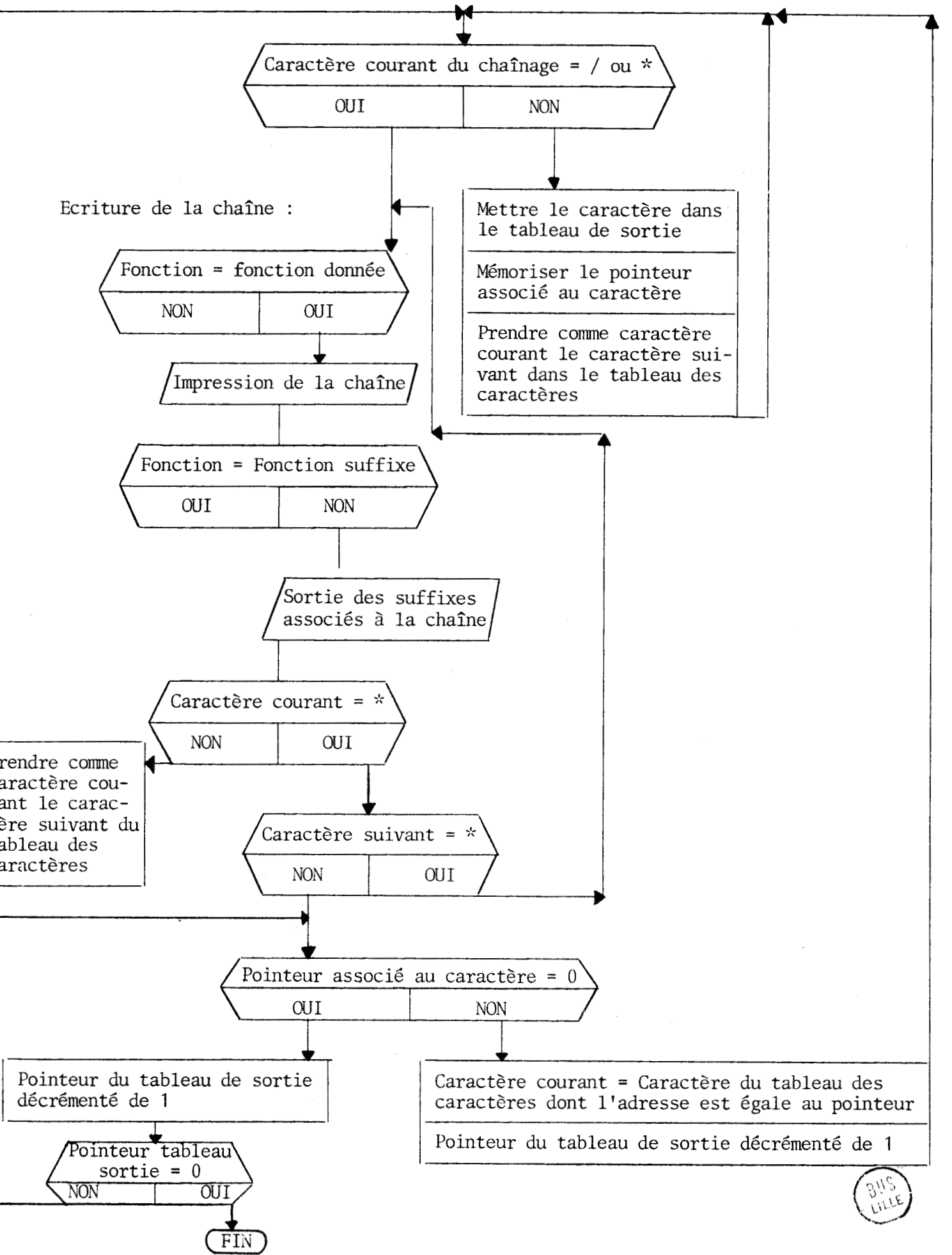
DETERMINATION DES CAS DE SUPPRESSION POUR UNE CHAINE EXISTANTE



IV.4 SORTIE DES CHAINES DU DICTIONNAIRE : "SORTSTEM - SORTMOT"

Cet algorithme permet de sortir toutes les chaînes contenues dans le dictionnaire qui ont une fonction sémantique donnée : mot-pivot, mot-vide, mot grammatical, ou suffixe, et de sortir pour un stem de mot les suffixes qui lui sont associés.

ORGANIGRAMME GENERAL SORTIE DES CHAINES



IV.5 PLURIEL DES MOTS COMPOSES

Les mots composés sont constitués par la juxtaposition de plusieurs mots, reliés par des traits d'union, ou séparés par des espaces.

Exemple : *borne-fontaine, timbre-poste, porte-crayon, porte à faux.*

Un mot composé peut toujours s'écrire en reliant les différents mots par des traits d'union. Par exemple : l'orthographe *"porte-à-faux"* est également acceptée. Cette convention d'écriture sera retenue pour les mots composés figurant dans le dictionnaire, ou dans les phrases à analyser.

Les mots composés suivent des règles différentes pour la formation du pluriel :

- certains sont invariables. Ex : *des porte-crayon.*
- d'autres prennent la marque du pluriel
  - . soit pour tous les mots : ex : *des bornes-fontaines*
  - . soit pour certains composants seulement, ex : *des timbres-poste.*

Pour déterminer le stem d'un mot composé, on effectue la troncature du suffixe sur le dernier mot composant seulement. Exemple : *porte-cray* $\xrightarrow{[NOM60]}$ *on* Si les mots qui précèdent le dernier mot peuvent prendre la marque du pluriel, ils sont stockés sous la forme singulier, ou sous la forme de stem si le pluriel ne résulte pas de l'adjonction de caractères au singulier. Par exemple : *borne-font* $\xrightarrow{[NOM10]}$ *aine ; chev-vap* $\xrightarrow{[NOM81]}$ *eur* pour *cheval-vapeur.*

L'algorithme de reconnaissance de la chaîne d'un mot composé est une extension de la méthode de reconnaissance d'une chaîne de caractères dans l'arborescence, vue précédemment.

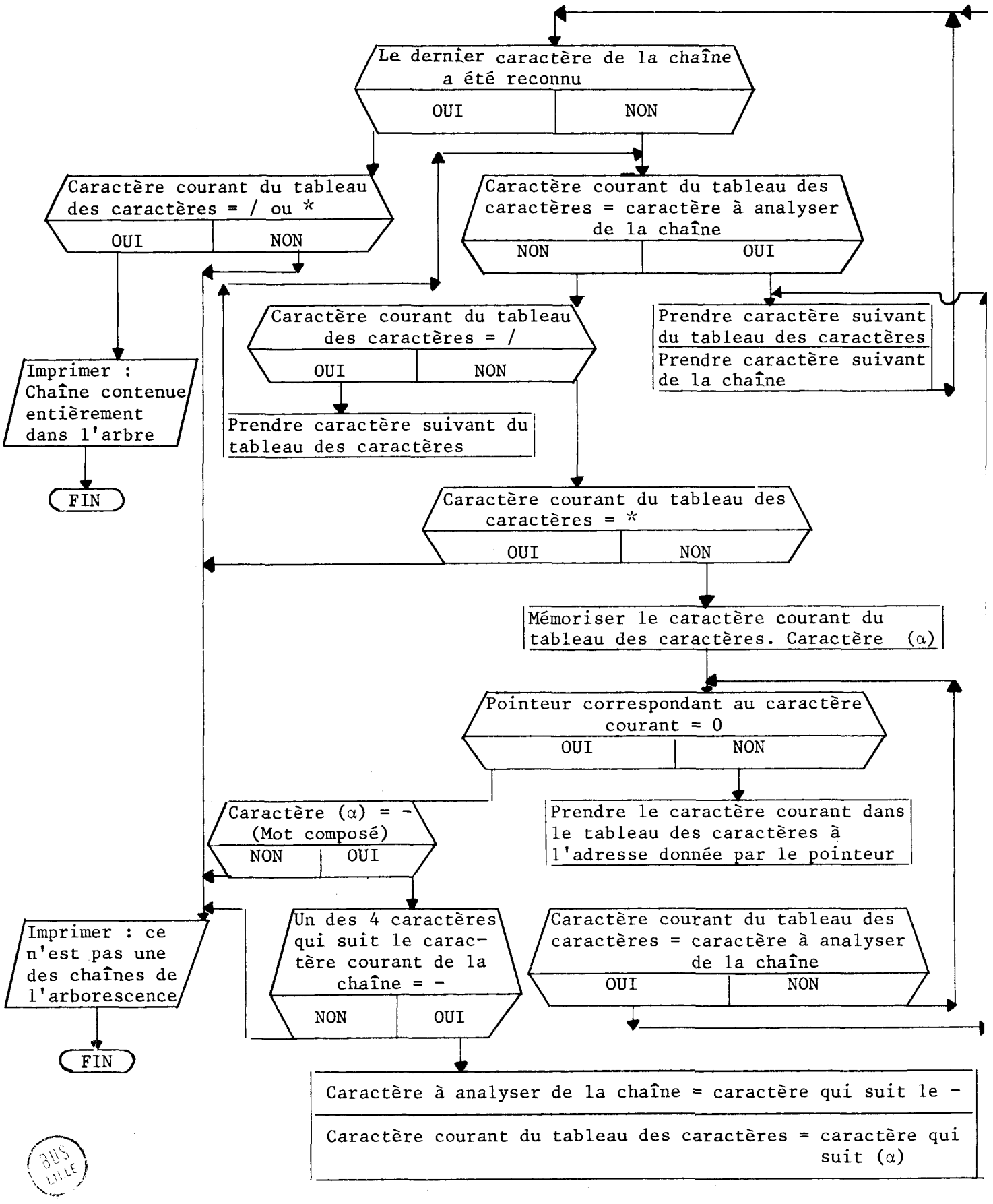
Cette méthode est basée sur la représentation sous forme singulier, ou de stem des différents composants d'un mot composé.

Soit, par exemple, le mot composé : *"cheval-vapeur"*. Soit y la chaîne qui lui correspond dans le dictionnaire ; y sera la chaîne : *"chev-vap"*.

Si on recherche le mot *cheval-vapeur* dans le dictionnaire, on va retrouver les 4 premiers caractères : *chev* de y. La recherche du caractère suivant du mot : *"a"* va montrer que le caractère ne correspond pas dans la chaîne y. Mais, dans l'ordre des codes EBCDIC, le caractère "-" est situé avant les caractères lettres ou chiffres. Dans la recherche du caractère "*a*", il suffit de noter que le premier caractère du chaînage d'alternant est un "-". Comme on n'a pas retrouvé le caractère "*a*" dans le dictionnaire, et qu'on a noté la présence du "-" correspondant à la chaîne y, on déterminera si le mot recherché est un mot composé en analysant les 5 caractères qui suivent le "*a*". Si dans ces caractères, il existe un "-", on reprendra la recherche du mot à partir de ce caractère. On aura ainsi ignoré les caractères du mot composant situés entre le premier caractère non trouvé et le "-". Ainsi, pour : *cheval-vapeur*, l'analyse portera sur les caractères : *chev*, puis *-vapeur*.

L'organigramme correspondant va résumer le principe de cette méthode.

ORGANIGRAMME : RECONNAISSANCE D'UNE CHAINE DE CARACTERES APPARTENANT A UN MOT SIMPLE OU A UN MOT COMPOSE, DE LONGUEUR DONNEE



IV.6 RECHERCHE D'UN MOT : "RECHMOT". RECHERCHE D'UNE PHRASE : "RECHPHRASE"

Ces deux algorithmes seront traités simultanément, car "Rechphrase" est une extension de "Rechmot".

IV.6.1 Recherche d'un mot : "Rechmot"

La reconnaissance d'un mot dans le dictionnaire est basée sur la concordance entre stem et suffixe, par l'intermédiaire des codes syntactiques de la partie stem et de la partie suffixe du mot, trouvés dans le dictionnaire.

Le dictionnaire contient des chaînes de stems et des chaînes de suffixes. La méthode de reconnaissance d'une chaîne composée dans l'arborescence permet à la fois de reconnaître la chaîne du stem, et celle du suffixe.

On détermine, par cette méthode, l'ensemble des chaînes contenues dans la chaîne du mot. Ces chaînes constituent des stems possibles pour le mot. On choisit parmi ces chaînes la plus longue, et on recherche si dans la chaîne du mot, la partie non commune avec celle du stem est une chaîne de suffixe figurant dans le dictionnaire, en réutilisant l'algorithme de reconnaissance d'une chaîne de caractères dans l'arbre. On recherche si parmi les codes associés à la partie stem et ceux associés à la partie suffixe, il existe un, ou plusieurs couples stem-suffixe qui ont même fonction syntactique. Si on n'a pas trouvé de suffixe associé à un stem, on prendra parmi les chaînes de stems possibles qui restent, celle qui est la plus longue et on recommence la recherche du suffixe.

La méthode s'arrête donc dès qu'on a trouvé un couple stem-suffixe qui a même fonction syntactique. Elle déterminera de plus si ce couple est unique ou non (cas ambigu). On retrouve ainsi la contrainte introduite pour les cas ambigus dans la justification du plus court stem possible : le choix du stem est indépendant des stems existants dans le dictionnaire, sauf pour les mots ambigus où les différentes chaînes ambiguës doivent être associées au même stem.

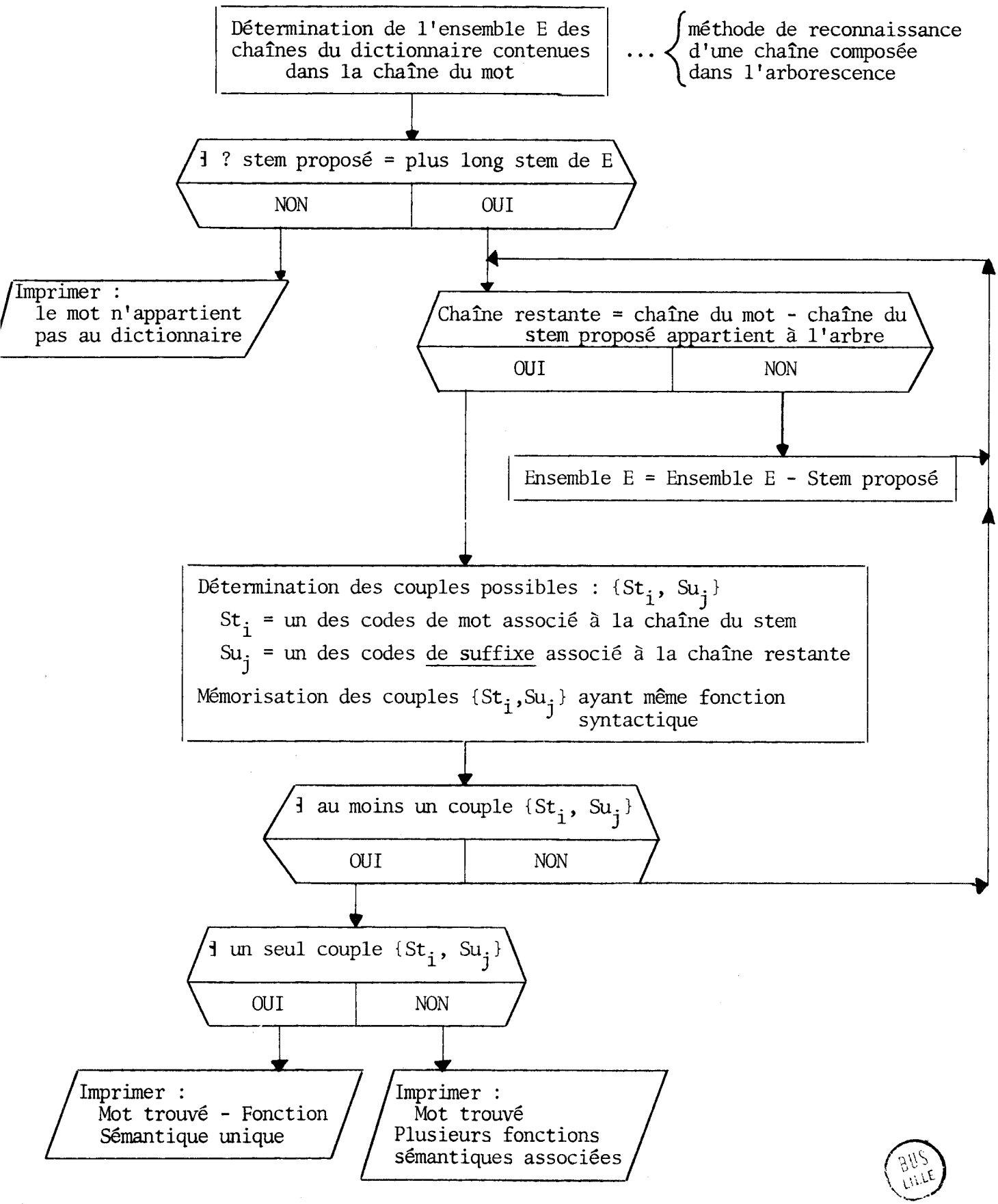
Si le plus long stem ne convient pas, la méthode recherchera le stem plus court qui est possible. Il n'est donc pas nécessaire d'associer systématiquement à un mot le plus long stem qu'il contient. La procédure de recherche, dans ce cas, est un peu plus longue. Mais l'expérience montre qu'en choisissant lors

de l'introduction des mots dans le dictionnaire les stems les plus courts possibles, on obtient, lors de la reconnaissance d'un mot dans le dictionnaire, le stem le plus long comme stem de mot, dans presque tous les cas.

Si le mot ne fait pas partie du dictionnaire, la méthode de recherche successivement si tous les stems déterminés dans la reconnaissance de la chaîne du mot sont possibles, et les rejettera. Le temps de recherche est augmenté, et on pénalise la recherche de mots qui n'appartiennent pas au dictionnaire.

L'organigramme général de la recherche d'un mot dans le dictionnaire résume cette procédure de recherche. On peut remarquer que cette méthode inclut la recherche d'un autre stem si le stem examiné n'est pas acceptable. Dans d'autres méthodes, ceci est considéré comme un cas d'erreur et fait appel à une procédure spéciale : "backtrap routine" du système Smart, par exemple [2]. On peut noter d'autre part, que les chaînes de suffixes figurent dans le dictionnaire lui-même, et non dans un dictionnaire spécial des suffixes. Ce choix permet de réutiliser le même algorithme pour rechercher stem et suffixe. Au point de vue volume du stockage, la fin de ce chapitre montrera que le volume du dictionnaire, contenant à la fois les chaînes de stems et de suffixes, est inférieur à la somme des volumes du dictionnaire des chaînes de stems et du dictionnaire des chaînes de suffixes.

ORGANIGRAMME GENERAL DE LA RECHERCHE D'UN MOT DANS LE DICTIONNAIRE : "RECHMOT"



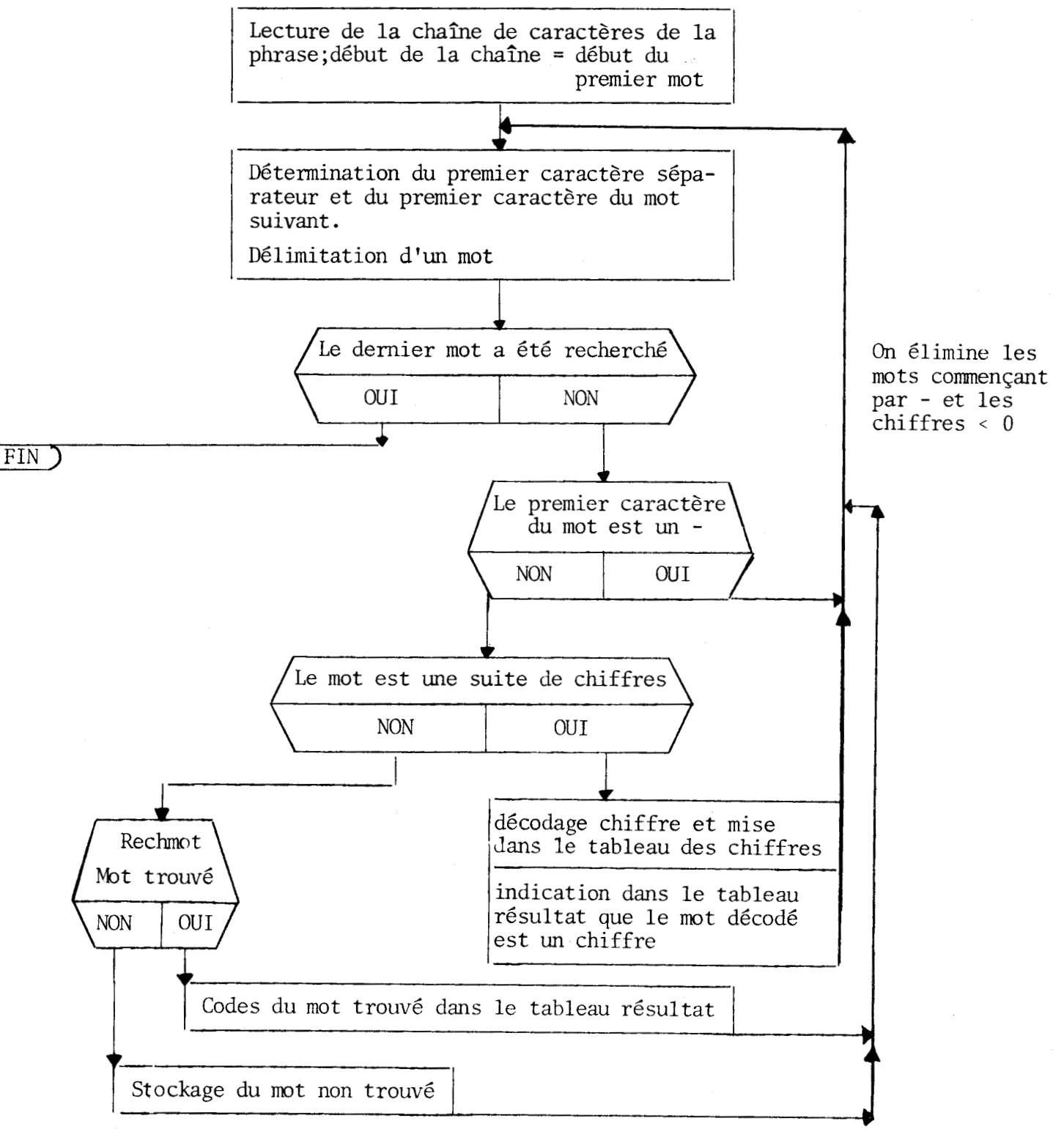
IV.6.2 RECHERCHE D'UNE PHRASE : "RECHPHRASE"

Cet algorithme est une extension de l'algorithme précédent. Il permet de rechercher l'ensemble des mots d'une phrase donnée, et de les ranger dans un tableau résultat, dans l'ordre où ils se présentent dans la phrase. Les mots non trouvés dans le dictionnaire seront rangés dans un autre tableau, et listés à la fin de l'analyse de la phrase. Le tableau résultat contiendra les pointeurs des mots trouvés. Il pourra être réutilisé par le système de documentation pour rechercher les informations associées aux mots de la phrase, éventuellement après une analyse syntactique de la phrase.

Dans l'algorithme qui est décrit ici, on a supposé que les chiffres n'étaient pas inclus dans le dictionnaire, mais étaient décodés directement à partir de la phrase. Cette convention n'a pas été utilisée dans l'exemple de documentation automatique décrit dans un des chapitres qui suit. Par contre, l'algorithme décrit ici a été réutilisé pour une des méthodes de détermination des mots vides qui sera exposée dans le chapitre qui suit.

Le principe de cette méthode de reconnaissance des mots d'une phrase a été résumé dans un organigramme.

ORGANIGRAMME GENERAL DE LA RECHERCHE D'UNE PHRASE



IV.7 REMARQUES SUR LE DICTIONNAIRE -1- CHAINAGE

Les algorithmes décrits ici sont les algorithmes de base permettant la création, la mise à jour, et la recherche des mots dans un dictionnaire.

Ces procédures ont été utilisées pour créer les dictionnaires qui ont servi à l'étude comparative des performances, qui sera décrite dans le chapitre qui suit. Ces créations et mise à jour ont été faites dans un contexte batch-processing à cause de la configuration de l'ordinateur, et du software utilisables, et de la surcharge du centre de calcul. Il serait possible d'utiliser ces procédures à l'aide d'un langage intéractif pour une mise à jour suivant un processus analogue à celui qui a été simulé pour le système de documentation décrit dans le chapitre VII.

Le principe de la recherche d'un caractère dans le dictionnaire est basé sur l'utilisation de chaînage d'alternants. En supposant que toutes les possibilités d'assemblage de caractères existent dans le dictionnaire, il faudrait 26 renvois successifs dans un chaînage pour retrouver le caractère "z" : A B ... Y Z. Si toutes les possibilités existent dans le dictionnaire, il faudra, en moyenne, 13 renvois pour retrouver un caractère. On pourrait penser à réduire le nombre d'opérations, en utilisant un système à plusieurs pointeurs. Mais l'exemple documentaire a montré que le nombre de mots recherchés pour chaque demande est assez faible (environ 20 à 30 mots). Par contre, dans un context de multiprogrammation, ou de time-sharing, la place mémoire nécessaire est un paramètre important. L'étude des performances du dictionnaire va montrer que le volume nécessaire pour représenter le dictionnaire par la méthode exposée dans ce chapitre est assez performante pour éviter de séparer le dictionnaire en un trop grand nombre de pages, et que le temps de recherche s'en trouve amélioré.

CHAPITRE V

ETUDE COMPARATIVE des PERFORMANCES du DICTIONNAIRE -1- CHAINAGE

V.1 - INTRODUCTION

V.2 - DICTIONNAIRES UTILISES POUR CETTE ETUDE COMPARATIVE

V.3 - PRINCIPAUX PARAMETRES

V.4 - TEMPS DE CREATION ET DE MISE-A-JOUR

V.5 - TEMPS DE RECHERCHE

V.6 - ETUDE SUR LES STEMS DE MOTS

V.7 - VOLUME DE STOCKAGE

V.1 INTRODUCTION

Ce chapitre est consacré à une étude des performances du dictionnaire. Les temps cités dans cette étude dépendent de l'ordinateur utilisé. Il a donc semblé nécessaire de faire des études comparatives de temps et de performances à l'aide du même ordinateur (un CII 10070), sur plusieurs dictionnaires.

A cause des difficultés dues à la perforation des données, et à la surcharge du centre de calcul, il n'a pas été possible de créer plus de trois dictionnaires, aux caractéristiques très différentes.

Le paragraphe qui suit va présenter les trois dictionnaires. On déterminera ensuite les principaux paramètres auxquels nous nous intéresserons.

V.2 DICTIONNAIRES UTILISES POUR CETTE ETUDE COMPARATIVE

Les trois dictionnaires qui ont été créés sont :

- "DIC B 159". Ce dictionnaire contient 313 stems de mots, et les chaînes des suffixes. Les stems ont été déterminés à la main, et entrés, dans l'ordre alphabétique, par cartes. DIC B 159 a été utilisé surtout pour tester les algorithmes de base du dictionnaire -1- chaînage, et comparer la méthode de création automatique des stems à une méthode manuelle. Les mots associés aux stems correspondent à un ensemble de descriptions de phénomènes statistiques ou économiques.

- "DIC 3600". Ce dictionnaire a été créé pour tester la méthode de représentatic de l'arborescence sur un ensemble de mots assez important. Il contient 3752 stems de mots et atteint donc l'importance d'un dictionnaire d'un système documentaire complet [2], [5]. Parmi les stems, 3693 ont été déterminés automatiquement. Les mots correspondant à ces stems ont été choisis au hasard, dans une sortie de K.W.I.C. index prêtée par le Laboratoire de Calcul. Les autres stems ont été introduits par carte ; ils correspondent à un certain nombre de mots-pivots, ou de mots-vides. Ils ont été ajoutés pour l'étude sur les temps de recherche citée dans un des paragraphes qui suit. DIC 3600 contient également l'ensemble des chaînes de suffixes.

- "DIC TS" a été créé pour l'exemple de documentation automatique décrit dans un des chapitres qui suit. Il permet de reconnaître un ensemble de mots contenus dans une collection de "titres", ou descriptifs de phénomènes statistiques ou économiques, repérés par le système documentaire proposé. Ce dictionnaire contient 903 stems de mots, dont 619 seulement, correspondant à une partie des mots-pivots, ont été déterminés automatiquement. Les autres ont été déterminés directement, et entrés par cartes, pour ne pas surcharger la perforation. Ils correspondent à des mots non significatifs, à des abréviations, à des noms propres, ou à des mots invariables.

Seuls, les deux derniers dictionnaires ont été créés suivant la même technique.

.3 PRINCIPAUX PARAMETRES

En ce qui concerne les dictionnaires, on peut distinguer deux problèmes différents :

- d'une part, la création et la mise à jour
- d'autre part, l'utilisation, dans un système de documentation.

La création, et la mise à jour se font en dehors de l'utilisation du système documentaire. Les paramètres qui concernent ce problème sont importants, mais leur optimisation n'est pas primordiale. Par contre, il est nécessaire d'optimiser les paramètres qui concernent l'utilisation. Les paramètres essentiels sont :

- le volume de stockage du dictionnaire, qui permettra, suivant les cas, de laisser en mémoire le dictionnaire pour des interrogations répétées, ou de le charger à partir d'un périphérique, soit en totalité, soit par morceaux, suivant un système de pagination du dictionnaire;

- le temps de recherche des mots, et la détermination des codes qui leur correspondent dans le dictionnaire.

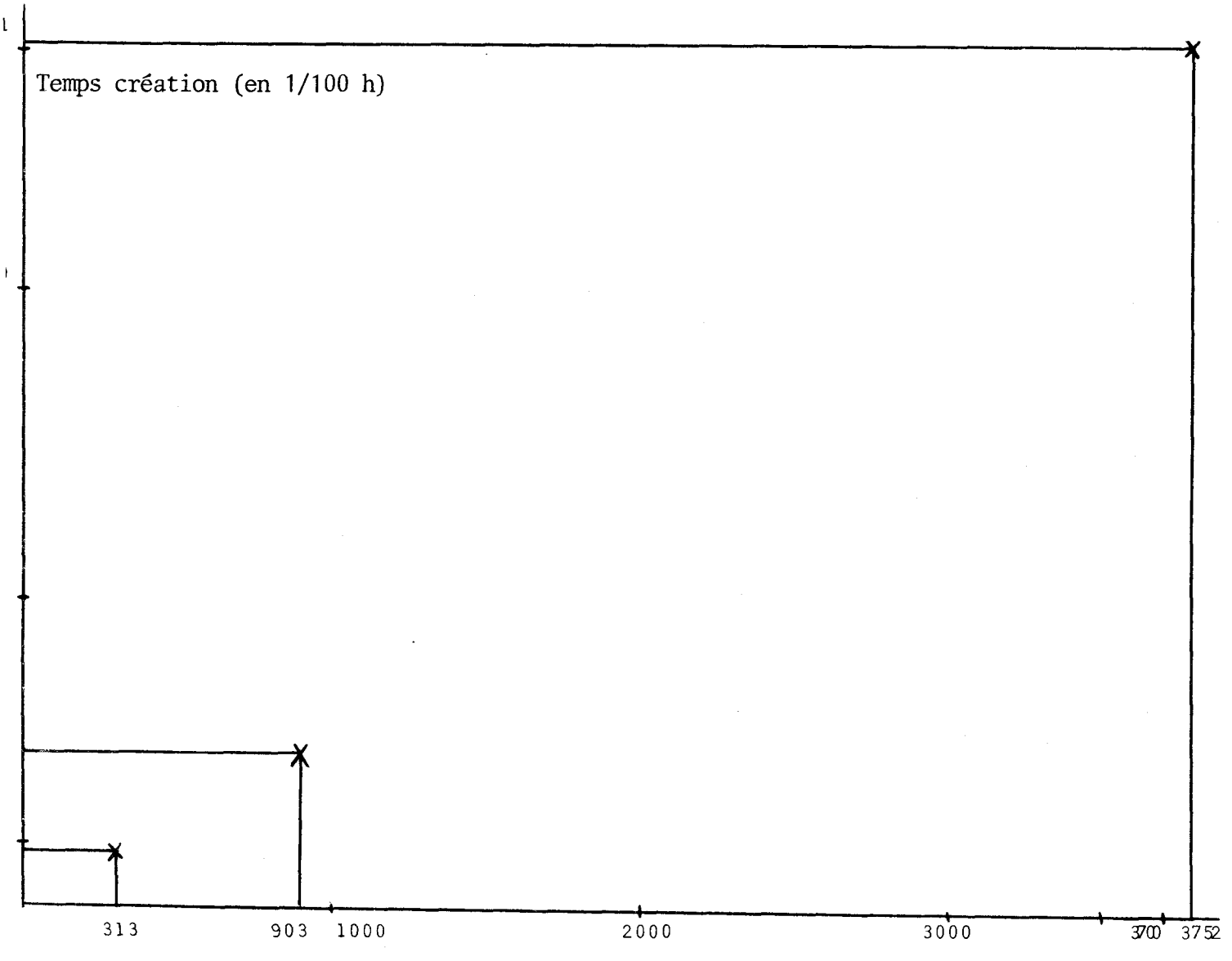
A titre indicatif, les temps de création des différents dictionnaires ont été relevés :

V.4 TEMPS DE CREATION ET DE MISE A JOUR

Un tableau résume les temps de création des trois dictionnaires. Les temps indiqués sont les temps d'unité centrale utilisés pour les calculs eux-mêmes. Les temps d'entrée-sortie sont décomptés à part, et varient suivant le contexte qui entoure le programme au moment de son exécution. Ils n'ont donc pas été pris en compte. Ces temps ne comprennent pas, non plus, les temps de compilation des programmes.

CREATION DES DICTIONNAIRES

	DIC B 159	DIC TS	DIC 3600
Détermination des stems automatiquement			
Nombre de stems déterminés		619	3693
Temps utilisateur (en 1/100 h).		0.92	5.549
Création du dictionnaire			
Nombre total de chaînes de stems (sans compter les chaînes de suffixes)	313	903	3752
Temps utilisateur (en 1/100 h).	0.91	2.657	14.151



Il est très difficile de tirer une conclusion concernant ces temps de création. Ils dépendent beaucoup de la longueur moyenne des chaînes introduites, de l'importance relative des parties communes de ces chaînes, et de l'ordre dans lequel elles ont été entrées.

Pour les dictionnaires "DIC B 159" et "DIC 3600", les stems de mots ont été entrés par ordre alphabétique, à cause de la façon dont ils ont été choisis. Les suffixes, et les stems supplémentaires (pour DIC 3600) ont été entrés par cartes, et insérés dans l'arborescence des chaînes de stems. On peut remarquer que le temps de création est à peu près proportionnel au nombre de mots contenus dans le dictionnaire (rapport : environ 1 à 12 dans les deux cas), ce qui semble acceptable.

Les temps de mise à jour sont évidemment très variables. Ils dépendent du volume du dictionnaire, et donc du nombre d'opérations nécessaires pour introduire les caractères nouveaux dans le tableau des caractères, et pour remettre en ordre les pointeurs.

On peut citer, par exemple, l'insertion de 700 chaînes de suffixes dans l'arborescence créée par les 3714 stems de mots de DIC 3600 en 3,7 centièmes d'heure. Ces chaînes de suffixes, réparties suivant tout l'alphabet, ou presque, ont nécessité des décalages de caractères, et des remises en ordre de pointeurs assez importants.

.5 TEMPS DE RECHERCHE

Un des buts de la représentation sous forme arborescente est de limiter les temps de recherche lorsque la taille du dictionnaire augmente.

Une étude comparative sur les temps de recherche a été effectuée sur les deux dictionnaires dont les caractéristiques différaient le plus : "DIC B 159" et "DIC 3600". L'étude a consisté à rechercher, à l'aide du programme "Rechphrase", les codes des mots contenus dans une phrase.

Un ensemble de 23 phrases, faisant partie du vocabulaire qui avait servi à créer "DIC B 159" ont été choisies, au hasard. On a introduit, ensuite dans "DIC 3600" les stems des mots de ces 23 phrases qui n'y figuraient pas, afin d'obtenir strictement les mêmes tableaux résultats de codes.

Ces 23 phrases contenaient 239 mots :

	DIC B 159	DIC 3600
Temps de constitution des 23 tableaux de codes en (1/100 h)	0.114	0.208
Nombre de stems dans le dictionnaire	313	3752

Le temps de recherche est à peine multiplié par 2, tandis que le nombre de stems du dictionnaire a été multiplié par plus de 10. Il n'y a donc pas proportionnalité entre le nombre de stems et le temps de recherche. On cite dans certains ouvrages [2] un temps de recherche proportionnel à $\log_2 N$, si N est le nombre de noeuds terminaux de l'arbre.

Une autre étude a été faite sur "DIC 3600" afin de déterminer, dans la recherche d'un mot, la proportion des cas où le stem du mot n'est pas le plus long stem contenu dans le mot. Les résultats seront exposés dans le paragraphe qui suit. La recherche a porté sur 3566 mots comportant en moyenne plus de 9 caractères, et le temps d'exécution a été de 2 158 centièmes d'heure (77,68s) soit 0,021s en moyenne pour déterminer les codes d'un mot, si ce mot peut être reconnu à l'aide du dictionnaire.

Il faut remarquer que les mots recherchés dans "DIC 3600" avaient, en moyenne, plus de 9 caractères, et que dans une phrase quelconque, il existe un nombre important de mots qui comportent peu de caractères, en particulier les articles et les mots de liaison.

V.6 ETUDE SUR LES STEMS DE MOTS

L'étude précédente a été faite afin de déterminer, dans la recherche d'un mot, la proportion des cas où le stem du mot n'est pas le plus long stem contenu dans le mot. C'est un essai de justification, à posteriori, du choix du plus court stem possible (cf. Chap II.4.3).

La procédure de recherche est telle que si le plus long stem contenu ne convient pas, on recherche si un des stems plus court convient. Dans le cas d'un mot qui figure dans le dictionnaire, le temps de recherche est donc augmenté si le plus long stem contenu dans le mot n'est pas le stem du mot.

Dans la recherche de 3566 mots dans "DIC 3600", pour 208 mots, le stem qui figure dans le dictionnaire n'est pas le plus long stem contenu. Ceci représente environ 5,8% de l'ensemble des mots recherchés.

Cette proportion peut donc être considérée comme acceptable, car elle ne pénalise pas trop les temps de recherche. Elle a l'avantage, par contre, de diminuer le volume du stockage, qui est un des paramètres principaux à optimiser

.7 VOLUME DE STOCKAGE

La méthode de mémorisation sous forme arborescente présente l'avantage de contenir sa propre clé de recherche : il suffit de parcourir les branches de l'arbre, en suivant éventuellement les chaînages pour retrouver les caractères contenus dans une chaîne de caractères. Il n'est pas nécessaire de créer, comme dans d'autres méthodes, des tables d'accès pour rechercher les codes de mots ; le chaînage contient sa propre clé d'accès. Les temps de recherche présentent également l'avantage de varier peu lorsque le nombre de chaînes stockées augmente; il n'y a pas proportionnalité avec le nombre de chaînes, mais avec le logarithme de ce nombre.

Le volume de stockage reste un des paramètres les plus importants à optimiser.

Dans un système de documentation automatique complet, un dictionnaire contient plusieurs milliers de mots, ou de stems de mots [2], [5]. Le volume représenté par l'ensemble de ces informations va influer sur le mode d'utilisation, et de stockage du dictionnaire.

Dans un système fonctionnant en batch-processing, il suffit de regrouper l'ensemble des demandes, d'effectuer en une seule fois les opérations de reconnaissance des mots et de recherche des pointeurs de mots dans le dictionnaire, et de stocker, éventuellement sur un périphérique, l'ensemble des pointeurs trouvés [5]. Le volume n'est pas,ici, le paramètre le plus important. Il suffit que la mémoire puisse contenir l'ensemble du dictionnaire.

On peut remarquer, cependant, que dans certains cas, la taille de mémoire disponible peut obliger à séparer le dictionnaire en plusieurs parties, ou pages. La pagination augmente les temps de recherche, car il faut trier les mots, avant la recherche des pointeurs de mots dans une page. On peut également séparer le dictionnaire en deux dictionnaires : un dictionnaire des mots fréquents, et un dictionnaire des mots rares [5], dont l'utilisation sera différente.

Par contre, dans un système possédant une partition Time-Sharing à ressources bloquées, le volume est un paramètre très important. Il en est de même pour un système à allocation dynamique de ressources, où plus le volume de mémoire nécessaire est important, plus le temps en file d'attente peut être augmenté.

C'est le cas, par exemple, avec un système d'allocation analogue au GECOS III du GE 635.

Dans un contexte de multiprogrammation, il peut y avoir interruption du programme chaque fois qu'il y aura une demande d'entrée-sortie. Si le dictionnaire est paginé, il y aura interruption de programme chaque fois qu'une nouvelle page du dictionnaire sera appelée en mémoire ; le temps d'exécution sera donc augmenté. Ceci est vrai également en Time-Sharing : il y a remise dans la file d'attente chaque fois qu'il y a une demande d'entrée-sortie, indépendamment du temps d'exécution alloué par le système pour chaque exécution du programme entre deux opérations de swapping (ex. Time-Sharing du GE 265, ou du XDS 940).

Il faudra donc, dans chacun des cas, optimiser le volume des données qui vont coexister en mémoire avec les programmes. Il s'agira, en batch processing, du volume mémoire non occupé par le système et les programmes, par exemple, dans lequel on cherchera à stocker le plus d'informations possibles.

En multiprogrammation, ou en Time-Sharing, le volume optimum dépendra du système d'allocation de ressources, ou du système de gestion de file d'attente. Si on doit paginer le dictionnaire, un autre paramètre interviendra pour déterminer le volume optimum : il faudrait minimiser le nombre de pages pour éviter d'augmenter trop de nombre d'entrées-sorties.

Le volume du stockage du dictionnaire est donc un paramètre très important ; c'est peut-être même le paramètre qu'il faut optimiser en premier.

Indépendamment de la simplification de la procédure de recherche, le système de représentation d'une arborescence à l'aide de pointeurs d'alternants permet de minimiser le volume nécessaire pour mémoriser cet arbre. D'autre part, la décomposition des mots en stem + suffixe, et le choix du plus court stem possible ont permis de réduire la longueur des chaînes de caractères qui figurent dans cet arbre, et donc leur partie non commune (cf. chapitre II).

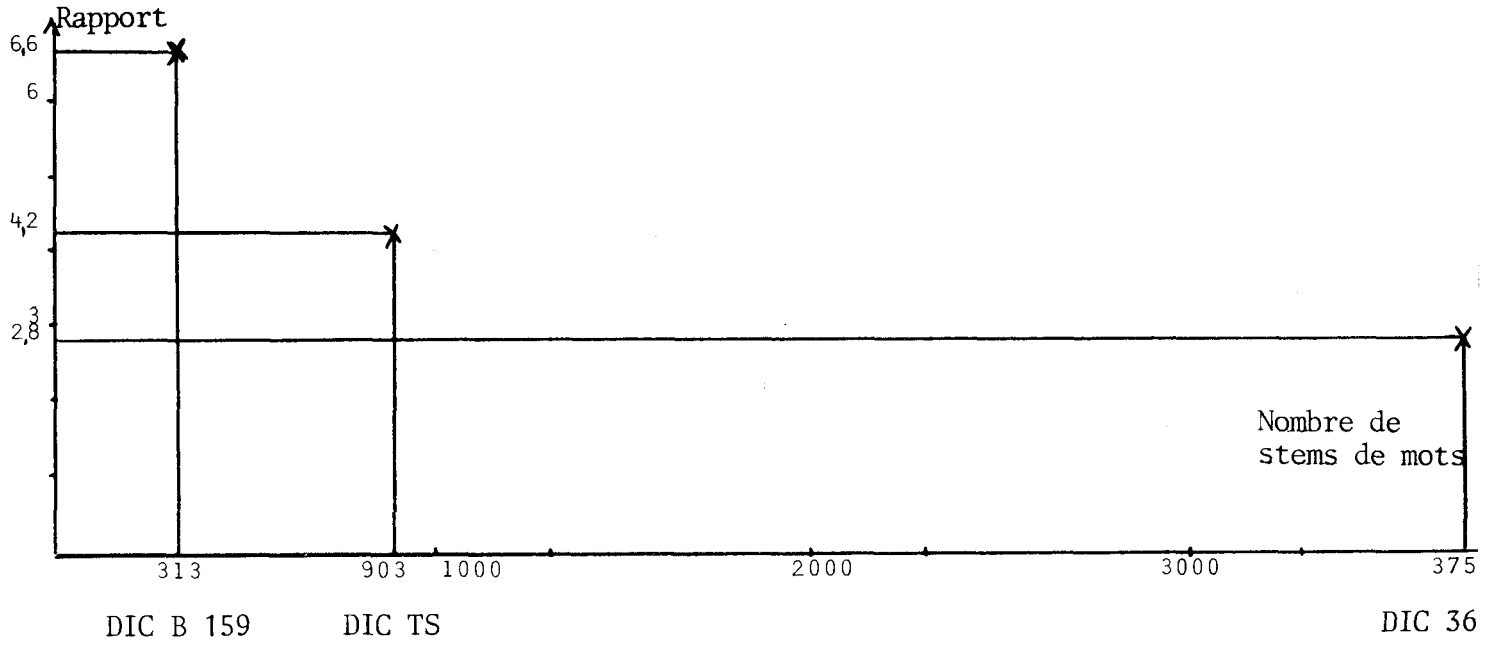
Une étude comparative a été effectuée sur les trois dictionnaires, pour déterminer les caractéristiques et les performances de la méthode de représentation en mémoire. Les principaux résultats de cette étude ont été regroupés dans un tableau des paramètres de stockage.

UDE COMPARATIVE DES PARAMETRES DU STOCKAGE

	DIC B 159	DIC TS	DIC 3600
Nombre de stems de mot déterminés automatiquement		619	3693
Nombre total de caractères des "mots" *		5076	33589
Longueur moyenne des "mots"		8,20	9,02
Nombre total de caractères des stems		3266	21125
Longueur moyenne des stems		5,27	5,72
Nombre de caractères du tableau des caractères après mise en arbre des chaînes de stems		2252	9329
Nombre total de chaînes de stems. (Les chaînes de suffixes n'entrent pas dans ce total)	313	903	3752
Nombre de caractères du tableau des caractères. Dictionnaire complet (avec les suffixes)	2073	3792	10682
Pourcentage de pointeurs non nuls	∿42	54,81	59,72
Rapport : $\frac{\text{Nombre de caractères du tableau des caractères}}{\text{nombre de stems}}$	6,6	4,2	2,8

* Les stems ont été déterminés par la méthode creatstems-double-forme (cf. chap. II). Le nombre total de caractères des "mots" a été obtenu en faisant la somme du nombre de caractères contenus dans les formes de mots données comme entrée de creatstems-double-forme, et en divisant le total obtenu par 2. En effet, deux formes du même mot sont nécessaires, dans cette méthode, pour déterminer le stem du mot.

Les trois dictionnaires étudiés ont des caractéristiques très différentes. Un des paramètres les plus intéressants du tableau de l'étude comparative est le rapport du nombre de caractères du tableau des caractères au nombre de stems du dictionnaire :



Ce rapport décroît lorsque le nombre de stems dans le dictionnaire augmente : il passe de 6,6 pour DIC B 159 à 2,8 pour DIC 3600.

La variation de ce rapport confirme l'hypothèse faite au chapitre II : plus l'arborescence contient de chaînes, plus le nombre de caractères nécessair pour introduire une nouvelle chaîne sera faible. Il va tendre vers une limite égale à 1 : il faut, en effet, au moins un caractère de fin de chaîne pour introduire une nouvelle chaîne dans l'arbre.

Un autre paramètre intéressant est le pourcentage de pointeurs non nuls. Ce pourcentage augmente avec la taille du dictionnaire. Il donne également une indication sur la structure de l'arbre : plus ce pourcentage est important, plu l'arbre comporte de noeuds, et plus les chaînes sont voisines. Par exemple, la mise en arbre des 700 chaînes de suffixes comporte 1431 caractères dans le tabl des caractères, et le pourcentage de pointeurs non nuls est de 67,7%. (A titre indicatif,le rapport du nombre de caractères au nombre de chaînes pour l'ensemb des suffixes est égal à 2).

Le suffixe déterminé automatiquement par la méthode de création desstems, réalisée sur les deux derniers dictionnaires, comporte en moyenne 3 caractères, qui sont enlevés à la fin de la chaîne du mot pour donner celle du stem. Les parties non communes entre les chaînes sont ainsi diminuées.

L'arbre correspondant se trouve ainsi réduit. Une comparaison a été faite à l'aide des éléments de DIC TS. Une forme de mot a été choisie pour chacun des stems contenus dans DIC TS, parmi les cartes de données qui ont servi à créer ce dictionnaire. La mise en arbre de ces chaînes de mots a donné un tableau de caractères comportant 4323 éléments, alors que le tableau du dictionnaire (stems + suffixes) comprend 3792 éléments seulement. L'arbre obtenu par la réunion des chaînes de stems et des chaînes de suffixes représente un volume plus faible que celui de l'arbre des chaînes de mots, ce qui correspond bien au principe de la mise sous forme arborescence exposé au chapitre II.

Les chaînes de suffixes ont été réunies avec les chaînes de stems pour constituer les dictionnaires. Cela a permis, comme l'a montré la description du dictionnaire -1- chaînage, de réutiliser pour la recherche d'un mot la partie de l'organigramme correspondant à la reconnaissance d'une chaîne de caractères dans l'arborescence, successivement pour la chaîne de stem et la chaîne de suffixe. Le programme ainsi réalisé est plus court, et n'utilise pas de procédure, pour éviter, en cours d'exécution, des appels de procédures qui sont assez longs. La réunion des chaînes de stems et de suffixes permet également de gagner de la place. Par exemple, pour DIC 3600, la mise en arbre de 3693 stems correspond à un tableau de caractères de 9329 éléments. L'adjonction des 700 chaînes de suffixes porte ce tableau à 10559 caractères. La différence représente 1230 caractères, alors que la mise en arbre des suffixes représente 1435 caractères. Le gain de place correspond à 200 caractères environ.

L'étude sur le volume du stockage a porté, jusqu'ici, sur une comparaison des dictionnaires réalisés. Les chaînages ont été évalués par le nombre de caractères du tableau des caractères. Le volume de mémoire réellement occupé est un multiple de ce nombre de caractères.

Dans le dictionnaire, à un caractère est associé un pointeur d'alternant, ou un pointeur de mot. Dans la réalisation en Fortran exposée précédemment, un caractère de chaîne était stocké dans un mot-machine 10070 ; de même pour les pointeurs. Pour un volume exprimé en mots-machine 10070, le facteur de multiplication est donc de 2. Par exemple, DIC TS, dont le tableau des caractères comporte 3792 éléments correspondra à 2 x 3792 = 7584 mots-machine 10070. Avec les programmes écrits en Fortran, la place perdue est assez importante, comme le montre l'annexe 4. Il est possible, à l'aide de procédures écrites en langage machine, et incluses dans les programmes Fortran, de stocker l'ensemble caractère-pointeur associé dans un seul mot 10070, suivant la configuration présentée

dans l'annexe 4, et adaptée au langage machine 10070. Avec ces hypothèses, DIC TS, par exemple, occupera 3792 mots 10070.

Il est donc possible d'indiquer quelques éléments de comparaison entre le stockage sous forme arborescente, et d'autres organisations de mémorisation.

La comparaison va porter sur les deux derniers dictionnaires. L'ensemble des résultats a été réuni dans un tableau où les volumes sont exprimés en octets. L'étude va porter sur le chaînage des stems déterminés automatiquement, avec adjonction des chaînes de suffixes. Le programme de mise à jour ne fait pas de statistique sur les volumes, car il a servi à mesurer les temps de création et de mise à jour. Dans l'étude sur les volumes qui suit, la comparaison va porter sur un ensemble de stems d'une part, et sur un ensemble de mots d'autre part, choisis à raison d'un mot pour chaque stem du premier ensemble. Il faut noter que l'ensemble stem-classe de suffixes permet de reconnaître les différentes formes possibles d'un mot. Une comparaison pour être exacte, devrait porter sur les stems d'une part, et sur l'ensemble des formes de mots qui peuvent être reconnues à l'aide de ces stems.

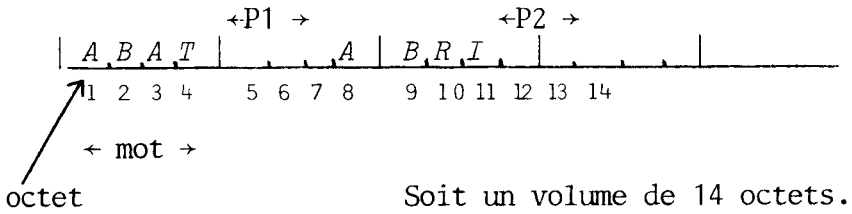
	DIC TS		DIC 3600	
		Volume (en octets)		Volume (en octets)
Nombre de stems	619		3693	
Nombre de caractères des "mots" *	5076	5076	33589	33589
Longueur moyenne des mots (en caractères)	8,20		9,02	
Nombre de caractères des stems	3266	3266	21125	21125
Longueur moyenne des stems (en caractères)	5,27		5,72	
Chaînage des stems (nb caractères) Avec les pointeurs	2252	10008	9329	37 316
Nombre de caractères du tableau des caractères. Chaînage Stems + Suffixes Avec les pointeurs	3482	13928	10560	42240
Volume des Pointeurs de mot (3 octets par pointeur)		1857		11079

Volume chaînage arborescent Stems + Suffixes	13 008		42 240
Volume représenté par les "mots" et les pointeurs de mots	6 993		44 668
Représentation des stems en format fixe (10 caractères/stem) et pointeurs (3 octets par pointeur)	8 047		48 009

* voir remarque du tableau de l'étude comparative des paramètres du stockage.

Le "volume représenté par les mots et les pointeurs" est le volume obtenu en stockant, à la suite les unes des autres, les chaînes de mots et les pointeurs de mots sans aucune organisation pour l'accès.

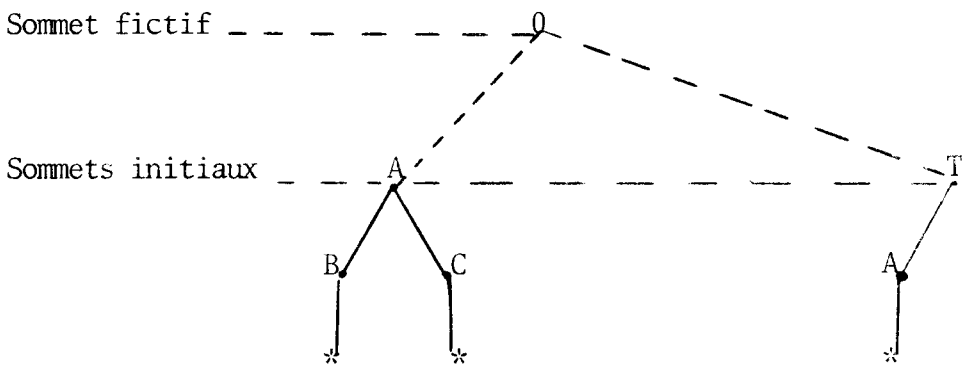
Par exemple, les chaînes *abat*, *abri* et leurs pointeurs $\{P_i\}$ seront représentés :



Soit un volume de 14 octets.

Pour DIC TS, ce volume est plus faible (6993 octets) que le volume du chaînage arborescent (13008 octets).

Par contre, pour DIC 3600, ce volume est plus important (44668 octets) que celui de l'arborescence des stems et des suffixes (42240 octets). En effet, plus l'arbre contient de chaînes, plus le nombre de caractères nécessaires pour introduire une nouvelle chaîne risque d'être faible. Il existe donc un seuil à partir duquel le stockage arborescent permet de gagner de la place en mémoire. Il faut remarquer que l'arbre des chaînes de caractères est la réunion de plusieurs arbres partiels, dont le noeud initial est un des caractères qui peut figurer en première position dans une chaîne. Ces noeuds initiaux ont été chaînés entre eux, à l'aide d'un sommet fictif, pour représenter l'ensemble des arbres en une seule arborescence.



Les résultats constatés pour l'arborescence ainsi constituée restent vrais pour les arbres partiels. Ce qui veut dire, en particulier, qu'une pagination du dictionnaire n'affecte pas les résultats constatés.

Donc, au-delà d'un seuil, la représentation sous forme arborescente donne des résultats intéressants pour le stockage, et en même temps, les temps d'accès ne sont pas fortement pénalisés (temps d'accès proportionnel au logarithme du nombre de chaînes). Il est donc intéressant de préciser, sur les résultats donnés par le plus gros des dictionnaires constitués les performances de ce mode de stockage.

Si on décide, par exemple, de ne stocker que les stems de mots, et de les organiser en vue d'une recherche séquentielle, ou séquentielle indexée, il est nécessaire de stocker chaque stem dans une zone de longueur fixe. La longueur moyenne des stems, pour DIC 3600 est de 5,72 caractères. Si on prend une zone de longueur fixe, égale à 9 caractères, le volume occupé par les chaînes de caractères sera de 3693 x 9 = 33 237 octets, auxquels il faudra ajouter les pointeurs de mot, soit 3693 x 3 = 11079 octets.

Le volume total sera donc de 33 237 + 11 079 = 44 316 octets, sans compter le volume occupé par les tables dans une méthode de séquentiel indexé.

Or, une zone de 9 octets n'est pas toujours suffisante pour distinguer deux stems : par exemple : *R A T I O N A L I | S* de *rationalis - er*

R A T I O N A L I | S de *rationalis - ation.*

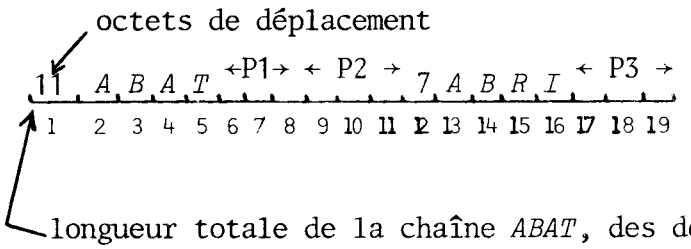
Il sera donc nécessaire de prendre une zone de longueur plus grande. Avec une zone de 11 octets, le volume occupé par les stems devient : 51 702 octets, alors que l'arborescence comprenant les stems et les suffixes n'occupe que 42 240 octets. On peut signaler, à titre indicatif, que le chaînage des stems seuls occupe un volume de 9329 x 4 = 37 316 octets.

On peut imaginer une autre méthode de représentation des stems, qui occupe moins de volume que la méthode précédente, qui obligeait à choisir une zone de longueur fixe plus importante que les 9 octets qui ont servi de base aux calculs de comparaison.

La méthode consiste à stocker les chaînes de stems, et les pointeurs de mot à la suite les uns des autres, mais, pour que les temps de recherche ne soient pas prohibitifs, on stocke dans un octet la longueur de la chaîne du

stem et du (ou des) pointeur de mot. Le déplacement est toujours inférieur à 256, et peut donc être représenté dans un octet.

Par exemple : la représentation de *abat, abri* avec deux pointeurs pour *abat*



Il n'est pas nécessaire de tester tous les caractères pour rechercher le début de la chaîne suivante, si la chaîne explorée ne convient pas.

Pour DIC 3600, le nombre total de caractères des stems est égal à 21 125. Le volume correspondant est donc de 21 125 octets. Chaque pointeur comporte 3 octets, et il y a 3693 stems. En comptant les octets de déplacement, le volume total est de : 21 125 + 3693 x 4 = 35 897 octets. Le volume correspondant du chaînage arborescent des chaînes de stems est de 37 316 octets. Les deux volumes sont donc assez voisins. Il faut noter, cependant, que le volume donné pour la méthode proposée serait plus faible, car plusieurs codes peuvent correspondre au même stem, d'après la méthode de détermination des stems. Mais cette méthode obligerait probablement à choisir des stems de mots plus longs que précédemment, car la procédure de recherche du stem du mot,dans le cas où le plus long stem contenu ne serait pas acceptable, risque d'être plus coûteux en temps. D'autre part, il faut prévoir des tables d'indexation pour réduire les balayages dans le tableau, et donc les temps d'accès.

Il est probable que pour un nombre de stems plus important, le volume du stockage arborescent deviendrait plus faible, que le volume correspondant à la méthode proposée. Les temps d'accès dépendent, dans la méthode proposée, du nombre de stems (n) repérés par un pointeur de la table d'indexation. Il y a proportionnalité du temps de recherche avec n, tandis que par la méthode par arbre, il y a proportionnalité du temps de recherche avec le logarithme du nombre de stems. Pour un grand nombre de stems, les temps d'accès correspondant à la méthode proposée risque d'être beaucoup plus importants que par la méthode de stockage arborescent.

La méthode de stockage sous forme arborescente permet, à partir d'un certain nombre de stems, d'obtenir un gain de volume par rapport à d'autres méthodes. Ce gain croît avec le nombre de mots contenus dans le dictionnaire, sans pénaliser les temps d'accès, qui restent proportionnels au logarithme du nombre de chaînes qui figurent dans l'arbre.

Les chapitres qui suivent vont être consacrés à l'exposé d'un exemple de système de documentation automatique utilisant le dictionnaire DIC TS qui a été cité ici. Auparavant, le chapitre qui suit est consacré au problème important de la détermination des mots non significatifs : mots-vides et mots grammaticaux.

C H A P I T R E VI

D E T E R M I N A T I O N des M O T S N O N S I G N I F I C A T I F S

I.1 INTRODUCTION

Le problème de la détermination des mots non significatifs est un problème très important en documentation automatique.

Il s'agit de mots qui portent des fonctions syntactiques uniquement, mais qui ne contribuent pas directement à préciser le concept sémantique de phrase. Ces mots peuvent servir, dans une analyse syntactique, pour accélérer la détermination du concept sémantique de phrase parmi l'ensemble des concepts repérés par le système de documentation automatique.

Plusieurs méthodes sont proposées dans ce chapitre, pour déterminer les mots non significatifs.

VI.2 MOTS NON SIGNIFICATIFS

On distingue du point de vue sémantique deux classes de mots :

- les mots significatifs ou mots-pivots, ou pivots : dont la réunion donne le concept sémantique de phrase. La suppression d'un pivot modifie le concept sémantique de phrase.

- les mots non significatifs, dont la suppression ne modifie pas le concept sémantique de phrase. Ce sont :

  . soit des mots ordinaires dont le sens sémantique n'est pas assez restrein pour contribuer à déterminer le concept sémantique de phrase.
  Par exemple, certains abverbes. Dans : " *personnes travaillant dans l'agriculture seulement*". L'adverbe *seulement* n'aide pas à déterminer le concept sémantique de la phrase.

  . soit des mots-outils, qui n'ont pas de signification propre, mais qui accompagnent les mots-pivots. Le chapitre I a dressé une énumération de ces mots-outils.
  Ce sont :
  - les articles
  - les adjectifs et pronoms possessifs
  - les adjectifs et pronoms démonstratifs
  - les pronoms personnels
  - les numéraux
  - les prépositions
  - les conjonctions
  - les pronoms relatifs.

  Ces mots-outils recevront un code d'utilisation documentaire différent suivant leur fonction syntactique ou sémantique dans la phrase.

Certains accompagnent les mots-pivots, mais n'apportent aucune information syntactique ou sémantique supplémentaire. Par exemple, certaines prépositions, ou certains articles. Dans ce cas, ils recevront un code d'utilisation documentaire de mot-vide.

D'autres introduisent, précisent, ou remplacent des mots-pivots. Par exemple, un pronom personnel peut remplacer un mot-pivot. Dans ce cas, le mot-outil apporte une information syntactique ou sémantique, et contribue à reconnaître la structure de la phrase. Dans ce cas, le code d'utilisation documentaire sera le code d'un mot grammatical, et le code de concept correspondant repérera ce mot parmi l'ensemble des mots grammaticaux.

Il est donc important de déterminer avec précision les mots non significatifs. En effet, ceux qui ont un code documentaire de mot grammatical servent de base pour l'analyse syntactique de la demande. D'autre part, un mot non significatif pris comme mot-pivot va augmenter le temps nécessaire pour déterminer le concept sémantique de phrase dans la mesure où il n'apporte aucune indication sémantique supplémentaire. Dans certains cas, même, il peut fausser la recherche.

Plusieurs méthodes sont proposées dans ce chapitre, pour déterminer les mots non significatifs d'un ensemble de documents ayant trait au même sujet. On constate, en effet, que dans les systèmes documentaires les plus évolués [2], l'analyse des mots non significatifs, la mise en évidence des relations entre concepts, la détermination des hiérarchies entre concepts, sont faits généralement à partir de l'ensemble des documents ayant trait au même sujet, et parfois à partir d'un échantillon pris dans cet ensemble.

Pour la création du dictionnaire, les mots-pivots vont être traités différemment des mots non significatifs. Les stems des mots-pivots seront créés automatiquement, et le code de concept attribué automatiquement lors de l'insertion de ces stems dans le dictionnaire. Les synonymies repérées vont être notées par modification des codes des stems. Pour les mots non significatifs, à priori moins nombreux que les mots-pivots, le stem du mot et l'ensemble des codes seront créés à la main, et entrés à partir de cartes. Comme bien souvent, il s'agit de mots invariables, il n'est pas nécessaire de créer automatiquement le stem, car celui-ci se confond avec le mot lui-même.

VI.3 DETERMINATION "A PRIORI" DES MOTS NON SIGNIFICATIFS

Cette méthode a été utilisée pour déterminer les mots-vides du dictionnaire DIC B 159.

Après examen des titres des documents à retrouver, une liste de mots-vides a été dressée à priori. Cette liste comportait les articles les plus couramment utilisés, et un certain nombre de prépositions, conjonctions et pronoms relatifs.

L'utilisation du dictionnaire DIC B 159 a montré que cette liste de mots non significatifs était incomplète. En effet, pour certain titres, des mots qui n'avaient aucun sens sémantique étaient reconnus comme des mots-pivots. D'autres mots non significatifs, par contre, n'étaient pas trouvés dans le dictionnaire, ce qui n'était pas grave pour la détermination du concept sémantique de la phrase mais pouvait être gênant pour une analyse syntactique, si ces mots non significatifs portaient une information syntactique importante.

Il était donc nécessaire de déterminer plus précisément les mots non pivots contenus dans les titres des documents.

Un programme de K.W.I.C. (Key word in Context) avait été créé pour étudier le contexte des mots dans les titres de documents. Ce programme a été utilisé pour sortir un listing de l'ensemble des mots contenus dans les titres.

A l'aide de ce listing, un ensemble de 56 mots non significatifs a été déterminé. Mais la création d'un listing de KWIC est très longue. Par exemple, pour un ensemble de 150 phrases, comprenant en moyenne, de dix à vingt mots, le temps de calcul en unité centrale est de l'ordre de 20 centièmes d'heure, car les tris et interclassements sont très nombreux dans un KWIC. D'autre part, le temps nécessaire pour dépouiller ce listing est assez long, car bien que le programme ne prenne pas comme pivot les mots contenus dans une liste de mots à exclure, entrée en donnée, le listing correspondant aux 150 phrases comportait plus de 1200 lignes.

L'utilisation du KWIC permet de déterminer complètement les mots non significatifs. Mais cette méthode demande des temps de calculs longs, et l'exploitation manuelle du listing de K.W.I.C. est fastidieuse. Cette méthode n'utilise pas le fait que l'ensemble des titres à analyser appartient à un même domaine. Une autre méthode, utilisant un échantillon de titres tirés au hasard a été testée pour déterminer plus rapidement les mots non significatifs contenus dans les titres de documents.

I.4 DETERMINATION PAR ITERATION DES MOTS NON SIGNIFICATIFS

Pour une itération, on constitue un dictionnaire "DIC-vide" des mots non significatifs, qui résulte de l'examen de l'itération précédente. On détermine ensuite, par tirage au hasard, un échantillon de titres dans l'ensemble des titres. On recherche, dans cet échantillon, les mots non significatifs qui figurent dans le dictionnaire "DIC-vide", et on liste, pour chaque titre, l'ensemble des mots non trouvés à l'aide de "DIC-vide". Ces mots sont supposés être des mots significatifs. L'examen de cet ensemble de mots permet de repérer des mots non significatifs, que l'on ajoute au dictionnaire "DIC-vide" pour l'itération suivante. Le programme de détermination des mots vides sera décrit dans l'annexe 6.

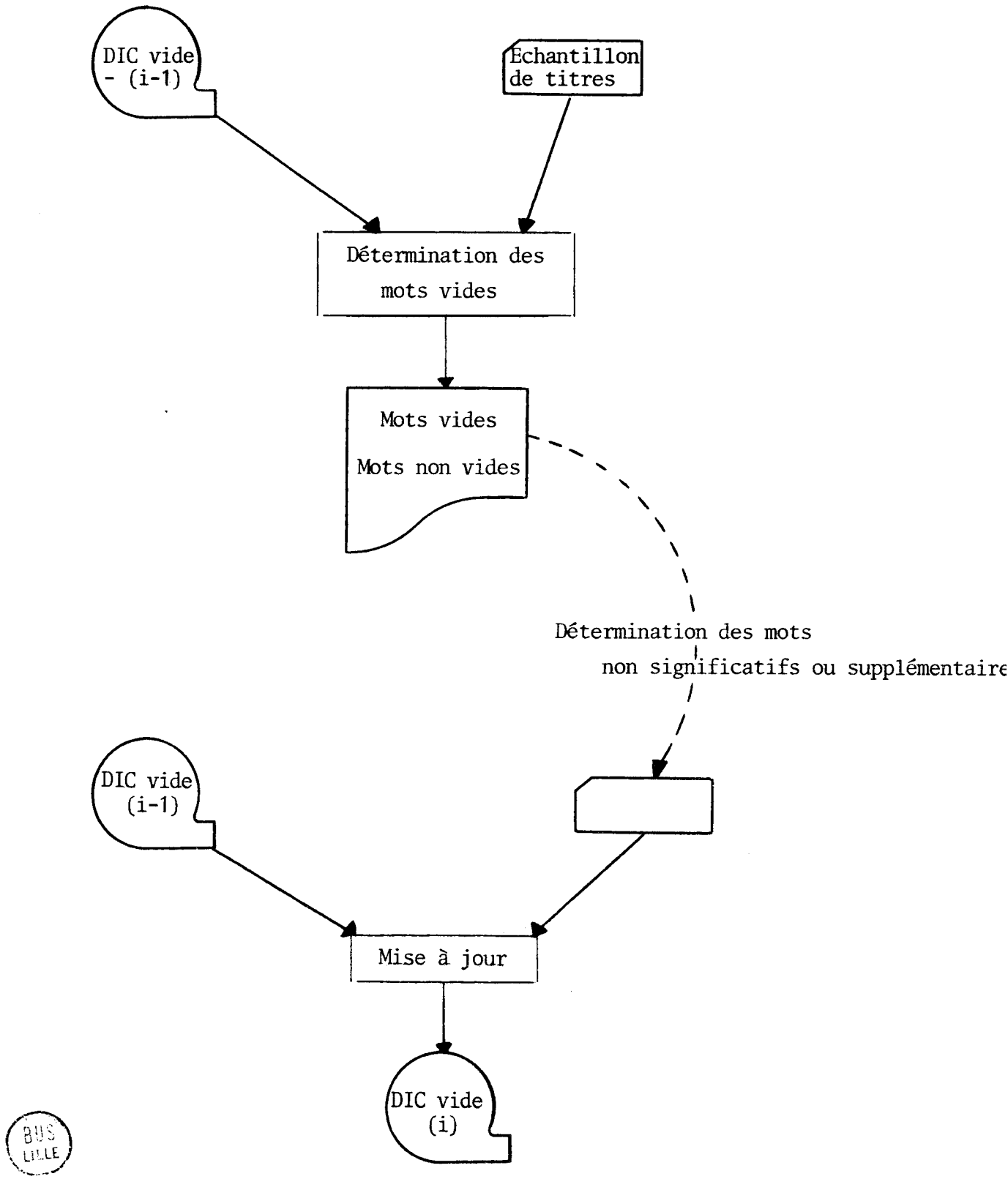
L'ordinogramme correspondant à une itération est décrit plus loin. L'examen de 7 phrases, dans la première itération, a permis de déterminer 33 mots non significatifs sur les 56 déterminés précédemment. Le temps de calcul correspondant a été de 0,056 centième d'heure. Au bout de deux itérations, utilisant chacune 5 titres, cette méthode a permis de déterminer une cinquantaine de mots non significatifs, sur les 56 déterminés par la méthode précédente.

Le temps de calcul nécessaire pour rechercher les mots non significatifs est beaucoup plus faible, car il n'y a plus de listing de K.W.I.C. à créer. Par contre, les premières itérations permettent de déterminer rapidement les mots non pivots les plus fréquents. Par contre, les mots non significatifs qui apparaissent rarement dans les titres sont plus difficiles à reconnaître, ce qui ne gêne pas trop pour la détermination du concept sémantique de phrase, dans la mesure où tous ces concepts appartiennent à un même domaine.

Par contre, pour la création du dictionnaire, la méthode précédente avait l'avantage de donner la liste des mots contenus dans l'ensemble des titres à rechercher, et de donner la fréquence d'apparition de ces mots.

La méthode qui va être exposée maintenant va permettre de déterminer les mots non significatifs à partir de l'ensemble des formes de mots contenues dans un ensemble de phrases, et du décompte de leur occurrence.

DETERMINATION DES MOTS NON SIGNIFICATIFS : ITERATION i



I.5 DETERMINATION DE LA FREQUENCE DES MOTS DANS UN ENSEMBLE DE PHRASES. CHOIX DES MOTS NON PIVOTS

Cette méthode a été utilisée pour créer les mots vides du dictionnaire DIC TS, créé pour l'exemple de système documentaire décrit dans les chapitres qui suivent.

Un programme : "Fréquence mots" lit un ensemble de phrases, et, en utilisant le principe de stockage sous forme arborescente décrit précédemment, dresse l'arbre de l'ensemble des formes de mot trouvées dans les phrases. Le pointeur associé à chaque caractère de fin de chaîne contient le nombre d'occurrences de la chaîne. Le programme sort ensuite l'ensemble des chaînes contenues dans l'arbre ainsi créé, et leur fréquence.

Ce listing va permettre de déterminer les mots non significatifs, soit d'après la fonction grammaticale qui est attachée à ces mots, comme dans la première méthode, soit d'après leur fréquence d'apparition dans les titres. En effet, on constate [2] que pour la détermination du concept sémantique de phrase, des mots qui apparaissent très fréquemment, ou qui ont un sens assez vague du point de vue sémantique n'aident pas à préciser le concept de phrase, et augmentent ainsi les temps de recherche. La détermination de la fréquence des mots permet ainsi de déterminer les mots trop fréquents.

Cette méthode a l'avantage de dresser la liste de l'ensemble des mots contenus dans les titres, ce qui est utile, comme le montrera le chapitre qui suit, pour la création du dictionnaire. Les temps de calculs nécessaires pour créer le listing de ces mots sont bien meilleurs que ceux obtenus par un KWIC : pour rechercher l'occurrence des différentes formes de mots dans 704 phrases, le temps d'unité centrale n'a été que de 2,76 centièmes d'heure, pour 975 formes de mots, avec, en moyenne, 10 occurrences par forme. La connaissance du nombre d'occurrences de chaque mot facilite la mise en place du fichier documentaire utilisant la technique du fichier inversé, comme le montrera le chapitre VII.

VI.6 C O N C L U S I O N

La dernière méthode exposée a été choisie pour déterminer les mots vides de DIC TS. 49 mots non significatifs ont été ainsi trouvés. Le chapitre qui suit va montrer que cette méthode facilite la mise en place d'un fichier documentaire inversé, et la répartition des articles de ce fichier, afin de diminuer les temps de recherche dans ce fichier.

C H A P I T R E VII

E X E M P L E de

D O C U M E N T A T I O N A U T O M A T I Q U E

II.1 INTRODUCTION

Ce chapitre est consacré à la description d'un système de documentation automatique, qui utilise un langage interactif pour formuler une demande, ou effectuer des opérations logiques sur des résultats de demandes précédentes, ou d'opérations logiques précédentes.

Les documents repérés dans l'exemple exposé, sont les "titres" décrivant l'ensemble des phénomènes statistiques et économiques contenus dans le document "projet de loi de finances" réalisé par la D.A.T.A.R., et l'INSEE [8]. Le programme de documentation automatique cité ici a été transcrit pour un système de Time-Sharing commercial. Il a été intégré dans un système plus important [7], qui comporte une partie de traitement des informations numériques repérées par les titres, associée à la partie documentaire de recherche de ces titres.

Les titres correspondant à une demande seront obtenus par l'exploitation d'un fichier inversé qui, pour chaque mot caractéristique associe l'ensemble des identificateurs des titres qui contiennent ce mot-clé. Le paragraphe qui suit va rappeler le principe du fichier inversé. Par la suite, on va montrer qu'il est possible de créer, et de mettre à jour automatiquement le fichier inversé à l'aide du dictionnaire.

VII.2 PRINCIPE DU FICHIER INVERSE

Une étude comparative sur les organisations de fichiers en documentation automatique sortirait du cadre de cet exposé [2], [6], [7]. Il est souhaitable, cependant, de préciser la notion de fichier inversé, et l'adaptation qui en sera faite dans l'exemple de documentation décrit dans ce chapitre.

Le fichier inversé est basé sur l'arrangement du fichier des informations par l'intermédiaire d'identificateurs d'information : mots-clés ou termes d'index. L'organisation est la suivante : tous les éléments indexés par le mot-clé A sont lestés en premier, suivis par ceux identifiés par B, puis ceux identifiés par C, et ainsi de suite, par ordre des mots-clés. Chaque élément d'information figure donc dans le fichier pour chaque mot-clé auquel il appartient.

Pour retrouver l'information stockée dans un tel fichier, il suffit d'extraire du fichier les sections qui correspondent aux termes d'index utilisés pour formuler la demande. L'identification des éléments correspondant à la demande est obtenue par comparaison des sections du fichier correspondant aux termes d'index trouvés.

Un tel fichier est assez simple à mettre en oeuvre. La création, et la maintenance du fichier peuvent être automatisés. Pour des fichiers de ce type, dont les informations sont stables, ce qui est le cas dans l'exemple traité ici, les temps de mise à jour ne sont pas très élevés. Par contre, la structure de fichier inversé ne convient pas pour des fichiers à fort taux de modification

Cette structure s'adapte bien à une utilisation "on-line". L'exemple exposé ici va permettre d'utiliser immédiatement les résultats donnés par le fichier inversé pour préciser les résultats obtenus par les questions précédentes, ou effectuer des opérations logiques sur ces résultats à l'aide d'un langage simple interactif.

Adaptation du principe du fichier inversé à l'exemple traité

Les titres à rechercher sont repérés par un identificateur numérique, qui figurera comme information élémentaire dans le fichier inversé.

Dans le dictionnaire, les stems des mots significatifs sont associés à un ensemble de codes. Dans la partie sémantique de ces codes (cf. chap III.4), le concept correspondant au mot est repéré par un numéro de concept. Ce numéro de concept va servir d'index au fichier inversé.

Par exemple :

Soient les titres : titre repéré par l'identificateur I1, et contenant les mots-pivots dont les codes de concept sont M1, M2

" " " " I2, et contenant M2, M3.

Le fichier inversé correspondant aura la structure :

Index	Article
M1	I1
M2	I1 I2
M3	I2

Le résultat de l'exploitation du fichier inversé sera un ensemble d'identificateurs de titres, et non les titres eux-mêmes, ce qui permet une exploitation beaucoup plus rapide du fichier, et un encombrement plus faible. On constate, en effet, que lors de l'utilisation du programme documentaire, l'opération demandant la sortie des titres correspondant à un ensemble d'identificateurs intervient généralement après un ensemble d'opérations logiques, destinées à préciser la demande, et utilisant des ensembles d'identificateurs.

VII.3 CREATION DES FICHIERS UTILISES PAR LE SYSTEME

Ce paragraphe va étudier la création, et la mise à jour du dictionnaire, puis montrer l'utilisation de ce dictionnaire pour la création du fichier inversé.

VII.3.1 Dictionnaire

L'ensemble des titres traités par le système documentaire est ici un ensemble fixe. Le dictionnaire a donc été créé en une seule fois.

Les mots-vides ont été déterminés par la méthode décrite dans le chapitre précédent, et utilisant la fréquence des mots dans l'ensemble des titres. Le listing obtenu va permettre de déterminer les mots-vides et en même temps à préparer les données nécessaires pour le programme de création des stems, et les données pour les mots-vides.

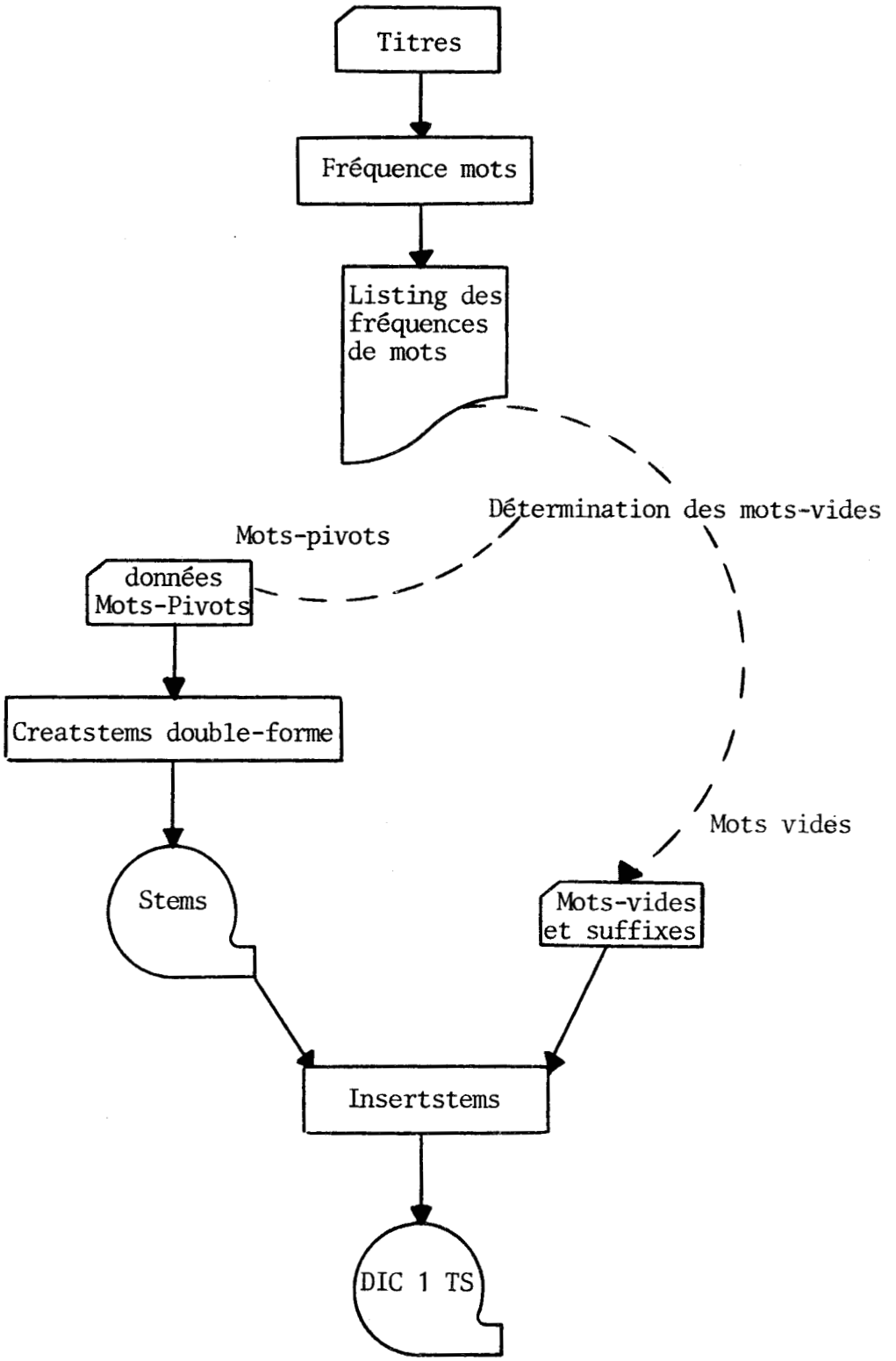
Pour accélérer les temps de recherche dans le fichier inversé, le listing de fréquence des mots a été également utilisé pour déterminer les mots-pivots les plus fréquents. Ceux-ci ont été entrés en premier dans le dictionnaire, afin que leur mémoire de concept attribué par le système soit faible (ils seront ainsi les index des premiers articles du fichier inversé).

Les synonymies, et les concepts non élémentaires (cf. Chap III.3) seront référés à l'aide du listing de sortie des mots caractéristiques.

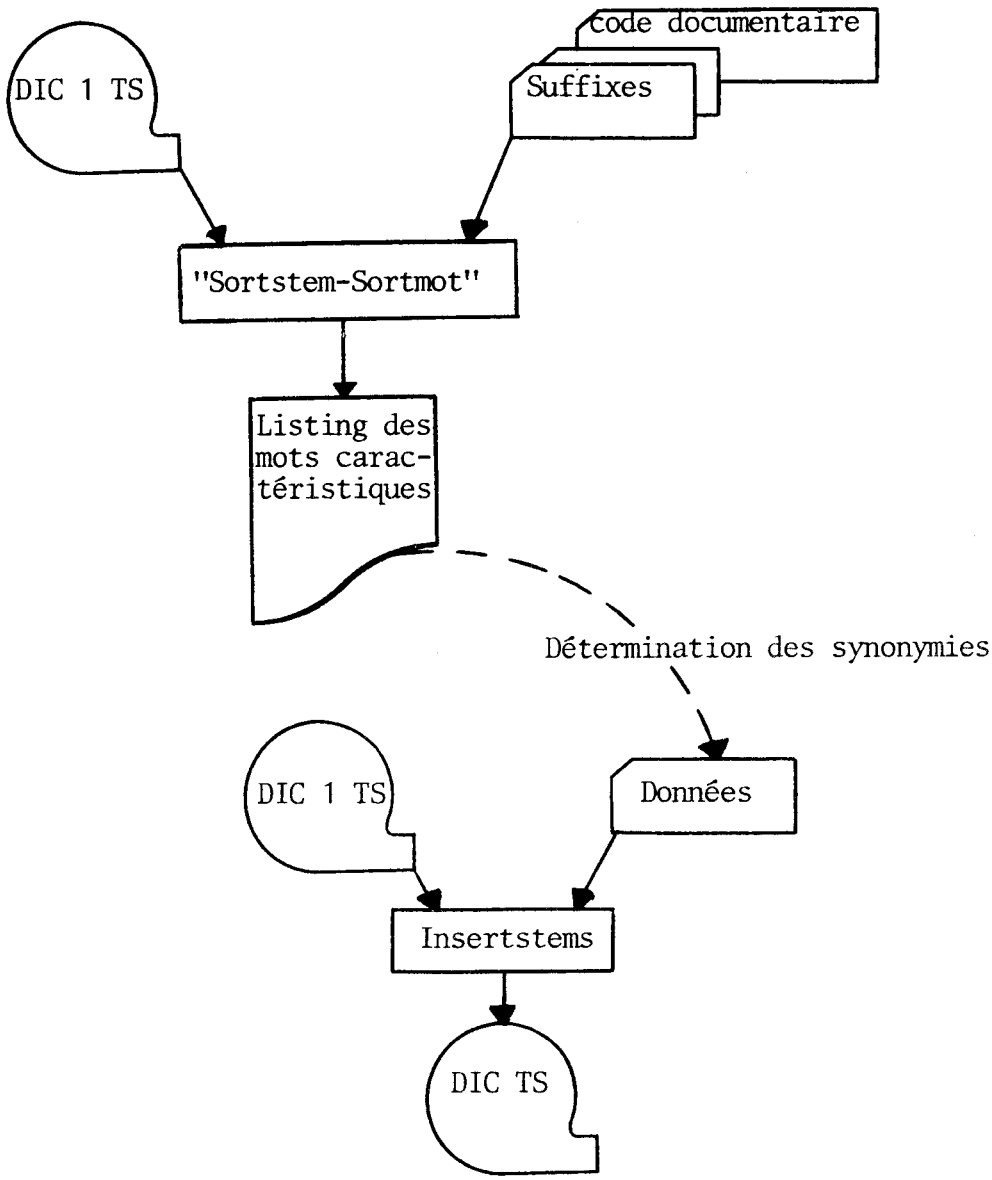
La création du dictionnaire est résumée par un ordinogramme.

Si d'autres titres sont ajoutés par la suite à l'ensemble initial, on peut utiliser le programme "rechphrase" pour déterminer les mots qui ne figurent pas dans le dictionnaire. On peut ensuite introduire ces mots nouveaux par le programme "insertstems", soit en déterminant les stems par un des deux programmes proposés (cf. chap II), soit en les entrant directement par carte, après détermination de ces stems manuellement.

ORDINOGRAMME DE CREATION DU DICTIONNAIRE



CREATION DU DICTIONNAIRE (suite)



VII.3.2 Fichier inversé

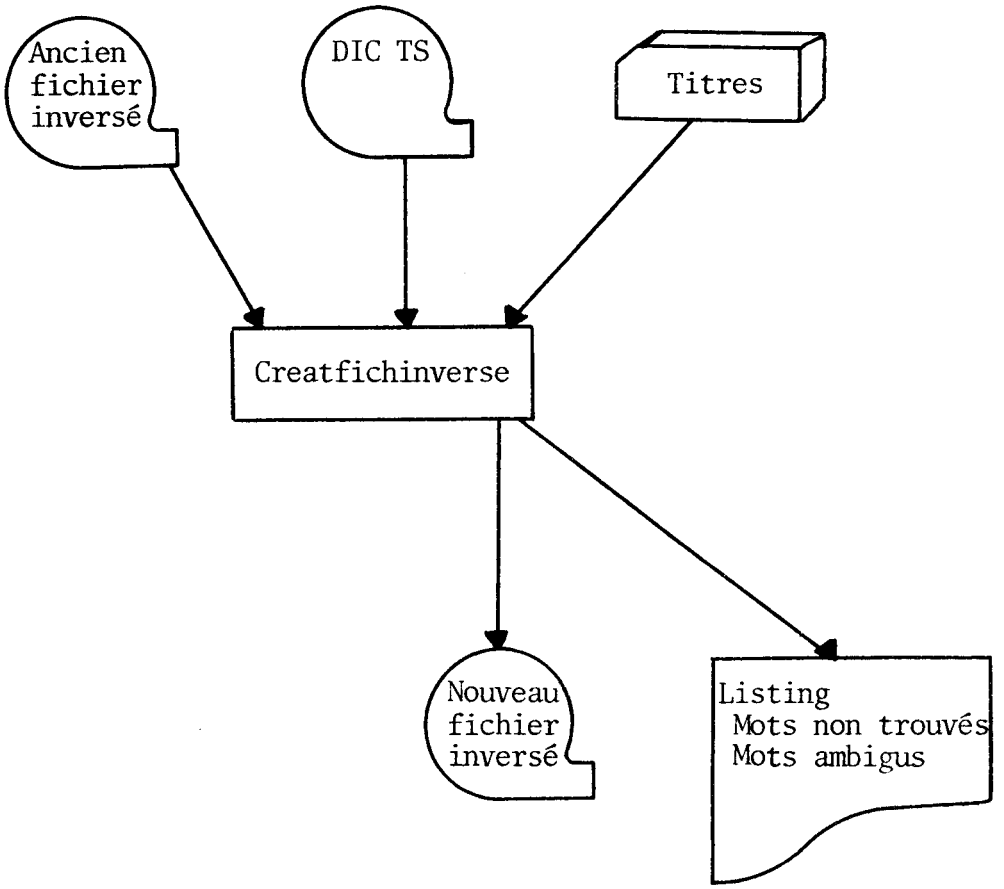
Le problème est le même pour la création et l'ajout de nouveaux titres au fichier inversé. L'ordinogramme correspondant montre que le dictionnaire est utilisé pour la maintenance de ce fichier. Cela permet, en particulier, de vérifier que tous les mots caractéristiques contenus dans les titres figurent bien dans le dictionnaire, en utilisant le listing des anomalies. Il n'est pas nécessaire, en effet, que tous les mots non significatifs figurent dans le dictionnaire, car il n'y a pas d'analyse syntactique dans l'exemple donné ici.

Les titres sont repérés par un identificateur numérique qui figure dans les cartes de données des titres. Le dictionnaire va permettre, pour chacun des mots-pivots d'un titre donné, de déterminer le, ou les numéros de concept qui lui sont associés. Si plusieurs numéros de concept correspondent au même mot, il s'agit d'un cas ambigu, ou d'un concept non-élémentaire. On a choisi, dans ce cas, de sortir sur un listing les renseignements trouvés à l'aide du dictionnaire, et de choisir le, ou les codes pertinents ; les autres codes surchargeraient le fichier inversé, et risqueraient de fausser, ou d'alourdir les recherches. Dans les autres cas, le dictionnaire va déterminer les couples associés : identificateur de titre $\leftrightarrow$ code de concept de mot-pivot. Le code de concept de mot-pivot sera la clé de l'article du fichier inversé ; l'identificateur de titre sera un des éléments de l'article du fichier inversé (cf. annexe 6). Ainsi le dictionnaire permet, pour chaque nouveau titre, de déterminer les éléments nécessaires pour la mise à jour du fichier inversé.

La description du fichier inversé, l'organigramme de création sont donnés dans l'annexe 6, ainsi que le temps CPU utilisé dans l'exemple traité.

Le paragraphe qui suit va montrer que le dictionnaire va être réutilisé, pour rechercher les titres répondant à la demande, conjointement avec le fichier inversé, de la même façon que pour la création du fichier inversé, ce qui permet d'éviter des erreurs dans les recherches.

ORDINOGRAMME CREATION - AJOUT AU FICHIER INVERSE



VII.3.3 Fichier des titres

Ce fichier permet, connaissant un identificateur de titre, de retrouver le libellé qui lui correspond.

VII.4 EXEMPLE DE DOCUMENTATION AUTOMATIQUE

Le système de documentation exposé dans ce paragraphe va rechercher parmi l'ensemble des "titres" repérés à l'aide du fichier inversé, ceux qui répondent à une demande formulée par un utilisateur. Un langage simple va permettre ensuite de faire des opérations logiques sur les résultats de demandes ou d'opérations logiques précédentes.

VII.4.1 Principe de la recherche des titres qui correspondent à une demande

Le dictionnaire permet, dans une demande, de reconnaître les mots significatifs. On va donc obtenir l'ensemble des codes de concepts qui correspondent à ces mots-pivots.

On a choisi, dans l'exemple exposé, de rechercher les titres qui contiennei tous les mots-pivots de la demande reconnus par le dictionnaire.

Dans les systèmes fonctionnant en batch processing, il est préférable de rechercher les documents intéressants dans un listing de sortie plus important, plutôt que d'attendre un autre traitement de questions par lot pour préciser la demande. Ces systèmes permettent de retrouver des documents qui ne contiennei qu'une partie des mots significatifs de la question [2]. Dans ce cas, l'ensembl des documents produits en sortie est plus important.

Dans l'exemple exposé ici, le langage interactif va permettre d'orienter la recherche "on-line", suivant les résultats déjà obtenus.

Pour accélérer le traitement des transactions, il est souhaitable de limit le volume des sorties.

L'exploitation du fichier inversé va permettre de déterminer les titres qui correspondent à une demande.

Par exemple, soit une question comportant trois mots-pivots, dont les codes sont M1, M2, M3. On va rechercher les articles du fichier inversé correspondant aux index M1, M2, M3. Les articles contiennent des identificateurs de titre (I_i), classés par ordre croissant suivant la configuration.

Article M1 :	I1	I2	I3	I5	I7		
Article M2 :	I2	I5	I7	I9			
Article M3 :	I2	I4	I5	I12	I15	I18	I30

Les titres correspondant à cette demande seront ceux dont l'identificateur figurera, pour l'exemple donné, à la fois dans les articles M1, M2 et M3. On trouvera, ici, deux titres repérées par I2 et I5. On obtiendra donc les titres pertinents en effectuant une intersection logique sur le contenu des articles du fichier inversé qui correspondent aux mots-pivots reconnus dans la question, dans le cas où ceux-ci correspondent chacun à un code unique.

Si un mot-pivot correspond à un cas ambigu, plusieurs codes de concept lui seront associés par le dictionnaire. Par exemple, un mot-pivot sera associé aux codes M4, M5. Les titres qui répondent à la demande sont ceux qui contiennent un des cas possibles du mot ambigu. Ils seront donc repérés, dans l'exemple donné, par les identificateurs qui figurent dans l'article indexé par M4, ou dans l'article indexé par M5. Le résultat sera alors l'union logique des deux articles indexés par M4 et M5.

On déterminera donc les titres répondant à une question en faisant l'union des cas possibles pour un mot ambigu, pour créer l'article correspondant à ce mot, et en faisant l'intersection logique de tous les articles des mots-pivots.

Par exemple, soit une demande correspondant à 4 mots-pivots reconnus par le dictionnaire :

- trois mots-pivots non ambigus, dont les codes de concept seront M1, M2, M3
- un mot-pivot ambigu, correspondant à deux codes : M4, M5.

On suppose que les articles du fichier inversé ont la structure suivante :

Article M1 :	I1	I2	I3	I5	I7		
Article M2 :	I2	I5	I7	I9			
Article M3 :	I2	I4	I5	I12	I15	I18	I30
Article M4 :	I2	I5	I7				
Article M5 :	I2	I9	I12	I30			

On va donc effectuer successivement :

A = M4 $\cup$ M5	I2	I5	I7	I9	I12	I30

R1 = A $\cap$ M1	I2	I5	I7

R2 = R1 $\cap$ M2	I2	I5	I7

R3 = R2 $\cap$ M3	I2	I5

Donc, deux titres seulement, repérés par les identificateurs I2 et I5 répondent à la demande de l'exemple.

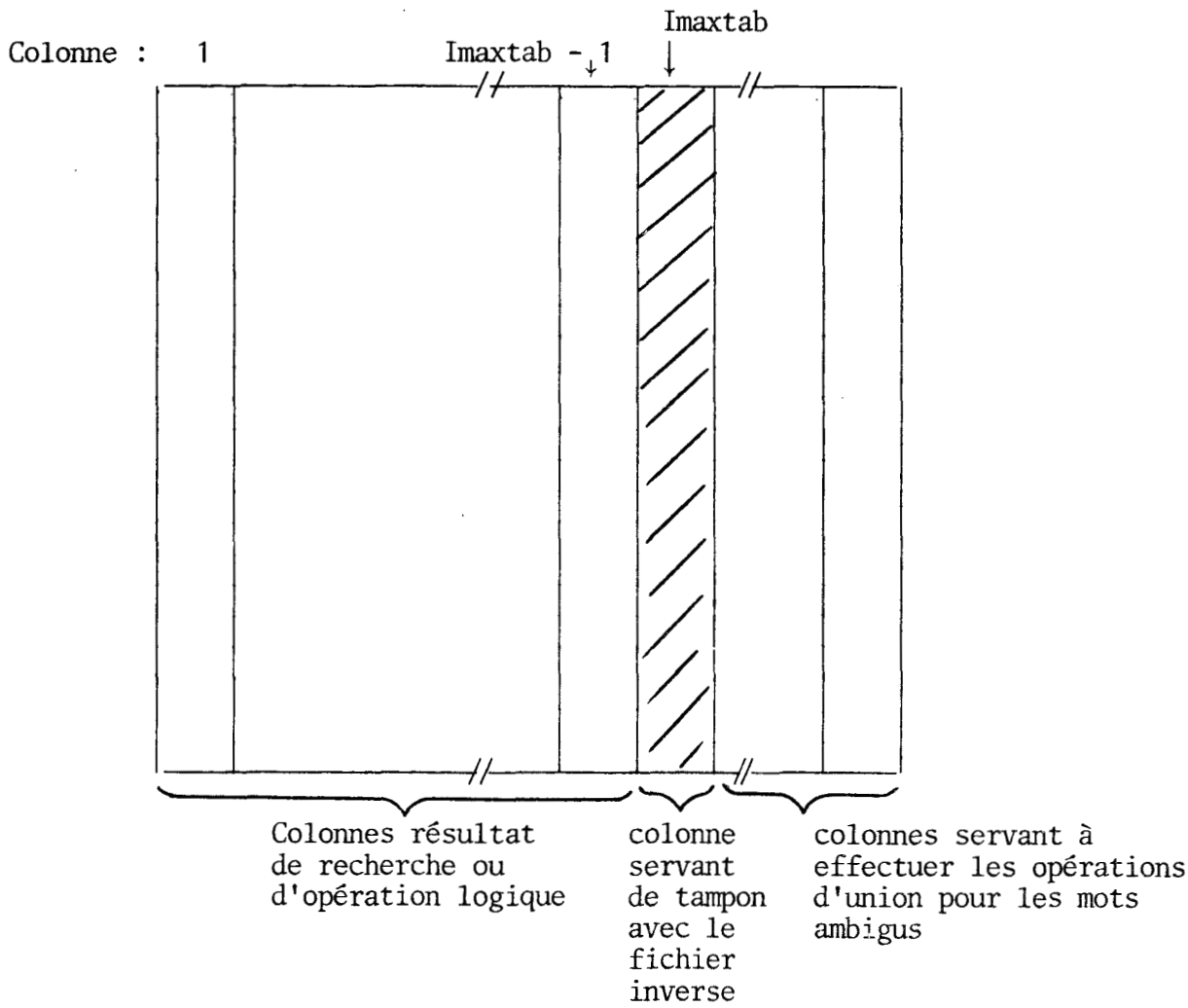
Si un des résultats intermédiaires Ri ne donne aucun identificateur possibl il est inutile de continuer les opérations logiques : aucun titre ne répond à la demande.

Ce principe de recherche va être appliqué par le système de documentation décrit dans le paragraphe qui suit.

VII.4.2 Langage de recherche des titres

L'utilisateur dispose d'une zone de travail constituée par un certain nombre de colonnes qui vont recevoir les résultats de recherches, ou d'opération logiques. Le langage de documentation comprend une dizaine d'opérations, résumée dans un tableau, et dont les paramètres correspondent à des numéros de colonne d la zone de travail.

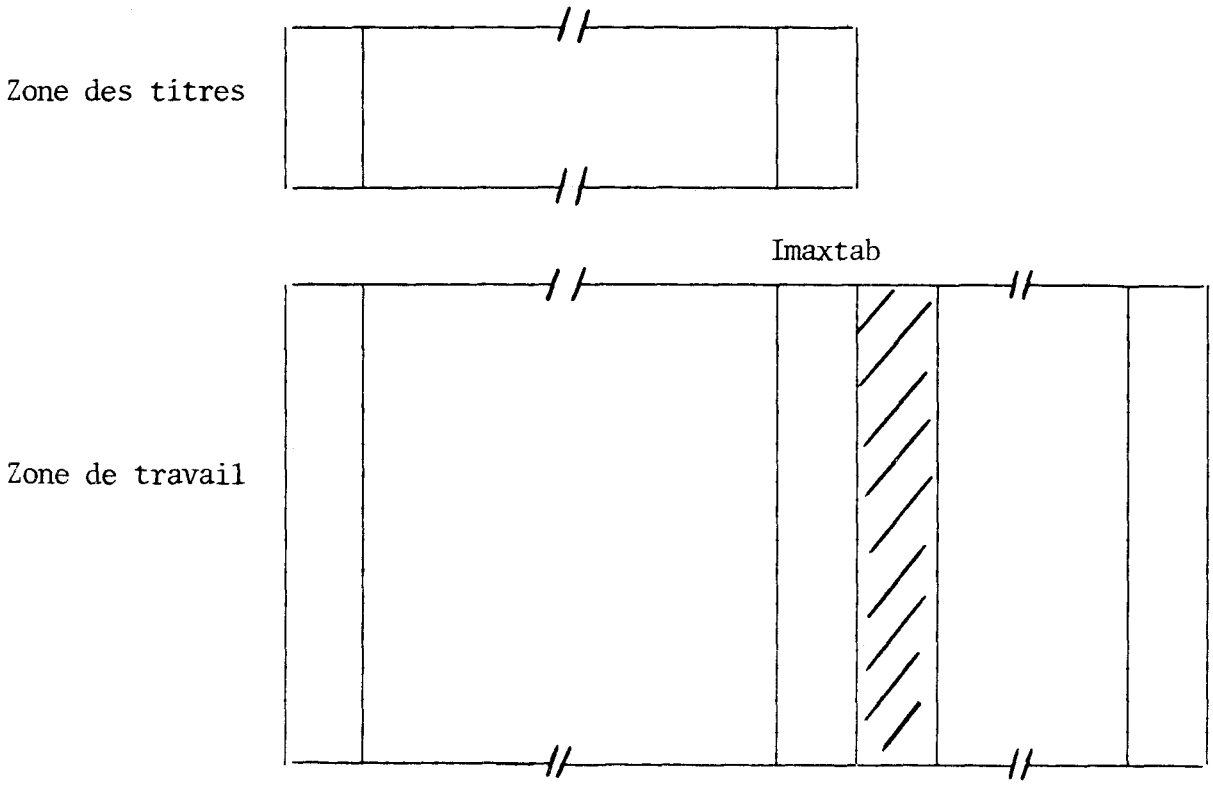
Cette zone a la structure suivante :



Les dimensions de la zone peuvent être modifiées lors de l'initialisation du programme. La colonne tampon sert pour extraire les articles du fichier inverse ; elle a été incluse dans cette zone pour ne pas différencier les calculs qui portent sur cette colonne.

Le nombre de colonnes servant à effectuer les opérations d'union pour les mots ambigus peut être donné à l'initialisation du programme. Il correspond au nombre maximum de mots-pivots ambigus que l'on accepte dans une demande. Si on trouve plus de mots ambigus que la limite fixée, il y a abandon de la demande avec sortie d'un message d'erreur.

Une autre zone est associée à cette zone de travail. Elle va recevoir un titre construit par le système pour chaque transaction. Cela permettra à l'utilisateur de vérifier que le contenu de la colonne correspond bien au contenu supposé.



La dimension des colonnes a été calculée de façon qu'un article du fichier inverse tienne dans une colonne, ce qui explique que les mots-pivots trés fréquents, donc peu significatifs ont été éliminés, comme dans d'autres systèmes [2]

Les opérations qui constituent le langage documentaire sont présentées dans le tableau des opérations de documentation. Lorsqu'un paramètre figure entre parenthèse, il a une valeur implicite égale à celle du pointeur automatique de colonne. Ce pointeur automatique est incrémenté de une unité chaque fois qu'un des paramètres mis entre parenthèses est omis. Cela permet d'accumuler les résultats sans avoir à gérer la zone de travail. Au contraire, la présence de ces paramètres permet de réutiliser des colonnes déjà occupées, et qui contiennen des résultats devenus sans valeur.

Pour augmenter la vitesse d'exécution, le système travaille sur les identificateurs de titres. Une colonne contiendra donc les identificateurs de titres qui correspondent à une transaction. On trouvera également le nombre d'identificateurs dans la colonne en tête de cette colonne.

Par exemple, soit la séquence de transactions suivante :

Numéro de la transaction	Nature de la transaction	Numéro de la colonne résultat
1	question	1
2	question	2
3	ET 1, 2	3
4	OU 1, 2	4
5	ET 3, 4, 2	2

La colonne 2 a été réutilisée pour y stocker le résultat de l'opération ET sur les colonnes 3 et 4.

Généralement, on utilise le pointeur automatique pour allouer les colonnes, car l'utilisation du système est beaucoup plus simple. Comme le nombre de colonnes est limité, on peut être amené à les réarranger au bout d'un certain nombre de transactions pour ne garder que les résultats importants.

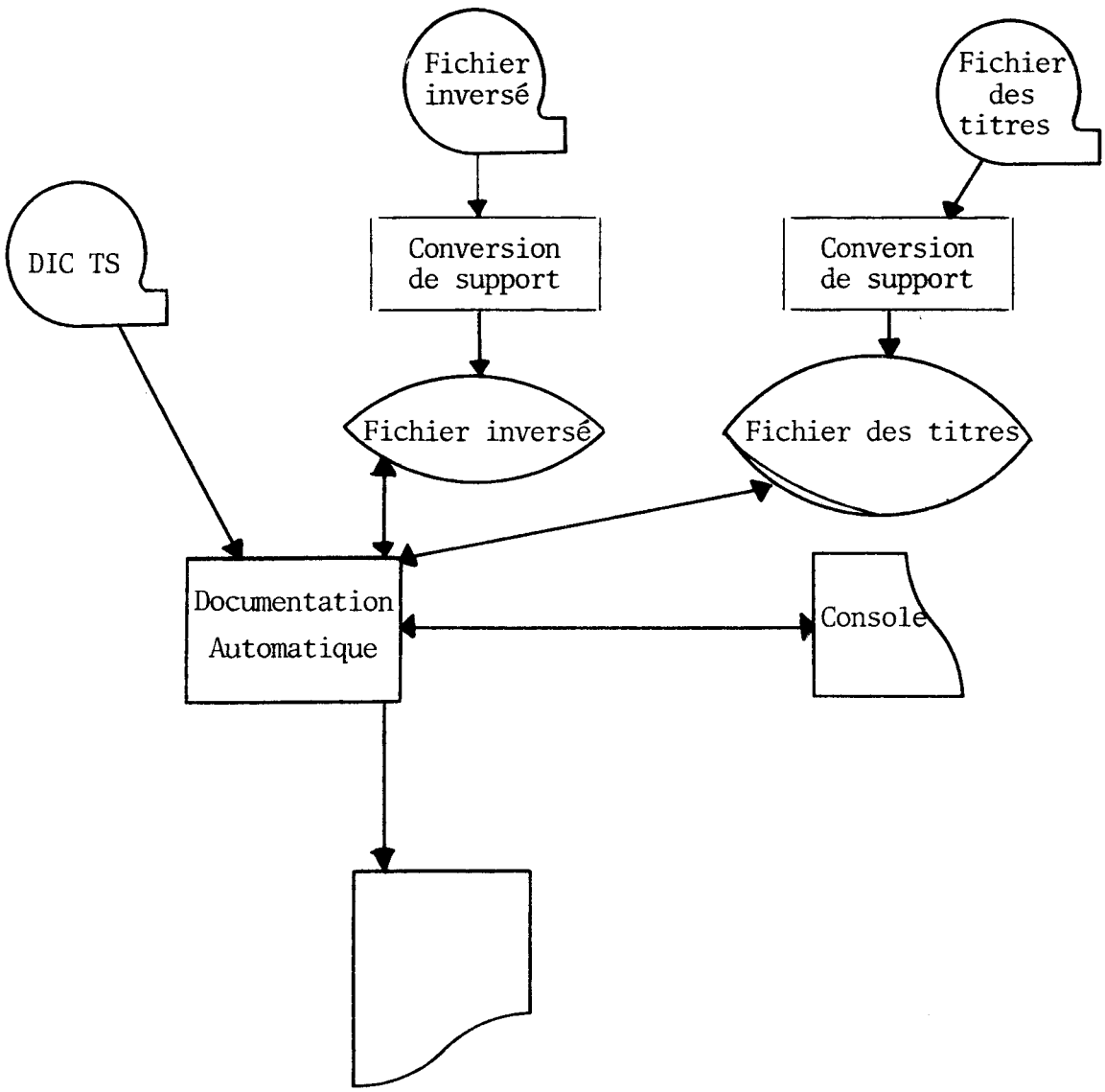
L'utilisation du système est résumée dans l'organigramme d'utilisation du système, qui est donné plus loin.

Tableau des Opérations de documentation

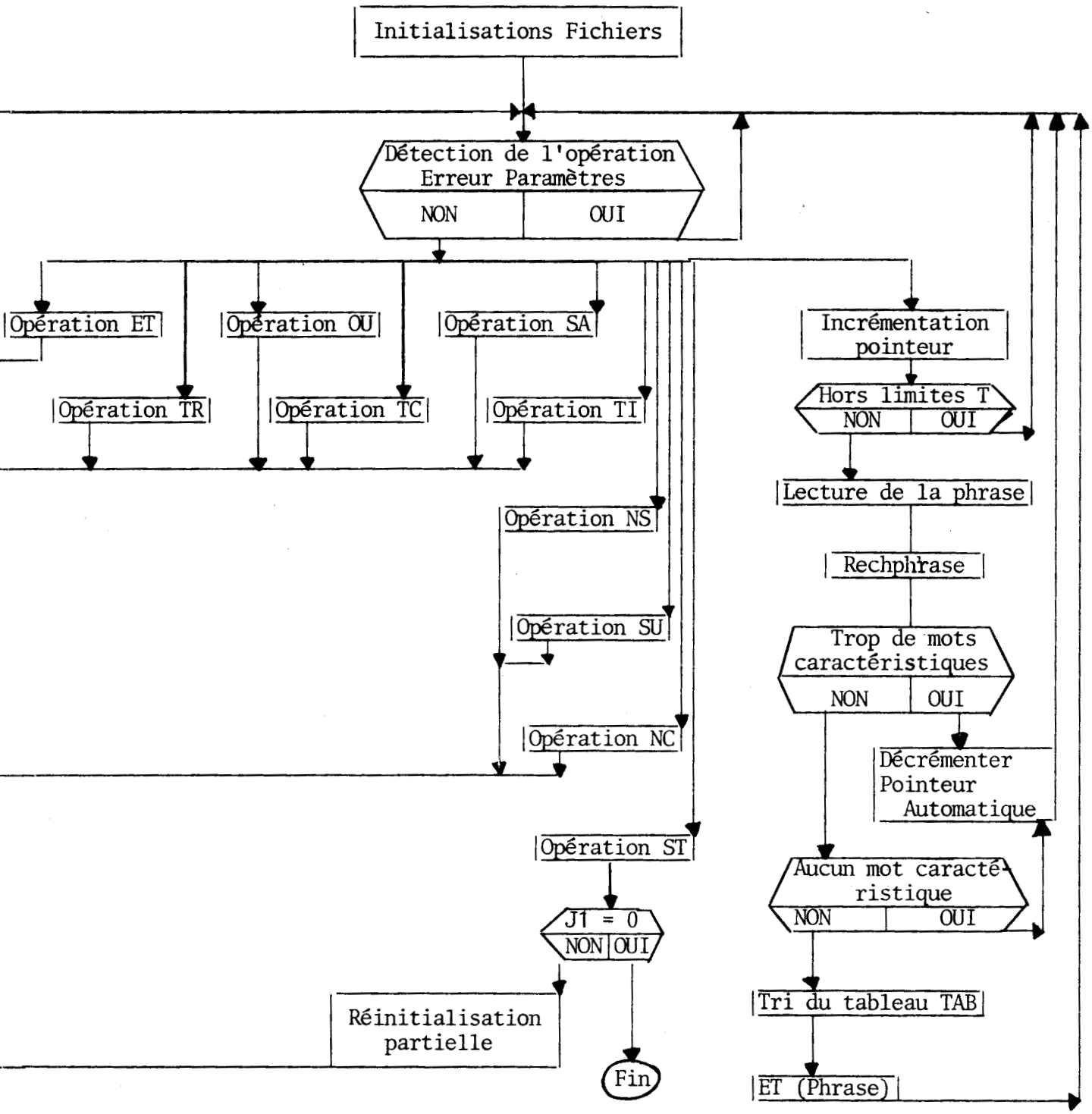
(J_i) : indique que le paramètre est facultatif. Dans le cas où le paramètre est omis, la valeur implicite est celle du pointeur du système.

Opération	Paramètres	Signification
ET	J1, J2, (J3)	J3 = ET (J1,J2)
OU	J1, J2, (J3)	J3 = OU (J1,J2)
SA	J1, J2, (J3)	J3 = J1 SAUF J2
NS	(J1)	sortir les identificateur de titres de la colonne J1
TI	(J1)	sortir les libellés des titres de la colonne J1
TC	(J1), (J2)	lister les titres des colonnes J1 à J2
SU	⌴	supprimer la transaction précédente
TR	J1, J2	transférer la colonne J1 dans la colonne J2
ST	J1	J1 = 0 arrêt des opérations J1 ≠ 0 réinitialisation des colonnes à partir de la colonne J1
NC	⌴	sortir le contenu du pointeur automatique de colonne
		Recherche des titres correspondant à la demande

ORDINOGRAMME GENERAL DOCUMENTATION AUTOMATIQUE



ORGANIGRAMME D'UTILISATION DU SYSTEME



VII.4.3 Description des opérations du langage

Le système reconnait une dizaine d'opérations. Celles-ci sont repérées par deux caractères, suivis d'un espace, et de 1 ou plusieurs paramètres numériques, suivant les cas, séparés par des virgules. Il y a détection d'erreur si le forma n'est pas respecté. Il y a également sortie d'un message d'erreur si une des valeurs données pour un des paramètres n'est pas comprise dans l'intervalle correspondant aux colonnes résultat de la zone de travail.

Recherche des identificateurs de titres correspondant à une demande

Le système lance une opération de recherche si les trois premiers caractère de la demande ne correspondent pas aux 2 caractères identificateurs d'opération suivi d'un blanc. La recherche s'effectue suivant le principe exposé précédemmen Il y a une opération préalable de réunion pour les articles du fichier inversé qui correspondent aux diverses possibilités d'un mot ambigu, puis intersection logique de l'ensemble des articles correspondant aux mots-pivots.

Les codes des mots-pivots sont préalablement triés en ordre croissant. Cela permet d'éliminer les codes des mots non ambigus qui apparaissent plusieurs fois dans la même demande car on n'analyse pas l'ordre relatif des mots.

Un même mot peut être associé à plusieurs codes sans que ce soit un mot ambigu. Il s'agit d'un cas de synonymie, par exemple, où un mot français, ou une abréviation est synonyme d'un ensemble de mots. Par exemple le mot *E.D.F.* qui correspond à l'ensemble des mots-pivots : "*Electricité de France*". Dans ce cas, les titres qui répondent à la demande seront ceux qui contiendront à la fois les trois mots-pivots. Il faudra faire l'opération logique d'intersection sur les trois articles qui correspondent dans le fichier inversé, et non l'opération d'union, comme dans le cas d'un mot ambigu. Le code de concept non élémentaire (cf. chap IV) permet de distinguer ce cas de celui d'un mot ambigu.

Le résultat de l'opération est placé dans une colonne déterminée automatiquement par le système.

Dans la description des opérations qui suivent, un paramètre noté entre parenthèses indique qu'il est facultatif. Dans le cas où il est omis, la valeur implicite est celle du pointeur du système.

Opération : ET J1, J2, (J3)

Cette opération réalise l'intersection logique sur les identificateurs de titres des deux colonnes J1 et J2 (J1 et J2 ≠ 0). Le nombre de titres obtenus est donné en réponse à l'utilisateur. On peut avoir J1 = J2, ce qui manque un peu d'intérêt, et également J3 = J1 ou J3 = J2.

Opération : OU J1, J2, (J3)

Cette opération réalise l'union logique sur les identificateurs de titres contenus dans les deux colonnes J1 et J2 (J1 et J2 ≠ 0). Le nombre d'identificateurs obtenus est donné en réponse à l'utilisateur. Le résultat est placé dans la colonne J3 qui peut être attribuée par le système si le paramètre est omis. On peut avoir J3 = J1 ou J3 = J2.

Opération : SAU J1, J2, (J3)

La colonne J3 va contenir les identificateurs de titres contenus dans J1 et qui ne sont pas contenus dans J2. L'utilisateur recevra en réponse le nombre d'identificateurs de titres de la colonne J3. Il faut, pour que l'opération soit exécutable,que J3 ≠ J2.

Opération : NS (J1)

Permet de lister le contenu de la colonne J1, et le titre attribué par le système à cette colonne.

Opération : TI (J1)

Permet de sortir sur l'imprimante le libellé des titres qui correspondent aux identificateurs de titres contenus dans la colonne J1.

Opération:TC (J1, J2)

Si J2 est omis, il a la valeur de J1. Cette opération permet de lister sur la console les titres donnés par le système pour les colonnes J1 à J2.

Opération : SU

Permet de décrémenter de une unité le pointeur de colonne automatique. La colonne correspondante est remise à zéro, ainsi que sa zone de titre.

Opération : TR J1, J2

Cette opération transfère le contenu de la colonne J1 dans la colonne J2, et transfère la zone titre de J1 dans celle de J2.

Opération : ST (J1)

Si J1 est omis, il a la valeur 0 par défaut.

Si J1 = 0, ST 0 arrête la série des transactions et interrompt le programme

Si J1 $\neq$ 0, les colonnes de la zone de travail, à partir de J1 sont remises à zéro, et le pointeur automatique de colonne est forcé à la valeur J1. Cela permet, par exemple après un réarrangement de certaines colonnes par des opérateurs TR de disposer, à nouveau,de place dans la zone de travail pour effectuer d'autres transactions avec le système.

Opération : NC

Donne à l'utilisateur la valeur actuelle du pointeur automatique de colonne.

Ce langage est donc un langage simple, assez facile à apprendre et à mettre en oeuvre. Il peut être étendu en fonction des besoins ; d'ailleurs, certaines opérations ont été introduites après des essais d'utilisation du système par des personnes n'ayant aucune expérience en informatique [7].

CONCLUSION

Les comparaisons faites sur les différents dictionnaires réalisés pour cette étude ont montré que la méthode de représentation était assez performante, tant du point de vue du volume occupé que des temps de recherche.

La méthode de recherche choisie permet de ne pas tenir compte du contexte déjà existant dans le thesaurus lors de l'introduction d'un nouveau stem. Ceci a permis de choisir des stems plus courts, et d'augmenter la performance de la méthode de représentation en mémoire. L'opération de détermination des stems de mots a été automatisée, sans nécessiter des volumes de données importants. Une des méthodes de détermination permet même l'élimination de certaines fautes dûes aux erreurs dans l'introduction des données.

Les classes de suffixes permettant d'associer à un stem une classe de suffixes par l'intermédiaire d'un ensemble de codes syntactiques qui précisent la fonction grammaticale du mot. Il est possible ainsi d'associer aux différentes formes orthographiques d'un même mot un stem unique, et de déterminer ce stem lorsqu'on cherchera à interpréter dans une question une des formes du mot, à l'aide du dictionnaire.

Dans l'exemple de système documentaire exposé ici, on peut remarquer que le dictionnaire ne sert uniquement qu'à déterminer les mots-pivots contenus dans une question, pour utiliser un fichier inversé sur ces mots-pivots. Cette procédure est assez rigide, car elle ne tient pas compte de l'ordre des mots, de la structure de la demande, et dans une certaine mesure, des synonymies. L'utilisation d'un langage interactif pour gérer les résultats de demandes permet de rendre plus souple l'utilisation du système.

Mais on peut remarquer, bien que les résultats obtenus sur l'exemple donné semblent suffisants, que toutes les possibilités offertes par le thesaurus ne sont pas utilisées : en effet, celui-ci permet de retrouver les codes nécessaires pour réaliser une analyse syntactique de la demande. Une telle analyse permettant

de déterminer de façon plus précise les concepts contenus dans une phrase, en tenant compte de l'ordre des mots et de leur fonction grammaticale.

Les documents repérés par le système documentaire appartenaient tous au même domaine : celui des statistiques et études économiques. La détermination des synonymies et des hiérarchies a été relativement simple, mais dans le cas où les documents appartiennent à des domaines différents, la méthode doit être complétée, et perfectionnée. On peut envisager, par exemple, l'emploi de méthodes définissant des proximités, ou des corrélations. Il serait alors souhaitable d'étudier un ensemble qui permettrait de définir des synonymies ou des hiérarchies entre mots ou groupes de mots, selon le contexte qui les entoure.

A N N E X E 1

1 - Suffixes caractéristiques de noms

2 - Suffixes caractéristiques d'adjectifs

3 - Classes de suffixes de noms

4 - Classes de suffixes d'adjectifs

5 - Classes de suffixes d'adverbes

6 - Suffixes caractéristiques des classes de verbes

7 - Ensemble des classes de suffixes ambiguës

1 - SUFFIXES CARACTERISTIQUES DE NOMS

ace	*populace*
ade	*reculade*
age	*feuillage*
aie	*roseraie*
ail	*éventail*
aille	*mangeaille*
ain - aine	*quatrain, dizaine*
aire	*légataire*
aison	*démangeaison*
ance	*délivrance*
ande	*multiplicande*
ard	*chauffard*
asse	*filasse*
at	*externat*
ateur	*dessinateur*
ation	*désignation*
âtre	*marâtre*
ature	*armature*
aud	*lourdaud*
cule	*animalcule*
eau	*chevreau*
ée	*assiettée*
éen	*lycéen*
elle	*ruelle*
ement	*règlement*
er	*vacher*
erie	*cajolerie*

eron	*forgeron*
esse	*sagesse*
et	*garçonnet*
eté	*netteté*
ette	*fillette*
ie	*jalousie*
ien	*collegien*
ier	*laitier*
iere	*laitière*
ille	*brindille*
illon	*négrillon*
is	*taillis*
ise	*franchise*
isme	*vocalisme*
ison	*trahison*
iste	*fleuriste*
ite	*gastrite*
ité	*nervosité*
ition	*prohibition*
itude	*exactitude*
ment	*blanchiment*
oir	*grattoir*
oire	*baignoire*
ole	*bestiole*
on	*ballon*
ot	*ilot*
sion	*mission*
te	*propreté*
tion	*action*

	tude	*servitude*
	ule	*florule*
	ure	*pointure*
	xion	*flexion*

2 - SUFFIXES D'ADJECTIFS

	able	*faisable*
	ain	*américain*
	ais	*français*
	al	*matinal*
	an	*roman*
	ard	*vantard*
	asse	*fadasse*
	atre	*blanchâtre*
	aud	*rougeaud*
	el	*mortel*
	elet	*aigrelet*
	esque	*pédantesque*
	et	*doucet*
	eux	*haineux*
	u	*chevelu*
	ible	*lisible*
	ien	*parisien*
	ier	*hospitalier*
	if	*maladif*
	in	*enfantin*
	ique	*chimique*
	iste	*réaliste*
	ois	*suédois*
	ot	*palot*
	uble	*soluble*

3 - CLASSES DE SUFFIXES DE NOMS

1	␣ , *s, x*	*pou*
86	*abilité, abilités*	*respect - abilité*
2	*ace, aces*	*popul-ace*
3	*ade, ades*	*recul - ade*
4	*age, ages*	*feuill - age*
5	*aie, aies*	*fut - aie*
6	*ail, ails, aux*	*évent - ail*
7	*aille, ailles*	*valet - aille*
9	*ain, ains*	*quatr - ain*
10	*aine, aines*	*diz - aine*
11	*aire, aires*	*légat - aire*
12	*aison, aisons*	*sal - aison*
74	*al, aux, als*	*chev - al ; chac - al*
13	*ance, ances*	*abond - ance*
14	*ande, andes*	*multiplic - ande*
75	*ant, ants*	*assist - ant*
76	*ante, antes*	*assist - ante*
15	*ard, ards*	*chauff - ard*
53	*arde, ardes*	*chauff - arde*
16	*asse, asses*	*fil - asse*
17	*at, ats*	*extern - at*
18	*ateur, ateurs*	*condens - ateur*
78	*atrice, atrices*	*dessin - atrice*
19	*ation, ations*	*désign - ation*

87	*atif, atifs*	*séd - atif*
20	*ative, atives*	*rot - ative*
21	*ature, atures*	*arm - ature*
22	*and, ands*	*lourd - aud*
23	*aude, audes*	*lourd - aude*
24	*cule, cules*	*animal - cule*
72	*e, es*	*livr - e*
25	*eau, eaux*	*chevr - eau*
27	*ee, ees*	*assiett - ee*
28	*een, eens*	*lyc - een*
29	*eenne, ennes*	*lyc - eenne*
30	*elle, elles*	*ru - elle*
31	*ement, ements*	*règl - ement*
88	*ence, ences*	*sem - ence*
32	*er, ers*	*vach - er*
79	*ere, eres*	*vach - ere*
33	*erie, eries*	*cajol - erie*
34	*eron, erons*	*forg - eron*
80	*eronne, eronnes*	*mouch - eronne*
89	*esque, esques*	*fr - esque*
35	*esse, esses*	*sag - esse*
36	*et, ets*	*garçonn - et*
37	*ete, etes*	*nett - eté*
38	*ette, ettes*	*fill - ette*
81	*eur, eurs*	*mang - eur*
82	*euse, euses*	*mang - euse*
47	*i, is*	*étourd - i*

39	*ie, ies*	*étourd - ie*
40	*ien, iens*	*colleg - ien*
41	*ienne, iennes*	*colleg - ienne*
42	*ier, iers*	*pomm - ier*
43	*iere, ieres*	*port - iere*
73	*ibilité, ibilités*	*suscept - ibilité*
44	*ille, illes*	*brind - ille*
45	*illon, illons*	*négr - illon*
46	*illonne, illonne*	*négr - illonne*
48	*ise, ises*	*franch - ise*
49	*isme, ismes*	*protestant - isme*
50	*ison, isons*	*trah - ison*
51	*iste, istes*	*calvin - iste*
52	*ite, ites*	*real - ité*
54	*ition, itions*	*prohib - ition*
55	*itude, itudes*	*exact - itude*
56	*ment, ments*	*blanchi - ment*
57	*oir, oirs*	*gratt - oir*
58	*oire, oires*	*baign - oire*
59	*ole, oles*	*besti - ole*
60	*on, ons*	*ball - on*
83	*onne, onnes*	*garç - onne*
61	*ot, ots*	*ball - ot*
62	*otte, ottes*	*palli - otte*
63	*sion, sions*	*mis - sion*
65	*teur, teurs*	*ac - teur*

66	*trice, trices*	*ac - trice*
64	*tion, tions*	*ac - tion*
68	*té, tés*	*propre - té*
69	*tude, tudes*	*servi - tude*
70	*ule, ules*	*flor - ule*
77	*u, us*	*pend - u*
84	*un, uns*	*import - un*
85	*une, unes*	*fort - une*
71	*ure, ures*	*peint - ure*

4 - CLASSES DE SUFFIXES : ADJECTIFS

00	␣ , e, s, es	*délicat*
01	*able, ables*	*effroy - able*
02	*ain, aine, ains, aines*	*américˌ - ain*
26	*aire, aires*	*second - aire*
03	*ais, aise, aises*	*franç - ais*
04	*al, ale, als, aux, ales*	*matin - al*
29	*anc, anche, ancs, anches*	*bl - anc*
05	*an, ane, ans,anne, annes*	*rom - an*
30	*ant, ante, ants, antes*	*effray - ant*
06	*ard, arde, ards, ardes*	*vant - ard*
07	*asse, asses*	*fad - asse*
31	*ateur, atrice, ateurs, atrices*	*déprad - ateur*
37	*atif, ative, atifs, atives*	*séd - atif*
08	*atre, atres*	*blanch - atre*
38	*atoir, atoire, atoirs, atoires*	*superfét - atoir*
09	*aud, aude, auds, audes*	*rouge - aud*
43	*aux, ausse, ausses*	
32	*c, che, cs, ches*	
39	*eau, eaux, elle, elles*	*b - eau*
25	*e, ee, es, ees*	*log - é*
46	*en, enne, ens, ennes*	*acché - en*
10	*el, els, elle, elles*	*mort - el*
11	*elet, elette, elets, elettes*	*aigr - elet*

40	*ent, ente, ents, entes*	*différ - ent*
41	*erien, erienne, eriens eriennes*	
12	*esque, esques*	*livr - esque*
13	*et, ette, ets, ettes*	*douc - et*
28	*euf, euve, eufs, euves*	*n - euf*
14	*eux, eur, euse, eurs, euses*	*hain - eux*
15	*ible, ibles*	*ris - ible*
42	*ieme, iemes*	*dix - ieme*
16	*ien, ienne, iens, iennes*	*paris - ien*
17	*ier, iere, iers, ieres*	*hospital - ier*
18	*if, ive, ifs, ives*	*malad - if*
19	*in, ine, igne, ins, ignes, ines*	*mal - in, enfant - in*
20	*ique, ic, ics, iques*	*chim - ique*
21	*iste, istes*	*fum - iste*
33	*on, onne, ons, onnes*	*boug - on*
22	*ois, oise, oises*	*suéd - ois*
27	*os, osse, osses*	*gr - os*
23	*ot, ots, otte, ottes*	*pal - ot*
44	*oux, ousse, ousses*	*r - oux*
34	*teur, trice, teurs, trices*	*dépréda - teur*
24	*uble, ubles*	*sol - uble*
35	*u, ue, us, ues*	*dissol - u*
36	*un, une, uns, unes*	*br - un*

5 - CLASSES DE SUFFIXES D'ADVERBES

400	- *ment*	*hardi - ment*
401	- *ement*	*pos - ement*
402	- *amment*	*sav - amment*
403	- *emment*	*prud - emment*

6 - SUFFIXES CARACTERISTIQUES DES CLASSES DE VERBES

Verbes du 1er groupe

320	*-er -ent*	conjugaison - type	*ex : aimer*
326	*-er -tent*	verbes en *-eter*	*ex : jeter*
327	*-er -lent*	verbes en *-eler*	*ex : appeler*
332	*-yer -ient*	verbes en *-yer*	*ex : payer*

Verbes du 2ème groupe

336	*-ir -issent*		*ex : haïr*

Verbes du 3° groupe

340	*-mir -ment*	verbes en *-mir*	*ex : dormir*
341	*-enir -iennent*	verbes en *-enir*	*ex : tenir*
342	*-ir -ent*	verbes en *-aillir* ou *-euillir* ou *-ourir* ou *-frir*	*ex : assaillir*
345	*-ouillir -ouillent*	verbes en *-ouillir*	*ex : bouillir*
346	*-tir -tent*	verbes en *-tir*	*ex : mentir*
347	*-etir -etent*	verbes en *-etir*	*ex : vêtir*
348	*-erir -ierent*	verbes en *-erir*	*ex : acquérir*
350	*-uir -uient*	verbes en *-uir*	*ex : fuir*
355	*-evoir -oivent*	verbes en *-evoir*	*ex : recevoir*
370	*-re -ent*	verbes en *-dre* 1er type *andre, erdre, ompre, ordre*	*ex : rendre*
371	*-endre, -ennent*	verbes en *-endre* 2ème type	*ex : prendre*
372	*-ndre, -gnent*	verbes en *-ndre, eindre, aindre oindre*	*ex : peindre*
374	*-aincre - ainquent*	verbes en *-aincre*	*ex : vaincre*
376	*-aire -ent*	verbes en *-aire* 1er type	*ex : faire*

377	*-aire,*	*-aisent*	verbes en *-aire* 2° type	*ex : plaire*
378	*-aire,*	*- aient*	verbes en *-aire* 3° type	*ex : traire*
379	*-aitre,*	*- aissent*	verbes en *-aitre*	*ex : connaître*
382	*-ettre,*	*-ettent*	verbes en *-ettre*	*ex : mettre*
383	*-attre,*	*-attent*	verbes en *-attre*	*ex : battre*
384	*-oire,*	*-oient*	verbes en *-oire* 1° type	*ex : croire*
385	*-oitre,*	*-oissent*	verbes en *-oitre*	*ex : croitre*
386	*-oire,*	*-oivent*	verbes en *-oire* 2° type	*ex : boire*
387	*-oudre,*	*-olvent*	verbes en *-oudre* 1° type	*ex : absoudre*
388	*-oudre,*	*-ousent*	verbes en *-oudre* 2° type	*ex : coudre*
389	*-oudre,*	*-oulent*	verbes en *-oudre* 3° type	*ex : moudre*
390	*-ivre,*	*-ivent*	verbes en *-ivre*	*ex : suivre*
394	*-ire,*	*-isent*	verbes en *-ire*	*ex : lire*
395	*-ire,*	*-ivent*	verbes en *-ire*	*ex : écrire*
396	*-uire,*	*-uisent*	verbes en *-uire*	*ex : cuire*

7 - ENSEMBLE DES CLASSES DE SUFFIXES AMBIGUËS

aire - *aires*

asse - *asses*

atre - *atres*

esque - *esques*

ible - *ibles*

ieme - *iemes*

iste - *istes*

uble - *ubles*

A N N E X E 2

C R E A T I O N des S T E M S

1. Création de l'arborescence des chaînes inverses de suffixes :
   "Creat dic invsuf"

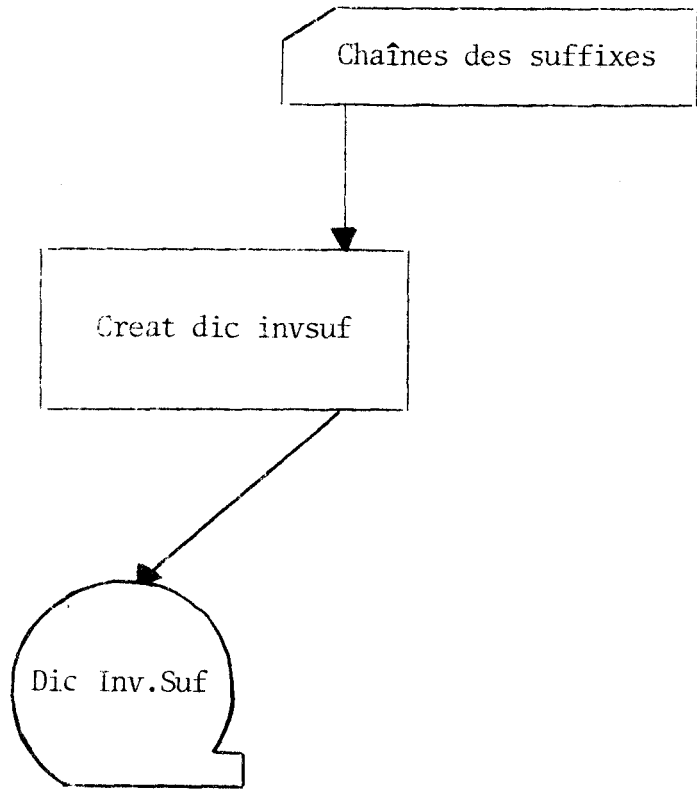
2. Création des stems : "creatstems - gramm< mot"

3. Création des stems : "creatstems-double-forme".

1. CREATION DE L'ARBORESCENCE DES CHAINES INVERSES DE SUFFIXES : "CREAT DIC INVSUF"

Ce programme crée l'arborescence des chaînes inverses de suffixes. C'est une extension du programme de création du dictionnaire décrit dans l'annexe 4, § 3. Ici, les suffixes sont transformés en chaînes inverses de suffixe avant insertion dans l'arbre.

L'ordinogramme est le suivant :



```
      INTEGER MAT(30),ADR(0:20000),CHAIN(0:20000)
      INTEGER TAM(30)
C*****************************************************************
C
C----------------------- CREAT DIC INVSUF ----------------------------
C
C*****************************************************************
      COMMON ADR,CHAIN
      LOGICAL F,FX
      LBUF=1000
C-----INITIALISATION.
      CHAIN(0)=4
      CHAIN(1)=1R
      CHAIN(2)=1R*
      CHAIN(3)=1R*
      CHAIN(4)=1R*
      ADR(0)=0
      ADR(1)=0
      ADR(2)=-90000
      ADR(3)=-90001
      ADR(4)=-90200
C-----------FIN INITIALISATION
C******************
      WRITE(108,109)
  109 FORMAT (1H1)
C     LECTURE ET SORTIE DU MOT
  984 CONTINUE
      DO 112 K=1,30
  112 TAM(K)=1R
      WRITE(108,104)
  104 FORMAT(1HO,$*********************************,***************$)
      ISUFFIXE=IMPASE=IGRAMM=ICHANGE=0
      READ(105,105)  ((TAM(J),J=1,20),ISUFFIXE,IGRAMM,IMPASE,ICHANGE)
  105 FORMAT(20R1,I10,I10,I10,I10)
      IF (TAM(1).EQ. 1R-)  GOTO 99$
      WRITE(108,106) (TAM(J),J=1,20)
  106 FORMAT(1HO,20R1)
C     CALCUL DE LANGMOT=PLACE DU DERNIER CARACTERE DE LA CHAINE
C-----------------------------------------------------------------
      M=1
   25 IF (TAM(M).EQ.1R )  GOTO 23
      M=M+1
      GOTO 25
   23 IF(TAM(M+1).EQ.1R )  GOTO 24
      M=M+2
      GOTO 25
   24 LANGMOT=M-1
X     WRITE (108,1007)  LANGMOT
X1007 FORMAT(1HO,$LANGUEUR DE LA CHAINE =$,I5)
C-----------------------------------------------------------------
      DO 113 K=LANGMOT+1,30
  113 MOT(K)=1R
      DO 111 K=1,LANGMOT
      J=LANGMOT-K+1
  111 MOT(J)=TAM(K)
```

```
      WRITE(108,106) (MOT(J),J=1,20)
      WRITE(108,110)  IMPOSE,ISUFFIXE,ICHANGE,IGRAMM
  110 FORMAT(1HO,$IMP=$,I5,5X,$SUF=$,I5,5X,$CHANGE=,$I5,5X,$GRAM=$,I5)
C-----------------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------INSERTION DU STEM DANS STOCK.STEMS---------------
C-----------------------------------------------------------------
      F=.FALSE.
      FX=.FALSE.
      M=ICHAIN=1
    9 LMOT=MOT(M)
      IF (M.EQ. LONGMOT+1 ) GOTO 1
    8 LCHAIN=CHAIN(ICHAIN)
      IF (LCHAIN.EQ.LMOT)   GOTO 2
      IF (LCHAIN.NE.1R/)    GOTO 7
C     SAUT DU /
X     WRITE  (108,1000) ICHAIN
X1000  FORMAT (1HO,$SAUT DE /  ICHAIN= $,I5)
      ICHAIN=ICHAIN+1
      GOTO 8
C     LES 2 CHAINES CONCORDENT
    2 ICHAIN=ICHAIN+1
      M=M+1
      GOTO 9
    7 IF (LCHAIN.EQ.1R*)  GOTO 3
      IF(LCHAIN.LT.LMOT)GOTO 17
C---- CAS OU CHAIN(ICHAIN)>MOT(M) : ON INSERE LE RESTE DU STEM **
X     WRITE (108,1010) ICHAIN,LCHAIN,LMOT
X1010 FORMAT(1HO,$CHAIN(ICHAIN).GT.LMOT,ICHAIN=$,I5,$CHAIN(ICHAIN)=$,1R1
X    -,$MOT(M)=$,1R1)
      FX=.TRUE.
      IALTERNANT=0
      GOTO 15
   17 IADR=ADR(ICHAIN)
      IF(IADR.NE.0)  GOTO 4
C-----CAS 3.1:RECHERCHER LE * SUIVANT ET INSERER LE RESTE DU STEM + *
X     WRITE (108,1003)  ICHAIN
X1003 FORMAT (1HO,$CAS 3.1  ICHAIN=$,I5)
      IALTERNANT=ICHAIN
      GOTO 10
C-----CAS2:REMPLACER LES * PAR DES / ET INSERER LE RESTE DU STEM + *
    3 CHAIN(ICHAIN)=1R/
      IALTERNANT=0
X     WRITE(108,1004) ICHAIN
X1004 FORMAT (1HO,$CAS 2  ICHAIN=  $,I5)
   31 ICHAIN=ICHAIN+1
      IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
      CHAIN(ICHAIN)=1R/
      GOTO 31
C-----CAS 3.2:RECHERCHER LA PLACE DE L'IALTERNANT
    4 LL=CHAIN(IADR)
X     WRITE(108,1005)  IADR
X1005 FORMAT(1HO,$RECHERCHE ALTERNANT  IADR=$,I5,
      IF(LL.LT.LMOT)  GOTO 16
      IF(LL.GT.LMOT)  GOTO 18
X     WRITE (108,1008) ICHAIN,IADR
```

```
112: X1008 FORMAT (1HO,$CHAIN(ICHAIN)=LMOT      ICHAIN=$,I5,10X,$IADR=$,I5)
113:       ICHAIN=IADR
114:       GOTO 2
115:    16 ICHAIN=IADR
116:       GOTO 17
117:    18 F=.TRUE.
118:       IALTERNANT=ICHAIN
119:       ICHAIN=IADR
120:       GOTO 15
121:     1 CONTINUE
122: C-----CAS 1: LE STEM EST CONTENU DANS UN STEM DE STOCK.STEMS
123: X     WRITE(108,1006 )  ICHAIN
124: X1006 FORMAT(1HO,$ STEM CONTENU DANS STOCK.STEMS  ICHAIN= $,I5)
125:       LCHAIN=CHAIN(ICHAIN)
126:       IF(LCHAIN.EQ.1R/)  GOTO 21
127:       IF(LCHAIN.EQ.1R*)  GOTO 21
128:       IALTERNANT=0
129:       LONGMOT=LONGMOT+1
130:       MOT(LONGMOT)=1R/
131:       GOTO 20
132: C     RECHERCHE DE LI* SUIVANT.
133: C     ICHAIN SERA L'ADRESSE DU CARACTERE SUIVANT LE DERNIER *
134:    10 ICHAIN=ICHAIN+1
135:       IF(CHAIN(ICHAIN).EQ.1R*)  GOTO 5
136:       IF(ICHAIN.GT.CHAIN(0))  GOTO 6
137:       GOTO 10
138:     6 WRITE (108,14)
139:    14 FORMAT(1HO,$ERREUR RECHERCHE *$)
140:       GOTO 29
141:     5 CONTINUE
142: X     WRITE(108,1001)  ICHAIN
143: X1001 FORMAT(1HO,$RECHERCHE DU PREMIER  *    ICHAIN= $,I5)
144: C     RECHERCHE DU DERNIER *
145:    32 ICHAIN=ICHAIN+1
146:       IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
147:       GOTO 32
148: C     ON AJOUTE UN * A LA CHAINE AVANT INSERTION
149:    15 LONGMOT=LONGMOT+1
150:       MOT(LONGMOT)=1R*
151: C     INSERTION DES CARACTERES M A LONGMOT DE MOT
152: C     LE PREMIER CARACTERE A DECALER DE CHAIN A POUR ADRESSE ICHAIN
153:    20 IDECAL=LONGMOT-M+1
154: X     WRITE(108,1002)  M,LONGMOT,ICHAIN
155: X1002 FORMAT(1HO,$INSERTION CHAINE,M=$,I5,$LONGMOT=$,I8,$   ICHAIN=$I5)
156:       II=CHAIN(0)
157:       CHAIN(0)=II+IDECAL
158:       DO 11 JJ=II,ICHAIN,-1
159:       ADR(JJ+IDECAL)=ADR(JJ)
160:    11 CHAIN(JJ+IDECAL)=CHAIN(JJ)
161:       DO 12 JJ=0,IDECAL-1
162:       ADR(ICHAIN+JJ)=0
163:    12 CHAIN(ICHAIN+JJ)=MOT(M+JJ)
164:       DO 13 JJ=1,II+IDECAL
165:    13 IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+IDECAL
166: C     CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
167:       IF(F)   GOTO 19
```

```
C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
       IF (FX) ADR(ICHAIN)=ICHAIN + IDECAL
       IF(IALTERNANT.NE.0)  ADR(IALTERNANT)=ICHAIN
       GOTO 40
C      CHAINAGE DANS LE CAS 3.2
   19  ADR(ICHAIN)=ADR(IALTERNANT)
       ADR(IALTERNANT)=ICHAIN
   40  ICHAIN=ICHAIN+IDECAL-1
       GOTO 28
C-------
   21  CONTINUE
       F=.FALSE.
C      ON DEVRA VERIFIER S'IL NE S'AGIT PAS D'UN NOUVEAU CODE INTRODUIT
C        POUR UN STEM FISTACK.STEMS .
X      WRITE(108,1011)
X1011 FORMAT(1HO,$LE STEM FIGURE DEJA DANS DIC-F_COD$)
       GOTO 30
   41  ICHAIN=ICHAIN+1
X      WRITE(108,1009)  ICHAIN
X1009 FORMAT(1HO,$ICHAIN=...$,I5)
       IF(ADR(ICHAIN).GE.0)  GOTO 38
   30  IA=ADR(ICHAIN)
       II=IA/1000
       ISUF=-IA+II*1000
       IF(ISUFFIXE.LT.ISUF)  GOTO 42
       IF(ISUF.EQ.ISUFFIXE)  GOTO 33
       GOTO 41
   33  IF(ICHANGE.EQ.0)  GOTO 34
       F=.TRUE.
       IN=-IA/10000
       IF(IN.NE.ICHANGE)  GOTO 41
       GOTO 28
   34  IF(IMPOSE.EQ.0)  GOTO 36
C      DECALAGE DE 1.
   42  IJ=CHAIN(0)
       DO 35 JJ=IJ,ICHAIN,-1
       ADR(JJ+1)=ADR(JJ)
   35  CHAIN(JJ+1)=CHAIN(JJ)
   45  CHAIN(0)=IJ+1
C      REMISE EN ORDRE DES POINTEURS.
       DO 39 JJ=1,CHAIN(0)
   39  IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+1
   28  CONTINUE
C***
C      INTRODUCTION DU MOT
 1012 FORMAT(1HO,$ADR(0) :   ICHAIN=$,I5,5X,1R1,6X,$ADR(ICHAIN)=$,I10)
 1013 FORMAT(1HO,$IMPOSE=$,I5,5X,$ICHAIN=$,I5,5X,1R1,5X,$ADR=$,I10)
       IF (IGRAMM.EQ.2)  GOTO   26
       IF (IGRAMM.EQ.9)  GOTO  27
       IF (IGRAMM.EQ.1)  GOTO  27
C      CAS OU IL S'AGIT D'UN MOT-CLE.
       IF (IMPOSE.EQ.0)  GOTO 47
       ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
       WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
       GOTO 29
   47  ADR(0)=ADR(0)-1
```

```
224:         ADR(ICHAIN)=ADR(0)*100000-ISUFFIXE
225:         WRITE(108,1012)  ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
226:         GOTO 29
227:      27 CONTINUE
228: C       CAS OU IL S'AGIT D'UN MOT-VIDE , OU D'UN SUFFIXE
229:         ADR(ICHAIN)=-IGRAMM*10000-ISUFFIXE
230:          IF (IMPOSE.EQ.0)  GOTO 29
231:         WRITE(108,48)
232:      48 FORMAT(1HO,$ON A CHERCHE A DONNER UN N. A UN SUFFIXE OU MOT VIDE$)
233:         GOTO 29
234:      26 CONTINUE
235: C       CAS OU IL S'AGIT D'UN MOT GRAMMATICAL.
236:         IF(IMPOSE.EQ.0)  GOTO 49
237:         ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
238:         WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
239:         GOTO 29
240:      49 IMAXGR=IMAXGR+1
241:         ADR(ICHAIN)=-IMAXGR*100000-ISUFFIXE-IGRAMM*10000
242:         WRITE(108,1012)  ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
243:         GOTO 29
244: C***
245:      36 CONTINUE
246:         WRITE(108,37)
247:      37 FORMAT(1HO,$LE MOT FIGURE DEJA DANS STOCK-ITEMS$)
248: X       WRITE(108,1015)  ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
249: X1015 FORMAT(1HO,$            ICHAIN=$,I5,5X,1R1,1X,$ADR(ICHAIN)=$,I10)
250:         GOTO 29
251:      38 CONTINUE
252: X       WRITE(108,1014)
253: X1014 FORMAT(1HO,$ON A EXPLORE TOUS LES CODES DU STEM$)
254:         IF( F )  GOTO 46
255:         IJ=CHAIN(0)
256:         DO 44 JJ=IJ,ICHAIN-1,-1
257:         ADR(JJ+1)=ADR(JJ)
258:      44 CHAIN(JJ+1)=CHAIN(JJ)
259:         GOTO 45
260:      46 CONTINUE
261:         WRITE(108,43)
262:      43 FORMAT(1HO,$ ON A VOULU CHANGER UN N. QUI N'EXISTE PAS$)
263:    29  CONTINUE
264: C-------------------------------------------------------------------
265: C       FIN INSERTION DU MOT DANS DICFOD---------------------------
266: C-------------------------------------------------------------------
267:         GOTO 98$
268:    99$ CONTINUE
269:          WRITE(108,109)
270: C       SORTIE DE LA CHAINE MODIFIEE
271:         DO 107 J=1,CHAIN(0)
272:    107 WRITE(108,108)  J,CHAIN(J),ADR(J)
273:    108 FORMAT (1HO,I5,10X,1R1,5X,I10)
274: C-------------------------------------------------------------------
275:         END LABELS
276: C$$$$$$$$$$$$$$
277: C        MISE SUR BANDE DES TABLEAUX ADR ET CHAIN.
278: C-------------------------------------------------------------------
279:         WRITE(108,100)
```

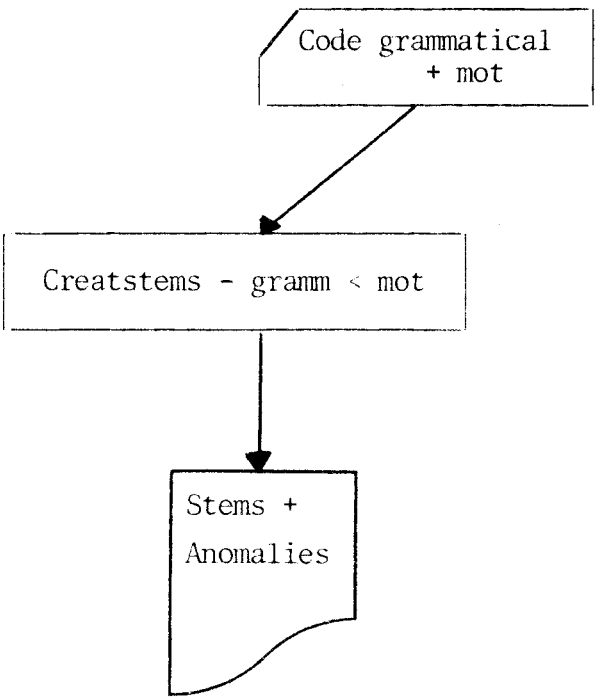
```
280:   100 FORMAT(1H1,50X,$MISE DES DONNEES SUR BANDE$)
281:       CALL BUFFER OUT (2,1,ADR,LBUF,ITEST,N)
282:     1 GOTO (1,2,3,4)  ITEST
283:     2 CONTINUE
284:       WRITE(108,102) N
285:        CALL BUFFER OUT(2,1,CHAIN,LBUF,ITEST,N)
286:     5 GOTO (5,6,3,4)  ITEST
287:     3 WRITE(108,200)
288:   200 FORMAT(1HO,$RENCONTRE DE FIN DE FICHIER$)
289:       GOTO 6
290:     4 WRITE(108,201)
291:   201 FORMAT(1HO,$  ERREUR  $)
292:     6 ENDFILE 2
293:       WRITE(108,102) N
294:   102 FORMAT(1HO,$   N=     $,I5)
295:       REWIND 2
296:       WRITE(108,101)
297:   101 FORMAT(1HO,$  FIN DE MISE DES DONNEES SUR BANDE $)
298: C-----------------------------------------------------------------
299:       END LABELS
300: C*********
301:       WRITE(108,1)
302:     1 FORMAT(1H1,20X,$CARACTERISTIQUES DE LA CHAINE$)
303:       WRITE(108,2) CHAIN(0)
304:     2 FORMAT(1HO,$LONGUEUR DE LA CHAINE=$,I10)
305:       WRITE(108,3)  ADR(0)
306:     3 FORMAT(1HO,$NOMBRE DE MOTS PIVOTS=$,I10)
307:       WRITE(108,4) IMAXGR
308:     4 FORMAT(1HO,$NOMBRE DE MOTS GRAMMATICAUX=$,I10)
309:       STOP
310:       END
```

2. CREATION DES STEMS : "CREATSTEMS - GRAMM < MOT"

Ce programme utilise l'algorithme de recherche de la plus longue chaîne de l'arborescence contenue dans une chaîne donnée.

Les principaux identificateurs figurent parmi ceux décrits dans l'annexe 4, § 2. Les codes des suffixes et de la fonction grammaticale sont ceux décrits dans l'annexe 3.

L'ordinogramme est le suivant :



```
      INTEGER MBT(40),ADRCHAIN(40),ADC(6,40),TAM(6,40)
      INTEGER CHAIN(0:20000),ADR(0:20000)
      COMMON ADR,CHAIN
      LOGICAL MSUF
C-----------------------------------------------------------------
C
C                    CREATSTEMS-GRAMM<MBT
C
C-----------------------------------------------------------------
C     LECTURE DE ADR ET CHAIN-
      LBUF=1000
      CALL BUFFER IN(1,1,ADR,LBUF,ITEST,N)
    1 GOTO (1,2,7,3) ITEST
    7 WRITE(108,103)
    2 WRITE(108,100) N
  100 FORMAT(1H1,$ENTREE DE ADR$,I10)
      CALL BUFFER IN(1,1,CHAIN,LBUF,ITEST,N)
    4 GOTO (4,5,8,3) ITEST
    8 WRITE(108,103)
    5 WRITE(108,101) N
  101 FORMAT(1HO,$ENTREE DE CHAIN$,I10)
      GOTO 6
    3 WRITE(108,102)
  102 FORMAT(1HO,$ ERREUR$)
  103 FORMAT(1HO,$ FIN DE FICHIER$)
      PAUSE 1
    6 CONTINUE
      WRITE(108,1111)
 1111 FORMAT(1H1,$FIN ENTREE DIC-INVSUF$)
      END LABELS
C*******
      CALL EBFSET(33S)
   13 READ(105,100) IG,(MBT(J),J=1,40)
  100 FORMAT(I1,40R1)
      DO 11 J=1,40
   11 ADRCHAIN(J)=0
      WRITE(108,101) IG,(MBT(J),J=1,40)
  101 FORMAT(1HO,$******$,I2,5X,40R1)
      IF(IG.NE.3)  GOTO 12
      WRITE(108,102)
  102 FORMAT(1HO,$LE MBT EST UN VERBE    $)
      GOTO 13
   12 CONTINUE
C-----------------------------------------------------------------
C**** RECHERCHE DU PLUS LONG SUFFIXE.
C-----INITIALISATION DE M
      DO 9 J=1,40
      IF(MBT(J).NE.1R )  GOTO 9
      IF(MBT(J+1).EQ.1R ) M=J-1; GOTO 10
    9 CONTINUE
   10 ADRCHAIN(M)=2
      INBL=INBL+M
      ICHAIN=1
    1 LMBT=MBT(M)
X     WRITE(108,200)  M,LMBT,ICHAIN
```

```
 56: X200  FORMAT(1HO,$M= $,I5,5X,1R1,5X,$ICHAIN=$,I5)
 57:     5 LCHAIN=CHAIN(ICHAIN)
 58:       IF(LCHAIN.EQ.LMOT)  GOTO 3
 59:       IF(LCHAIN.NE.1R/)   GOTO 4
 60: X     WRITE(108,201) ICHAIN
 61: X201  FORMAT(1HO,$SAUT DU / ,ICHAIN=$,I5)
 62: C.... ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER /.
 63:       IF(ADRCHAIN(M).EQ.0) ADRCHAIN(M)=ICHAIN
 64:       ICHAIN=ICHAIN+1
 65:       GOTO 5
 66:     3 M=M-1
 67:       IF(M.EQ.0) GOTO 8
 68:       ICHAIN=ICHAIN+1
 69:       GOTO 1
 70:     4 IF(LCHAIN.EQ.1R*) GOTO 7
 71: C.... SUIVI DU CHAINAGE
 72:     6 IADR=ADR(ICHAIN)
 73: X     WRITE(108,202)  ICHAIN,IADR
 74: X202  FORMAT(1HO,$SUIVI DU CHAINAGE  ICHAIN=$,I5,5X,$IADR=$,I5)
 75:       IF(IADR.EQ.0) GOTO 8
 76:       ICHAIN=IADR
 77:       LCHAIN=CHAIN(ICHAIN)
 78:       IF(LCHAIN-LMOT) 6,2,8
 79:     2 CONTINUE
 80: C.... APRES SUIVI DU CHAINAGE LCHAIN=LMOT
 81: X     WRITE(108,203)  IADR
 82: X203  FORMAT(1HO,$LCHAIN=LMOT , ICHAIN= $,I5)
 83:       GOTO 3
 84:     7 ADRCHAIN(M)=ICHAIN
 85:     8 CONTINUE
 86: C**** FIN RECHERCHE DU PLUS LONG SUFFIXE
 87: C-------------------------------------------------------------
 88: C     IMPRESSION INTERMEDIAIRE.
 89:       DO 24 J=1,30
 90:    24 WRITE(108,25) J,MOT(J),ADRCHAIN(J)
 91:    25 FORMAT(1HO,20X,I10,5X,1R1,I10)
 92: C-------------------------------------------------------------
 93:       IG=-IG-900
 94:       DO 14 J=1,40
 95:       IF(ADRCHAIN(J).EQ.0)  GOTO 14
 96: C-----IL EXISTE UN STEM POSSIBLE.RECHERCHE DU SUFFIXE.
 97:       WRITE(108,205) J
 98:  205  FORMAT(1HO,$STEM POSSIBLE  J=$,I5)
 99:       IC1=ADRCHAIN(J)
100:       MSUF=.FALSE.
101:    18 IS1=ADR(IC1)
102:       IF(IS1.GE.0)  GOTO 15
103:       IS2=IS1/100
104:       IF(IS2-IG) 15,17,16
105:    16 IC1=IC1+1
106:       GOTO 18
107:    17 CONTINUE
108: C   ON A TROUVE UN SUFFIXE.
109:       IF(MSUF) GOTO 19
110:       MSUF=.TRUE.
111:       ISTEM=J
```

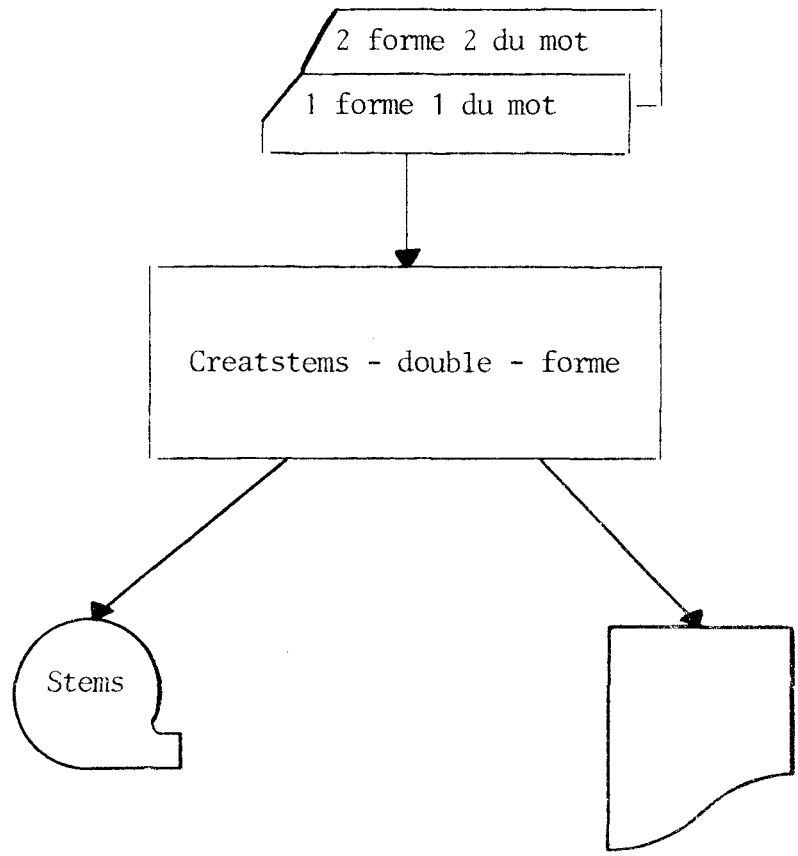
```
112:        NSUF=IS1
113:        GOTO 16
114:     15 IF(MSUF) GOTO 20
115:     14 CONTINUE
116: C-----IL N'EXISTE PAS DE SUFFIXE.
117:        WRITE(108,150)
118:    150 FORMAT(1HO,$ERREUR-1 PAS DE SUFFIXE$)
119:        GOTO 13
120:     19 CONTINUE
121: C-----IL EXISTE 2 SUFFIXES POUR LE MEME STEM.
122:        WRITE(108,151) ISTEM
123:    151 FORMAT(1HO,$ERREUR-2 2SUFFIXES J=$,I5)
124:        GOTO 13
125: C-----ON A TROUVE UN SUFFIXE.
126:     20 CONTINUE
127:        DO 21 J=ISTEM+1,40
128:     21 MOT(J)=1R
129:        WRITE(108,152) (MOT(J),J=1,40),NSUF
130:    152 FORMAT(1HO,10X,40R1,5X,I10)
131:        GOTO 13
132:     33 STOP
133:        END
```

3. CREATION DES STEMS : "CREATSTEMS - DOUBLE - FORME "

Comme précédemment, les principaux identificateurs, et les codes sont décrits dans l'annexe 3.

Les stems sont ici stockés sur bande magnétique, pour être réutilisés dans le programme de création du dictionnaire. Pour les raisons signalées dans la fin du chapitre II, seule cette méthode de création des stems a été utilisée pour déterminer les stems des mots à introduire dans les différents dictionnaires qui ont été créés.

L'ordinogramme est le suivant :



```
      INTEGER MOT(40),ADRCHAIN(40),ADC(6,40),TOM(6,40)
      INTEGER MAT(40)
      INTEGER CHAIN(0:20000),ADR(0:20000)
      COMMON ADR,CHAIN
      LOGICAL MSUF
C*************************************************************************
C
C                    CREATSTEMS-DOUBLE-FORME
C
C*************************************************************************
C*************************************************************************
C     LECTURE DE ADR ET CHAIN-
      LBUF=1000
      CALL BUFFER IN(1,1,ADR,LBUF,ITEST,N)
    1 GOTO (1,2,7,3) ITEST
    7 WRITE(108,103)
    2 WRITE(108,100) N
  100 FORMAT(1H1,$ENTREE DE ADR$,I10)
      CALL BUFFER IN(1,1,CHAIN,LBUF,ITEST,N)
    4 GOTO (4,5,8,3) ITEST
    8 WRITE(108,103)
    5 WRITE(108,101) N
  101 FORMAT(1HO,$ENTREE DE CHAIN$,I10)
      GOTO 6
    3 WRITE(108,102)
  102 FORMAT(1HO,$ ERREURS)
  103 FORMAT(1HO,$ FIN DE FICHIERS)
      PAUSE 1
    6 CONTINUE
      WRITE(108,1111)
 1111 FORMAT(1H1,$FIN ENTREE DIC-INVSUF$)
      END LABELS
      IERR=-1
C*******
      READ(105,100) I,(MOT(J),J=1,40)
  100 FORMAT(I1,40R1)
   23 IERR=IERR+1
   27 DO 22 J=1,40
      DO 22 KK=1,6
   22 ADC(KK,J)=TOM(KK,J)=0
      KK=0
   13 KK=KK+1
X     WRITE(108,204) I,(MOT(J),J=1,40)
C**** RECHERCHE DU PLUS LONG SUFFIXE.
C-----INITIALISATION DE M
      DO 9 J=1,40
      IF(MOT(J).NE.1R ) GOTO 9
      IF(MOT(J+1).EQ.1R ) M=J-1; GOTO 10
    9 CONTINUE
   10 ADRCHAIN(M)=2
      INBL=INBL+M
      ICHAIN=1
    1 LMOT=MOT(M)
X     WRITE(108,200)  M,LMOT,ICHAIN
X200  FORMAT(1HO,$M= $,I5,5X,1R1,5X,$ICHAIN=$,I5)
```

```
      5 LCHAIN=CHAIN(ICHAIN)
        IF(LCHAIN.EQ.LMOT)  GOTO 3
        IF(LCHAIN.NE.1R/)   GOTO 4
X       WRITE(108,201) ICHAIN
X201    FORMAT(1HO,$SAUT DU / ,ICHAIN=$,I5)
C.... ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER /.
        IF(ADRCHAIN(M).EQ.0) ADRCHAIN(M)=ICHAIN
        ICHAIN=ICHAIN+1
        GOTO 5
      3 M=M-1
        IF(M.EQ.0) GOTO 8
        ICHAIN=ICHAIN+1
        GOTO 1
      4 IF(LCHAIN.EQ.1R*) GOTO 7
C.... SUIVI DU CHAINAGE
      6 IADR=ADR(ICHAIN)
X       WRITE(108,202)  ICHAIN,IADR
X202    FORMAT(1HO,$SUIVI DU CHAINAGE  ICHAIN=$,I5,5X,$IADR=$,I5)
        IF(IADR.EQ.0) GOTO 2
        ICHAIN=IADR
        LCHAIN=CHAIN(ICHAIN)
        IF(LCHAIN-LMOT) 6,2,8
      2 CONTINUE
C.... APRES SUIVI DU CHAINAGE LCHAIN=LMOT
X       WRITE(108,203)  IADR
X203    FORMAT(1HO,$LCHAIN=LMOT ,  ICHAIN= $,I5)
        GOTO 3
      7 ADRCHAIN(M)=ICHAIN
      8 CONTINUE
C**** FIN RECHERCHE DU PLUS LONG SUFFIXE
X204    FORMAT(1HO,$*** $,I2,5X,40R1)
        DO 11 J=1,40
        ADC(KK,J)=ADRCHAIN(J)
        ADRCHAIN(J)=0
     11 TOM(KK,J)=MOT(J)
        CALL EOFSET(33S)
        READ(105,100)  I,(MOT(J),J=1,40)
        IF(I.EQ.1) GOTO 12
        GOTO 13
     12 CONTINUE
C**** RECHERCHE DU SUFFIXE COMPATIBLE
C-----IMPRESSION INTERMEDIAIRE
        WRITE(108,104) (TOM(1,J),J=1,40)
        WRITE(108,104) (TOM(2,J),J=1,40)
  104   FORMAT(1HO,40R1)
        DO 14 J=1,40
        IF(ADC(1,J).EQ.0) GOTO 14
        IF(ADC(2,J).EQ.0) GOTO 14
C---- IL EXISTE UN STEM POSSIBLE.RECHERCHE D'UN SUFFIXE
X       WRITE(108,205) J
X205    FORMAT(1HO,$STEM POSSIBLE  J=$,I5)
        IC1=ADC(1,J)
        IS1=ADR(IC1)
        MSUF=.FALSE.
     18 IC2=ADC(2,J)
        IS2=ADR(IC2)
```

```
112:     19 IF(IS2-IS1)  15,16,17
113:     17 IC2=IC2+1
114:        IS2=ADR(IC2)
115:        IF(IS2.LT.0) GOTO 19
116: C---- ON A FINI D'EXPLORER LES POSSIBILITES DU 2-SUFFIXE.
117:     15 IC1=IC1+1
118:        IS1=ADR(IC1)
119:        IF(IS1.LT.0) GO TO 18
120: C---- ON A FINI D'EXPLORER LES POSSIBILITES DU 1-SUFFIXE.
121:        IF(MSUF) GO TO 21
122:        GOTO 14
123:     16 CONTINUE
124: C---- ON A TROUVE UN SUFFIXE
125:        IF(MSUF) GOTO 20
126:        MSUF=.TRUE.
127:        ISTEM=J
128:        NSUF=IS2
129:        GOTO 17
130:     20 CONTINUE
131: C---- ON A TROUVE 2 SUFFIXES POUR LE MEME STEM.
132:        WRITE(108,101) J
133:    101 FORMAT(1H1,50X,$ ERREUR 1 : 2 SUFFIXES    J=$,I5)
134:        GO TO 23
135:     14 CONTINUE
136: C---- ON SORT PAR EPUISEMENT DES SUF.
137:        WRITE(108,102)  (TOM(KK,J),J=1,40)
138:    102 FORMAT(1H150X,$ERREUR 2 : PAS DE SUFFIXE  :$,40R1)
139:        GOTO 23
140:     21 CONTINUE
141:        DO 25 J=1,ISTEM
142:        IF(TOM(1,J)-TOM(I,J)) 26,25,26
143:     25 CONTINUE
144:        GOTO 28
145:     26 WRITE(108,105)
146:    105 FORMAT(1H1,50X,$ERREUR 3 : CARTES DONNEES FAUSSES$)
147:        GOTO 23
148: C---- ON A TROUVE UN SUFFIXE
149:     28 ICOMPT=ICOMPT+1
150:        INLST=INLST+ISTEM
151:        DO 24 J=ISTEM+1,40
152:     24 TOM(1,J)=1K
153:        DO 47 J=1,40
154:     47 MAT(J)=TOM(1,J)
155:        WRITE(108,103) ICOMPT,NSUF,(MAT(J),J=1,40)
156:    103 FORMAT(1HO,$***$,I5,10X,I10,2X,40R1)
157:        J=NSUF/1000
158:        NSUF=-NSUF+J*1000
159:        MAT(40)=NSUF
160:        CALL BUFFER OUT(2,1,MAT,40,ITEST)
161:     41 GOTO (41,27,43,44) ITEST
162:     43 WRITE(108,45)
163:     45 FORMAT(1H1,$ERREUR : F-F$)
164:     44 WRITE(108,46)
165:     46 FORMAT(1H1,$ ERREUR-TRANSMISSION$)
166:     33 CONTINUE
167:        ENDFILE 2
```

```
C------CALCULS ------------------------------------------------------------
      WRITE(108,106) ICOMPT
  106 FORMAT(1H1,$NB. DE STEMS TROUVES=$,I10)
      INBL=INBL/2
      WRITE(108,107) INBL
  107 FORMAT(1HO,$NB. DE LETTRES CONTENUES DANS LES MOTS=$,I10)
      BL=INBL
      BL=BL/ICOMPT
       WRITE(108,108) BL
  108 FORMAT(1HO,$LONG. MOYENNE DES MOTS =$,F8.3)
      WRITE(108,109) IERR
  109 FORMAT(1HO,$NB. ERREURS=$,I6)
      WRITE(108,110) INLST
  110 FORMAT(1HO,$NB. DE LETTRES DES STEMS =$,I10)
      BL=INLST
      BL=BL/ICOMPT
      WRITE(108,111) BL
  111 FORMAT(1HO,$LONG. MOYENNE DES STEMS =$,F8.3)
      STOP
      END
```

ANNEXE 3

DESCRIPTION

DICTIONNAIRE -1- CHAINAGE

1. Structure des pointeurs associés aux chaînes
2. Identificateurs communs aux programmes du dictionnaire -1- chaînage
3. Programmes de création et de mise à jour du dictionnaire "insertstems". Ordinogramme création du dictionnaire
4. Programme de suppression d'une chaîne "concacstems"
5. Programme de sortie des chaînes du dictionnaire "sortstem - sortmot"
6. Programme de recherche des mots d'une phrase qui figurent dans le dictionnaire "Rechphrase"

1. STRUCTURE DES POINTEURS ASSOCIES AUX CHAINES

Les pointeurs de mot et les pointeurs d'alternance figurent dans le même tableau. Ces pointeurs ont été différenciés afin d'accélérer les opérations de remise en ordre des pointeurs d'alternance, sans avoir à examiner le caractère auquel chaque pointeur correspond :

- les pointeurs d'alternance sont ≥ 0

- les pointeurs de mot sont < 0.

Les pointeurs de mot contiennent l'ensemble des codes syntactiques et sémantiques associés à une chaîne de caractères. Chaque pointeur de mot est stocké dans un mot-machine de l'ordinateur CII-10070. Ce matériel a l'avantage d'effectuer très rapidement les opérations de multiplication et de division en décimal, grâce à des dispositifs d'arithmétique décimale cablés.

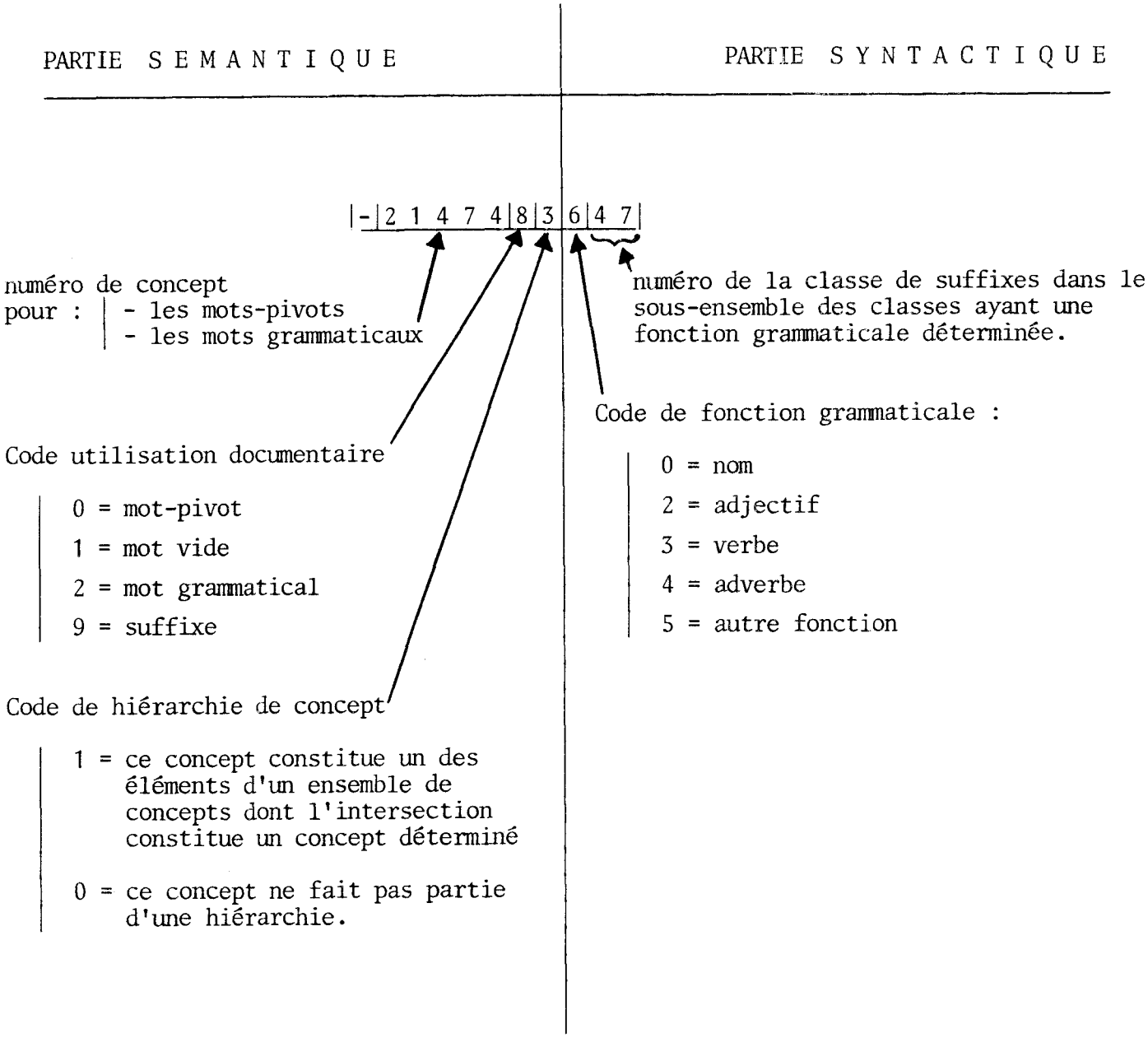
Les mots-machine 10070 comportent 32 bits, et permettent de représenter dans un mot un nombre entier ≤ 2147483647 $(2^{32}-1)$ soient 10 positions décimales caractéristiques.

Afin de simplifier les programmes écrits en Fortran, les 10 positions décimales du pointeur de mot ont été subdivisées en un certain nombre de positions décimales pour recevoir des codes ; l'ensemble des codes est divisé en deux sous-ensembles :

- partie syntactique : comprend les trois positions décimales de droite du mot
- partie sémantique : sur 7 positions décimales.

Les parties sémantiques et syntactiques du codage sont elles-mêmes subdivisées en ensembles de codes, suivant la structure donnée ci-après.

STRUCTURE DES POINTEURS DE MOT



. IDENTIFICATEURS COMMUNS AUX PROGRAMMES DU DICTIONNAIRE -1- CHAINAGE

Afin de faciliter la compréhension des programmes, certains identificateurs ont été réutilisés pour désigner, d'un programme à l'autre, les mêmes éléments, ou des éléments voisins :

Le dictionnaire est constitué par l'association de deux tableaux :

- CHAIN : tableau contenant les caractères des chaînes
- ADR : tableau des pointeurs d'alternance et des pointeurs de mots.

La chaîne de caractères à analyser, soit pour l'introduire, ou la supprimer de la chaîne, soit pour déterminer les mots du dictionnaire qu'elle contient sera stockée dans le tableau MOT. A ce tableau sera éventuellement associé le tableau ADRCHAIN, qui recevra des pointeurs pris dans ADR, pour l'analyse du contenu de MOT.

Les pointeurs courants des tableaux précédents seront :

- ICHAIN : pointeur courant du tableau CHAIN
- M : pointeur courant du tableau MØT.

Il existe un certain nombre d'identificateurs associés à ces tableaux :

- LCHAIN : contient le caractère courant du tableau CHAIN
- LMØT : contient le caractère courant à analyser du tableau MØT
- Iadr : pointeur courant du chaînage d'alternant
- Chain(o) : contient la longueur de la chaîne.

Les codes sonr répétés par un certain nombre d'identificateurs :

- ISUFFIXE : représente la partie syntactique du codage
- IGRAMM : représente le code d'utilisation documentaire
- ICHANGE : représente le code de concept à changer dans le cas d'insertion, ou de concaténation d'une chaîne
- IMPØSE : représente le numéro de concept imposé.

3. PROGRAMMES DE CREATION ET DE MISE A JOUR DU DICTIONNAIRE - "INSERTSTEMS" -

Ces deux programmes ne diffèrent que par la lecture des données :

- le programme de création crée une chaîne initiale, et insère dans le chaînage des données lues sur bande (données créées par le programme de création des stems), ou sur cartes

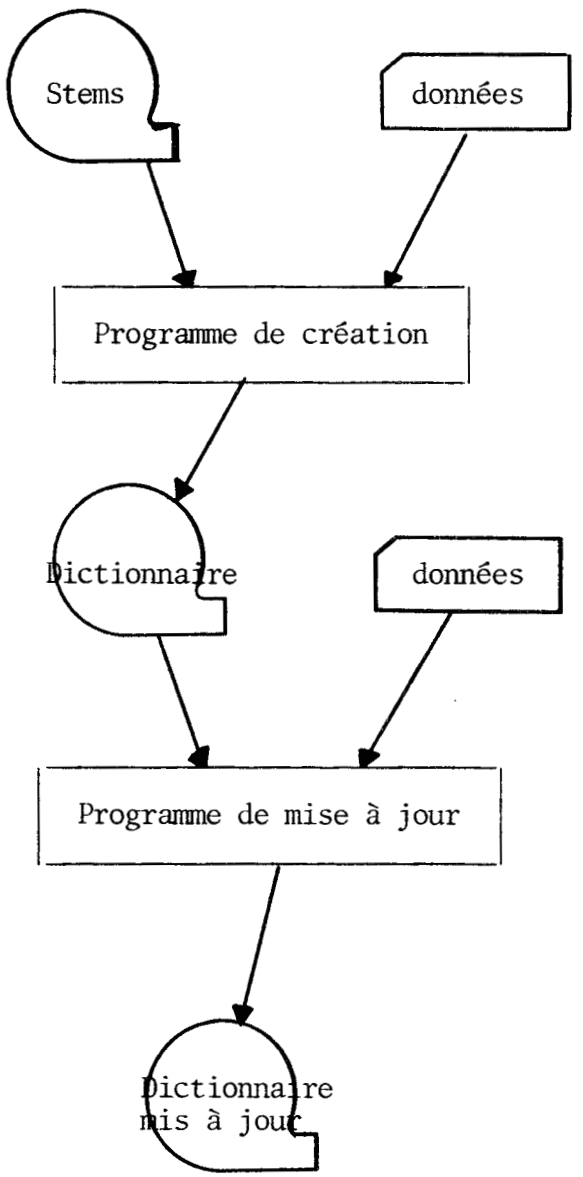
- le programme de mise à jour n'utilise,ici, que des données sur carte, car il a été utilisé pour les modifications du dictionnaire, en particulier pour introduire les synonymies.

Les deux programmes se terminent par le calcul des principales caractéristiques de la chaîne :
longueur de la chaîne
nombre de mots-pivots
nombre de mots grammaticaux
pourcentage d'éléments de adr non nuls.

Ces deux programmes auraient pu être réunis en un seul, mais on peut voir sur l'organigramme que la bande intermédiaire peut servir de point de reprise si les modifications introduites par le programme de mise à jour sont erronées, et surtout s'il y a des erreurs de manipulations dûes aux opérateurs, lors du passage en batch, ce qui a été, hélas, assez souvent le cas !

Le code de concept des mots-pivots est obtenu en incrémentant de 1 le contenu de l'identificateur ADR(0), sauf si le code est imposé (impose $\neq$ 0). On peut modifier le code de concept d'un mot-pivot en introduisant, en donnée, dans "ichange" le code que l'on veut changer, et dans "impose" le nouveau code qui figurera dans le dictionnaire.

ORDINOGRAMME CREATION DU DICTIONNAIRE



```
      INTEGER MOT(40),ADR(0:6000),CHAIN(0:6000)
      COMMON ADR,CHAIN
      LOGICAL F,FX
      LOGICAL BANDE
      BANDE=.TRUE.
C*****************************************************************************
C
C        CREATION DICTIONNAIRE CARREZ
C
C*****************************************************************************
C-----INITIALISATION.
      CHAIN(0)=5
      CHAIN(1)=1R
      CHAIN(2)=1R*
      CHAIN(3)=1R*
      CHAIN(4)=1R*
      CHAIN(5)=1R*
      ADR(0)=0
      ADR(1)=0
      ADR(2)=-90000
      ADR(3)=-90001
      ADR(4)=-90200
      ADR(5)=-90370
C-----------FIN INITIALISATION
C******************
      WRITE(108,109)
  109 FORMAT (1H1)
C     LECTURE ET SORTIE DU MOT
   98$ CONTINUE
      WRITE(108,104)
   104 FORMAT(1HO,$**************************$)
      ISUFFIXE=IMPOSE=IGRAMM=ICHANGE=0
      IF(BANDE) GOTO 50
      READ(105,105)  ((MOT(J),J=1,20),ISUFFIXE,IGRAMM,IMPOSE,ICHANGE)
   105 FORMAT(20R1,I10,I10,I10,I10)
      IF (MOT(1).EQ. 1R-)  GOTO 99$
      GOTO 55
    50 CALL BUFFER IN(1,1,MOT,40,ITEST)
    51 GOTO (51,52,53,54) ITEST
    54 WRITE(108,200)
   200 FORMAT(1H1,$ERREUR TRANSMISSION$)
      GOTO 99$
    53 WRITE(108,201)
   201 FORMAT(1H1,$ FIN DES DONNEES SUR BANDE$)
      WRITE(108,109)
      BANDE=.FALSE.
      GOTO 98$
    52 ISUFFIXE=MOT(40)
    55 INMOT=INMOT+1
       WRITE(108,106) INMOT,(MOT(J),J=1,20)
   106 FORMAT(1HO,I10,1H;,20R1)
      WRITE(108,110)  IMPOSE,ISUFFIXE,ICHANGE,IGRAMM
   110 FORMAT(1HO,$IMP=$,I5,5X,$SUF=$,I5,5X,$CHANGE=,$I5,5X,$GRAM=$,I5)
      IF(MOT(1).GE.176)  GOTO 22
C  LE MOT NE COMMENCE PAS PAR UNE LETTRE OU UN CHIFFRE.
```

```
      WRITE(108,108)
  108 FORMAT(1HO,50X,$**********MOT INCORRECT$)
      INMOT=INMOT-1
      GOTO 98$
   22 CONTINUE
C     CALCUL DE LONGMOT=PLACE DU DERNIER CARACTERE DE LA CHAINE
C-------------------------------------------------------------------
      M=1
   25 IF (MOT(M).EQ.1R )  GOTO 23
      M=M+1
      GOTO 25
   23 IF(MOT(M+1).EQ.1R )  GOTO 24
      M=M+2
      GOTO 25
   24 LONGMOT=M-1
X     WRITE (108,1007)  LONGMOT
X1007 FORMAT(1HO,$LONGUEUR DE LA CHAINE =$,I5)
C-------------------------------------------------------------------
C-------------------------------------------------------------------
C-----------------INSERTION DU STEM DANS STOCK.STEMS-----------------
C-------------------------------------------------------------------
      F=.FALSE.
      FX=.FALSE.
      M=ICHAIN=1
    9 LMOT=MOT(M)
      IF (M.EQ. LONGMOT+1 ) GOTO 1
    8 LCHAIN=CHAIN(ICHAIN)
      IF (LCHAIN.EQ.LMOT)   GOTO 2
      IF (LCHAIN.NE.1R/)   GOTO 7
C     SAUT DU /
X     WRITE  (108,1000) ICHAIN
X1000  FORMAT (1HO,$SAUT DE /  ICHAIN= $,I5)
      ICHAIN=ICHAIN+1
      GOTO 8
C     LES 2 CHAINES CONCORDENT
    2 ICHAIN=ICHAIN+1
      M=M+1
      GOTO 9
    7 IF (LCHAIN.EQ.1R*)  GOTO 3
      IF(LCHAIN.LT.LMOT)GOTO 17
C---- CAS OU CHAIN(ICHAIN)>MOT(M) : ON INSERE LE RESTE DU STEM **
X     WRITE (108,1010) ICHAIN,LCHAIN,LMOT
X1010 FORMAT(1HO,$CHAIN(ICHAIN).GT.LMOT,ICHAIN=$,I5,$CHAIN(ICHAIN)=$,1R1
X    =,$MOT(M)=$,1R1)
      FX=.TRUE.
      IALTERNANT=0
      GOTO 15
   17 IADR=ADR(ICHAIN)
      IF(IADR.NE.0)  GOTO 4
C-----CAS 3.1:RECHERCHER LE * SUIVANT ET INSERER LE RESTE DU STEM + *
X     WRITE (108,1003)  ICHAIN
X1003 FORMAT (1HO,$CAS 3.1  ICHAIN=$,I5)
      IALTERNANT=ICHAIN
      GOTO 10
C-----CAS2:REMPLACER LES * PAR DES / ET INSERER LE RESTE DU STEM + *
```

```
112:      3 CHAIN(ICHAIN)=1R/
113:        IALTERNANT=0
114: X      WRITE(108,1004) ICHAIN
115: X1004 FORMAT (1HO,$CAS 2  ICHAIN=  $,I5)
116:     31 ICHAIN=ICHAIN+1
117:        IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
118:        CHAIN(ICHAIN)=1R/
119:        GOTO 31
120: C-----CAS 3.2:RECHERCHER LA PLACE DE L'ALTERNANT
121:      4 LL=CHAIN(IADR)
122: X      WRITE(108,1005)  IADR
123: X1005 FORMAT(1HO,$RECHERCHE ALTERNANT  IADR=$,I5)
124:        IF(LL.LT.LMOT)  GOTO 16
125:        IF(LL.GT.LMOT)  GOTO 18
126: X      WRITE (108,1008) ICHAIN,IADR
127: X1008 FORMAT (1HO,$CHAIN(ICHAIN)=LMOT     ICHAIN=$,I5,10X,$IADR=$,I5)
128:        ICHAIN=IADR
129:        GOTO 2
130:     16 ICHAIN=IADR
131:        GOTO 17
132:     18 F=.TRUE.
133:        IALTERNANT=ICHAIN
134:        ICHAIN=IADR
135:        GOTO 15
136:      1 CONTINUE
137: C-----CAS 1: LE STEM EST CONTENU DANS UN STEM DE STOCK.STEMS
138: X      WRITE(108,1006 )  ICHAIN
139: X1006 FORMAT(1HO,$ STEM CONTENU DANS STOCK.STEMS  ICHAIN= $,I5)
140:        LCHAIN=CHAIN(ICHAIN)
141:        IF(LCHAIN.EQ.1R/)  GOTO 21
142:        IF(LCHAIN.EQ.1R*)  GOTO 21
143:        IALTERNANT=0
144:        LONGMOT=LONGMOT+1
145:        MOT(LONGMOT)=1R/
146:        GOTO 20
147: C      RECHERCHE DE L'* SUIVANT.
148: C      ICHAIN SERA L'ADRESSE DU CARACTERE SUIVANT LE DERNIER *
149:     10 ICHAIN=ICHAIN+1
150:        IF(CHAIN(ICHAIN).EQ.1R*)  GOTO 5
151:        IF(ICHAIN.GT.CHAIN(0))  GOTO 6
152:        GOTO 10
153:      6 WRITE (108,14)
154:     14 FORMAT(1HO,$ERREUR RECHERCHE *$)
155:        GOTO 29
156:      5 CONTINUE
157: X      WRITE(108,1001)  ICHAIN
158: X1001 FORMAT(1HO,$RECHERCHE DU PREMIER  *    ICHAIN= $,I5)
159: C      RECHERCHE DU DERNIER *
160:     32 ICHAIN=ICHAIN+1
161:        IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
162:        GOTO 32
163: C      ON AJOUTE UN * A LA CHAINE AVANT INSERTION
164:     15 LONGMOT=LONGMOT+1
165:        MOT(LONGMOT)=1R*
166: C      INSERTION DES CARACTERES M A LONGMOT DE MOT
167: C      LE PREMIER CARACTERE A DECALER DE CHAIN A POUR ADRESSE ICHAIN
```

```
168;     20 IDECAL=LONGMOT-M+1
169; X      WRITE(108,1002)  M,LONGMOT,ICHAIN
170; X1002 FORMAT(1HO,$INSERTION CHAINE,M=$,I5,$LONGMOT=$,I8,$   ICHAIN=$I5)
171;        II=CHAIN(0)
172;        CHAIN(0)=II+IDECAL
173;        DO 11 JJ=II,ICHAIN,-1
174;        ADR(JJ+IDECAL)=ADR(JJ)
175;     11 CHAIN(JJ+IDECAL)=CHAIN(JJ)
176;        DO 12 JJ=0,IDECAL-1
177;        ADR(ICHAIN+JJ)=0
178;     12 CHAIN(ICHAIN+JJ)=MOT(M+JJ)
179;        DO 13 JJ=1,II+IDECAL
180;     13 IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+IDECAL
181; C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
182;        IF(F)   GOTO 19
183; C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
184;        IF (FX) ADR(ICHAIN)=ICHAIN + IDECAL
185;        IF(IALTERNANT.NE.0)  ADR(IALTERNANT)=ICHAIN
186;        GOTO 40
187; C      CHAINAGE DANS LE CAS 3.2
188;     19 ADR(ICHAIN)=ADR(IALTERNANT)
189;        ADR(IALTERNANT)=ICHAIN
190;     40 ICHAIN=ICHAIN+IDECAL-1
191;        GOTO 28
192; C-------
193;     21 CONTINUE
194;        F=.FALSE.
195; C      ON DEVRA VERIFIER S'IL NE S'AGIT PAS D'UN NOUVEAU CODE INTRODUIT
196; C        POUR UN STEM FISTOCK.STEMS .
197; X      WRITE(108,1011)
198; X1011 FORMAT(1HO,$LE STEM FIGURE DEJA DANS DIC-F-COD$)
199;        GOTO 30
200;     41 ICHAIN=ICHAIN+1
201; X      WRITE(108,1009)  ICHAIN
202; X1009 FORMAT(1HO,$ICHAIN=...$,I5)
203;        IF(ADR(ICHAIN).GE.0)  GOTO 38
204;     30 IA=ADR(ICHAIN)
205;        II=IA/1000
206;        ISUF=-IA+II*1000
207;        IF(ISUFFIXE.LT.ISUF)  GOTO 42
208;        IF(ISUF.EQ.ISUFFIXE)  GOTO 33
209;        GOTO 41
210;     33 IF(ICHANGE.EQ.0)  GOTO 34
211;        F=.TRUE.
212;        IN=-IA/10000
213;        IF(IN.NE.ICHANGE)  GOTO 41
214;        GOTO 28
215;     34 IF(IMPOSE.EQ.0)  GOTO 36
216; C      DECALAGE DE 1.
217;     42 IJ=CHAIN(0)
218;        DO 35 JJ=IJ,ICHAIN,-1
219;        ADR(JJ+1)=ADR(JJ)
220;     35 CHAIN(JJ+1)=CHAIN(JJ)
221;     45 CHAIN(0)=IJ+1
222; C      REMISE EN ORDRE DES POINTEURS.
223;        DO 39 JJ=1,CHAIN(0)
```

```
224:     39 IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+1
225:     28 CONTINUE
226: C***
227: C      INTRODUCTION DU MOT
228:   1013 FORMAT(1HO,$IMPOSE=$,I5,5X,$ICHAIN=$,I5,5X,1R1,5X,$ADR=$,I10)
229:        IF (IGRAMM.EQ.2)  GOTO   26
230:        IF (IGRAMM.EQ.9)  GOTO  27
231:        IF (IGRAMM.EQ.1)  GOTO  27
232: C      CAS OU IL S'AGIT D'UN MOT-CLE.
233:        IF (IMPOSE.EQ.0)  GOTO 47
234:        ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
235:        WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
236:        GOTO 29
237:     47 ADR(0)=ADR(0)-3
238:        ADR(ICHAIN)=ADR(0)*100000-ISUFFIXE
239:        GOTO 29
240:     27 CONTINUE
241: C      CAS OU IL S'AGIT D'UN MOT-VIDE , OU D'UN SUFFIXE
242:        ADR(ICHAIN)=-IGRAMM*10000-ISUFFIXE
243:         IF (IMPOSE.EQ.0)  GOTO 29
244:        WRITE(108,48)
245:     48 FORMAT(1HO,$ON A CHERCHE A DONNER UN N. A UN SUFFIXE OU MOT VIDE$)
246:        GOTO 29
247:     26 CONTINUE
248: C      CAS OU IL S'AGIT D'UN MOT GRAMMATICAL.
249:        IF(IMPOSE.EQ.0)  GOTO 49
250:        ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
251:        WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
252:        GOTO 29
253:     49 IMAXGR=IMAXGR+1
254:        ADR(ICHAIN)=-IMAXGR*100000-ISUFFIXE-IGRAMM*10000
255:        GOTO 29
256: C***
257:     36 CONTINUE
258:        WRITE(108,37)
259:     37 FORMAT(1HO,$LE MOT FIGURE DEJA DANS STOCK-STEMS$)
260: X      WRITE(108,1015)  ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
261: X1015 FORMAT(1HO,$            ICHAIN=$,I5,5X,1R1,5X,$ADR(ICHAIN)=$,I10)
262:        GOTO 29
263:     38 CONTINUE
264: X      WRITE(108,1014)
265: X1014 FORMAT(1HO,$ON A EXPLORE TOUS LES CODES DU STEM$)
266:        IF( F )  GOTO 46
267:        IJ=CHAIN(0)
268:        DO 44 JJ=IJ,ICHAIN-1,-1
269:        ADR(JJ+1)=ADR(JJ)
270:     44 CHAIN(JJ+1)=CHAIN(JJ)
271:        GOTO 45
272:     46 CONTINUE
273:        WRITE(108,43)
274:     43 FORMAT(1HO,$ ON A VOULU CHANGER UN N. QUI NIEXISTE PAS$)
275:   29   CONTINUE
276: C-------------------------------------------------------------------
277: C      FIN INSERTION DU MOT DANS DICFCOD-----------------------------
278: C-------------------------------------------------------------------
279:        WRITE(108,103) CHAIN(0)
```

```
  103 FORMAT(1HO,$*** LONGUEUR DE LA CHAINE =  $,I10)
      GOTO 98$
  99$ CONTINUE
      END LABELS
C*********
      WRITE(108,100)
  100 FORMAT(1H1)
      WRITE(108,1)
    1 FORMAT(1H1,20X,$CARACTERISTIQUES DE LA CHAINE$)
      WRITE(108,2) CHAIN(0)
    2 FORMAT(1HO,$LONGUEUR DE LA CHAINE=$,I10)
      WRITE(108,3)  ADR(0)/3
    3 FORMAT(1HO,$NOMBRE DE MOTS PIVOTS=$,I10)
      WRITE(108,4) IMAXGR
    4 FORMAT(1HO,$NOMBRE DE MOTS GRAMMATICAUX=$,I10)
      WRITE(108,100)
       WRITE(108,101)
  101 FORMAT(1HO,20X,$PRINCIPALES CARACTERISTIQUES DE LA CHAINE$)
       WRITE(108,102)  CHAIN(0)
  102 FORMAT(1HO,$LONGUEUR DE LA CHAINE=  $,I5)
      DO 10  I=1,CHAIN(0)
   10 IF(ADR(I).EQ.0)  IZER=IZER+1
       WRITE(108,105) IZER
  105 FORMAT(1HO,$NOMBRE D'ELEMENTS DE ADR NULS  $,I5)
      IXY=CHAIN(0)-IZER
       WRITE (108,107)  IXY
  107 FORMAT(1HO,$NOMBRE D'ELEMENTS NON NULS DANS ADR  $,I5)
      XY=(IXY*100.)/CHAIN(0)
      WRITE(108,106)  XY
  106 FORMAT(1HO,$POURCENTAGE D'ELEMENTS NON NULS DANS ADR  $,F8.3)
C     FIN DES CALCULS DES PARAMETRES DE CHAIN-ADR----------------------
C     -----------------------------------------------------------------
C     $$$$$$$$$$$$
      WRITE(108,100)
C     SORTIE DE LA CHAINE MODIFIEE
C-----------------------------------------------------------------------
      END LABELS
      WRITE (2)  CHAIN
      WRITE (2)  ADR
      ENDFILE 2
      REWIND 2
       STOP
      END
```

```
      INTEGER MOT(40),ADR(0:6000),CHAIN(0:6000)
      COMMON ADR,CHAIN
      LOGICAL F,FX
C*****************************************************************
C
C              MISE A JOUR DU DICTIONNAIRE CARREZ.
C
C*****************************************************************
C   --LECTURE  DU  DICTIONNAIRE A METTRE A JOUR.
      READ(3) CHAIN
      READ(3) ADR
C******************
      WRITE(108,109)
  109 FORMAT (1H1)
C     LECTURE ET SORTIE DU MOT
   98$ CONTINUE
      WRITE(108,104)
  104 FORMAT(1HO,$*************************$)
      ISUFFIXE=IMPOSE=IGRAMM=ICHANGE=0
      READ(105,105)  ((MOT(J),J=1,20),ISUFFIXE,IGRAMM,IMPOSE,ICHANGE)
  105 FORMAT(20R1,I10,I10,I10,I10)
      IF (MOT(1).EQ. 1R-)  GOTO 99$
      INMOT=INMOT+1
       WRITE(108,106) INMOT,(MOT(J),J=1,20)
  106 FORMAT(1HO,I10,1H:,20R1)
      WRITE(108,110)  IMPOSE,ISUFFIXE,ICHANGE,IGRAMM
  110 FORMAT(1HO,$IMP=$,I5,5X,$SUF=$,I5,5X,$CHANGE=,$I5,5X,$GRAM=$,I5)
      IF(MOT(1).GE.176)  GOTO 22
C  LE MOT NE COMMENCE PAS PAR UNE LETTRE OU UN CHIFFRE.
      WRITE(108,108)
  108 FORMAT(1HO,50X,$*********************MOT INCORRECT$)
      INMOT=INMOT-1
      GOTO 98$
   22 CONTINUE
C     CALCUL DE LONGMOT=PLACE DU DERNIER CARACTERE DE LA CHAINE
C-----------------------------------------------------------------
      M=1
   25 IF (MOT(M).EQ.1R )  GOTO 23
      M=M+1
      GOTO 25
   23 IF(MOT(M+1).EQ.1R )  GOTO 24
      M=M+2
      GOTO 25
   24 LONGMOT=M-1
X     WRITE (108,1007)  LONGMOT
X1007 FORMAT(1HO,$LONGUEUR DE LA CHAINE =$,I5)
C-----------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------------------------------------------------------
C-----------------INSERTION DU STEM DANS STOCK.STEMS-------------
C-----------------------------------------------------------------
      F=.FALSE.
      FX=.FALSE.
      M=ICHAIN=1
    9 LMOT=MOT(M)
```

```
56:         IF (M.EQ. LONGMOT+1 ) GOTO 1
57:       8 LCHAIN=CHAIN(ICHAIN)
58:         IF (LCHAIN.EQ.LMOT)   GOTO 2
59:         IF (LCHAIN.NE.1R/)    GOTO 7
60: C       SAUT DU /
61: X       WRITE  (108,1000) ICHAIN
62: X1000   FORMAT (1HO,$SAUT DE /  ICHAIN= $,I5)
63:         ICHAIN=ICHAIN+1
64:         GOTO 8
65: C       LES 2 CHAINES CONCORDENT
66:       2 ICHAIN=ICHAIN+1
67:         M=M+1
68:         GOTO 9
69:       7 IF (LCHAIN.EQ.1R*)  GOTO 3
70:         IF(LCHAIN.LT.LMOT)GOTO 17
71: C---- CAS OU CHAIN(ICHAIN)>MOT(M) : ON INSERE LE RESTE DU STEM +*
72: X       WRITE (108,1010) ICHAIN,LCHAIN,LMOT
73: X1010 FORMAT(1HO,$CHAIN(ICHAIN).GT.LMOT,ICHAIN=$,I5,$CHAIN(ICHAIN)=$,1R1
74: X     -,$MOT(M)=$,1R1)
75:         FX=.TRUE.
76:         IALTERNANT=0
77:         GOTO 15
78:      17 IADR=ADR(ICHAIN)
79:         IF(IADR.NE.0)  GOTO 4
80: C-----CAS 3.1:RECHERCHER LE * SUIVANT ET INSERER LE RESTE DU STEM + *
81: X       WRITE (108,1003)  ICHAIN
82: X1003 FORMAT (1HO,$CAS 3.1  ICHAIN=$,I5)
83:         IALTERNANT=ICHAIN
84:         GOTO 10
85: C-----CAS2:REMPLACER LES * PAR DES / ET INSERER LE RESTE DU STEM + *
86:       3 CHAIN(ICHAIN)=1R/
87:         IALTERNANT=0
88: X       WRITE(108,1004) ICHAIN
89: X1004 FORMAT (1HO,$CAS 2  ICHAIN=  $,I5)
90:      31 ICHAIN=ICHAIN+1
91:         IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
92:         CHAIN(ICHAIN)=1R/
93:         GOTO 31
94: C-----CAS 3.2:RECHERCHER LA PLACE DE LIALTERNANT
95:       4 LL=CHAIN(IADR)
96: X       WRITE(108,1005)  IADR
97: X1005 FORMAT(1HO,$RECHERCHE ALTERNANT  IADR=$,I5)
98:         IF(LL.LT.LMOT)  GOTO 16
99:         IF(LL.GT.LMOT)  GOTO 18
00: X       WRITE (108,1008) ICHAIN,IADR
01: X1008 FORMAT (1HO,$CHAIN(ICHAIN)=LMOT      ICHAIN=$,I5,10X,$IADR=$,I5)
02:         ICHAIN=IADR
03:         GOTO 2
04:      16 ICHAIN=IADR
05:         GOTO 17
06:      18 F=.TRUE.
07:         IALTERNANT=ICHAIN
08:         ICHAIN=IADR
09:         GOTO 15
10:       1 CONTINUE
11: C-----CAS 1: LE STEM EST CONTENU DANS UN STEM DE STOCK.STEMS
```

```
112: X      WRITE(108,1006 )  ICHAIN
113: X1006 FORMAT(1HO,$ STEM CONTENU DANS STOCK.STEMS  ICHAIN= $,I5)
114:        LCHAIN=CHAIN(ICHAIN)
115:        IF(LCHAIN.EQ.1R/)  GOTO 21
116:        IF(LCHAIN.EQ.1R*)  GOTO 21
117:        IALTERNANT=0
118:        LONGMOT=LONGMOT+1
119:        MOT(LONGMOT)=1R/
120:        GOTO 20
121: C      RECHERCHE DE LI* SUIVANT.
122: C      ICHAIN SERA LIADRESSE DU CARACTERE SUIVANT LE DERNIER *
123:    10 ICHAIN=ICHAIN+1
124:        IF(CHAIN(ICHAIN).EQ.1R*)  GOTO 5
125:        IF(ICHAIN.GT.CHAIN(0))  GOTO 6
126:        GOTO 10
127:     6 WRITE (108,14)
128:    14 FORMAT(1HO,$ERREUR RECHERCHE *$)
129:        GOTO 29
130:     5 CONTINUE
131: X      WRITE(108,1001)  ICHAIN
132: X1001 FORMAT(1HO,$RECHERCHE DU PREMIER  *    ICHAIN= $,I5)
133: C      RECHERCHE DU DERNIER *
134:    32 ICHAIN=ICHAIN+1
135:        IF(CHAIN(ICHAIN).NE.1R*)  GOTO 15
136:        GOTO 32
137: C      ON AJOUTE UN * A LA CHAINE AVANT INSERTION
138:    15 LONGMOT=LONGMOT+1
139:        MOT(LONGMOT)=1R*
140: C      INSERTION DES CARACTERES M A LONGMOT DE MOT
141: C      LE PREMIER CARACTERE A DECALER DE CHAIN A POUR ADRESSE ICHAIN
142:    20 IDECAL=LONGMOT-M+1
143: X      WRITE(108,1002)  M,LONGMOT,ICHAIN
144: X1002 FORMAT(1HO,$INSERTION CHAINE,M=$,I5,$LONGMOT=$,I8,$   ICHAIN=$I5)
145:        II=CHAIN(0)
146:        CHAIN(0)=II+IDECAL
147:        DO 11 JJ=II,ICHAIN,-1
148:        ADR(JJ+IDECAL)=ADR(JJ)
149:    11 CHAIN(JJ+IDECAL)=CHAIN(JJ)
150:        DO 12 JJ=0,IDECAL-1
151:        ADR(ICHAIN+JJ)=0
152:    12 CHAIN(ICHAIN+JJ)=MOT(M+JJ)
153:        DO 13 JJ=1,II+IDECAL
154:    13 IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+IDECAL
155: C      CREATION DIUN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
156:        IF(F)   GOTO 19
157: C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
158:        IF (FX) ADR(ICHAIN)=ICHAIN + IDECAL
159:        IF(IALTERNANT.NE.0)  ADR(IALTERNANT)=ICHAIN
160:        GOTO 40
161: C      CHAINAGE DANS LE CAS 3.2
162:    19 ADR(ICHAIN)=ADR(IALTERNANT)
163:        ADR(IALTERNANT)=ICHAIN
164:    40 ICHAIN=ICHAIN+IDECAL-1
165:        GOTO 28
166: C-------
167:    21 CONTINUE
```

```
168:         F=.FALSE.
169: C       ON DEVRA VERIFIER S'IL NE S'AGIT PAS D'UN NOUVEAU CODE INTRODUIT
170: C         POUR UN STEM FISTOCK.STEMS .
171: X       WRITE(108,1011)
172: X1011   FORMAT(1HO,$LE STEM FIGURE DEJA DANS DIC-F_COD$)
173:         GOTO 30
174:      41 ICHAIN=ICHAIN+1
175: X       WRITE(108,1009)  ICHAIN
176: X1009   FORMAT(1HO,$ICHAIN=...$,I5)
177:         IF(ADR(ICHAIN).GE.0)  GOTO 38
178:      30 IA=ADR(ICHAIN)
179:         II=IA/1000
180:         ISUF=-IA+II*1000
181:         IF(ISUFFIXE.LT.ISUF)  GOTO 42
182:         IF(ISUF.EQ.ISUFFIXE)  GOTO 33
183:         GOTO 41
184:      33 IF(ICHANGE.EQ.0)  GOTO 34
185:         F=.TRUE.
186:         IN=-IA/10000
187:         IF(IN.NE.ICHANGE)  GOTO 41
188:         GOTO 28
189:      34 IF(IMPOSE.EQ.0)  GOTO 36
190: C       DECALAGE DE 1.
191:      42 IJ=CHAIN(0)
192:         DO 35 JJ=IJ,ICHAIN,-1
193:         ADR(JJ+1)=ADR(JJ)
194:      35 CHAIN(JJ+1)=CHAIN(JJ)
195:      45 CHAIN(0)=IJ+1
196: C       REMISE EN ORDRE DES POINTEURS.
197:         DO 39 JJ=1,CHAIN(0)
198:      39 IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+1
199:      28 CONTINUE
200: C***
201: C       INTRODUCTION DU MOT
202:    1013 FORMAT(1HO,$IMPOSE=$,I5,5X,$ICHAIN=$,I5,5X,1R1,5X,$ADR=$,I10)
203:         IF (IGRAMM.EQ.2)  GOTO   26
204:         IF (IGRAMM.EQ.9)  GOTO  27
205:         IF (IGRAMM.EQ.1)  GOTO  27
206: C       CAS OU IL S'AGIT D'UN MOT-CLE.
207:         IF (IMPOSE.EQ.0)  GOTO 47
208:         ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
209:         WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
210:         GOTO 29
211:      47 ADR(0)=ADR(0)-1
212:         ADR(ICHAIN)=ADR(0)*100000-ISUFFIXE
213:         GOTO 29
214:      27 CONTINUE
215: C       CAS OU IL S'AGIT D'UN MOT-VIDE , OU D'UN SUFFIXE
216:         ADR(ICHAIN)=-IGRAMM*10000-ISUFFIXE
217:          IF (IMPOSE.EQ.0)  GOTO 29
218:         WRITE(108,48)
219:      48 FORMAT(1HO,$ON A CHERCHE A DONNER UN N. A UN SUFFIXE OU MOT VIDE$)
220:         GOTO 29
221:      26 CONTINUE
222: C       CAS OU IL S'AGIT D'UN MOT GRAMMATICAL.
223:         IF(IMPOSE.EQ.0)  GOTO 49
```

```
224:        ADR(ICHAIN)=-IMPOSE*100000-ISUFFIXE
225:        WRITE(108,1013) IMPOSE, ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
226:        GOTO 29
227:     49 IMAXGR=IMAXGR+1
228:        ADR(ICHAIN)=-IMAXGR*100000-ISUFFIXE-IGRAMM*10000
229:        GOTO 29
230: C***
231:     36 CONTINUE
232:        WRITE(108,37)
233:     37 FORMAT(1HO,$LE MOT FIGURE DEJA DANS STOCK=>ITEMS$)
234: X      WRITE(108,1015)  ICHAIN,CHAIN(ICHAIN),ADR(ICHAIN)
235: X1015 FORMAT(1HO,$           ICHAIN=$,I5,5X,1R1,8X,$ADR(ICHAIN)=$,I10)
236:        GOTO 29
237:     38 CONTINUE
238: X      WRITE(108,1014)
239: X1014 FORMAT(1HO,$ON A EXPLORE TOUS LES CODES DU STEM$)
240:        IF( F )  GOTO 46
241:        IJ=CHAIN(0)
242:        DO 44 JJ=IJ,ICHAIN-1,-1
243:        ADR(JJ+1)=ADR(JJ)
244:     44 CHAIN(JJ+1)=CHAIN(JJ)
245:        GOTO 45
246:     46 CONTINUE
247:        WRITE(108,43)
248:     43 FORMAT(1HO,$ ON A VOULU CHANGER UN N. QUI N'EXISTE PAS$)
249:   29   CONTINUE
250: C-----------------------------------------------------------------
251: C      FIN INSERTION DU MOT DANS DICFCOD---------------------------
252: C-----------------------------------------------------------------
253:        WRITE(108,103) CHAIN(0)
254:    103 FORMAT(1HO,$*** LONGUEUR DE LA CHAINE =  $,I10)
255:        GOTO 98$
256:    99$ CONTINUE
257:        END LABELS
258: C**********
259:        WRITE(2) CHAIN
260:        WRITE(2) ADR
261:        ENDFILE 2
262:        REWIND 2
263:        END LABELS
264: C-----------------------------------------------------------------
265:        WRITE(108,100)
266:    100 FORMAT(1H1)
267:        WRITE(108,1)
268:      1 FORMAT(1H1,20X,$CARACTERISTIQUES DE LA CHAINE$)
269:        WRITE(108,2) CHAIN(0)
270:      2 FORMAT(1HO,$LONGUEUR DE LA CHAINE=$,I10)
271:        WRITE(108,3)  ADR(0)
272:      3 FORMAT(1HO,$NOMBRE DE MOTS PIVOTS=$,I10)
273:        WRITE(108,4) IMAXGR
274:      4 FORMAT(1HO,$NOMBRE DE MOTS GRAMMATICAUX=$,I10)
275:        WRITE(108,100)
276:         WRITE(108,101)
277:    101 FORMAT(1HO,20X,$PRINCIPALES CARACTERISTIQUES DE LA CHAINE$)
278:         WRITE(108,102)  CHAIN(0)
279:    102 FORMAT(1HO,$LONGUEUR DE LA CHAINE=  $,I5)
```

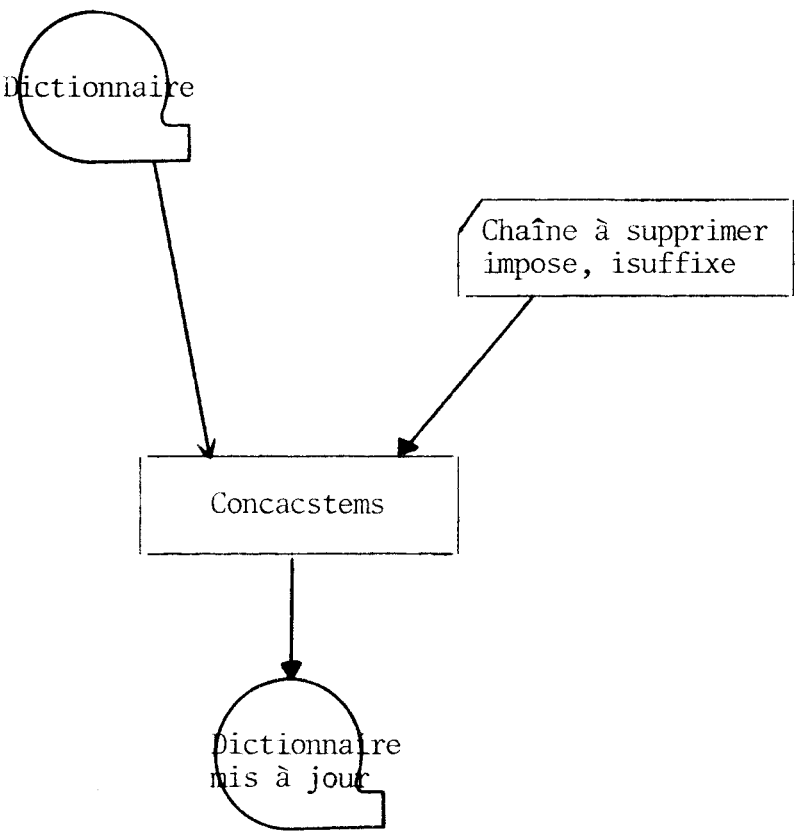
```
      DO 10  I=1,CHAIN(0)
   10 IF(ADR(I).EQ.0)  IZER=IZER+1
       WRITE(108,105) IZER
  105 FORMAT(1HO,$NOMBRE D'ELEMENTS DE ADR NULS  $,I5)
      IXY=CHAIN(0)-IZER
       WRITE (108,107)  IXY
  107 FORMAT(1HO,$NOMBRE D'ELEMENTS NON NULS DANS ADR  $,I5)
      XY=(IXY*100.)/CHAIN(0)
      WRITE(108,106)  XY
  106 FORMAT(1HO,$POURCENTAGE D'ELEMENTS NON NULS DANS ADR  $,F8.3)
C     FIN DES CALCULS DES PARAMETRES DE CHAIN-ADR----------------------
C     ---------------------------------------------------------------
C     $$$$$$$$$$$$
      WRITE(108,100)
C     SORTIE DE LA CHAINE MODIFIEE
       STOP
      END
```

4. PROGRAMME DE SUPPRESSION D'UNE CHAINE : "CONCACSTEMS"

Les données nécessaires sont, ici :

- la chaîne à retirer, telle qu'elle figure dans le dictionnaire ; la chaîne du stem d'un mot, par exemple
- le code du suffixe, et le numéro du concept.

L'ordinogramme est le suivant :



```
      INTEGER ADR(0:20000),CHAIN(0:20000)
      COMMON ADR,CHAIN
      INTEGER MOT(30),ADRCHAIN(30),ADRADR(30)
      LOGICAL PLUR,SLATCH
C*************************************************************************
C
C-------------------------- CONCACSTEMS ----------------------------------
C
C*************************************************************************
C     LECTURE DE ADR ET CHAIN.
C-------------------------------------------------------------------------
      CALL BUFFER IN(1,1,ADR,3501,ITEST,N)
    1 GOTO (1,2,7,3) ITEST
    7 WRITE(108,103)
    2 WRITE(108,100) N
  100 FORMAT(1H1,$ENTREE DE ADR$,I10)
      CALL BUFFER IN(1,1,CHAIN,3501,ITEST,N)
    4 GOTO (4,5,8,3)  ITEST
    8 WRITE(108,103)
    5 WRITE(108,101) N
  101 FORMAT(1H0,$ENTREE DE CHAIN$,I10)
      GOTO 6
    3 WRITE(108,102)
  102 FORMAT(1H0,$  ERREUR$)
  103 FORMAT(1H0,$  FIN DE FICHIER$)
      PAUSE 1
    6 CONTINUE
      WRITE(108,1111)
 1111 FORMAT(1H1,$ FIN ENTREE DU DICTIONNAIRE$)
C------------FIN DE LA LECTURE DE LA CHAINE
      END LABELS
C$$$$$$$$$$$$$$
  999 CONTINUE
C     LECTURE DES MOTS
  999 READ(105,104) ((MOT(J),J=1,20),IMPOSE,ISUFFIXE)
  104 FORMAT(20(R1),I10,I10)
      IF(MOT(1).EQ.1R ) GOTO 1005
      WRITE(108,105) ((MOT(J),J=1,20),IMPOSE,ISUFFIXE)
  105 FORMAT(1H0,20R1,5X,$IMPOSE =$,I5,5X,$ISUFFIXE =$,I5)
      IADR=IADRSLATCH=0
      II=CHAIN(0)
      M=ICHAIN=1
    9 LMOT=MOT(M)
X     WRITE (108,201) L,ICHAIN
X201  FORMAT (1H0,$M= $,I4,10X,$ICHAIN= $,I4)
      IF((LMOT.EQ.1R ).AND.(MOT(M+1).EQ.1R )) GOTO 1
    8 LCHAIN=CHAIN(ICHAIN)
      IF(LCHAIN.EQ.LMOT) GOTO 2
      IF (LCHAIN.NE. 1R/) GOTO 7
X     WRITE (108,202)   ICHAIN
X202  FORMAT (1H0,$SAUT DU /,ICHAIN= $,I5)
      IADRSLATCH=ICHAIN
      ICHAIN=ICHAIN+1
      GOTO 8
    7 IF(LCHAIN.EQ.1R*) GOTO 3
    4 IADR=ADR(ICHAIN)
X     WRITE (108,203)    IADR
  203 FORMAT (1H0$SUIVI DE LA CHAINE$,I5)
      IF(IADR.EQ.0) GOTO 3
```

```
      ICHAIN=IADR
      IF(CHAIN(ICHAIN).NE.LMOT)  GOTO 4
    2 ADRADR(M)=ADR(ICHAIN)
      ADRCHAIN(M)=ICHAIN
      ICHAIN=ICHAIN+1
      M=M+1
      GOTO 9
    1 CONTINUE
X     WRITE (108,200)ICHAIN
X200  FORMAT(1HO,$SORTIE PAR 1     ICHAIN=$,I4)
      IF((CHAIN(ICHAIN).EQ.1R/).OR.(CHAIN(ICHAIN).EQ.1R*))   GOTO 5
    3 WRITE (108,100)
  100 FORMAT(1HO,$LE MOT NE FAIT PAS PARTIE DU DICTIONNAIRE$)
      GOTO 6
    5 CONTINUE
X     WRITE(108,204)
X204  FORMAT(1HO,$LE MOT FIGURE DANS LE DICTIONNAIRE$)
C     SORTIE DES TABLEAUX MOT,ADRCHAIN,ADRADR
X     WRITE(108,207)
X207  FORMAT(1HO,$TABLEAUX  MOT,ADRCHAIN,ADRADR$)
X     DO 206 I=1, M-1
X     WRITE(108,208) I,MOT(I),ADRCHAIN(I),ADRADR(I)
X208  FORMAT(1HO,I2,5X,1R1,6X,I4,5X,I4)
X206  CONTINUE

C-----CONCATENATION DE LA CHAINE-----------------------------------
C     ICHAIN CONTIENT L'ADRESSE DU CARACTERE DE FIN DE MOT:/ OU *
C     IADRSLATCH : ADRESSE DU DERNIER SLATCH RENCONTRE AVANT LA FIN
C        DE MOT
      SLATCH=.TRUE.
      PLUR=.FALSE.
      IF(CHAIN(ICHAIN+1).EQ.CHAIN(ICHAIN))  PLUR=.TRUE.
      IF(.NOT.PLUR)  GOTO 25
      WRITE(108,102) ICHAIN
  102 FORMAT(1HO,$PLUSIEURS MOTS ONT LE MEME STEM   ,ICHAIN=$,I5)
   25 IZ=-ADR(ICHAIN)
      IMP=IZ/10000
      IMPZ=IZ-IMP*10000
      IMPV=IMPZ/1000
      ISUF=IMPZ-IMPV*1000
X     WRITE(108,215)  ICHAIN,IMP,ISUF
X215  FORMAT(1HO,$ICHAIN=$,I5,5X,$N.STEM=$,I10,5X,$SUFFIXE=*,I10)
      IF((ISUF.EQ.ISUFFIXE).AND.(IMP.EQ.IMPOSE)) GOTO 23
      IF(CHAIN(ICHAIN+1).NE.CHAIN(ICHAIN))    GOTO 24
      ICHAIN=ICHAIN+1
      GOTO 25
   24 WRITE(108,26)
   26 FORMAT(1HO,$ERREUR RECHERCHE : MOT NE FIGURANT PAS DANS DIC$)
      GOTO 6

   23 CONTINUE
C       ON A RETROUVE LA CHAINE.
        IF(PLUR)  GOTO 27
C     CAS OU LE TERME DE FIN DE MOT EST UNIQUE
      IF(CHAIN(ICHAIN).EQ.1R*)  GOTO 10
C******CAS 1.
C     CAS OU LE CARACTERE DE FIN DE MOT EST UN /
X     WRITE (108,205)  CHAIN(ICHAIN)
X205  FORMAT(1HO,$LE CARACTERE DE FIN DE MOT EST UN $,1R1)
   27 IPREM=ICHAIN
```

```
      ICHAIN=ICHAIN+1
      GOTO 19
   10 CONTINUE
C******CAS 2.
C     CAS OU LE CARACTERE DE FIN DE MOT EST UNE *
X     WRITE(108,205)  CHAIN,ICHAIN)
      IPILE=0
      ICHAIN=ICHAIN+1
      DO 11 I=1,M-1
   11 IF(ADRADR(I).EQ.ICHAIN)   IPILE=I
X     WRITE(108,209) IADRSLATCH,IPILE,IADR
X209  FORMAT(1HO,$IADRSLATCH=$,I5,10X,$IPILE=$,I5,10X,$IADR=$,I5)
C     IADR: ADRESSE DU DERNIER CARACTERE CHAINE
C     IADRSLATCH: ADRESSE DU SLATCH
C     ADRCHAIN(IPILE): ADRESSE DU CARACTERE DONT LE ADR RENVOIT AU
C       CARACTERE D'ADRESSE ICHAIN+1
      IF(IPILE.EQ.0)  GOTO 12
C******CAS 2.1
C     IL EXISTE UN CARACTERE DONT LE ADR RENVOIT AU CARACT D'ADR ICHAIN+1
X     WRITE(108,211)  ADRCHAIN(IPILE)
X211  FORMAT(1HO,$ADRCHAIN(IPILE)=$,I5)
      IF(IADR.LE.ADRCHAIN(IPILE))  GOTO 13
C******CAS 2.1.1
X     WRITE(108,210)
X210  FORMAT(1HO,$IADR>ADRCHAIN(IPILE)$)
      WRITE(108,101)
      IPREM=IADR
  101 FORMAT(1HO,$ARBRE NON REGULIER$)
      ICHAIN=ICHAIN-1
      ADR(ICHAIN)=-0
      DO 22 I=1,II
   22 IF(ADR(I).EQ.IADR)  ADR(I)=0
      GOTO 19
C******CAS 2.1.2
   13 IF(IADRSLATCH.GT.ADRCHAIN(IPILE)) GOTO 14
C******CAS 2.1.2.1
      IPREM=ADRCHAIN(IPILE)
      GOTO 19
C******CAS 2.1.2.2
   14 IPREM=IADRSLATCH
      ICHAIN=ICHAIN-1
      ADR(ICHAIN)=ADR( IPREM)
      SLATCH=.FALSE.
      GOTO 19
   12 CONTINUE
C******CAS 2.2
C     IL N'EXISTE PAS DE CARACTERE DONT LE ADR RENVOIT AU CARACT D'ADR
C         ICHAIN
      IF(IADR.LE.IADRSLATCH)  GOTO 16
C******CAS 2.2.2
      DO 17 I=1,M
   17 IF(ADRCHAIN(I).EQ.IADR)  I1=I
      IPREM=IADR
      DO 18 I=I1,M
   18 IF(ADRADR(I).NE.0)  IPREM=I
      GOTO 19
C******CAS 2.2.1
   16 IPREM=IADRSLATCH
      ICHAIN=ICHAIN-1
      ADR(ICHAIN)=ADR( IPREM)
```

```
180:         SLATCH=.FALSE.
181:      19 CONTINUE
182:  C      DECALAGE DE LA CHAINE
183:  X      WRITE(108,212)
184:  X212   FORMAT(1HO,$DECALAGE DE LA CHAINE$)
185:  X      WRITE(108,213)  IPREM,ICHAIN
186:  X213   FORMAT(1HO,$IPREM=$,I5,5X,$ICHAIN=$,I5)
187:  C      LE PREMIER CARACTERE A DECALER DE CHAIN A POUR ADRESSE ICHAIN
188:         IDECAL=ICHAIN-IPREM
189:         CHAIN(0)=II+IDECAL
190:  X      WRITE(108,214)  CHAIN(0)
191:  X214   FORMAT(1HO,$LONGUEUR DE LA NOUVELLE CHAINE=$,I5)
192:         DO 20 J=ICHAIN,II+IDECAL
193:         CHAIN(J-IDECAL)=CHAIN(J)
194:      20 ADR(J-IDECAL)=ADR(J)
195:         DO 21 J=1,CHAIN(0)
196:         IF(ADR(J).GE.ICHAIN)  ADR(J)=ADR(J)-IDECAL
197:      21 IF(ADR(J).GE.CHAIN(0))  ADR(J)=0
198:         IF(SLATCH)  GOTO 6
199:  C      MODIFICATION DES CARACTERS DE FIN DE MOT .CAS 2.1.2)1 ET 2.2.1
200:         ICHAIN=IPREM-1
201:      28 IF(CHAIN(ICHAIN).NE.'R/')  GOTO 6
202:         CHAIN(ICHAIN)=CHAIN(ICHAIN+1)
203:         ICHAIN=ICHAIN-1
204:         GOTO 28
205:      6  CONTINUE
206:  C-------------------------------------------------------------------
207:          GOTO 99$
208:   100$ CONTINUE
209:         END LABELS
210:  C*******
211:         WRITE(108,2)
212:       2 FORMAT(1H1,$SORTIE DE LA CHAINE MODIFIEE$)
213:         DO 1 J=1,200
214:       1 WRITE(108,3) J,CHAIN(J),ADR(J)
215:       3 FORMAT(1HO,I5,5X,1R1,5X,I7)
216:         STOP
217:          END
```

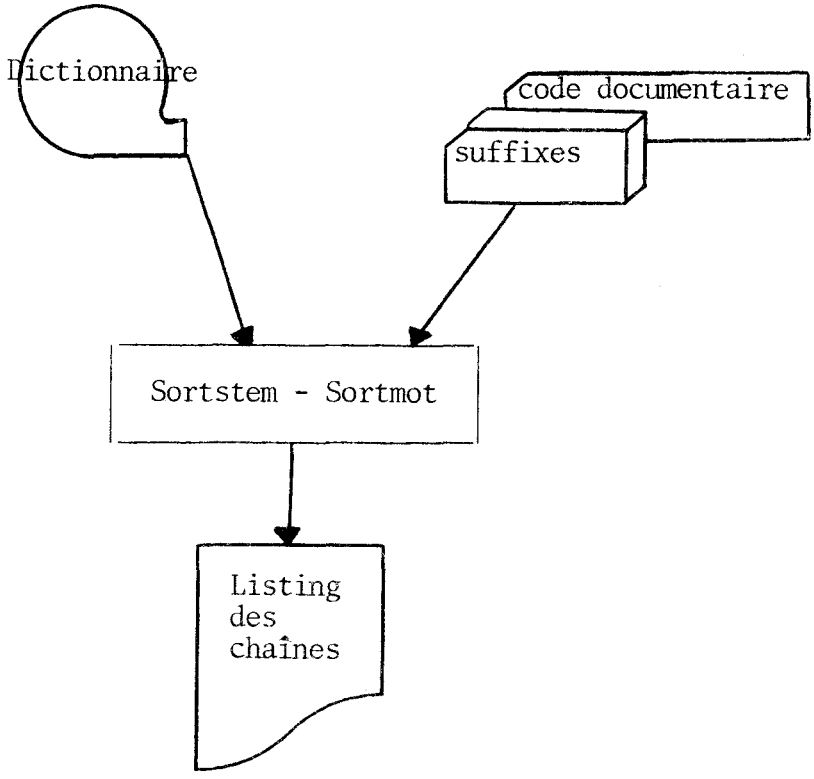
5. PROGRAMME DE SORTIE DES CHAINES DU DICTIONNAIRE. "SORTSTEM - SORTMOT"

Dans ce programme, un certain nombre de suffixes sont rangés dans le tableau TABSUF, et leur code dans NSUF, pour sortir avec un stem de mot les suffixes qui lui sont associés.

Le tableau MARK va servir à répéter les classes de suffixe qui ont été utilisées au moins une fois dans le dictionnaire.

Les données nécessaires pour ce programme comportent, outre les suffixes qui seront placés dans TABSUF, une ou plusieurs cartes contenant le code documentaire des ensembles de chaînes que l'on veut sortir.

L'ordinogramme est très simple :



```
 1; C******************
 2;       INTEGER MARK(400)
 3;        INTEGER NSUF(700),TABSUF(700,3)
 4;       INTEGER MOT(0:30),ADRCHAIN(0:30)
 5;       INTEGER ADR(0:6000),CHAIN(0:6000)
 6;       COMMON ADR,CHAIN
 7; C******************
 8; C     LECTURE DES SUFFIXES.
 9;       I=1
10;     3 READ(105,1)((TABSUF(I,J),J=1,3),NSUF(I))
11;     1 FORMAT(3R4,8X,I10)
12;       IF(NSUF(I).LT.0)  GOTO 2
13;       I=I+1
14;       GOTO 3
15;     2 NSUF(I)=0
16;       INSUF=I-1
17; X     WRITE(108,1000)  INSUF
18; X1000 FORMAT(1H1,'FIN DE LECTURE DES SUFFIXES',I5)
19; C     IMPRESSION DES SUFFIXES
20;       DO 4 I=1,INSUF
21;     4 WRITE(108,5)(NSUF(I),(TABSUF(I,J),J=1,3))
22;     5 FORMAT(1HO,5X,I5,5X,3R4)
23; C     FIN D'ENTREE DES SUFFIXES.
24;       END LABELS
25; C******************
26; C     LECTURE DU DICTIONNAIRE.
27; C-----------------------------------------------------------------------
28; C     LECTURE DE ADR ET CHAIN.
29;       READ(1) CHAIN
30;       READ(1) ADR
31;       WRITE(108,105)
32;   105 FORMAT(1H1)
33; C*************************************************************************
34;       END LABELS
35; C
36; C                                        SORTSTEM-SORTMOT
37; C
38; C*************************************************************************
39;    10 READ(105,98) ISORT,(ADRCHAIN(I),I=1,18)
40;    98 FORMAT(I2,18A4)
41;       IF(ISORT.LT.0)  GOTO 11
42;       WRITE(108,99)  (ADRCHAIN(I),I=1,18)
43;    99 FORMAT(1H1,20X,18A4)
44;       I=0
45;       ICHAIN=1
46;     3 I=I+1
47;       LCHAIN=CHAIN(ICHAIN)
48;       IF((LCHAIN.EQ.1R/).OR.(LCHAIN.EQ.1R*)) I=I-1; GOTO 1
49;       MOT(I)=LCHAIN
50;       ADRCHAIN(I)=ADR(ICHAIN)
51;       ICHAIN=ICHAIN+1
52;       GOTO 3
53;     1 CONTINUE
54; C     ECRITURE DU STEM.
55;     5 IMPZ=ADR(ICHAIN)
```

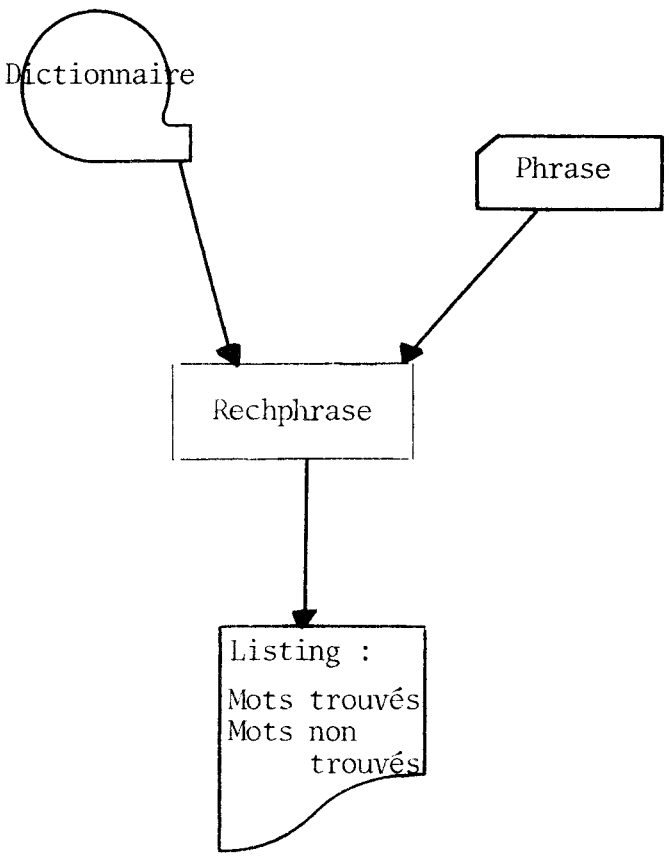
```
      IMP=IMPZ/10000
      IMP=-IMP
      IMPG=IMP/10
      IGR=IMP-IMPG*10
      IF(IGR.NE.ISORT)  GOTO 9
      IMPW=IMPZ/1000
      ISUF=-IMPZ+IMPW*1000
      WRITE(108,100)  I,(MOT(J),J=1,I),ISUF,IMP
  100 FORMAT(1HO,10X,N(1R1),20X,I5,$  N. : $,I10)
      IF (IMP.EQ.9)  GOTO 7
C     SORTIE DES SUFFIXES.
      MARK(ISUF)=1
      DO 6 J=1,INSUF
      IF(NSUF(J).LT.ISUF) GOTO 6
      IF(NSUF(J).GT.ISUF) GOTO 7
      WRITE(108,103) I,(TABSUF(J,K),K=1,3)
  103 FORMAT (1H ,10X,N(X),3R4)
    6 CONTINUE
    7 CONTINUE
      WRITE(108,104)
  104 FORMAT(1HO,$****************$)
    9 CONTINUE
      IF(LCHAIN.EQ.1R*)  GOTO 2
      ICHAIN=ICHAIN+1
      GOTO 3
    2 IF(CHAIN(ICHAIN+1).EQ.1R*)  ICHAIN=ICHAIN+1 ; GOTO 5
   12 IF(ADRCHAIN(I).EQ.0)  GOTO 4
      ICHAIN=ADRCHAIN(I)
      MOT(I)=0
      I=I-1
      GOTO 3
    4 MOT(I)=0
      I=I-1
      IF(I.EQ.0)  GOTO 10
      IF(I.NE.1)  GOTO 12
      WRITE(108,97)
   97 FORMAT(1H1)
      GOTO 12
   11  WRITE(108,102)
       WRITE(108,101)
       WRITE(108,102)
  101 FORMAT(1HO,$* FIN DE LA SORTIE DES STEMS *$)
  102 FORMAT(1HO,$******************************$)
C-----------------------------------------------------------------
C--------------------FIN DE SORT MOT------------------------------
C-----------------------------------------------------------------
      WRITE(108,106)
  106 FORMAT(1H1,60(1H*),$SUFFIXES UTILISES$,40(1H*))
      DO 20 J=1,400
      IF(MARK(J).NE.1) GOTO 20
      WRITE(108,107) J
  107 FORMAT(1HO,I10)
   20 CONTINUE
      STOP
       END
```

6. PROGRAMME DE RECHERCHE DES MOTS D'UNE PHRASE QUI FIGURENT DANS LE DICTIONNAIRE

"RECHPHRASE"

Le tableau MØT contient les caractères qui constituent la phrase. Les pointeurs des mots trouvés dans le dictionnaire seront rangés dans le tableau RESULT, le libellé des mots trouvés dans le tableau MØTROUVE, et les mots non trouvés dans le tableau NONTROUVE.

L'ordinogramme est le suivant :



```
      INTEGER MOT(88),ADRCHAIN(0:82),ADR(0:30000),CHAIN(0:30000)
      COMMON ADR,CHAIN
      INTEGER RESULT(200,5),NONTROUVE(50,20)
      INTEGER NOMBRE(20)
      INTEGER MOTROUVE(50,20)
      LOGICAL STEM,TROUVE,SEPAR
C*****************************************************************
C
C                              RECHPHRASE.
C
C*****************************************************************
      READ(1)  CHAIN
      READ(1)  ADR
      WRITE(108,1111)
 1111 FORMAT(1H1,$ FIN ENTREE DU DICTIONNAIRE$)
C-----------FIN DE LA LECTURE DE LA CHAINE
      END LABELS
C$$$$$$$$$$$$$
      IM=80
      MOT(IM+1)=1R.
      MOT(IM+2)=1RA
C--------------------
 1001 CONTINUE
C     LECTURE DE LA PHRASE.
      DO 25 J=1,IM
   25 ADRCHAIN(J)=MOT(J)=0
      READ(105,105) (MOT(J),J=1,IM)
  105 FORMAT(80R1)
      IF(MOT(1).EQ.1F-)  GOTO 1000
      WRITE(108,106)  (MOT(J),J=1,80)
  106 FORMAT(1H1,80R1)
C-----------------------------------
      INMOT=0
      IXMOT=IZMOT=1
      M=MDEBMOT=1
C-----------------------------------
   30 SEPAR=.FALSE.
      IF(MDEBMOT.GE.IM)  GOTO 13
C     DETERMINATION DU CARACTERE SEPARATEUR-FIN DE MOT,DEBUT MOT SUIVANT
      DO 26 J=MDEBMOT,IM+2
C     LE - N'EST PAS UN CARACTERE SEPARATEUR : MOTS COMPOSES
      LMOT=MOT(J)
      IF(LMOT.EQ.1F-)  GOTO 24
      IF(LMOT.LT.128) GOTO 27
   24 IF(SEPAR) MSUIVMOT=J ; GOTO 35
      GOTO 26
   27 IF(.NOT.SEPAR) MSEPAR=J
      SEPAR=.TRUE.
   26 CONTINUE
   35 LMOT=MOT(MDEBMOT)
C     ON ELIMINE LES MOTS COMMENCANT PAR-,ET LES CHIFFRES <0
      IF(LMOT.EQ.1F-)  GOTO 12
      IF(LMOT.LT.240) GOTO 28
C     LE MOT EST UNE SUITE DE CHIFFRES.
      N1=0
```

```
     36 JX=LMOT-240
        N1=N1*10+JX
        MDEBMOT=MDEBMOT+1
        IF(MDEBMOT.EQ.MSEPAR)  GOTO 37
        LMOT=MOT(MDEBMOT)
        GOTO 36
C       ON A DECODE UN NOMBRE ENTIER<2 147 483 648
     37 INMOT=INMOT+1
        NOMBRE(INMOT)=N1
        RESULT(IXMOT,1)=2147483000+INMOT
        M=MDEBMOT=MSUIVMOT
        IXMOT=IXMOT+1
        GOTO 30
C*****************************************************************
C
C                                    RECH-MOT
C
C*****************************************************************
     28 STEM=.FALSE.
        ICHAIN=1
        MSTEM=MDEBMOT
      1 LMOT=MOT(M)
X       WRITE (108,200) M,LMOT,ICHAIN
X200    FORMAT (1HO,'M= ',I5,5X,1R1,5X,'ICHAIN= ',I5)
        IF(M.GE.MSEPAR)  GOTO 2
      5 LCHAIN=CHAIN(ICHAIN)
        IF(LCHAIN.EQ.LMOT) GOTO 3
        IF (LCHAIN.NE.1R/)  GOTO 4
X       WRITE (108,201) ICHAIN
X201    FORMAT(1HO,' SAUT DU /  ',ICHAIN= ',I5)
        IF(ADRCHAIN(M).NE.0)  GOTO 19
        IF(STEM) GOTO 19
C       ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER LATCH.
        ADRCHAIN(M)=ICHAIN
        MSTEM=M
     19 ICHAIN=ICHAIN+1
        GOTO 5
      3 M=M+1
        ICHAIN=ICHAIN+1
        GOTO 1
      4 IF(LCHAIN.NE.1R*)  GOTO  39
        IF(STEM) GOTO 22
        GOTO 6
     39 ICCHAIN=ICHAIN
        LPRCHAIN=LCHAIN
C       SUIVI DU CHAINAGE.
      8 IADR=ADR(ICCHAIN)
X       WRITE(108,202) ICHAIN,IADR
X202    FORMAT(1HO,'SUIVI DU CHAINAGE      ICHAIN=',I5,5X,'IADR= ',I5)
        IF(IADR.EQ.0) GOTO 7
        ICCHAIN=IADR
        LCHAIN=CHAIN(ICCHAIN)
        IF (LCHAIN.LT.LMOT)  GOTO 8
        IF (LCHAIN.NE.LMOT)  GOTO 7
C       APRES SUIVI DU CHAINAGE,LCHAIN=LMOT.
X       WRITE(108,203) ICCHAIN
```

```
112: X203  FORMAT(1HO,$LCHAIN=LMOT    ,ICHAIN=   $,I5)
113:       ICHAIN=ICCHAIN+1
114:       M=M+1
115:       GOTO 1
116:     7 CONTINUE
117: X     WRITE(108,206) LPRCHAIN,ICCHAIN
118: X206  FORMAT(1HO,$LPRCHAIN =$,1R1,$    ICCHAIN=$,,5)
119:        IF(LPRCHAIN.EQ.1R-)  GOTO 17
120:       IF (LPRCHAIN.NE.1R )  GOTO 9
121: C     RECHERCHE DU PRECHAIN - OU 'BLANC'.
122:    17 DO 10 J=M,M+7
123:       IF(MOT(J).NE.LPRCHAIN)  GOTO 10
124:       M=J
125:       GOTO 1
126:    10 CONTINUE
127:     2 LCHAIN=CHAIN(ICHAIN)
128: X     WRITE(108,208)  LCHAIN,ICHAIN
129: X208  FORMAT (1HO,$SORTIE PAR 2: LCHAIN=$,1R1,5X,$    ICHAIN=$,I5)
130:       IF(LCHAIN.EQ.1R/) GOTO 21
131:       IF(LCHAIN.NE.1R*)  GOTO 9
132:    21 CONTINUE
133: X     WRITE(108,204)
134: X204  FORMAT(1HO,$MOT COMPLET DANS DIC=STEM$)
135:       IF(STEM)  GOTO 11
136:       ICSTEM=ICHAIN
137:       ICHAIN=2
138:       STEM=.TRUE.
139:       GOTO 11
140:     9 IF(STEM)  GOTO 22
141: C     RECHERCHE DU PLUS LONG STEM
142:    23 DO 12 J=MSTEM,MDEBMOT,-1
143:       IF(ADRCHAIN(J).EQ.0)  GOTO 12
144:       ICHAIN=ADRCHAIN(J)
145:       M=J
146:       ADRCHAIN(J)=0
147:       GOTO 6
148:    12 CONTINUE
149:       WRITE (108,100)
150:   100 FORMAT(1HO,$ERREUR RECHERCHE 1:PAS DE STEM CONTENU    MOT NON
151:      -TROUVE$)
152: C     STOCKAGE DU MOT NON TROUVE.
153:       JX=0
154:       DO 29 J=MDEBMOT,MSEPAR-1,1
155:       JX=JX+1
156:    29 NONTROUVE(IZMOT,JX)=MOT(J)
157:       IF(MSUIVMOT.GT.IM)    GOTO 13
158:       M=MDEBMOT=MSUIVMOT
159:       IZMOT=IZMOT+1
160:       GOTO 30
161:     6 STEM=.TRUE.
162: C     ON A LE PLUS LONG STEM.
163: X     WRITE(108,205) M-MDEBMOT,(MOT(J),J=MDEBMOT,M-1)
164: X205  FORMAT(1HO,$PLUS LONG STEM =     $,N(1R1))
165:       IF(STEM)  ICSTEM=ICHAIN
166: C     RECHERCHE DU SUFFIXE.
167: X     WRITE(108,207) M
```

```
168: x207  FORMAT(1HO,$RECHERCHE DU SUFFIXE  M=$,I5)
169:       ICHAIN=1
170:       GOTO 1
171: C***
172:    11 CONTINUE
173: C     RECHERCHE DU SUFFIXE COMPATIBLE
174: x     WRITE(108,209) ICSTEM
175: x209  FORMAT(1HO,$RECHERCHE DU N° SUFFIXE   ICSTEM =$,I5)
176:       I=0
177:       TROUVE=.FALSE.
178: C     RECHERCHE DE G° SUFFIXE
179:       J=ADR(ICHAIN)
180:    20 IMP=J/100000
181:       N1=-J+IMP*100000
182:       IGS=N1/10000
183:       IF(IGS.NE.9)   GOTO 15
184: C     RECHERCHE DU N° SUFFIXE
185:       IMP=J/1000
186:       N1=-J+IMP*1000
187: C     RECHERCHE ISUF.
188:       ICC=ICSTEM
189:       IADRSTEM=ADR(ICC)
190:    16 IMP=-IADRSTEM/1000
191:       ISUF=-IADRSTEM-IMP*1000
192:       IF(N1-ISUF) 18,40,18
193: C     MOT TROUVE
194:    40 TROUVE=.TRUE.
195:       I=I+1
196:       RESULT(IXMOT,I)=IADRSTEM
197: x     WRITE(108,210) M
198: x210  FORMAT(1H ,$     M=  $,I5)
199: x     WRITE(108,211)  IMP
200: x211  FORMAT(1HO,10X,$ ** MOT TROUVE **          N°: $,I5)
201:    18 ICC=ICC+1
202:       IADRSTEM=ADR(ICC)
203:       IF(IADRSTEM.GE.0)   GOTO 15
204:       GOTO 16
205:    15 ICHAIN=ICHAIN+1
206:       J=ADR(ICHAIN)
207:       IF(J.LT.0)  GOTO 20
208: C     EXPLORATION TERMINEE.
209:       IF(TROUVE)   GOTO 14
210:       WRITE(108,101)
211:   101 FORMAT(1HO,$ERREUR RECHERCHE 2:LE MOT NE FAIT PAS PARTIE DE DIC$)
212: C     LE PLUS LONG STEM ASSOCIE NE CONVIENT PAS:RECHERCHE STEM PLUS COURT
213:       GOTO 23
214:    22 CONTINUE
215: C     ON A TROUVE UN STEM,MAIS LE SUFFIXE ASSOCIE N'EXISTE PAS
216:       WRITE(108,103)
217:   103 FORMAT(1HO,$ERREUR RECHERCHE 3:LE SUFFIXE ASSOCIE N'EXISTE PAS$)
218:       GOTO 23
219:    14 CONTINUE
220: C     ON A TROUVE LE MOT.
221:       JX=0
222: C     STOCKAGE DU MOT TROUVE
223:       DO 42 J=MDEBMOT,MSEPAR-1,1
```

```
      JX=JX+1
   42 MOTROUVE(IXMOT,JX)=MOT(J)
      IXMOT=IXMOT+1
      IF(MSUIVMOT.GT.IM)  GOTO 13
      M=MDEBMOT=MSUIVMOT
      GOTO 30
   13 CONTINUE
C*****************************************************************
C
C                              FIN DE RECH-MOT
C
C*****************************************************************
C     ON A FINI L'ANALYSE DE LA PHRASE
C     IXMOT=NB. DE MOTS TROUVES
C     IZMOT=NB. DE MOTS NON TROUVES
      WRITE(108,107)
  107 FORMAT(1H1,20X,'MOTS NON TROUVES')
      DO 31 J=1,IZMOT
   31 WRITE(108,108) (NONTROUVE(J,M),M=1,20)
  108 FORMAT(1H0,10X,20R1)
C     IMPRESSION DES MOTS TROUVES.
      WRITE(108,109)
  109 FORMAT(1H1,20X,'MOTS TROUVES')
      DO 32 J=1,IXMOT
   32 WRITE(108,110) (MOTROUVE(J,M),M=1,20),(RESULT(J,M),M=1,5)
  110 FORMAT(1H0,20R1,5(5X,I10))
      WRITE(108,111)
  111 FORMAT(1H1,20X,'CHIFFRES')
      DO 38 J=1,INMOT
  112 FORMAT(1H0,10X,I12)
      WRITE(108,112) NOMBRE(J)
   38 NOMBRE(J)=0
C     MISE A ZERO DE RESULT ET NONTROUVE.
      DO 41 J=1,IXMOT
      DO 33 M=1,5
   33 RESULT(J,M)=0
      DO 41 M=1,20
   41 MOTROUVE(J,M)=0
      DO 34  J=1,IZMOT
      DO 34  M=1,20
   34 NONTROUVE(J,M)=0
      GOTO 1001
 1000 STOP
      END
```

ANNEXE 4

REPRESENTATION 10070

LANGAGE MACHINE

1. Principe

2. Codage

. PRINCIPE

Les programmes de création et de mise à jour du dictionnaire ont été écrits en Fortran IV. Le traitement des chaînes de caractères est assez délicat dans ce langage, et pour simplifier ce traitement, le tableau des caractères a été constitué à raison d'un caractère par mot-machine 10070.

Deux procédures incluses dans le Fortran IV permettent d'atteindre, ou de ranger un caractère dans un des quatre octets qui constituent un mot 10070. Mais des essais ont montré que les temps d'appel de ces procédures augmentent notablement les temps de recherche dans le dictionnaire ainsi constitué. Comme la taille des dictionnaires créés n'était pas très importante, la solution utilisant un caractère par mot 10070 a été conservée. En effet, la mémoire centrale disponible était de 128 K mots, et le système ne fonctionnait qu'en monoprogrammation, et donc le problème du volume des données n'était pas fondamental pour le système expérimental ainsi créé.

Par contre, pour un système opérationnel, où le temps de recherche, et également le volume sont des paramètres fondamentaux à optimiser, il est possible d'utiliser dans le programme de recherche et d'exploitation des codes, écrit en Fortran, une procédure de recherche de mot écrite en langage machine, et ajoutée à la bibliothèque. Il est possible, en assembleur 10070, d'atteindre facilement le niveau de l'octet, et de réduire ainsi le volume du dictionnaire.

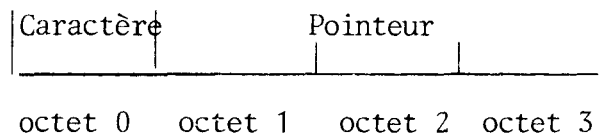
Un codage utilisant un mot 10070 pour représenter le caractère et son pointeur associé sera proposé dans le paragraphe suivant. Il est adopté au langage machine 10070, et il est possible de créer ce codage à partir d'un programme Fortran, utilisant les procédures encode/décode. On peut donc adapter les programmes de création et de mise à jour déjà vus pour constituer le dictionnaire suivant cette représentation. Etant donné qu'il n'est pas nécessaire d'optimiser, dans un premier stade, du moins, les temps de création et de mise à jour, ces programmes peuvent être écrits en Fortran.

2. CODAGE

Le 10070 est une machine essentiellement à mot. Il existe, en langage machine, des opérations manipulant des octets. Elles nécessitent l'emploi d'un registre pour indexation. Par exemple : LB, 2 alpha, 0. Si le registre 0 contient la valeur 2, LB va charger dans le registre 2 l'octet 2 du mot d'adresse alpha.

Si l'octet adressé est l'octet gauche du mot, l'indexation n'est pas nécessaire : LB, 2 alpha va charger dans le registre 2 l'octet de gauche du mot d'adresse alpha.

L'indexation augmente le temps d'exécution ; la recherche sera donc accélérée si le caractère est stocké dans l'octet gauche du mot :



Il est possible de représenter les codes des pointeurs dans les 3 octets qui suivent.

- En ce qui concerne les pointeurs d'alternants, trois octets permettent de représenter un nombre entier < 16 777 215 (2^{24}-1), ce qui est surabondant.

Deux octets, soit un demi-mot suffisent : il est possible ainsi de représenter un entier < 65 536 (2^{16}-1). Il est ainsi possible de manipuler les pointeurs d'alternants à l'aide des opérations travaillant sur les demi-mots, ce qui accélère le traitement.

- Pour les pointeurs de mots, l'annexe 3 a montré que les codes se partageaient en deux sous-ensembles.

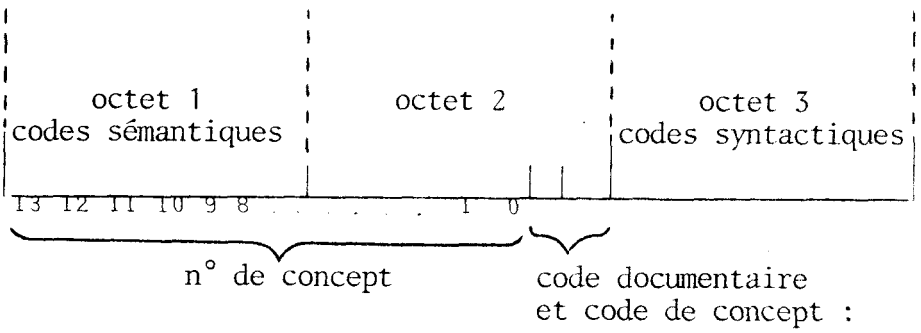
. codes syntactiques, donnant la fonction grammaticale et la classe de suffixes

. codes sémantiques, permettant une utilisation ultérieure des informations associées au mot par le système de documentation :

- code documentaire
- code de concept
- code de concept appartenant à un ensemble de concepts dont la réunion constitue un concept.

Un octet suffit pour représenter l'ensemble des codes syntactiques : il est possible de repérer 256 classes de suffixes différentes, or, le chapitre I a déterminé l'existence de 91 classes de suffixes de nom, 47 classes d'adjectif, 34 classes de verbe, 4 classes d'adverbe, soit 175 classes de suffixes, en tout.

Deux octets suffisent pour représenter les codes sémantiques :

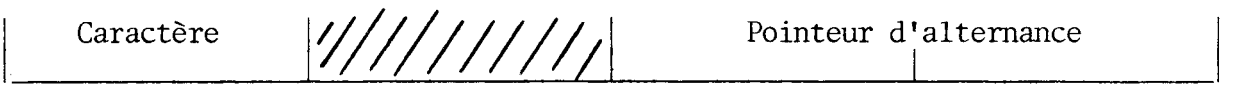


0 0	mot pivot concept élémentaire
0 1	mot pivot concept appartenant à un ensemble
1 0	mot vide ou mot grammatical
1 1	suffixe

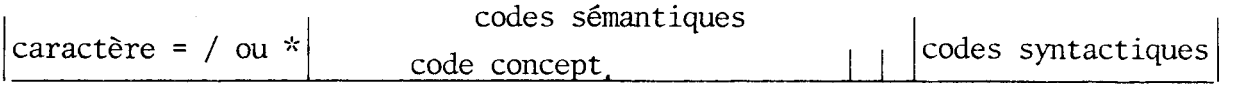
Le code de concept est représenté sur 14 bits donc entier < 16 383 (2^{14}-1) ce qui est suffisant, car pour plus de 16000 concepts, il est probable qu'une pagination sera nécessaire ; dans ce cas le code de concept sera complété par le code de la page. A titre indicatif pour l'ensemble du système, Smart [2], les concepts sont repérés par un nombre < 32000 (2^{15}).

Le code documentaire et le code de concept donnant la configuration 10 déterminent un mot grammatical ou un mot vide. Un mot vide aura un numéro de concept = 0, et un mot grammatical un numéro de concept ≠ 0, ce qui permettra de les différencier.

Le codage proposé est donc :



ou :



```
          ****** DEBUT DE PROGRAMME ******

###   OPERATION  N:    1
AUTOMOBILE
$

*** PHRASE N :    1
COLONNE : 1             5  SERIES REPONDENT A LA QUESTION

###   OPERATION  N:    2
'NS' 1,0,0,

***  NUMERO  DES SERIES  COLONNE :  1
* OPERATION N:  1       1 - PHRASE N,  1
   5  SERIES :
/   1  : 4020150057
/   2  : 4020250018
/   3  : 4020342280
/   4  :1102011111
/   5  :1102011123

###   OPERATION  N:    3
CAMIONS
$

*** PHRASE N :    2
COLONNE : 2             1  SERIES REPONDENT A LA QUESTION

###   OPERATION  N:    4
'NS' 0,0,0,

***  NUMERO  DES SERIES  COLONNE :  2
* OPERATION N:  3       2 - PHRASE N,  2
   1  SERIES :
/   1  : 6010450045

###   OPERATION  N:    5
VOITURES
$

*** PHRASE N :    3
COLONNE : 3             4  SERIES REPONDENT A LA QUESTION

###   OPERATION  N:    6
'NS' 0,0,0,

***  NUMERO  DES SERIES  COLONNE :  3
* OPERATION N:  5       3 - PHRASE N,  3
   4  SERIES :
/   1  : 6010240600
/   2  : 6010251020
/   3  : 6010251950
/   4  :1102032292

###   OPERATION  N:    7
'OU' 1,2,0,

COLONNE  4 :     6 SERIES REPONDENT A L'OPERATION
```

```
###   OPERATION  N:  49
'ST' 11,0,0,

0*****************************LA TRANSACTION RECOMMENCE A LA COLONNE :11

###   OPERATION  N:  50
'ET' 11,3,2,

COLONNE 2  AUCUNE SERIE NE REPOND A L'OPERATION

###   OPERATION  N:  51
'ET' 13,2,2,

0 ERREUR : HORS LIMITES T

###   OPERATION  N:  52
123

ON SORT DES LIMITES DE T : INTAB=IMAXTAB ; UTILISER   L'OPERATION 'ST'

###   OPERATION  N:  53
123
123
123
123
123

** PHRASE TROP LONGUE,RECOMMENCEZ

###   OPERATION  N:  54
AGC,AGE,CA,CAE,CAECL,CAEL,CCP,CDC,CE,CEE,CEG,CES,CET,CETA,CF,CII,
CMCC,CNE,CSP,CV,DENS,FPA,GDF,GW,HLM,IBM,ICL,IRPP,IVD,
$

*** PHRASE N :  11
0****** TROP DE MOTS CARACTERISTIQUES,RECOMMENCEZ

###   OPERATION  N:  55
'SA' 2,3,3,

DANS L'OPERATION 'SA' ON DOIT AVOIR J2#J3

###   OPERATION  N:  56
'TR' 0,0,0,

LES PARAMETRES DE TR DOIVENT ETRE NON NULS

###   OPERATION  N:  57
'ET' 1,10,0,

ON SORT DES LIMITES DE T : INTAB=IMAXTAB ; UTILISER   L'OPERATION 'ST'

###   OPERATION  N:  58
'ET'1,9,0,

ON SORT DES LIMITES DE T : INTAB=IMAXTAB ; UTILISER   L'OPERATION 'ST'

###   OPERATION  N:  59
'NC' 0,0,0,

0*** DERNIERE COLONNE ATTRIBUEE :13

###   OPERATION  N:  60
'ST' 1,0,0,

0*****************************LA TRANSACTION RECOMMENCE A LA COLONNE : 1
```

ANNEXE 5

PROGRAMMES de DETERMINATION des MOTS NON SIGNIFICATIFS

1. Détermination par itération

2. Fréquence mots

1. DETERMINATION PAR ITERATION

Le programme de recherche des mots vides contenus dans une phrase est dérivé du programme rechphrase (cf. A.3). Le dictionnaire utilisé ici est le dictionnaire des mots vides déterminés à l'étape précédente. Les mots trouvés sont les mots vides. Les mots non trouvés sont supposés être pivots, et permettront de choisir de nouveaux mots vides.

Le programme est donné ici avec un exemple d'analyse de phrase. Il faut noter que dans cet exemple les codes alphanumériques situés avant la virgule n'ont pas été pris en compte.

```
 1:       INTEGER MOT(168),ADRCHAIN(0:162),ADR(0:20000),CHAIN(0:20000)
 2:       COMMON ADR,CHAIN
 3:       INTEGER KARTE(80),RESULT(50,5),NONTROUVE(50,20)
 4:       INTEGER NOMBRE(20)
 5:       INTEGER MOTROUVE(50,20)
 6:       LOGICAL STEM,TROUVE,SEPAR
 7: C-----------------------------------------------------------------
 8: C
 9: C--------------------   DETERMINATION DES MOTS VIDES   ----------
10: C
11: C-----------------------------------------------------------------
12: C     LECTURE DE ADR ET CHAIN.
13:       CALL BUFFER IN(1,1,ADR,3501,ITEST,N)
14:     1 GOTO (1,2,7,3) ITEST
15:     7 WRITE(108,103)
16:     2 WRITE(108,100) N
17:   100 FORMAT(1H1,$ENTREE DE ADR$,I10)
18:       CALL BUFFER IN(1,1,CHAIN,3501,ITEST,N)
19:     4 GOTO (4,5,8,3)  ITEST
20:     8 WRITE(108,103)
21:     5 WRITE(108,101) N
22:   101 FORMAT(1HO,$ENTREE DE CHAIN$,I10)
23:       GOTO 6
24:     3 WRITE(108,102)
25:   102 FORMAT(1HO,$  ERREUR$)
26:   103 FORMAT(1HO,$  FIN DE FICHIER$)
27:       PAUSE 1
28:    6  CONTINUE
29:       WRITE(108,1111)
30:  1111 FORMAT(1H1,$ FIN ENTREE DU DICTIONNAIRE$)
31:       REWIND 1
32: C-----------FIN DE LA LECTURE DE LA CHAINE
33:       END LABELS
34: C$$$$$$$$$$$$$$$
35:       IM=160
36:       MOT(IM+1)=1R.
37:       MOT(IM+2)=1RA
38: C--------------------
39:       CALL EOFSET(1000S)
40:       READ(105,105) (KARTE(J),J=1,80)
41:   105 FORMAT(80R1)
42: C-----LECTURE DE LA PHRASE.
43:  1001 ID=0
44:    45 DO 25 J=1,80
45:    25 MOT(J+ID)=KARTE(J)
46:       READ(105,105) (KARTE(J),J=1,80)
47:       IF(KARTE(1).EQ.1R$)  GOTO 44
48:       ID=ID+80
49:       GOTO 45
50:    44 CONTINUE
51: C     ECRITURE DE LA PHRASE.
52:       WRITE(108,106) (MOT(J),J=1,160)
53:   106 FORMAT(1H1,80R1,/,80R1)
54: C-----------------------------------
55:       INMOT=0
```

```
 56:        IXMOT=IZMOT=1
 57:        M=19
 58:        MDEBMOT=19
 59: C-----------------------------------
 60:     30 SEPAR=.FALSE.
 61: C     DETERMINATION DU CARACTERE SEPARATEUR=FIN DE MOT,DEBUT MOT SUIVANT
 62:        DO 26 J=MDEBMOT,IM+2
 63: C      LE - N'EST PAS UN CARACTERE SEPARATEUR : MOTS COMPOSES
 64:        LMOT=MOT(J)
 65:        IF(LMOT.EQ.1R-)  GOTO 24
 66:        IF(LMOT.LT.128) GOTO 27
 67:     24 IF(SEPAR) MSUIVMOT=J ; GOTO 35
 68:        GOTO 26
 69:     27 IF(.NOT.SEPAR) MSEPAR=J
 70:        SEPAR=.TRUE.
 71:     26 CONTINUE
 72:     35 LMOT=MOT(MDEBMOT)
 73: C      ON ELIMINE LES MOTS COMMENCANT PAR-,ET LES CHIFFRES <0
 74:        IF(LMOT.EQ.1R-)  GOTO 12
 75:        IF(LMOT.LT.240) GOTO 28
 76: C      LE MOT EST UNE SUITE DE CHIFFRES.
 77:        N1=0
 78:     36 JX=LMOT-240
 79:        N1=N1*10+JX
 80:        MDEBMOT=MDEBMOT+1
 81:        IF(MDEBMOT.EQ.MSEPAR)  GOTO 37
 82:        LMOT=MOT(MDEBMOT)
 83:        GOTO 36
 84: C      ON A DECODE UN NOMBRE ENTIER<2 147 483 648
 85:     37 INMOT=INMOT+1
 86:        NOMBRE(INMOT)=N1
 87:        RESULT(IXMOT,1)=2147483000+INMOT
 88:        M=MDEBMOT=MSUIVMOT
 89:        IXMOT=IXMOT+1
 90:        IF(MDEBMOT.GE.IM)  GOTO 13
 91:        GOTO 30
 92: C*****************************************************************
 93: C
 94: C                                  RECH-MOT
 95: C
 96: C*****************************************************************
 97:     28 STEM=.FALSE.
 98: X      WRITE(108,212) MSUIVMOT,MSEPAR
 99: X212   FORMAT(1H1,$MSUIVMOT=$,I5,5X,$MSEPAR=$,I5)
100:        ICHAIN=1
101:        MSTEM=MDEBMOT
102:      1 LMOT=MOT(M)
103: X      WRITE (108,200) M,LMOT,ICHAIN
104: X200   FORMAT (1HO,$M= $,I5,5X,1R1,5X,$ICHAIN= $,I5)
105:        IF(M.GE.MSEPAR)  GOTO 2
106:      5 LCHAIN=CHAIN(ICHAIN)
107:        IF(LCHAIN.EQ.LMOT) GOTO 3
108:        IF (LCHAIN.NE.1R/)  GOTO 4
109: X      WRITE (108,201) ICHAIN
110: X201   FORMAT(1HO,$ SAUT DU /  ,ICHAIN= $,I5)
111:        IF(ADRCHAIN(M).NE.0)  GOTO 19
```

```
112:          IF(STEM) GOTO 19
113:  C       ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER cLATCH.
114:          ADRCHAIN(M)=ICHAIN
115:          MSTEM=M
116:       19 ICHAIN=ICHAIN+1
117:          GOTO 5
118:        3 M=M+1
119:          ICHAIN=ICHAIN+1
120:          GOTO 1
121:        4 IF(LCHAIN.NE.1R*)  GOTO  39
122:          IF(STEM) GOTO 22
123:          GOTO 6
124:       39 ICCHAIN=ICHAIN
125:          LPRCHAIN=LCHAIN
126:  C       SUIVI DU CHAINAGE.
127:        8 IADR=ADR(ICCHAIN)
128:  X       WRITE(108,202) ICHAIN,IADR
129:  X202    FORMAT(1HO,$SUIVI DU CHAINAGE      ICHAIN=$,I5,5X,$IADR= $,I5)
130:          IF(IADR.EQ.0) GOTO 7
131:          ICCHAIN=IADR
132:          LCHAIN=CHAIN(ICCHAIN)
133:          IF (LCHAIN.LT.LMOT)  GOTO 8
134:          IF (LCHAIN.NE.LMOT)  GOTO 7
135:  C       APRES SUIVI DU CHAINAGE,LCHAIN=LMOT.
136:  X       WRITE(108,203) ICCHAIN
137:  X203    FORMAT(1HO,$LCHAIN=LMOT   ,ICHAIN=  $,I5)
138:          ICHAIN=ICCHAIN+1
139:          M=M+1
140:          GOTO 1
141:        7 CONTINUE
142:  X       WRITE(108,206) LPRCHAIN,ICCHAIN
143:  X206    FORMAT(1HO,$LPRCHAIN =$,1R1,$   ICCHAIN=$,I5)
144:           IF(LPRCHAIN.EQ.1R-)  GOTO 17
145:          IF (LPRCHAIN.NE.1R )  GOTO 9
146:  C       RECHERCHE DU PROCHAIN = OU 'BLANC'.
147:       17 DO 10 J=M,M+7
148:          IF(MOT(J).NE.LPRCHAIN)  GOTO 10
149:          M=J
150:          GOTO 1
151:       10 CONTINUE
152:        2 LCHAIN=CHAIN(ICHAIN)
153:  X       WRITE(108,208)  LCHAIN,ICHAIN
154:  X208    FORMAT (1HO,$SORTIE PAR 2: LCHAIN=$,1R1,5X,$   ICHAIN=$,I5)
155:          IF(LCHAIN.EQ.1R/) GOTO 21
156:          IF(LCHAIN.NE.1R*)  GOTO 9
157:       21 CONTINUE
158:  X       WRITE(108,204)
159:  X204    FORMAT(1HO,$MOT COMPLET DANS DIC-STEM$)
160:          IF(STEM)  GOTO 11
161:          ICSTEM=ICHAIN
162:          ICHAIN=2
163:          STEM=.TRUE.
164:          GOTO 11
165:        9 IF(STEM)  GOTO 22
166:  C       RECHERCHE DU PLUS LONG STEM
167:       23 DO 12 J=MSTEM,MDEBMOT,-1
```

```
      IF(ADRCHAIN(J).EQ.0)  GOTO 12
      ICHAIN=ADRCHAIN(J)
      M=J
      ADRCHAIN(J)=0
      GOTO 6
   12 CONTINUE
C     STOCKAGE DU MOT NON TROUVE.
      JX=0
      DO 29 J=MDEBMOT,MSEPAR-1,1
      JX=JX+1
   29 NONTROUVE(IZMOT,JX)=MOT(J)
      IF(MSUIVMOT.GT.IM)    GOTO 13
      M=MDEBMOT=MSUIVMOT
      IZMOT=IZMOT+1
      GOTO 30
    6 STEM=.TRUE.
C     ON A LE PLUS LONG STEM.
X     WRITE(108,205) M=MDEBMOT,(MOT(J),J=MDEBMOT,M-1)
X205  FORMAT(1HO,$PLUS LONG STEM =     $,N(1R1))
      IF(STEM)  ICSTEM=ICHAIN
C     RECHERCHE DU SUFFIXE.
X     WRITE(108,207) M
X207  FORMAT(1HO,$RECHERCHE DU SUFFIXE  M=$,I5)
      ICHAIN=1
      GOTO 1
C***
   11 CONTINUE
C     RECHERCHE DU SUFFIXE COMPATIBLE
X     WRITE(108,209) ICSTEM
X209  FORMAT(1HO,$RECHERCHE DU N. SUFFIXE   ICSTEM =$,I5)
      I=0
      TROUVE=.FALSE.
C     RECHERCHE DE G. SUFFIXE
      J=ADR(ICHAIN)
   20 IMP=J/100000
      N1=-J+IMP*100000
      IGS=N1/10000
      IF(IGS.NE.9)   GOTO 15
C     RECHERCHE DU N. SUFFIXE
      IMP=J/1000
      N1=-J+IMP*1000
C     RECHERCHE ISUF.
      ICC=ICSTEM
      IADRSTEM=ADR(ICC)
   16 IMP=-IADRSTEM/1000
      ISUF=-IADRSTEM-IMP*1000
      IF(N1-ISUF) 18,40,18
C     MOT TROUVE
   40 TROUVE=.TRUE.
      I=I+1
      RESULT(IXMOT,I)=IADRSTEM
X     WRITE(108,210) M
X210  FORMAT(1H ,$    M=  $,I5)
X     WRITE(108,211)  IMP
X211  FORMAT(1HO,10X,$ ** MOT TROUVE **          N.: $,I5)
   18 ICC=ICC+1
```

```
224:         IADRSTEM=ADR(ICC)
225:         IF(IADRSTEM.GE.0)   GOTO 15
226:         GOTO 16
227:      15 ICHAIN=ICHAIN+1
228:         J=ADR(ICHAIN)
229:         IF(J.LT.0)  GOTO 20
230:  C      EXPLORATION TERMINEE.
231:         IF(TROUVE)   GOTO 14
232:  C      LE PLUS LONG STEM ASSOCIE NE CONVIENT PAS:RECHERCHE STEM PLUS COU
233:         GOTO 23
234:      22 CONTINUE
235:  C      ON A TROUVE UN STEM,MAIS LE SUFFIXE ASSOCIE N'EXISTE PAS
236:         GOTO 23
237:      14 CONTINUE
238:  C      ON A TROUVE LE MOT.
239:  C      STOCKAGE DU MOT TROUVE
240:         JX=0
241:         DO 42 J=MDEBMOT,MSEPAR-1,1
242:         JX=JX+1
243:      42 MOTROUVE(IXMOT,JX)=MOT(J)
244:         IXMOT=IXMOT+1
245:         IF(MSUIVMOT.GT.IM)  GOTO 13
246:         M=MDEBMOT=MSUIVMOT
247:         GOTO 30
248:      13 CONTINUE
249:  C*****************************************************************
250:  C
251:  C                              FIN DE RECH-MOT
252:  C
253:  C*****************************************************************
254:  C      ON A FINI L'ANALYSE DE LA PHRASE
255:  C      IXMOT=NB. DE MOTS TROUVES
256:  C      IZMOT=NB. DE MOTS NON TROUVES
257:  C      IMPRESSION DES MOTS TROUVES.
258:          WRITE(108,109)
259:     109 FORMAT(1HO,$********************  MOTS VIDES$)
260:         DO 32 J=1,IXMOT
261:      32 WRITE(108,110) (MOTROUVE(J,M),M=1,20),(RESULT(J,M),M=1,5)
262:     110 FORMAT(1HO,20R1,5(5X,I10))
263:         WRITE(108,107)
264:     107 FORMAT(1HO,$******************  MOTS NON VUES$)
265:         DO 31 J=1,IZMOT
266:      31 WRITE(108,108) (NONTROUVE(J,M),M=1,20)
267:     108 FORMAT(1HO,10X,20R1)
268:         WRITE(108,111)
269:     111 FORMAT(1HO,$************************  CHIFFRES$)
270:         DO 38 J=1,INMOT
271:     112 FORMAT(1HO,10X,I12)
272:         WRITE(108,112) NOMBRE(J)
273:      38 NOMBRE(J)=0
274:  C      MISE A ZERO DE RESULT ET NONTROUVE.
275:         DO 41 J=1,IXMOT
276:         DO 33 M=1,5
277:      33 RESULT(J,M)=0
278:         DO 41 M=1,20
279:      41 MOTROUVE(J,M)=1R
```

```
280:         DO 34  J=1,IZMOT
281:         DO 34  M=1,20
282:      34 NONTROUVE(J,M)=0
283:         DO 43 J=1,IM
284:      43 MOT(J)=1R
285:         GOTO 1001
286:    1000 STOP
287:         END
```

```
$C 001004(2  )    ,DU SECTEUR HABITATION A LOYER MODERE LOCATION

********************  MOTS VIDES

DU                              =520000              0              0

A                               =120000              0              0

                                      0              0              0

******************  MOTS NON VIDES

          SECTEUR

          HABITATION

          LOYER

          MODERE

          LOCATION

************************  CHIFFRES

                       0
```

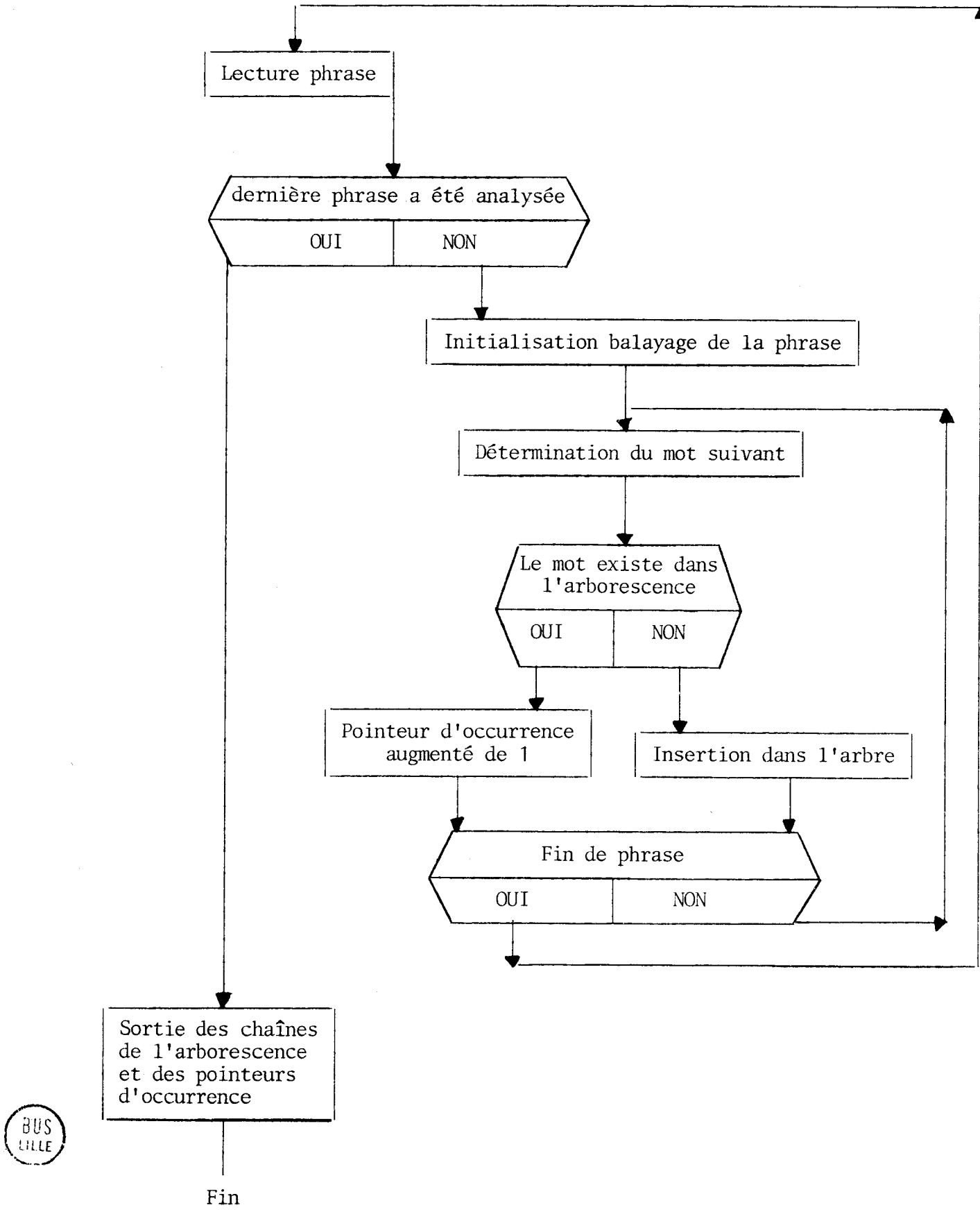
2. FREQUENCE MOTS

Ce programme analyse un ensemble de phrases.Pour chaque phrase, les chaînes des mots qui ne figurent pas dans l'arborescence sont insérées dans l'arbre. Pour les mots qui figurent déjà, le pointeur d'occurrence, associé au caractère de fin de chaîne est incrémenté de 1.

Lorsque toutes les phrases sont analysées, le programme liste l'ensemble des formes de mot contenues dans l'arborescence, et leur occurrence.

L'organigramme décrit plus loin reprend les organigrammes d'insertion de chaîne dans une arborescence, et celui de sortie des mots du chaînage.

ORDINOGRAMME. FREQUENCE MOTS



```
      INTEGER MOT(163),ADR(0:20000),CHAIN(0:20000)
      COMMON ADR,CHAIN
      LOGICAL F,FX,SEPAR
      INTEGER SORT(30),ADRSORT(30)
      INTEGER KARTE(80)
      LOGICAL SLATCH
C-----------------------------------------------------------------
C
C------------------------------FREQUENCEMOTS----------------------
C
C-----------------------------------------------------------------
      CHAIN(0)=2
       CHAIN(1)=0
      CHAIN(2)=1R*
      ADR(0)=0
      ADR(1)=0
      ADR(2)=-20000
C   **LECTURE DU DICTIONNAIRE DES MOTS VIDES.
C---------
      IM=160
      MOT(IM+1)=1R.
      MOT(IM+2)=1RA
C------------------
      WRITE(108,102)
  102 FORMAT(1H1)
      CALL EOFSET(50S)
C------LECTURE DE LA PHRASE.
      ID=0
   69 CONTINUE
      MDEBMOT=21
      READ(105,100) (MOT(J),J=1,160)
  100 FORMAT(80R1)
C-----ECRITURE DE LA PHRASE.
      ID=ID+1
      WRITE(108,170) ID
  170 FORMAT(1HO,$*** PHRASE ;$,I5)
      WRITE(108,101) (MOT(J),J=1,160)
  101 FORMAT(1HO,6R1,2X,4R1,2X,10R1,5X,60R1,/,30X,80R1)
C------------------
   68 SEPAR=.FALSE.
      IF(MDEBMOT.GE.IM)  GOTO 69
C-----DETERMINATION DU CARACTERE SEPARATEUR-FIN DE MOT,DEBUT MOT SUIVANT.
      DO 66 J=MDEBMOT,IM+2
      LMOT=MOT(J)
      IF(LMOT.EQ.1R-) GOTO 64
      IF(LMOT.LT.128)  GOTO 67
   64 IF(SEPAR) MSUIVMOT=J;GOTO 65
      GOTO 66
   67 IF(.NOT.SEPAR)  LONGMOT=J-1
      SEPAR=.TRUE.
   66 CONTINUE
   65 M=MDEBMOT
C-----------------------------------------------------------------
C     INSERTION DU MOT DANS DICFCOD.
C-----------------------------------------------------------------
```

```
C     CE PROGRAMME UTILISE DES ETIQUETTES DE 1 A 30  ET DE X1000 A X1011
C-------------------
C     CE PROGRAMME NECESSITE LES DECLARATIONS  :
C     INTEGER CHAIN(0:N),ADR(0:N),MOT(20)
C          LOGICAL  F,FX
C-------
C     LES VARIBLES UTILISEES SONT:
C       IADR,IALTERNANT,ICAS,ICCHAIN,ICHAIN,ICOMPT,IDECAL,II,IZ,I1,I2,
C       JJ,KZ,LL,LMOT,LONGMOT,M,
C       ----------
      F=.FALSE.
      FX=.FALSE.
      ICHAIN=1
    9 LMOT=MOT(M)
      IF(M.EQ.LONGMOT+1)  GOTO 1
    8 LCHAIN=CHAIN(ICHAIN)
      IF (LCHAIN.EQ.LMOT)   GOTO 2
      IF (LCHAIN.NE.1R/)   GOTO 7
C     SAUT DU /
X     WRITE  (108,1000) ICHAIN
X1000  FORMAT (1HO,$SAUT DE /  ICHAIN= $,I5)
      ICHAIN=ICHAIN+1
      GOTO 8
C     LES 2 CHAINES CONCORDENT
    2 ICHAIN=ICHAIN+1
      M=M+1
      GOTO 9
    7 IF (LCHAIN.EQ.1R*)  GOTO 3
      IF(LCHAIN.LT.LMOT)GOTO 17
C     CAS OU CHAIN(ICHAIN)>MOT(M) : ON INSERE LE RESTE DU MOT **
X     WRITE (108,1010) ICHAIN,LCHAIN,LMOT
X1010 FORMAT(1HO,$CHAIN(ICHAIN).GT.LMOT,ICHAIN=$,I5,$CHAIN(ICHAIN)=$,1R1
X    *,$MOT(M)=$,1R1)
      FX=.TRUE.
      IALTERNANT=0
      GOTO 15
   17 IADR=ADR(ICHAIN)
      IF(IADR.NE.0)  GOTO 4
C-----CAS 3.1:RECHERCHER LE * SUIVANT ET INSERER LE RESTE DU MOT + *
X     WRITE (108,1003)  ICHAIN
X1003 FORMAT (1HO,$CAS 3.1  ICHAIN=$,I5)
      IALTERNANT=ICHAIN
      GOTO 10
C-----CAS 2:REMPLACER LE * PAR / ET INSERER LE RESTE DU MOT + *
    3 CHAIN(ICHAIN)=1R/
      IALTERNANT=0
      ICHAIN=ICHAIN+1
X     WRITE(108,1004) ICHAIN
X1004 FORMAT (1HO,$CAS 2  ICHAIN=  $,I5)
      GOTO 15
C-----CAS 3.2:RECHERCHER LA PLACE DE L'ALTERNANT
    4 LL=CHAIN(IADR)
X     WRITE(108,1005)  IADR
X1005 FORMAT(1HO,$RECHERCHE ALTERNANT  IADR=$,I5)
      IF(LL.LT.LMOT)  GOTO 16
      IF(LL.GT.LMOT)  GOTO 18
```

```
112: X      WRITE (108,1008) ICHAIN,IADR
113: X1008 FORMAT (1HO,$CHAIN(ICHAIN)=LMOT      ICHAIN=$,I5,10X,$IADR=$,I5)
114:        ICHAIN=IADR
115:        GOTO 2
116:    16  ICHAIN=IADR
117:        GOTO 17
118:    18  F=.TRUE.
119:        IALTERNANT=ICHAIN
120:        ICHAIN=IADR
121:        GOTO 15
122:     1  CONTINUE
123: C-----LE MOT EST CONTENU DANS UN MOT DE DICFCOD
124: X      WRITE(108,1006 )  ICHAIN
125: X1006 FORMAT(1HO,$CHAINE CONTENUE DANS DICFCOD  ICHAIN= $,I5)
126:        LCHAIN=CHAIN(ICHAIN)
127:        IF(LCHAIN.EQ.1R/)  GOTO 21
128:        IF(LCHAIN.EQ.1R*)  GOTO 21
129:        IALTERNANT=0
130:        LONGMOT=LONGMOT+1
131:        MOT(LONGMOT)=1R/
132:        GOTO 20
133:    21  CONTINUE
134: C******LE MOT FIGURE DANS LE DICTIONNAIRE.
135:        ADR(ICHAIN)=ADR(ICHAIN)-1
136:        GOTO 29
137: C      RECHERCHE DE LI* SUIVANT.
138: C      ICHAIN SERA L'ADRESSE DU CARACTERE SUIVANT LE *.
139:    10  ICHAIN=ICHAIN+1
140:        IF(CHAIN(ICHAIN).EQ.1R*)  GOTO 5
141:        IF(ICHAIN.GT.CHAIN(0))  GOTO 6
142:        GOTO 10
143:     6  WRITE (108,14)
144:    14  FORMAT(1HO,$ERREUR RECHERCHE *$)
145:        GOTO 29
146:     5  CONTINUE
147: X      WRITE(108,1001)  ICHAIN
148: X1001 FORMAT(1HO,$RECHERCHE DU PROCHAIN *   ICHAIN= $,I5)
149:        ICHAIN=ICHAIN+1
150: C      ON AJOUTE UN * A LA CHAINE AVANT INSERTION
151:    15  LONGMOT=LONGMOT+1
152:        MOT(LONGMOT)=1R*
153: C      INSERTION DES CARACTERES M A LONGMOT DE MOT
154: C      LE PREMIER CARACTERE A DECALER DE CHAIN A POUR ADRESSE ICHAIN
155:    20  IDECAL=LONGMOT-M+1
156: X      WRITE(108,1002)  M,LONGMOT,ICHAIN
157: X1002 FORMAT(1HO,$INSERTION CHAINE,M=$,I5,$LONGMOT=$,I8,$   ICHAIN=$I5)
158:        II=CHAIN(0)
159:        CHAIN(0)=II+IDECAL
160:        DO 11 JJ=II,ICHAIN,-1
161:        ADR(JJ+IDECAL)=ADR(JJ)
162:    11  CHAIN(JJ+IDECAL)=CHAIN(JJ)
163:        DO 12 JJ=0,IDECAL-1
164:        ADR(ICHAIN+JJ)=0
165:    12  CHAIN(ICHAIN+JJ)=MOT(M+JJ)
166:        DO 13 JJ=1,II+IDECAL
167:    13  IF(ADR(JJ).GE.ICHAIN)  ADR(JJ)=ADR(JJ)+IDECAL
```

```
168: C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
169:        IF(F)   GOTO 19
170: C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN
171:        IF (FX) ADR(ICHAIN)=ICHAIN + IDECAL
172:        IF(IALTERNANT.NE.0)  ADR(IALTERNANT)=ICHAIN
173:        GOTO 27
174: C      CHAINAGE DANS LE CAS 3.2
175:     19 ADR(ICHAIN)=ADR(IALTERNANT)
176:        ADR(IALTERNANT)=ICHAIN
177: C******ON DETERMINE LE N. MOT ET LE NB. D'ITERATIONS.
178:     2/ ADR(0)=ADR(0)-1
179:        ADR(ICHAIN+IDECAL-1)=ADR(0)*100000-1
180: C      CREATION D'UN CHAINAGE SUPPLEMENTAIRE ENTRE IALTERNANT ET ICHAIN.
181:     29 CONTINUE
182: C-------------------------------------------------------------------
183: C      FIN INSERTION DU MOT DANS DICFCOD----------------------------
184: C-------------------------------------------------------------------
185:        MDEBMOT=MSUIVMOT
186:        GOTO 68
187: C------ON A FINI DE CREER LE DICTIONNAIRE; SORTIE DES MOTS.
188: C-------------------------------------------------------------------
189:     50 CONTINUE
190:        END LABELS
191: C**************
192: C-------------------------------------------------------------------
193: C-----SORTIE DE TOUS LES MOTS DE DIC.F.COD----------------------------
194: C-------------------------------------------------------------------
195:        WRITE(108,99)
196:     99 FORMAT(1H1)
197:        SLATCH=.FALSE.
198:        I=ICHAIN=1
199:      3 LCHAIN=CHAIN(ICHAIN)
200:        IF(LCHAIN.EQ.1R/) SLATCH=.TRUE.;ICHAIN=ICHAIN+1;GOTO 2
201:        IF (LCHAIN.EQ.1R*)  ICHAIN=ICHAIN+1;GOTO 2
202:        SORT(I)=LCHAIN
203:        ADRSORT(I)=ADR(ICHAIN)
204:        I=I+1
205:        ICHAIN=ICHAIN+1
206:        GOTO 3
207:      2 IA=-ADR(ICHAIN-1)
208:        IN=IA/10000
209:        IG1=IN/10
210:        IG=IN-IG1*10
211:        IF(IG.NE.0)  GOTO 1
212:        IOC=IA-IN*10000
213:        ITOC=ITOC+IOC
214:        NMOT=NMOT+1
215:        WRITE(108,100) NMOT,(SORT(J),J=1,30),IN,IOC
216:    100 FORMAT(1HO,I5,$****$,30R1,$N. MOT=$,I10,$  NB. OCCURENCES=$,I10)
217:      1 IF(SLATCH) SLATCH=.FALSE.;GOTO 3
218:      5 IF (ADRSORT(I).EQ.0)  GOTO 4
219:        ICHAIN=ADRSORT(I)
220:        ADRSORT(I)=0
221:        GOTO 3
222:      4 SORT(I)=0
223:        I=I-1
```

```
224:         IF(I.NE.0)   GOTO 5
225:         WRITE (108,101)
226:    101  FORMAT(1H1,$FIN DE LA SORTIE DE DIC.F.COD$)
227:         END LABELS
228: C       SORTIE DU DIC
229:         WRITE(108,100) CHAI\(0)
230:    100  FORMAT(1H1,50(1H*),,LONGUEUR DE LA CHAINE :$,I10)
231:         ITOC=ITOC/NMOT
232:         WRITE(108,103) ITOC
233:    103  FORMAT(1HO,$MOYENNE OCCURENCE =$,I3)
234:         WRITE(108,101)
235:    101  FORMAT(1H1)
236:         WRITE(108,53)
237:     53  FORMAT(1H1,$*** SORTIE DE CHAIN$)
238:         STOP
239:         END
```

ANNEXE 6

PROGRAMMES

de DOCUMENTATION AUTOMATIQUE

---

1. Création du fichier inversé
   - 1.1 Structure du fichier
   - 1.2 Organigramme de création
   - 1.3 Programme "creatfichinverse"
2. Programme de documentation automatique

1. CREATION DU FICHIER INVERSE

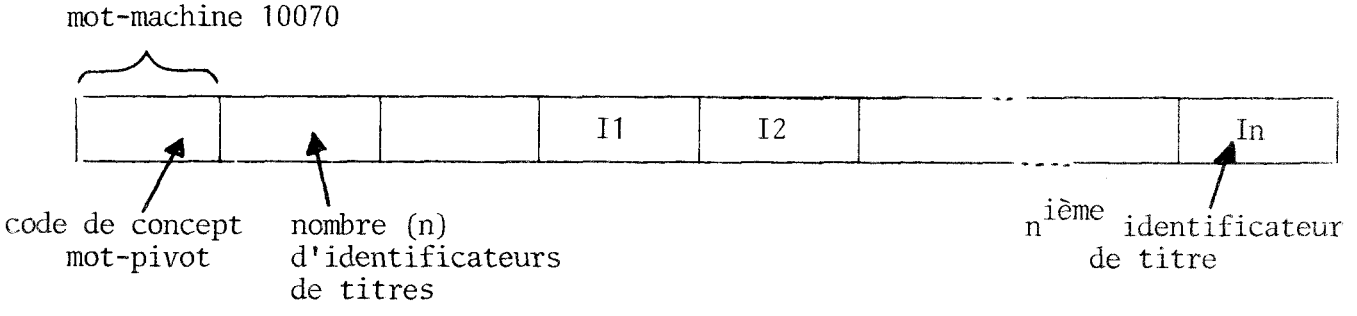
Comme le chapitre VII l'a montré, le fichier inversé va être créé à l'aide des titres, et du dictionnaire.

1.1 Structure du fichier

La clé de chaque article sera le code de concept d'un mot pivot. Ce code, numérique, sera stocké dans un mot-machine 10070, représenté sous forme d'entier. Chaque article contiendra les identificateurs des titres, représentés dans l'exemple, par des nombres entiers. Les identificateurs, représentés chacun sous forme entière dans un mot 10070, n'ont pas été choisi pour ce programme, mais provenaient d'un choix préalable fait pour la partie de traitement des informations numériques [7]. On peut dire, simplement, que pour l'ensemble des titres traités, deux titres n'avaient pas le même identificateur.

Pour faciliter les opérations de recherche documentaire effectuées à l'aide du fichier inversé, les identificateurs ont été triés en ordre croissant dans chaque article, et le nombre d'identificateurs notés dans un mot 10070.

La structure du fichier inversé est donc :



Dans l'exemple de programme de documentation donné ici, le fichier inversé est organisé comme un fichier séquentiel à articles de longueur fixe. Cette structure a été choisie, car elle s'adaptait le mieux à une utilisation sous Fortran IV, avec le software alors disponible sur le 10070. IL était prévu d'organiser par la suite le fichier sous forme séquentielle indexée, à l'aide du nouveau software SIRIS 7 du 10070, qui malheureusement a été implanté trop tard. On peut signaler, par contre, que dans la transcription sur Time-Sharing, le fichier inversé a été organisé en accès direct, sur disques magnétiques.

Le dernier article du fichier inversé a un index égal à 999999. Cet article a été créé pour faciliter les opérations d'insertion dans le programme de création, et pour tester la fin de fichier dans la transcription du programm sur time-sharing.

1.2 Organigramme de création

Pour chaque titre, le programme rechphrase permet de déterminer pour les mots pivots contenus dans le titre, les codes de concept qui leur correspondent dans le dictionnaire.

Les cas ambigus sont éliminés et sortis sur un listing d'erreur.

L'ensemble des codes de concept trouvés pour le titre sont ensuite triés par ordre croissant, dans le tableau résultat (tableaux TAB).

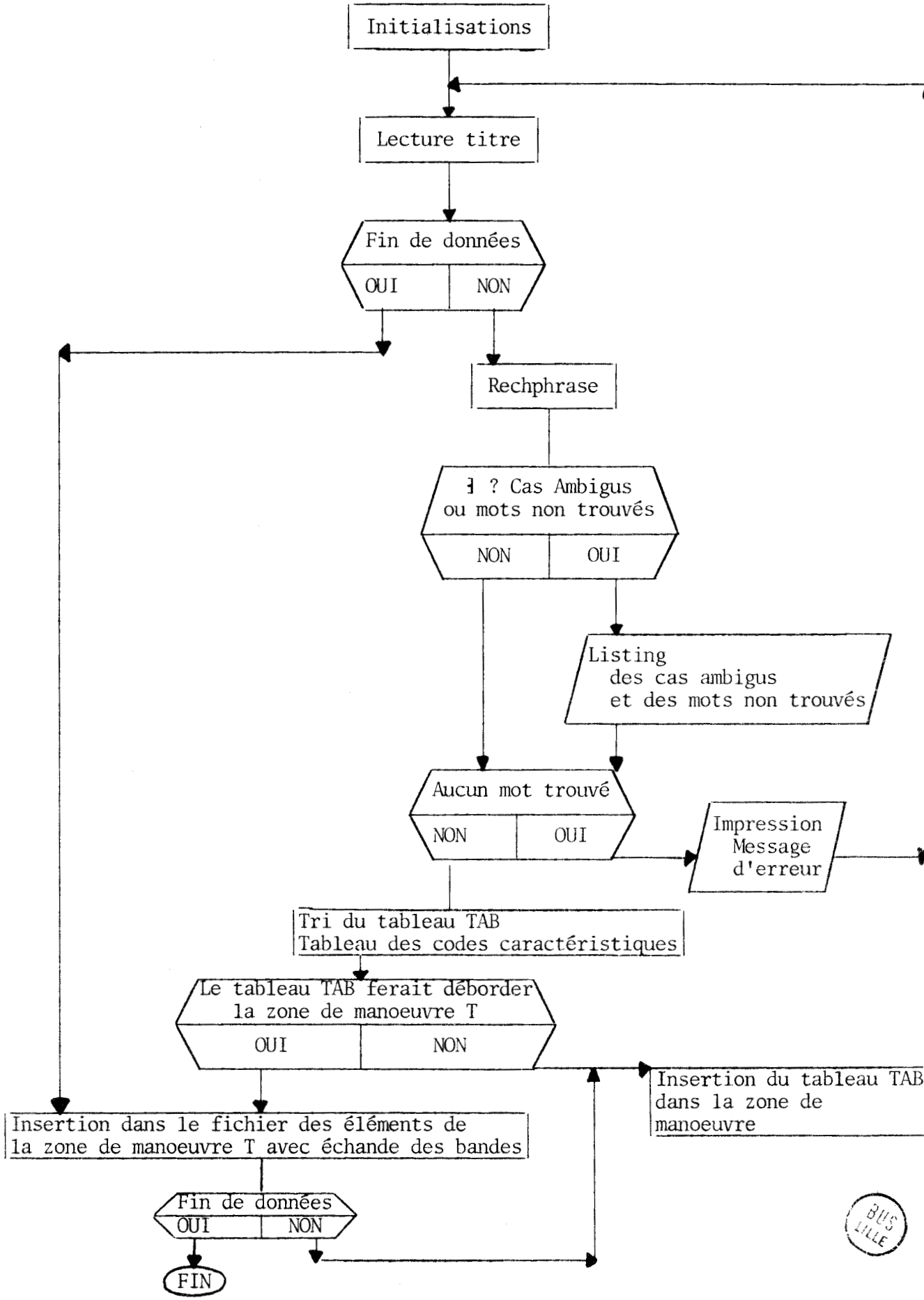
Pour limiter les balayages, les couples code de concept - identificateur de titre sont rangés dans une zone de manoeuvre (tableau T), de longueur maximum donné, et les couples sont triés par code de concept croissant.

Chaque tableau résultat est inséré dans cette zone de manoeuvre, si le volume total n'est pas supérieur à la longueur maximum donnée.

Les éléments stockés dans la zone de manoeuvre sont ensuite insérés dans le fichier. Cette méthode permet de minimiser les balayages et les mouvements de bandes magnétiques.

L'organigramme de création résume cette méthode.

ORGANIGRAMME GENERAL CREATION DU FICHIER INVERSE



1.3 Programme "creatfichinverse"

Les principaux identificateurs utilisés dans ce programme ont déjà été décrits dans l'annexe 2.

A titre indicatif, on peut signaler que la création du fichier inversé, pour 704 titres a demandé 8,7 centièmes de temps C.P.U. avec une zone de manoeuvre de longueur égale à 40 000 mots.

```
      INTEGER ADR(0:6000),CHAIN(0:6000)
      COMMON ADR,CHAIN
      INTEGER KARTE(80),RESULT(0:4),NONTROUVE(40,20),MOT(168)
      INTEGER ADRCHAIN(0:162),AMBIGU(20),NOMBRE(10),TAB(50)
      INTEGER TAMPON(0:200),NTAMPON(0:200)
      INTEGER T(2,0:40000)
      INTEGER GRAMM(4)
      LOGICAL STEM,TROUVE,SEPAR
      LOGICAL FINDONNEES
      LOGICAL FINFICHIER
C-----------------------------------------------------------------
C
C----------------------   CREAT. FICH. INVERSE.  ---------------------
C
C-----------------------------------------------------------------
C     LECTURE DE ADR ET CHAIN.
      READ(1) CHAIN
      READ(1) ADR
      END LABELS
C$$$$$$$$$$$$$$$
      LONGT=40000.
      LTAMP=200
      FINDONNEES=.FALSE.
      IBAND1=2
      IBAND2=3
      IM=160
      MOT(IM+1)=1R.
      MOT(IM+2)=1RA
C--------------------
C----CREATION DE L'ARTICLE  FOND DE FICHIER  DE FICH-INVERSE.
      TAMPON(0)=999999
      DO 4 I=2,3
      CALL BUFFER OUT(I,1,TAMPON,LTAMP+1,ITEST)
    1 GOTO (1,2,3,3) ITEST
    3 PAUSE 1
    2 CONTINUE
      ENDFILE 2
    4 CONTINUE
      REWIND 2,3
      END LABELS
C******************
C     LECTURE DE LA PHRASE.
C-----------------------------------------------------------------
      CALL EOFSET (1000S)
 1001 CONTINUE
      INPHRASE=INPHRASE+1
      WRITE(108,104) INPHRASE
  104 FORMAT(1H1,60(1H*),$ PHRASE N:$,I8,30(1H*),
      READ(105,100) NSERIE,(MOT(J),J=1,150)
  100 FORMAT(I10,70R1,/,80R1)
C     ECRITURE DE LA PHRASE.
      WRITE(108,101) NSERIE,(MOT(J),J=1,150)
  101 FORMAT(1H0,I10,2X,10R1,2X,107R1,/,X,33R1)
C---------------------------------
      INMOT=0
```

```
 56:        IXMOT=IZMOT=1
 57:        M=MDEBMOT=11
 58: C-----------------------------------
 59:     30 SEPAR=.FALSE.
 60: C     DETERMINATION DU CARACTERE SEPARATEUR-FIN DE MOT,DEBUT MOT SUIVAN
 61:        DO 26 J=MDEBMOT,IM+2
 62: C      LE - N'EST PAS UN CARACTERE SEPARATEUR : MOTS COMPOSES
 63:        LMOT=MOT(J)
 64:        IF(LMOT.EQ.1R-)  GOTO 24
 65:        IF(LMOT.LT.128) GOTO 27
 66:     24 IF(SEPAR) MSUIVMOT=J ; GOTO 35
 67:        GOTO 26
 68:     27 IF(.NOT.SEPAR) MSEPAR=J
 69:        SEPAR=.TRUE.
 70:     26 CONTINUE
 71:     35 LMOT=MOT(MDEBMOT)
 72: C      ON ELIMINE LES MOTS COMMENCANT PAR-,ET LES CHIFFRES <0
 73:        IF(LMOT.EQ.1R-)  GOTO 12
 74: C*****************************************************************
 75: C
 76: C                                   RECH-MOT
 77: C
 78: C*****************************************************************
 79:     28 STEM=.FALSE.
 80: X      WRITE(108,212) MSUIVMOT,MSEPAR
 81: X212  FORMAT(1H1,$MSUIVMOT=$,I5,5X,$MSEPAR=$,I5)
 82:        ICHAIN=1
 83:        MSTEM=MDEBMOT
 84:      1 LMOT=MOT(M)
 85: X      WRITE (108,200) M,LMOT,ICHAIN
 86: X200  FORMAT (1HO,$M= $,I5,5X,1R1,5X,$ICHAIN= $,I5)
 87:        IF(M.GE.MSEPAR)  GOTO 2
 88:      5 LCHAIN=CHAIN(ICHAIN)
 89:        IF(LCHAIN.EQ.LMOT) GOTO 3
 90:        IF (LCHAIN.NE.1R/)  GOTO 4
 91: X      WRITE (108,201) ICHAIN
 92: X201  FORMAT(1HO,$ SAUT DU /  ,ICHAIN= $,I5)
 93:        IF(ADRCHAIN(M).NE.0)  GOTO 19
 94:        IF(STEM) GOTO 19
 95: C      ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER <LATCH.
 96:        ADRCHAIN(M)=ICHAIN
 97:        MSTEM=M
 98:     19 ICHAIN=ICHAIN+1
 99:        GOTO 5
100:      3 M=M+1
101:        ICHAIN=ICHAIN+1
102:        GOTO 1
103:      4 IF(LCHAIN.NE.1R*)  GOTO  39
104:        IF(STEM) GOTO 22
105:        GOTO 6
106:     39 ICCHAIN=ICHAIN
107:        LPRCHAIN=LCHAIN
108: C      SUIVI DU CHAINAGE.
109:      8 IADR=ADR(ICCHAIN)
110: X      WRITE(108,202) ICHAIN,IADR
111: X202  FORMAT(1HO,$SUIVI DU CHAINAGE      ICHAIN=$,I5,5X,$IADR= $,I5)
```

```
112:         IF(IADR.EQ.0) GOTO 7
113:         ICCHAIN=IADR
114:         LCHAIN=CHAIN(ICCHAIN)
115:         IF (LCHAIN.LT.LMOT)  GOTO 8
116:         IF (LCHAIN.NE.LMOT)  GOTO 7
117: C       APRES SUIVI DU CHAINAGE,LCHAIN=LMOT.
118: X       WRITE(108,203) ICCHAIN
119: X203    FORMAT(1HO,$LCHAIN=LMOT   ,ICHAIN=  $,I5)
120:         ICHAIN=ICCHAIN+1
121:         M=M+1
122:         GOTO 1
123:       7 CONTINUE
124: X       WRITE(108,206) LPRCHAIN,ICCHAIN
125: X206    FORMAT(1HO,$LPRCHAIN =$,1R1,$   ICCHAIN=$,I5)
126:          IF(LPRCHAIN.EQ.1R=)  GOTO 17
127:         IF (LPRCHAIN.NE.1R )  GOTO 9
128: C       RECHERCHE DU PRCHAIN - OU 'BLANC'.
129:      17 DO 10 J=M,M+7
130:         IF(MOT(J).NE.LPRCHAIN)  GOTO 10
131:         M=J
132:         GOTO 1
133:      10 CONTINUE
134:       2 LCHAIN=CHAIN(ICHAIN)
135: Y       WRITE(108,208)  LCHAIN,ICHAIN
136: X208    FORMAT (1HO,$SORTIE PAR 2: LCHAIN=$,1R1,5X,$   ICHAIN=$,I5)
137:         IF(LCHAIN.EQ.1R/) GOTO 21
138:         IF(LCHAIN.NE.1R*)  GOTO 9
139:      21 CONTINUE
140: X       WRITE(108,204)
141: X204    FORMAT(1HO,$MOT COMPLET DANS DIC-STEM$)
142:         IF(STEM)  GOTO 11
143:         ICSTEM=ICHAIN
144:         ICHAIN=2
145:         STEM=.TRUE.
146:         GOTO 11
147:       9 IF(STEM)  GOTO 22
148: C       RECHERCHE DU PLUS LONG STEM
149:      23 DO 12 J=MSTEM,MDEBMOT,-1
150:         IF(ADRCHAIN(J).EQ.0)  GOTO 12
151:         ICHAIN=ADRCHAIN(J)
152:         M=J
153:         ADRCHAIN(J)=0
154:         GOTO 6
155:      12 CONTINUE
156: C+++++STOCKAGE DU MOT NON TROUVE.
157:         JX=0
158:         DO 32 J=1,20
159:      32 NONTROUVE(IZMOT,J)=1R
160:         DO 29 J=MDEBMOT,MSEPAR-1,1
161:         JX=JX+1
162:      29 NONTROUVE(IZMOT,JX)=MOT(J)
163:         IZMOT=IZMOT+1
164:         IF(MSUIVMOT.GT.IM)     GOTO 13
165:         M=MDEBMOT=MSUIVMOT
166:         GOTO 30
167:       6 STEM=.TRUE.
```

```
C       ON A LE PLUS LONG STEM.
X       WRITE(108,205) M-MDEBMOT,(MOT(J),J=MDEBMOT,M-1)
X205    FORMAT(1HO,$PLUS LONG STEM =      $,N(1R1))
        IF(STEM)  ICSTEM=ICHAIN
C       RECHERCHE DU SUFFIXE.
X       WRITE(108,207) M
X207    FORMAT(1HO,$RECHERCHE DU SUFFIXE  M=$,I5)
        ICHAIN=1
        GOTO 1
C***
   11 CONTINUE
C       RECHERCHE DU SUFFIXE COMPATIBLE
X       WRITE(108,209) ICSTEM
X209    FORMAT(1HO,$RECHERCHE DU N. SUFFIXE   ICSTEM =$,I5)
        I=0
        TROUVE=.FALSE.
C       RECHERCHE DE G. SUFFIXE
        J=ADR(ICHAIN)
   20   IMP=J/100000
        N1=-J+IMP*100000
        IGS=N1/10000
        IF(IGS.NE.9)   GOTO 15
C       RECHERCHE DU N. SUFFIXE
        IMP=J/1000
        N1=-J+IMP*1000
C       RECHERCHE ISUF.
        ICC=ICSTEM
        IADRSTEM=ADR(ICC)
   16   IMP=-IADRSTEM/1000
        ISUF=-IADRSTEM-IMP*1000
        IF(N1-ISUF) 18,40,18
C       MOT TROUVE
   40   TROUVE=.TRUE.
X       WRITE(108,210) M
X210    FORMAT(1H ,$    M=  $,I5)
X       WRITE(108,211)  IMP
X211    FORMAT(1HO,10X,$ ** MOT TROUVE **          N.: $,I5)
        IGX=IMP/100
        IG=IMP-IGX*100
        IF(IG.EQ.90) GOTO 18
        I=I+1
        GRAMM(I)=IG
        RESULT(0)=I
        RESULT(I)=IGX
   18   ICC=ICC+1
        IADRSTEM=ADR(ICC)
        IF(IADRSTEM.GE.0)   GOTO 15
        GOTO 16
   15   ICHAIN=ICHAIN+1
        J=ADR(ICHAIN)
        IF(J.LT.0)  GOTO 20
C       EXPLORATION TERMINEE.
        IF(TROUVE)   GOTO 14
C+++++LE PLUS LONG STEM ASSOCIE NE CONVIENT PAS:RECHERCHE STEM PLUS COU
        GOTO 23
   22   CONTINUE
```

```
224: C+++++ON A TROUVE UN STEM,MAIS LE SUFFIXE ASSOCIE N'EXISTE PAS
225:        GOTO 23
226:     14 CONTINUE
227: C+-+--ON A TROUVE LE MOT.
228:        IF(RESULT(0).EQ.0)  GOTO 23
229:        JX=0
230:        DO 41 J=MDEBMOT,MSEPAR-1
231:        JX=JX+1
232:     41 AMBIGU(JX)=MOT(J)
233:        WRITE(108,102) (AMBIGU(J),J=1,20),((RESULT(J),GRAMM(J)),J=1,RESULT
234:       -(0))
235:    102 FORMAT(1HO,$***$,20R1,4(5X,I8,2X,I2))
236: C      REMISE A ZERO DU TABLEAU AMBIGU.
237:        DO 34 J=1,20
238:     34 AMBIGU(J)=1R
239:        DO 58 J=1,RESULT(0)
240:        IF(GRAMM(J).GE.10) GOTO 58
241:        TAB(IXMOT)=RESULT(J)
242:        IXMOT=IXMOT+1
243:     58 CONTINUE
244: C      REMISE A ZERO DU TABLEAU RESULT.
245:        DO 33 J=0,4
246:     33 RESULT(J)=0
247:     42 IF(MSUIVMOT.GT.IM)  GOTO 13
248:        M=MDEBMOT=MSUIVMOT
249:        GOTO 30
250:     13 CONTINUE
251: C*****************************************************************
252: C
253: C                              FIN DE RECH-MOT
254: C
255: C*****************************************************************
256: C      ON A FINI L'ANALYSE DE LA PHRASE
257:         DO 43 J=1,IM
258:        ADRCHAIN(J)=0
259:     43 MOT(J)=1R
260: C      IXMOT=NB. DE MOTS TROUVES+1
261: C      IZMOT=NB. DE MOTS NON TROUVES+1
262: C      IMPRESSION DES MOTS TROUVES.
263: C      LE TABLEAU TAB CONTIENT LES N. DES MOTS NON AMBIGUS
264:        IF(IZMOT.EQ.1) GOTO 47
265:        IZMOT=IZMOT-1
266:        WRITE(108,106)
267:    106 FORMAT(1HO,$******************  MOTS NON TROUVES$)
268:        DO 31 J=1,IZMOT
269:     31 WRITE(108,103) (NONTROUVE(J,M),M=1,20)
270:    103 FORMAT(1HO,10X,20R1)
271:     47 CONTINUE
272:        IF(IXMOT.NE.1)  GOTO 46
273:        WRITE(108,107)
274:    107 FORMAT(1HO,$--------------AUCUN MOT CARACTERISTIQUE TROUVE-----$)
275:        GOTO 1001
276:     46 IXMOT=IXMOT-1
277: C
278: C  ------------FIN DE L'ANALYSE DE LA PHRASE
279: C
```

```
280: C*****************************************************************
281: C
282: C                    TRI DU TABLEAU TAB
283: C
284: C*****************************************************************
285:       IF(IXMAT.EQ.1) GOTO 50
286:       DO 51 I=2,IXMAT
287:       DO 52 J=1,I-1
288:       IF(TAB(I).GT.TAB(J)) GOTO 52
289:       IF(TAB(I).EQ.TAB(J))  GOTO 55
290:       IGS=J
291:       GOTO 53
292:    52 CONTINUE
293:       GOTO 51
294:    53 IG=TAB(I)
295:       DO 54 IGX=I-1,IGS,-1
296:    54 TAB(IGX+1)=TAB(IGX)
297:       TAB(IGS)=IG
298:       GOTO 51
299:    55 IXMAT=IXMAT-1
300:       DO 56 J=I,IXMAT
301:    56 TAB(J)=TAB(J+1)
302:       I=I-1
303:    51 CONTINUE
304:    50 CONTINUE
305:       WRITE(108,110)
306:   110 FORMAT(1HO,$***SORTIE DE TAB TRIEE$)
307:       DO 57 J=1,IXMAT
308:    57 WRITE(108,111) J,TAB(J)
309:   111 FORMAT(1HO,I2,5X,I10)
310:       GOTO 69
311: C*****************************************************************
312: C
313: C                INSERTION DE TAB DANS T
314: C
315: C*****************************************************************
316:  1000  FINDONNEES=.TRUE.
317:       WRITE(108,1003)
318:  1003 FORMAT(1H1,50(1H*),$FIN DES DONNEES $)
319:       GOTO 77
320:    69 CONTINUE
321:       IMAX =T(2,0)
322:        IF(IMAX+IXMAT.GT.LONGT)  GOTO 1002
323:       I=0
324:       DO 70 J=1,IXMAT
325:    73 IF(T(1,I)-TAB(J)) 71,71,72
326:    71 I=I+1
327:       IF(I.LE.IMAX) GOTO 73
328:       IMAX=IMAX+1
329:       T(1,I)=TAB(J)
330:       T(2,I)=NSERIE
331:       GOTO 70
332:    72 DO 74 K=IMAX,I,-1
333:       T(1,K+1)=T(1,K)
334:    74 T(2,K+1)=T(2,K)
335:       IMAX=IMAX+1
```

```
336:         T(1,I)=TAB(J)
337:         T(2,I)=NSERIE
338:         I=I+1
339:         IF(I.GT.IMAX) I=IMAX
340:      70 CONTINUE
341:         T(2,0)=IMAX
342: C       ECRITURE DE T.
343:      77 CONTINUE
344:         WRITE(108,105) IMAX
345:     105 FORMAT(1H0,60(1H*),$NB. D'ELEMENTS DANS T$,I10)
346:         IF(FINDONNEES)  GOTO 1004
347:         GOTO 1001
348:    1002 WRITE(108,1008)
349:    1008 FORMAT(1H1,50X,$ARRET REMPLISSAGE T$)
350: C              FIN    INSERTION  DE  TAB   DANS    T.
351: C-------------------------------------------------------------------
352: C*******************************************************************
353: C
354: C              INSERTION DANS LE FICHIER DES ELEMENTS DE T.
355: C
356: C*******************************************************************
357:    1004 CONTINUE
358:         FINFICHIER=.FALSE.
359:         WRITE(108,300) IBAND2
360:     300 FORMAT(1H1,50(1H*),$NOUVEAU FICHIER SUR BANDE$,I2,50(1H*))
361:         WRITE(108,400)
362:     400 FORMAT(1H1)
363:         JMAX=T(2,0)
364:         J=1
365:         ASSIGN 1005 TO ITL1
366:         ASSIGN 307 TO ITL2
367: C------------------------ LECTURE D'UN ARTICLE ---------------------
368:     319 CONTINUE
369:         CALL BUFFER IN(IBAND1,1,TAMPON,LTAMP+1,ITEST)
370:     301 GOTO(301,302,303,304) ITEST
371:     303 WRITE(108,306)
372:     306 FORMAT(1H1,60X,$**** FIN DE FICHIER LECTURE$)
373:         FINFICHIER=.TRUE.
374:         GOTO ITL1
375:     304 WRITE(108,305)
376:     305 FORMAT(1H1,60X,$ ERREUR=2 LECTURE : TRANSMISSION$)
377:         GOTO 1005
378:     302 GOTO ITL2
379: C---------------- FIN DE LECTURE D'UN ARTICLE ---------------------
380:     307 CONTINUE
381:         IF(TAMPON(0)-T(1,J)) 308,309,310
382: C*******
383:     310 CONTINUE
384: C    CAS N.ARTICLE > N.MAT ; CREATION D'UN NOUVEL ARTICLE.
385:         DO 311 I=1,LTAMP
386:     311 NTAMPON(I)=0
387:         NTAMPON(0)=T(1,J)
388:         NTAMPON(1)=1
389:         NTAMPON(2)=T(2,J)
390:     330 NANCIENMAT=T(1,J)
391:         J=J+1
```

```
392:        IF(J.LE.JMAX )  GOTO 312
393:        ASSIGN 320 TO IEC1
394: C------------------- ECRITURE D'UN NOUVEL ARTICLE. --------------------
395:    329 CONTINUE
396:        WRITE(108,401) NTAMPON(0)
397:    401 FORMAT(1HO,$ARTICLE N:$,I10)
398:        CALL BUFFER OUT(IBAND2,1,NTAMPON,LTAMP+1,ITEST)
399:    313 GOTO (313,314,315,316) ITEST
400:    315 WRITE(108,317) NTAMPON(0)
401:    317 FORMAT(1H1,60X,I4,$ERREUR ECRITURE NOUVEL ARTICLE : FF$)
402:        GOTO 1005
403:    316 WRITE(108,318) NTAMPON(0)
404:    318 FORMAT(1H1,60X,I4,$ERREUR ECRITURE NOUVEL ARTICLE : TRANSMISSION$)
405:        GOTO 1005
406:    314 CONTINUE
407:        GOTO IEC1
408:    ---------------------- FIN ECRITURE NOUVEL ARTICLE ------------------
        320 CONTINUE
            ASSIGN 1006 TO ITL1
            ASSIGN 325 TO ITL2
  2: C************** ON RECOPIE LA FIN DU FICHIER ***********************
 13: C--------------------- ECRITURE D'UN ARTICLE DANS N-FICHIER ------------
 14:    325 CONTINUE
 15:    321 GOTO (321,322,323,324) ITEST
 16:    323 WRITE(108,326)
 17:    326 FORMAT(1H1,60X,$ERREUR ECRITURE ARTICLE : FF$)
 18:         CALL BUFFER OUT(IBAND2,1,TAMPON,LTAMP+1,ITEST)
 19:        GOTO 1005
 20:    324 WRITE(108,327)
 21:    327 FORMAT(1H1,60X,$ERREUR ECRITURE ARTICLE : TRANSMISSION$)
 22:        GOTO 1005
 23:    322 CONTINUE
 24:        IF(FINFICHIER)  GOTO ITL1
 25: C------ LECTURE ARTICLE SUIVANT.
 26:        GOTO 319
 27: C************ FIN RECOPIE DE FIN DU FICHIER ***********************
 28:    312 IF(NANCIEN#OT.EQ.T(1,J)) GOTO 328
 29:        ASSIGN 307 TO IEC1
 30: C      ECRITURE NOUVEL ARTICLE DANS LE FICHIER.
 31:        GOTO 329
 32:    328 CONTINUE
 33: C------------ INSERTION DE T(2,J) DANS LE NOUVEL ARTICLE .
 34:        DO 331 I=2,NTAMPON(1)+1
 35:        IF(TAMPON(I)-T(2,J)) 331,332,333
 36:    333 II=I
 37:        GOTO 334
 38:    331 CONTINUE
 39:        II=NTAMPON(1)+2
 40:        GOTO 336
 41:    334 DO 335 I=NTAMPON(1)+1,II,-1
 42:    335 NTAMPON(I+1)=NTAMPON(I)
 43:    336 NTAMPON(II)=T(2,J)
 44:        NTAMPON(1)=NTAMPON(1)+1
 45:    332 GOTO 330
 46: C*******
 47:    309 CONTINUE
```

```
448: C------------ INSERTION DE T(2,J) DANS L'ARTICLE.
449:       DO 341 I=2,TAMPON(1)+1
450:       IF(TAMPON(I)-T(2,J)) 341,342,343
451:   343 II=I
452:       GOTO 344
453:   341 CONTINUE
454:       II=TAMPON(1)+2
455:       GOTO 346
456:   344 DO 345 I=TAMPON(1)+1,II,-1
457:   345 TAMPON(I+1)=TAMPON(I)
458:   346 TAMPON(II)=T(2,J)
459:       TAMPON(1)=TAMPON(1)+1
460:   342 CONTINUE
461:       J=J+1
462:       IF(J.LE.JMAX ) GOTO 307
463: C   ON RECOPIE LA FIN DU FICHIER.
464:       GOTO 320
465: C*******
466:   308 CONTINUE
467:       ASSIGN 1006 TO ITL1
468:       ASSIGN 307 TO ITL2
469:       GOTO 325
470: C***************************** FIN INSERTION ***********************
471:  1006 ENDFILE IBAND2
472:       REWIND 2,3
473:       IF(FINDONNEES) GOTO 1007
474: C                  ECHANGE DES BANDES
475:       J=IBAND1
476:       IBAND1=IBAND2
477:       IBAND2=J
478:       T(2,0)=0
479:       WRITE(108,347)
480:   347 FORMAT(1H1,45(1H*),$ECHANGE DES BANDES$,45(1H*))
481:       GOTO 69
482:  1007 WRITE(108,1009) IBAND2
483:  1009 FORMAT(1H1,60(1H*),$FIN DE CREATFICHINVERSE;FICHIER SUR DEROULEUR
484:     -:$,I2,25(1H*))
485:  1005 CONTINUE
486:       STOP
487:       END
```

2. PROGRAMME DE DOCUMENTATION AUTOMATIQUE

Le programme décrit dans ce paragraphe a été testé sur le CII 10070. Comme le software et le hardware disponibles ne permettaient pas une utilisation en time-sharing, le dialogue utilisateur-ordinateur a été simulé à l'aide de la console-maître du pupitre. Ce qui explique que la plupart des ordres de sortie ont été doublés : il y a sortie simultanée sur une des imprimantes du système, et sur la console. Par contre, toutes les entrées ont lieu sur la console. Un exemple de dialogue, comportant plusieurs transactions avec le système est donné à la suite du programme.

Ce programme a été transcrit, sur time-sharing, en respectant la même structure. Comme la place mémoire disponible ne permettait pas de loger le programme complet, il a été séparé en deux liens, tout en gardant la même organisation.

Les principaux identificateurs utilisés ici sont déjà décrits dans l'annexe 3. On peut remarquer que les fichiers utilisés sur le 10070 ont été organisés sur disque, en accès séquentiel, car c'est pratiquement le seul mode d'organisation accessible à partir du Fortran avec le software qui était alors disponible. Cette organisation avait été retenue, car les fichiers traités dans l'exemple ne sont pas très volumineux, et la réorganisation en accès direct sur disque pour le système de Time-sharing ne posait pas de difficulté de programmation.

```
      INTEGER CHAIN(0:6000),ADR(0:6000)
      COMMON ADR,CHAIN
      INTEGER KARTE(80),RESULT(2,0:6),NONTROUVE(40,20),MOT(0:220)
      INTEGER ADRCHAIN(0:218),TAB(2,50),AMBIGU(20)
      INTEGER T(0:300,15),TIT(11,0:4)
      LOGICAL STEM,TROUVE,SEPAR
C-----------------------------------------------------------------
C
C------------------------------ DOC.AUTO  ------------------------------
C
C-----------------------------------------------------------------
C  IMAXTAB= NB. MAXIMUM DE COLONNES DE T.
C  INTAB= N. DE LA COLONNE OU S'EFFECTUE LA TRANSACTION.
C  LTAMP=LONGUEUR DES ARTICLES DU FICHIER INVERSE.
C  IMAXAMB= NB. MAXIMUM DE MOTS AMBIGUS ADMIS.
C  IM1= LONGUEUR DE L'ARTICLE DU FICHIER DES TITRES .
C  IM=LONGUEUR MAXI. DE LA PHRASE A ANALYSER.
C  IX=NB. MAXIMUM DE MOTS-PIVOTS A STOCKER DANS TAB.
C  LE DICTIONNAIRE SE COMPOSE DE 2 TABLEAUX :
C     CHAIN= TABLEAU DES CHAINES DE CARACTERES.
C     ADR=TABLEAU DES POINTEURS ET DES IDENTIFICATEURS.
C.................................................................
C     LECTURE DE ADR ET CHAIN.
      READ(1) CHAIN
      READ(1)  ADR
      WRITE(108,1111)
 1111 FORMAT(1H1,' FIN ENTREE DU DICTIONNAIRE')
      REWIND 1,2,3
C-----------FIN DE LA LECTURE DE LA CHAINE
      IM=216
      IX=20
      IM1=160
      MOT(IM+1)=1R.
      MOT(IM+2)=1RA
      IMAXTAB=11
      IMAXAMB=4
      INTAB=0
      LTAMP=200
C     INITIALISATION DU TABLEAU TIT.
      DO 1019 J=1,IMAXTAB
 1019 TIT(J,2)=TIT(J,3)=4H
C*****************************************************************
C
C        DETECTION DE L'OPERATION.
C
C*****************************************************************
 1001 CONTINUE
      IP=IP+1
C  LECTURE D'UNE OPERATION,OU DU DEBUT D'UNE PHRASE.
      DO 49 J=1,18
   49 AMBIGU(J)=0
      READ(101,100) (AMBIGU(J),J=1,18)
  100 FORMAT(18A4)
      DECODE(72,320,AMBIGU) KARTE
  320 FORMAT(72R1)
```

```
      IF(KARTE(1).NE.1R') GOTO 1002
C ON A DETECTE UNE OPERATION DIFFERENTE DE LA RECHERCHE D'UNE PHRASE.
      DECODE(72,323,AMBIGU) IOP,J1,J2,J3
  323 FORMAT(A4,3I)
      WRITE(108,322) IP,IOP,J1,J2,J3
  322 FORMAT(1H1,'**** OPERATION N:',I3,' = ',A4,',',I3,',',I3,',',I3)
      IF(J1.GT.IMAXTAB)  GOTO 1010
      IF(J1.LT.0)  GOTO 1010
      IF(J2.GT.IMAXTAB)  GOTO 1010
      IF(J2.LT.0)  GOTO 1010
      IF(J3.GT.IMAXTAB)  GOTO 1010
      IF(J3.LT.0)  GOTO 1010
C  DETECTION DU CODE OPERATION.
      IF(IOP.EQ.4H'ET')    GOTO  1003
      IF(IOP.EQ.4H'OU')    GOTO  1004
      IF(IOP.EQ.4H'NS')    GOTO  1008
      IF(IOP.EQ.4H'TI')    GOTO  1005
      IF(IOP.EQ.4H'SA')    GOTO  1009
      IF(IOP.EQ.4H'TC')    GOTO 1017
      IF(IOP.EQ.4H'NC')      GOTO  1020
      IF(IOP.EQ.4H'SU')    GOTO  1006
      IF(IOP.EQ.4H'TR')      GOTO  1013
      IF(IOP.EQ.4H'ST')    GOTO  1007
      WRITE(108,321) IOP
  321 FORMAT(1HO,'******** ERREUR TYPE OPERATION',A4)
      GOTO 1001
 1010 WRITE(108,1011)
 1011 FORMAT(1HO,' ERREUR : HORS LIMITES T')
      GOTO 1001
 1015 WRITE(108,1016)
 1016 FORMAT(1H1,'ON SORT DES LIMITES DE T : INTAB=IMAXTAB ; UTILISER
     -L'OPERATION 'ST'')
      GOTO 1001
 1021 WRITE(108,1022)
 1022 FORMAT(1HO,'**LES 2 PREMIERS PARAMETRES DOIVENT ETRE NON NULS')
      GOTO 1001
C-------------------
C     LECTURE DE LA PHRASE.
C-----------------------------------------------------------------------
 1002 INTAB=INTAB+1
      IF(INTAB.EQ.IMAXTAB) GOTO 1015
      ID=0
   45 DO 25 J=1,72
   25 MOT(J+ID)=KARTE(J)
C  LECTURE DE LA SUITE DE LA PHRASE.
      READ(101,108)  (KARTE(J),J=1,72)
  108 FORMAT(72R1)
C  LA PHRASE SE TERMINE PAR UN * EN COLONNE 1.
      IF(KARTE(1).EQ.1R*)  GOTO 44
      ID=ID+72
      IF(ID.LE.IM) GOTO 45
      WRITE(108,105)
  105 FORMAT(1H1,'**** PHRASE TROP LONGUE , RECOMMENCEZ')
   47 INTAB=INTAB-1
      DO 48 J=0,IM
   48 MOT(J)=0
```

```
      GOTO 1001
   44 CONTINUE
C     ECRITURE DE LA PHRASE.
      INPHRASE=INPHRASE+1
      WRITE(108,109) INPHRASE
  109 FORMAT(1H1,$*** PHRASE N ! *,I3)
      WRITE(108,101) (MOT(J),J=1,160)
  101 FORMAT(1HO,80R1,/,80R1)
C-----------------------------------
      INAMB=1
      IXMOT=0
      IZMOT=1
      M=MDEBMOT=1
C-----------------------------------

   30 SEPAR=.FALSE.
C    DETERMINATION DU CARACTERE SEPARATEUR=FIN DE MOT,DEBUT MOT SUIVANT
      DO 26 J=MDEBMOT,IM+2
C     LE . N'EST PAS UN CARACTERE SEPARATEUR ! MOTS COMPOSES
      LMOT=MOT(J)
      IF(LMOT.EQ.1R.)  GOTO 24
      IF(LMOT.LT.128) GOTO 27
   24 IF(SEPAR) MSUIVMOT=J ; GOTO 35
      GOTO 26
   27 IF(.NOT.SEPAR) MSEPAR=J
      SEPAR=.TRUE.
   26 CONTINUE
   35 LMOT=MOT(MDEBMOT)
C     ON ELIMINE LES MOTS COMMENCANT PAR.,ET LES CHIFFRES <0
      IF(LMOT.EQ.1R.)  GOTO 12
C.....................................................................
C
C                                   RECH=MOT
C
C.....................................................................
      STEM=.FALSE.
X     WRITE(108,212) MSUIVMOT,MSEPAR
X212  FORMAT(1H1,$MSUIVMOT=$,I5,5X,$MSEPAR=$,I5)
      ICHAIN=1
      MSTEM=MDEBMOT
    1 LMOT=MOT(M)
X     WRITE (108,200) M,LMOT,ICHAIN
X200  FORMAT (1HO,$M= $,I5,5X,1R1,5X,$ICHAIN= $,I5)
      IF(M.GE.MSEPAR)  GOTO 2
    5 LCHAIN=CHAIN(ICHAIN)
      IF(LCHAIN.EQ.LMOT) GOTO 3
      IF (LCHAIN.NE.1R/)  GOTO 4
X     WRITE (108,201) ICHAIN
X201  FORMAT(1HO,$ SAUT DU /  ,ICHAIN= $,I5)
      IF(ADRCHAIN(M).NE.0)  GOTO 19
      IF(STEM) GOTO 19
C     ADRCHAIN VA CONTENIR L'ADRESSE DU PREMIER SLATCH.
      ADRCHAIN(M)=ICHAIN
      MSTEM=M
   19 ICHAIN=ICHAIN+1
      GOTO 5
```

```
168:      3 M=M+1
169:        ICHAIN=ICHAIN+1
170:        GOTO 1
171: C      LCHAIN='* ',ET ON N'A PAS M=MSEPAR.
172:      4 IF(LCHAIN.NE.1R*)  GOTO  39
173:        IF(STEM) GOTO 22
174:        GOTO 6
175:     39 ICCHAIN=ICHAIN
176:        LPRCHAIN=LCHAIN
177: C      SUIVI DU CHAINAGE.
178:      8 IADR=ADR(ICCHAIN)
179: X      WRITE(108,202) ICHAIN,IADR
180: X202  FORMAT(1HO,'SUIVI DU CHAINAGE      ICHAIN=',I5,5X,'IADR= ',I5)
181:        IF(IADR.EQ.0) GOTO 7
182:        ICCHAIN=IADR
183:        LCHAIN=CHAIN(ICCHAIN)
184:        IF (LCHAIN.LT.LMOT)  GOTO 8
185:        IF (LCHAIN.NE.LMOT)  GOTO 7
186: C      APRES SUIVI DU CHAINAGE,LCHAIN=LMOT.
187: X      WRITE(108,203) ICCHAIN
188: X203  FORMAT(1HO,'LCHAIN=LMOT   ,ICHAIN=  ',I5)
189:        ICHAIN=ICCHAIN+1
190:        M=M+1
191:        GOTO 1
192:      7 CONTINUE
193: X      WRITE(108,206) LPRCHAIN,ICCHAIN
194: X206  FORMAT(1HO,'LPRCHAIN =',1R1,'   ICCHAIN=',I5)
195:         IF(LPRCHAIN.EQ.1R=)  GOTO 17
196:        IF (LPRCHAIN.NE.1R )  GOTO 9
197: C      RECHERCHE DU PROCHAIN = OU 'BLANC'.
198:     17 DO 10 J=M,M+4
199:        IF(MOT(J).NE.LPRCHAIN)  GOTO 10
200:        M=J
201:        GOTO 1
202:     10 CONTINUE
203: C      ON RECHERCHE UN MOT COMPOSE SANS = .
204:        ICHAIN=ICCHAIN+1
205:        GOTO 1
206:      2 LCHAIN=CHAIN(ICHAIN)
207: X      WRITE(108,208)  LCHAIN,ICHAIN
208: X208  FORMAT (1HO,'SORTIE PAR 2: LCHAIN=',1R1,5X,'   ICHAIN=',I5)
209:        IF(LCHAIN.EQ.1R/) GOTO 21
210:        IF(LCHAIN.NE.1R*)  GOTO 9
211:     21 CONTINUE
212: X      WRITE(108,204)
213: X204  FORMAT(1HO,'MOT COMPLET DANS DIC=STEM')
214:        IF(STEM)  GOTO 11
215:        ICSTEM=ICHAIN
216:        ICHAIN=2
217:        MSTEM=M
218:        STEM=.TRUE.
219:        GOTO 11
220:      9 IF(STEM)  GOTO 22
221: C      RECHERCHE DU PLUS LONG STEM
222:     23 DO 12 J=MSTEM,MDEBMOT,-1
223:        IF(ADRCHAIN(J).EQ.0)  GOTO 12
```

```
      ICHAIN=ADRCHAIN(J)
      M=J
      ADRCHAIN(J)=0
      GOTO 6
   12 CONTINUE
C+++++STOCKAGE DU MOT NON TROUVE.
      JX=0
      DO 32 J=1,20
   32 NONTROUVE(IZMOT,J)=1R
      DO 29 J=MDEBMOT,MSEPAR-1,1
      JX=JX+1
   29 NONTROUVE(IZMOT,JX)=MOT(J)
      IZMOT=IZMOT+1
      IF(MSUIVMOT.GT.IM)    GOTO 13
      M=MDEBMOT=MSUIVMOT
      GOTO 30
    6 STEM=.TRUE.
      MSTEM=M
C     ON A LE PLUS LONG STEM.
X     WRITE(108,205) M=MDEBMOT,(MOT(J),J=MDEBMOT,M-1)
X205  FORMAT(1HO,$PLUS LONG STEM =     $,N(1R1))
      IF(STEM)  ICSTEM=ICHAIN
C     RECHERCHE DU SUFFIXE.
X     WRITE(108,207) M
X207  FORMAT(1HO,$RECHERCHE DU SUFFIXE  M=$,I5)
      ICHAIN=1
      GOTO 1
C***
   11 CONTINUE
C     RECHERCHE DU SUFFIXE COMPATIBLE
X     WRITE(108,209) ICSTEM
X209  FORMAT(1HO,$RECHERCHE DU N. SUFFIXE   ICSTEM =$,I5)
      I=0
      RESULT(1,0)=0
      TROUVE=.FALSE.
C     RECHERCHE DE G. SUFFIXE
      J=ADR(ICHAIN)
   20 IMP=J/100000
      N1=J+IMP*100000
      IGS=N1/10000
C  IGS=CODE GRAMM. DE LA PARTIE #SUFFIXE#.
      IF(IGS.NE.9)   GOTO 15
C     RECHERCHE DU N. SUFFIXE
      IMP=J/1000
      N1=J+IMP*1000
C     RECHERCHE ISUF=SUFFIXE DE LA PREMIERE PARTIE #STEM#.
      ICC=ICSTEM
      IADRSTEM=ADR(ICC)
   16 IMP=IADRSTEM/1000
      ISUF=IADRSTEM-IMP*1000
      IF(N1-ISUF) 18,40,18
C     MOT TROUVE
   40 TROUVE=.TRUE.
X     WRITE(108,210) M
X210  FORMAT(1H ,$     M=  $,I5)
X     WRITE(108,211)  IMP
```

```
X211  FORMAT(1HO,10X,$ ** MOT TROUVE **          N.I $,I5)
C     IGX=N. DU MOT.
      IGX=IMP/100
C     IG=CODE GRAMM.*10+CODE SIGMA=MOTS.
      IG=IMP-IGX*100
      IF(IG.EQ.90) GOTO 18
      I=I+1
      RESULT(1,0)=I
      RESULT(1,I)=IGX
      RESULT(2,I)=IG
   18 ICC=ICC+1
      IADRSTEM=ADR(ICC)
      IF(IADRSTEM.GE.0)   GOTO 15
      GOTO 16
   15 ICHAIN=ICHAIN+1
      J=ADR(ICHAIN)
      IF(J.LT.0)  GOTO 20
C     EXPLORATION TERMINEE.
      IF(TROUVE)   GOTO 14
C+++++LE PLUS LONG STEM ASSOCIE NE CONVIENT PAS;RECHERCHE STEM PLUS COURT
      GOTO 23
   22 CONTINUE
C+++++ON A TROUVE UN STEM,MAIS LE SUFFIXE ASSOCIE N'EXISTE PAS
      GOTO 23
   14 CONTINUE
C+-+--ON A TROUVE LE MOT.
C    ON REMPLIT LE TABLEAU TAB.
      GOTO (23,36),RESULT(1,0)+1
C    ON A UN CAS AMBIGU
      ICOMPT=0
      DO 34 J=1,RESULT(1,0)
      IG=RESULT(2,J)
C     ON ELIMINE LES MOTS NON PIVOTS.
      IF(IG.GE.10) GOTO 34
C     ON DETERMINE LES MOTS-PIVOTS=SIGMA(MOTS-PIVOTS)
      IF(IG.EQ.1) GOTO 38
      ICOMPT=ICOMPT+1
      IXMOT=IXMOT+1
      IF(IXMOT.GT.IX) GOTO 37
      TAB(1,IXMOT)=RESULT(1,J)
      TAB(2,IXMOT)=INAMB
      GOTO 34
   38 IXMOT=IXMOT+1
      IF(IXMOT.GT.IX) GOTO 37
      TAB(1,IXMOT)=RESULT(1,J)
      TAB(2,IXMOT)=0
   34 CONTINUE
      IF(ICOMPT.LE.1) TAB(2,IXMOT)=0 ; GOTO 42
C     REMISE A ZERO DU TABLEAU AMBIGU.
      DO 33 J=1,20
   33 AMBIGU(J)=1R
C     IMPRESSION DES CAS AMBIGUS.
      JX=0
      DO 41 J=MDEBMOT,MSEPAR-1
      JX=JX+1
   41 AMBIGU(JX)=MOT(J)
```

```
      WRITE(108,102) (AMBIGU(J),J=1,20),(RESULT(1,J),J=1,RESULT(1,0))
  102 FORMAT(1HO,$*** MOT AMBIGU=*,2X,20R1,6(2X,I4))
      INAMB=INAMB+1
      IF(INAMB.LE.IMAXAMB) GOTO 42
      WRITE(108,104)
  104 FORMAT(1H/,40(1H*),$TROP DE MOTS AMBIGUS $,60(1H*))
      GOTO 47
   36 IF(RESULT(2,1).GE.10) GOTO 42
      IXMOT=IXMOT+1
      IF(IXMOT.GT.IX) GOTO 37
      TAB(1,IXMOT)=RESULT(1,1)
      TAB(2,IXMOT)=0
   42 IF(MSUIVMOT.GT.IM)  GOTO 13
      M=MDEBMOT=MSUIVMOT
      GOTO 30
   37 WRITE(108,112)
  112 FORMAT(1HO,$******* TROP DE MOTS CARACTERISTIQUES,RECOMMENCEZ$)
      INPHRASE=INPHRASE-1
      GOTO 47
   13 CONTINUE
      INAMB=INAMB-1
C.....
C
C                              FIN DE RECH-MOT
C
C.....
C     ON A FINI L'ANALYSE DE LA PHRASE
      DO 43 J=0,IM
      ADRCHAIN(J)=0
   43 MOT(J)=1R
C     IXMOT=NB. DE MOTS TROUVES
C     IZMOT=NB. DE MOTS NON TROUVES+1
C     IMPRESSION DES MOTS TROUVES.
C     LE TABLEAU TAB CONTIENT LES N. DES MOTS NON AMBIGUS
      IF(IZMOT.EQ.1) GOTO 28
      IZMOT=IZMOT-1
      WRITE(108,106)
  106 FORMAT(1HO,$******************  MOTS NON TROUVES$)
      DO 31 J=1,IZMOT
   31 WRITE(108,103) (NONTROUVE(J,M),M=1,20)
  103 FORMAT(1HO,10X,20R1)
   28 IF(IXMOT.NE.0) GOTO 46
      WRITE(108,107)
  107 FORMAT(1HO,$--------------AUCUN MOT CARACTERISTIQUE TROUVE-----$)
      INTAB=INTAB-1
      GOTO 1001
   46 CONTINUE
C
C  ------------FIN DE L'ANALYSE DE LA PHRASE
C
C.....
C
C                    TRI DU TABLEAU TAB
C
      WRITE(108,111) (TAB(I,J),I=1,2),J=1,IXMOT
C.....
```

```
392:        IF(IXMOT.EQ.1) GOTO 50
393:        DO 51 I=2,IXMOT
394:        DO 52 J=1,I-1
395:        IF(TAB(1,I)-TAB(1,J)) 57,58,52
396:     57 IGS=J
397:        GOTO 53
398:     58 IF(TAB(2,I)-TAB(2,J))  52,55,52
399:     52 CONTINUE
400:        GOTO 51
401:     53 IG1=TAB(1,I)
402:        IG2=TAB(2,I)
403:        DO 54 IGX=I-1,IGS,-1
404:        TAB(2,IGX+1)=TAB(2,IGX)
405:     54 TAB(1,IGX+1)=TAB(1,IGX)
406:        TAB(1,IGS)=IG1
407:        TAB(2,IGS)=IG2
408:        GOTO 51
409:     55 IXMOT=IXMOT-1
410:        DO 56 J=I,IXMOT
411:        TAB(2,J)=TAB(2,J+1)
412:     56 TAB(1,J)=TAB(1,J+1)
413:        I=I-1
414:     51 CONTINUE
415:     50 CONTINUE
416:        WRITE(108,110)
417:    110 FORMAT(1HO,$***SORTIE DE TAB TRIEE$)
418:        WRITE(108,111) (TAB(I,J),I=1,2),J=1,IXMOT
419:    111 FORMAT(1HO,I10,I10)
420: C*****************************************************************
421: C
422: C         ET (PHRASE )
423: C
424: C*****************************************************************
425:        TIT(INTAB,0)=IP
426:        TIT(INTAB,1)=INTAB
427:        TIT(INTAB,2)=4HPHRA
428:        TIT(INTAB,3)=4HSE N
429:        TIT(INTAB,4)=INPHRASE
430:
431:        T(1,INTAB)=0
432:
433:        J=1
434:     72 JX=TAB(1,J)
435:        IF(TAB(2,J).EQ.0) JY=INTAB ; GOTO 76
436: C  CAS AMBIGU
437:        ASSIGN 77 TO IT1
438:        GOTO 80
439:     76 CALL RECHARTICLE
440:        L=T(1,JY)
441:        IF(IXMOT.EQ.1) GOTO 70
442:        IF(TROUVE) GOTO 71
443:     77 J=J+1
444:        IF(J.GT.IXMOT) GOTO 81
445:        GOTO 72
446:     71 J=J+1
447:     69 CONTINUE
```

```
448:        JX=TAB(1,J)
449:        IF(TAB(2,J).EQ.0) GOTO 78
450: C   TRAITEMENT DES MOTS AMBIGUS
451:        ASSIGN 73 TO IT1
452:     80 J1=IMAXTAB+TAB(2,J)
453:        IF(T(1,J1).EQ.0) GOTO 79
454: C   CE N'EST PAS LE PREMIER TERME DU MOT AMBIGU
455:        JY=IMAXTAB
456:        CALL RECHARTICLE
457:        IF(.NOT.TROUVE) GOTO IT1
458:        J2=IMAXTAB
459:        CALL INTERCLASS
460:        GOTO IT1
461:     79 JY=J1
462:        CALL RECHARTICLE
463:        GOTO IT1
464:     78 CONTINUE
465: C   CAS D'UN MOT NON AMBIGU
466:        JY=IMAXTAB
467:        CALL RECHARTICLE
468:        IF(.NOT.TROUVE) GOTO 73
469:        J1=IMAXTAB
470:        J2=J3=INTAB
471:        CALL ET
472:        IF(L.EQ.0) GOTO 83
473:     73 J=J+1
474:        IF(J.LE.IXMOT) GOTO 69
475: C   TRAITEMENT DE FIN : ET(MOTS AMBIGUS)
476:     81 IF(INAMB.EQ.0) GOTO 70
477:        IF(T(1,INTAB).NE.0) JJ=1 ; GOTO 84
478:        J2=IMAXTAB+1
479:        L=T(1,J2)
480:        DO 85 J=1,L+1
481:     85 T(J,INTAB)=T(J,J2)
482:        JJ=2
483:        IF(JJ.GT.INAMB) GOTO 86
484:     84 DO 82 J=JJ,INAMB
485:        J1=J3=INTAB
486:        J2=IMAXTAB+J
487:        CALL ET
488:        IF(L.EQ.0) GOTO 83
489:     82 CONTINUE
490:     86 DO 75 J=1,INAMB
491:     75 T(1,IMAXTAB+J)=0
492:     70 WRITE(108,131) INTAB,L
493:    131 FORMAT(1HO,$COLONNE :$,I2,5X,I8,$  SERIES REPONDENT A LA QUESTION
494:       -$)
495:        GOTO 74
496:     83 WRITE(108,130)
497:    130 FORMAT(1HO,40(1H*),,$AUCUNE SERIE NE REPOND A LA QUESTION$)
498:        WRITE(108,132)
499:    132 FORMAT(1HO,$SUPPRESSION DE LA COLONNE$)
500:        INTAB=INTAB-1
501:     74 REWIND 2
502:        GOTO 1001
503: C*****************************************************************
```

```
504:  C
505:  C         OPERATION ET.
506:  C
507:  C*************************************************************************
508:   1003 IF(J1.EQ.0) GOTO 1021
509:        IF(J2.EQ.0) GOTO 1021
510:        IF(J3.NE.0) GOTO 330
511:        J3=INTAB=INTAB+1
512:        IF(INTAB.GT.IMAXTAB) GOTO 1015
513:    330 CONTINUE
514:        CALL TITCOLONNE
515:        CALL ET
516:        IF(L.NE.0) GOTO 331
517:        WRITE(108,332) J3
518:    332 FORMAT(1HO,$COLONNE$,I2,$     AUCUNE SERIE NE REPOND A L'OPERATION
519:       -)
520:        GOTO 1001
521:    331 WRITE(108,333) J3,L
522:    333 FORMAT(1HO,$COLONNE$,I2,$ :$,I4,$     SERIES REPONDENT A L'OPERATI
523:       -N$)
524:        GOTO 1001
525:  C*************************************************************************
526:  C
527:  C         OPERATION OU.
528:  C
529:  C*************************************************************************
530:   1004 IF(J1.EQ.0) GOTO 1021
531:        IF(J2.EQ.0) GOTO 1021
532:        IF(J3.EQ.0) GOTO 340
533:        IF(J2.EQ.J3) GOTO 343
534:        IF(J1.EQ.J3) GOTO 341
535:        GOTO 342
536:    340 J3=INTAB=INTAB+1
537:        IF(INTAB.GT.IMAXTAB) GOTO 1015
538:    342 CONTINUE
539:        CALL TITCOLONNE
540:        CALL OU
541:        WRITE(108,333) J3,L
542:        GOTO 1001
543:    343 J2=J1
544:        J1=J3
545:    341 CALL INTERCLASS
546:        CALL TITCOLONNE
547:        WRITE(108,333) J3,L1
548:        GOTO 1001
549:  C*************************************************************************
550:  C
551:  C         OPERATION TC: TITRE DE COLONNE.
552:  C
553:  C*************************************************************************
554:   1017 IF(J1.EQ.0) J1=INTAB
555:        IF(J2.EQ.0) J2=J1
556:        DO 600  J=J1,J2
557:    600 WRITE(108,601) J,(TIT(J,JX),JX=0,4)
558:    601 FORMAT(1HO,$TITRE DE LA TRANSACTION,COLONNE :$,I3,5X,$OPERATION N
559:       -$,I3,5X,I2,$ = $,2R4,$,$,I3)
```

```
560:        GOTO 1001
561: C*************************************************************************
562: C
563: C          OPERATION SA : J3=J1 SAUF J2
564: C
565: C*************************************************************************
566:  1009 IF(J1.EQ.0) GOTO 1021
567:        IF(J2.EQ.0) GOTO 1021
568:        IF(J3.EQ.0) GOTO 440
569: C  J3=N. DE LA COLONNE RESULTANT
570:        IF(J3.EQ.J2) GOTO 441
571:        GOTO 442
572:   441 WRITE(108,443)
573:   443 FORMAT(1HO,$DANS L'OPERATION 'SA' ON DOIT AVOIR J2#J3$)
574:        GOTO 1001
575:   440 J3=INTAB=INTAB+1
576:        IF(INTAB.GT.IMAXTAB) GOTO 1015
577:   442 CONTINUE
578:        CALL TITCOLONNE
579:        L1=T(1,J1)
580:        L2=T(1,J2)
581:        L=0
582:        IF(L1.EQ.0) GOTO 450
583:        I1=I2=2
584:        IF(L2.EQ.0) M=9999999 ; GOTO 448
585:   446 M=T(I2,J2)
586:        IF(T(I1,J1)-M)  444,448,445
587:   444 L=L+1
588:        T(L+1,J3)=T(I1,J1)
589:   447 I1=I1+1
590:        IF(I1.LE.L1+1)  GOTO 446
591:        GOTO 450
592:   448 I2=I2+1
593:        IF(I2.LE.L2+1) GOTO 447
594:        I1=I1+1
595:        GOTO 451
596:   445 I2=I2+1
597:        IF(I2.LE.L2+1) GOTO 446
598:   451 CONTINUE
599:        IF(I1.GT.L1+1) GOTO 450
600:   452 CONTINUE
601:        L=L+1
602:        T(L+1,J3)=T(I1,J1)
603:        I1=I1+1
604:        IF(I1.LE.L1+1) GOTO 452
605:   450 T(1,J3)=L
606:        WRITE(108,333) J3,L
607:        GOTO 1001
608: C*************************************************************************
609: C
610: C          OPERATION TI=RECHERCHE DES TITRES.
611: C
612: C*************************************************************************
613:  1005 IF(J1.NE.0) GOTO 370
614:        J1=INTAB
615:   370 L=T(1,J1)
```

```
      WRITE(108,360) J1
  360 FORMAT(1HO,$*** TITRES DES SERIES   COLONNE:  $,I2)
      WRITE(108,376) (TIT(J1,J),J=0,4)
  376 FORMAT(1HO,$* OPERATION N:$,I3,5X,I2,$ = $,2R4,$,$,I3)
      IF(L.EQ.0) GOTO 369
C     RECHERCHE DES TITRES DANS LE FICHIER DES TITRES.
      DO 368 J=2,L+1
       JX=T(J,J1)
  365 CALL BUFFER IN(3,1,MOT,IM1+1,ITEST)
  361 GOTO (361,362,363,364) ITEST
  364 WRITE(108,371)
  371 FORMAT(1H1,$ ERREUR-TRANSMISSION$)
  363 WRITE(108,372)
  372 FORMAT(1H1,$  FIN-DE-FICHIER$)
      GOTO 378
  362 CONTINUE
      IF(MOT(0)-JX) 365,366,367
  367 WRITE(108,373) JX
  373 FORMAT(1H1,$ERREUR : TITRE INEXISTANT DANS LE FICHIER DES TITRES
     -$,I10)
      BACKSPACE 3
      GOTO 368
  366 WRITE(108,374) (MOT(I),I=0,IM1)
  374 FORMAT(1HO,$** SERIE N :  $,I10,4X,80R1,/,20X,80R1)
  368 CONTINUE
  378 DO 379 I=1,IM1
  379 MOT(I)=0
      REWIND 3
      GOTO 1001
  369 WRITE(108,375)
  375 FORMAT(1HO,$   AUCUNE SERIE DANS LA COLONNE$)
      GOTO 1001
C*****************************************************************
C
C        OPERATION NS=SORTIR LES N. DES SERIES D'UNE COLONNE
C
C*****************************************************************
 1008 IF(J1.NE.0) GOTO 394
      J1=INTAB
  394 CONTINUE
      WRITE(108,390) J1
  390 FORMAT(1HO,$***NUMEROS DES SERIES    COLONNE:$,I3)
      WRITE(108,376) (TIT(J1,J),J=0,4)
      L=T(1,J1)
      WRITE(108,391) L
  391 FORMAT(1HO,I4,$  SERIES :$)
      IF(L.EQ.0) GOTO 1001
      DO 393 J=2,L+1
  393 WRITE(108,392) J-1,T(J,J1)
  392 FORMAT(1H/,I4,$  :$,I10)
      GOTO 1001
C*****************************************************************
C
C        OPERATION SU=SUPPRESSION DE LA TRANSACTION PRECEDENTE.
C
C*****************************************************************
```

```
 1006 WRITE(108,350) INTAB
  350 FORMAT(1HO,$SUPPRESSION DE LA COLONNE  $,I2)
      T(1,INTAB)=0
      TIT(INTAB,0)=TIT(INTAB,1)=TIT(INTAB,4)=0
      TIT(INTAB,2)=TIT(INTAB,3)=4H
      INTAB=INTAB-1
      GOTO 1001
C*****************************************************************
C
C        OPERATION TR : TRANSFERT DE LA COLONNE J1 DANS LA COLONNE J2
C
C*****************************************************************
 1013 IF(J1.EQ.0) GOTO 420
      IF(J2.EQ.0) GOTO 420
      WRITE(108,421) J1,J2
  421 FORMAT(1HO,$TRANSFERT DE LA COLONNE  $,I2,$DANS LA COLONNE  $,I2)
C     TRANSFERT DU TITRE DE LA COLONNE.
      DO 424 J=0,4
  424 TIT(J2,J)=TIT(J1,J)
      JY=T(1,J1)+1
      DO 422 JX=0,JY
  422 T(JX,J2)=T(JX,J1)
      GOTO 1001
  420 WRITE(108,423)
  423 FORMAT(1HO,$LES PARAMETRES DE TR DOIVENT ETRE NON NULS$)
      GOTO 1001
C*****************************************************************
C
C        OPERATION NC=NUMERO DE LA DERNIERE COLONNE ATTRIBUE AUTOMATIQUEMENT
C
C*****************************************************************
 1020 WRITE(108,610) INTAB
  610  FORMAT(1HO,$*** DERNIERE COLONNE ATTRIBUEE :$,I2)
      GOTO 1001
C*****************************************************************
C
C        OPERATION ST=STOP
C
C*****************************************************************
 1007 IF(J1.EQ.0) GOTO 1000
C  ON ARRETE LE PROGRAMME PAR L'OPERATION: 'ST'0,
      INTAB=J1-1
      WRITE(108,1012) J1
 1012 FORMAT(1HO,30(1H*),$LA TRANSACTION RECOMMENCE A LA COLONNE :$,I2)
      IF(J1.EQ.1) INPHRASE=0
      DO 1018 J=J1,IMAXTAB
      T(1,J)=0
      TIT(J,0)=TIT(J,1)=TIT(J,4)=0
 1018 TIT(J,2)=TIT(J,3)=4H
      GOTO 1001
 1000 WRITE(108,1014)
 1014 FORMAT(1H1,40(1H*),$FIN DE LA TRANSACTION$,20(1H*))
      STOP
C
C
C
```

```
C*****************************************************************
C
C
      SUBROUTINE RECHARTICLE
C.....................................................................
C  JX=N. DE L'ARTICLE A RECHERCHER.
C  JY=N. DE LA COLONNE DE T OU ON RANGE L'ARTICLE TROUVE.
C  LTAMP=LONG. DES ARTICLES DU FICHIER INVERSE.
      TROUVE=.TRUE.
   65 CALL BUFFER IN(2,1,T(0,JY),LTAMP,ITEST)
   61 GOTO (61,62,63,64) ITEST
   64 WRITE(108,120)
  120 FORMAT (1H1,$ ERREUR-TRANSMISSION$)
   63 WRITE(108,121)
  121 FORMAT(1H1,$FIN DE FICHIER$)
      TROUVE=.FALSE.
      RETURN
   62 IF(T(0,JY)-JX) 65,66,67
   67 WRITE(108,122) JX
  122 FORMAT(1H1,$ERREUR : MOT INEXISTANT DANS LE FICHIER INVERSE$,I10)
      TROUVE=.FALSE.
   66 CONTINUE
      BACKSPACE 2
      RETURN
C
C
C
C
C
      SUBROUTINE ET
C.....................................................................
C  J1,J2=N. DES COLONNES DE ET.
C  J3=N. DE LA COLONNE RESULTANT.
      L1=T(1,J1)
      L2=T(1,J2)
      L=0
      IF(L1.EQ.0) GOTO 304
      IF(L2.EQ.0) GOTO 304
      I1=I2=2
  303 IF(T(I1,J1)-T(I2,J2))  300,301,302
  300 I1=I1+1
      IF(I1.LE.L1+1)  GOTO 303
      GOTO 304
  301 L=L+1
      T(L+1,J3)=T(I1,J1)
      I1=I1+1
      IF(I1.GT.L1+1)  GOTO 304
      I2=I2+1
      IF(I2.GT.L2+1)  GOTO 304
      GOTO 303
  302 I2=I2+1
      IF(I2.LE.L2+1)  GOTO 303
  304 T(1,J3)=L
      RETURN
C
C
```

```
C
C
C
      SUBROUTINE OU
C.....................................................................
C  J1,J2=N. DES COLONNES DE OU
C  J3=N. DE LA COLONNE RESULTANT.
      L1=T(1,J1)
      L2=T(1,J2)
      I1=I2=2
      L=0
      IF(L1.EQ.0) GOTO 405
      IF(L2.EQ.0) GOTO 407
  403 IF(T(I1,J1)-T(I2,J2)) 400,401,402
  400 L=L+1
      T(L+1,J3)=T(I1,J1)
  406 I1=I1+1
      IF(I1.LE.L1+1) GOTO 403
C  FIN-1
  405 IF(I2.GT.L2+1) GOTO 404
      L=L+1
      T(L+1,J3)=T(I2,J2)
      I2=I2+1
      GOTO 405
  401 L=L+1
      T(L+1,J3)=T(I1,J1)
      I2=I2+1
      IF(I2.GT.L2+1)  GOTO 407
      GOTO 406
  402 L=L+1
      T(L+1,J3)=T(I2,J2)
      I2=I2+1
      IF(I2.LE.L2+1) GOTO 403
C  FIN-2
  407 IF(I1.GT.L1+1) GOTO 404
      L=L+1
      T(L+1,J3)=T(I1,J1)
      I1=I1+1
      GOTO 407
  404 T(1,J3)=L
      RETURN
C
C
C
C
C
C
      SUBROUTINE INTERCLASS
C.....................................................................
C  J1,J2= N. DES COLONNES
C  J1= N. DE LA COLONNE RESULTAT
      L2=T(1,J2)
      IF(L2.EQ.0) GOTO 507
      L1=T(1,J1)
      IF(L1.EQ.0) GOTO 503
      I1=I2=2
```

```
840:    503 IF(T(I1,J1)-T(I2,J2)) 500,501,502
841:    500 I1=I1+1
842:        IF(I1.LE.L1+1)  GOTO 503
843:  C FIN=1:
844:    505 IF(I2.GT.L2+1)  GOTO 504
845:        L1=L1+1
846:        T(I1,J1)=T(I2,J2)
847:        I1=I1+1
848:        I2=I2+1
849:        GOTO 505
850:    501 I2=I2+1
851:        IF(I2.GT.L2+1) GOTO 504
852:        GOTO 500
853:    502 L1=L1+1
854:  C DECALAGE DE T1 DE I1 A L1
855:        DO 506 L=L1,I1,-1
856:    506  T(L+1,J1)=T(L,J1)
857:        T(I1,J1)=T(I2,J2)
858:        I1=I1+1
859:        I2=I2+1
860:        IF(I2.LE.L2+1) GOTO 503
861:  C FIN=2:
862:    504 T(1,J1)=L1
863:    507 CONTINUE
864:        RETURN
865:    508 DO 509 I1=1,L2+1
866:    509 T(I1,J1)=T(I1,J2)
867:        RETURN
868:  C
869:  C
870:  C
871:  C
872:  C
873:  C
874:  C
875:        SUBROUTINE TITCOLONNE
876:  C.....
877:        TIT(J3,0)=IP
878:        TIT(J3,1)=J3
879:        TIT(J3,2)=IOP
880:        TIT(J3,4)=J2
881:        ENCODE(4,520,J) J1
882:    520 FORMAT(2H  ,I)
883:        TIT(J3,3)=J
884:        RETURN
885:
886:  C.....
887:
888:  C.....
889:
890:
891:        END
```

REFERENCES

[1] *Documentation automatique sur une banque de statistiques nationales - Système d'accumulation de données.*
Laboratoire de Mathématiques Appliquées de l'Université de Grenoble - 1967

[2] G. SALTON *Automatic Information Organization and Retrieval.*

[3] G. SALTON. M.E. LES *Information Analysis and Dictionary Construction. ISR 11.*

[4] P.C.LEECH - R.C. MATLACK *Information Retrieval : Dictionary Representations and Cluster Evaluation. ISR 12.*

[5] G. MARZIN - J. VINCENT-CARREFOUR - J.C. MONESTIEZ *Le système de documentation automatique du CNET.*

[6] *Etude sur l'expérience américaine des systèmes intégrés et réseaux informatiques.*
Worldwide Information System Inc. Los-Angeles

[7] T. GIRARD *Conception et expérimentation d'une banque de données économiques.*
Thèse 3° cycle IEEA. Avril 1971.

[8] *Statistiques et indicateurs des régions françaises.*
DATAR - INSEE

[9] KEEN *An analysis of the documentation requests.*
ISR 13

[10] J.J. ROCCHIO *Document retrieval system. Optimization and evaluation.*
ISR 10

[11] *Software des banques de données.*
Compte-rendu du colloque AFCET - IRIA. Aix en Provence, Juin 197(

[12] *L'Information Scientifique, l'Informatique et la Documentation Automatique.*
Colloque CNRS - IRIA. Novembre 1968.