Etude et réalisation d'une méthode de transport : traduction de programmes PL 360 en LP 80

Claudine Chassagne

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR

par

Claudine CHASSAGNE

ETUDE ET REALISATION D'UNE METHODE DE TRANSPORT: TRADUCTION DE PROGRAMMES PL360 EN LP80.

Soutenue le 26 Janvier 1979 devant la commission d'examen.

L. BOLLIET Président

M. BENNETT
J. COURTIN
G. VEILLON
} Examinateurs

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1977-1978

Président : M. Philippe TRAYNARD

Vice-présidents : M. René PAUTHENET

M. Georges LESPINARD

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Electronique - automatique
	BESSON Jean	Chimie minérale
	BLOCH Daniel	Physique du solide - cristallographie
	BONNETAIN Lucien	Génie chimique
	BONNIER Etienne	Métallurgie
*	BOUDOURIS Georges	Electronique - automatique
	BRISSONNEAU Pierre	Physique du solide - cristallographie
	BUYLE-BODIN Maurice	Electronique - automatique
	COUMES André	Electronique - automatique
	DURAND Francis	Métallurgie
	FELICI Noël	Electronique - automatique
	FOULARD Claude	Electronique - automatique
	LANCIA Roland	Electronique - automatique
	LONGEQUEUE Jean-Pierre	Physique nucléaire corpusculaire
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie - physique
	PAUTHENET René	Electronique - automatique
	PERRET René	Electronique - automatique
	POLOUJADOFF Michel	Electronique - automatique
	TRAYNARD Philippe	Chimie - physique
	VEILLON Gérard	Informatique fondamentale et appliquée

* en congé pour études

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuël	Electronique - automatique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electronique - automatique
	GUYOT Pierre	Métallurgie physique
	LACOUME Jean-Louis	Electronique - automatique
	JOUBERT Jean-Claude	Physique du solide - cristallographie

.../...

MM.	ROBERT André	Chimie appliquée et des matériaux
	ROBERT François	Analyse numérique
	ZADWORNY François	Electronique - automatique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	CHARTIER Germain	Electronique - automatique
	CHIAVERINA Jean	Biologie, biochimie, agronomie
	IVANES Marcel	Electronique - automatique
	LESIEUR Marcel	Mécanique
	MORET Roger	Physique nucléaire - corpusculaire
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie Physique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahim	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)
E.N.S.E.E.G.

MM.	BISCONDI Michel	Ecole des Mines St. Etienne (dépt. Métallurgie)
	BOOS Jean-Yves	Ecole des Mines St. Etienne (Métallurgie)
	DRIVER Julian	Ecole des Mines St. Etienne (Métallurgie)

.../...

MM.	KOBYLANSKI André	Ecole des Mines St. Etienne (Métallurgie)
	LE COZE Jean	Ecole des Mines St. Etienne (Métallurgie)
	LESBATS Pierre	Ecole des Mines St. Etienne (Métallurgie)
	LEVY Jacques	Ecole des Mines St. Etienne (Métallurgie)
	RIEU Jean	Ecole des Mines St. Etienne (Métallurgie)
	SAINFORT	C.E.N. Grenoble (Métallurgie)
	SOUQUET	U.S.M.G.
	CAILLET Marcel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	COULON Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	GUILHOT Bernard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LALAUZE René	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	LANCELOT Francis	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SARRAZIN Pierre	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	SOUSTELLE Michel	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THEVENOT François	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	THOMAS Gérard	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TOUZAIN Philippe	Ecole des Mines St. Etienne (Chim. Min. Ph.)
	TRAN MINH Canh	Ecole des Mines St. Etienne (Chim. Min. Ph.)

E.N.S.E.R.G.

MM.	BOREL	Centre d'études nucléaires de Grenoble
	KAMARINOS	Centre national recherche scientifique

E.N.S.E.G.P.

M.	BORNARD	Centre national recherche scientifique
Mme	CHERUY	Centre national recherche scientifique
MM.	DAVID	Centre national recherche scientifique
	DESCHIZEAUX	Centre national recherche scientifique

Je tiens à remercier,

Monsieur Louis BOLLIET, Professeur à l'Université des Sciences Sociales de Grenoble qui m'a fait l'honneur de présider le jury de cette thèse,

Monsieur Gérard VEILLON, Professeur à l'Institut National Polytechnique de Grenoble qui m'a donné les moyens de mener à bien cette recherche,

Monsieur Jacques COURTIN, Maître de conférences à l'Université des Sciences Sociales de Grenoble qui a encouragé ces travaux. Je veux lui exprimer ma profonde gratitude pour la confiance qu'il m'a toujours témoignée et pour ses nombreux conseils et encouragements,

Monsieur Malcom BENNETT, Ingénieur de recherches à la Compagnie CII-HB qui a accepté de juger cette thèse et de participer au jury.

Ce travail a aussi bénéficié dans les périodes où les erreurs de programme faisaient mon désespoir, de l'aide de mes collègues de laboratoire et en particulier de celle de Danielle DUJARDIN.

Je voudrais aussi remercier Madame H. DIAZ qui a assuré la dactylographie de cette thèse ainsi que le Service de reproduction qui en a réalisé le tirage.

S O M M A I R E

INTRODUCTION

PREMIERE PARTIE

PRESENTATION DU TRADUCTEUR PIAFTRAD

I - LE SYSTEME PIAF

II - ADAPTATION DU SYSTEME PIAF A LA TRADUCTION

DEUXIEME PARTIE

DESCRIPTION DES PROBLEMES DE TRADUCTION

TROISIEME PARTIE

EVALUATION

INTRODUCTION

La diversification croissante des ordinateurs, souvent incompatibles entre eux, a donné naissance à des études [POWA],[POOL], sur la transportabilité et l'adaptabilité des programmes et plus précisément sur le transport de logiciels. Considérons l'approche naturelle utilisée pour créer un programme. Après avoir examiné le problème, on détermine un ensemble appnoprié d'opérations de base et de types de données, puis on construit un algorithme qui utilise les opérations définies pour manipuler les données. L'algorithme ne donne aucune indication sur la représentation des types de données, ni sur l'obtention des résultats. En d'autres termes l'algorithme est indépendant de toute réalisation particulière des opérations et des types de données.

Un algorithme peut donc être réalisé sur n'importe quel calculateur par création de ses opérations et types de données.Pour transporter un algorithme sur une nouvelle machine, il faut donc:

- définir une représentation des opérations de base et des types de données pour le problème,
- exprimer l'algorithme dans les termes de sa représentation . Les techniques de transport permettent de supprimer totalement cette deuxième étape

I - LES PRINCIPALES METHODES DE TRANSPORT

1/ Transport par codage du programme dans des langages évolués disponibles

sur plusieurs machines, par exemple les langages FORTRAN, ALGOL, COBOL ou PASCAL dont l'aspect important est la portabilité elevée des compilateurs.

Cette méthode traditionnelle impose plusieurs conditions:

- les opérations de base et les types de données exigés par le problème doivent être disponibles dans le langage choisi.
- le langage choisi doit avoir une définition standard qui doit être largement implantée.
- il faut éviter des constructions qui ne sont acceptées que localement et qui sont interdites par la définition standard.

Exemple: Considérons les deux langages FORTRAN définis respectivement sur IBM 360 [FOR1] et sur IRIS 50 [FOR2].
Le compilateur FORTRAN développé sur IBM 360 n'accepte pas les identificateurs de plus de six caractères, tandis que le compilateur FORTRAN développé sur IRIS 50 accepte les identificateurs d'une longueur quelconque et tronque éventuellement à huit caractères, ceux qui possèdent une longueur supérieure. Il faut éviter dans ce cas, d'utiliser des identificateurs de longueur supérieure à six caractères, ce qui n'est toléré que localement à l'IRIS 50.

Ces conditions sont satisfaites par une grande majorité des programmes qui résolvent des problèmes scientifiques ou des programmes standard de gestion.

La première condition est la plus difficile à satisfaire: En effet, une grande partie des programmes (parmi lesquels, les programmes de traitement de textes) utilisent des opérations de base et des types de données qui ne sont pas disponibles dans le langage choisi, mais qui peuvent être néanmoins réalisés par combinaison des éléments existant dans le langage. Ceci rend le programme résultant inefficace.

Bien que le langage ne fournisse pas un type particulier de données, il se peut, dans certains cas, que cette donnée soit implémentée sur le calculateur.
Par exemple, ANSI FORTRAN [ANSI], ne possède ni le type chaîne, ni les opérations relatives à ce type. Cependant, le calculateur IBM360 [I360] fournit ces facilités. L'extension du langage ANSI FORTRAN par inclusion du type "chaîne" permet d'améliorer l'efficacité des programmes dans ce cas.

Donc les extensions définies en termes de la machine cible, réduisent la transportabilité des programmes, tandis que lorsqu'elles sont définies par la combinaison des éléments disponibles dans le langage, elles conservent la transportabilité au détriment de l'efficacité des programmes.

2/ Transport par utilisation d'un modèle de machine abstraite.

Le principe de cette étude est le suivant:

Les opérations de base et les types de données sont utilisés pour définir un calculateur fictif et l'algorithme est alors codé dans un certain langage défini pour ce calculateur. Celui-ci est appelé "machine abstraite". On dit qu'il "modélise" les exigences du problème. Pour exécuter le programme sur un ordinateur réel, on réalise la machine abstraite sur celui-ci.

Pour transporter un programme par cette méthode, il faut donc définir un modèle de machine abstraite adapté à ce problème, créer un langage approprié et construire un traducteur à la fois transportable et adaptable.

Parmi les machines abstraites qui ont été étudiées, on peut citer FLUB [POWA] décrite spécifiquement pour l'implémentation du système transportable STAGE2 [WAIT] ainsi que la machine abstraite TEXED [PØWA] utilisée pour l'implémentation d'un programme de traitement de textes MITEM.

II - PRESENTATION DU PROBLEME

Le choix de l'une ou l'autre méthode dépend donc des programmes considérés. Dans notre cas, le problème est différent puisque les programmes à transporter sont situés dans un cadre précis. Nous devions réaliser le transport sur IRIS 80, des programmes constituant le logiciel PIAFDOC [COU1,GRAN] opérationnel sur IBM360. Or ces programmes sont écrits dans le langage PL360 [WISU] et sont des programmes utilisant essentiellement des instructions de manipulation de chaînes de caractères. Choisir la méthode de codage dans un langage évolué réduit donc considérablement l'efficacité des algorithmes ainsi transportés. D'autre part, le contexte particulier offert par les deux machines IBM360 et IRIS80 nous a permis d'envisager une méthode de traduction qui est un cas particulier du transport par modèle de machine abstraite.

En effet ces deux machines IBM360 et IRIS80

- appartiennent à la même génération d'ordinateurs,
- disposent de langages de même niveau: en particulier IRIS80 dispose du langage de programmation LP80 [PEQU], qui appartient à la même famille de langages que PL360, dans lequel sont écrits les programmes à transporter.

1/Caractéristiques communes des deux machines:

Ce sont les suivantes:

- organisation processeur/registre: elles possèdent toutes les deux des registres arithmétiques multiples (les instructions arithmétiques peuvent prendre leurs opérandes en mémoire ou dans des registres) ;
- organisation-mémoire: l'espace d'adressage est linéaire. Cette caractéristique est importante en ce qui concerne la limitation de la taille-mémoire. Les deux machines ont toutes les deux des mots de 32 bits.
- mécanisme d'adressage: (par modification d'index) - L'adresse effective est obtenue après addition du déplacement à la base, adressage indirect et indexation.

2/Caractéristiques communes des langages PL360 et LP80

Ces deux langages de programmation (appelés langages de type dans la suite) possèdent les caractéristiques suivantes:

- ils conservent les possibilités d'un assembleur, à savoir:
  - . accès à tous les registres du calculateur,
  - . accès à tous les niveaux d'information,
  - . production d'un code machine optimisé.

- ils possèdent des notions indépendantes des calculateurs, liées aux traitements qu'ils ont à exprimer:
  - . un ensemble complet de types de données,
  - . des instructions évoluées puissantes (boucles, branchements entrées-sorties, etc...),
  - . une structure de programme adaptée à la structure d'un traitement (procédures),
  - . une organisation des données très souples et en accord avec l'organisation des traitements qui permet notamment:
    - l'accès à des variables déclarées communes à plusieurs traitements,
    - la définition des variables d'intérêt local pour un traitement.
  - . la possibilité de faire des déclarations de synonymie.
- ils atténuent les risques d'erreur en fournissant à la compilation un contrôle serré de la cohérence du programme.

Le choix du langage LP80 pour transporter le logiciel PIAFDOC sur IRIS80 possède le double avantage de conserver les notions d'efficacité et de clarté et de posséder une syntaxe voisine à celle du langage PL360, ce qui permet de réduire la complexité du traducteur.

Les différences existant entre les deux langages PL360 et LP80 sont dues aux différences liées aux deux machines, en particulier à l'adressage et aux primitives.

3/ D'après ces considérations, la méthode utilisée consiste à construire un traducteur permettant de traduire en LP80 des programmes écrits en PL360. La machine abstraite utilisée est une machine possédant les caractéristiques communes des deux ordinateurs IBM360 et IRIS80. On a défini pour cette machine abstraite un langage pivot de type "PL", intermédiaire entre les deux langages PL360 et LP80.

Le traducteur est construit à l'aide du transducteur général d'états finis utilisé dans le système PIAF [COU1] (système de traitement des langues naturelles).

En fait le système PIAF est composé de deux modules principaux: "un transducteur général d'états finis" et "un système d'analyse syntaxique". Ce dernier module qui a pour fonction de construire des structures arborescentes, n'est pas utilisé par le traducteur, puisque la syntaxe des deux langages PL360 et LP80 est voisine.

Dans une première partie, nous exposons le traducteur PIAFTRAD ainsi défini. Le premier chapitre de cette partie (chapitre I) est consacré à la description du système PIAF qui a permis de réaliser le traducteur. Dans le chapitre II, nous décrivons les modifications nécessaires pour obtenir le traducteur tandis que dans le chapitre III, nous abordons le fonctionnement proprement dit de celui-ci.

La seconde partie traite des problèmes posés par la traduction, qui se décomposent en problèmes liés au langage. La troisième partie est consacrée à l'analyse des mesures sur les performances du traducteur ainsi qu'à une appréciation du travail présenté. Enfin, nous conclurons sur les perspectives offertes par cette méthode de transport.

PREMIERE PARTIE

PRESENTATION DU TRADUCTEUR

I - LE SYSTEME PIAF

Le système PIAF (Programme Interactif d'Analyse du Français) est un système de traitement assisté des langues naturelles dont le but est de construire des structures interprétables. Il est composé de deux modules principaux, un transducteur général d'états finis conçu pour l'analyse morphologique et un système d'analyse syntaxique. Comme nous l'avons vu précédemment, seul le transducteur général d'états finis est utilisé pour construire le traducteur, les langages source et cible faisant partie de la même famille des langages de programmation (la structure des programmes écrits dans les deux langages étant identique : voir Deuxième partie). C'est pourquoi nous ne décrivons pas l'analyseur syntaxique.

Nous exposons d'abord les caractéristiques principales du système, puis décrivons ensuite le transducteur général d'états finis qui a été utilisé par le système de traduction. Ces définitions ainsi que les aspects théoriques ont été extraits de la thèse de Jacques COURTIN [COU1].

1 - Caractéristiques principales du système PIAF

L'objectif du système PIAF est de définir des outils informatiques puissants et simples pour aider au maximum l'utilisateur à maîtriser la complexité des paramètres linguistiques.

Dans cette perspective, les caractéristiques essentielles du système sont les suivantes:

a/ c'est un système composé de deux modules informatiques indépendants, chacun d'eux étant composé de sous-modules dont l'enchaînement peut être séquentiel ou parallèle (c'est-à-dire que le module suivant est activé dès qu'un sous-résultat est déterminé par le module précédent).

b/ c'est un système interactif qui permet un dialogue entre l'utilisateur et la machine: deux éditeurs, un éditeur lexicographique pour l'utilisation du transducteur général d'états finis et un éditeur général pour l'utilisation du système complet, permettent à l'utilisateur:

- la création et la modification en mode conversationnel des paramètres linguistiques, règles de grammaire, dictionnaire, liste de paradigmes (modèles) relations de dépendances.

- l'exécution interactive des différents modules d'analyse: c'est-à-dire qu'en cours d'exécution, on peut remettre en cause les paramètres linguistiques et ainsi les modifier.
- la détection et la correction manuelle des erreurs.
- la correction automatique de certaines classes d'erreurs liées à la généralité et à la réversibilité des modèles proposés.

c/ c'est un système réversible: on peut à partir de la même grammaire engendrer tout ou partie des formes d'une même base ou de plusieurs bases. Cette réversibilité est très importante, car:

- elle évite la réécriture d'un modèle génératif à l'aide d'un autre formalisme,
- lors de la mise au point de la grammaire, le linguiste peut vérifier même en cours d'exécution, l'adéquation de l'analyseur morphologique en demandant la génération des modèles qu'il vient de définir.

d/ la conception des modules constituant le système étant très générale, ceux-ci peuvent être utilisés pour l'aide à la traduction d'un langage de programmation dans un autre langage de la même famille et d'autres applications, telles que:

- la génération morphologique,
- la transduction phonétique,
- la correction automatique des erreurs lexicales par un système d'invariants (par exemple invariant phonétique),
- la traduction d'un texte en Braille abrégé.

2 - Présentation du transducteur général d'états finis

Le transducteur général d'états finis a été conçu pour réaliser l'analyse morphologique d'une langue.

Nous présentons d'abord les objectifs de l'analyse morphologique, puis décrivons les paramètres créés pour réaliser ces objectifs.

2.1. Objectifs de l'analyse morphologique

Les objectifs de l'analyse morphologique d'une phrase sont les suivants:

1/ Segmenter une phrase pour obtenir des mots ou des groupes de mots. Pour cela on considère que:

- le blanc n'est pas un séparateur de mots, afin de laisser la possibilité de reconnaître des locutions telles que "au fur et à mesure" comme un seul segment.
- tout segment est insegmentable, ce qui a pour conséquence de segmenter en un nombre de segments uniques.

Chaque mot de la langue analysée peut être découpé en une base précédée ou non de préfixes, suivie ou non de suffixes, et se terminant par une désinence. La détermination d'un segment consiste donc en sa décomposition en préfixe, base, suffixe et désinence.

Pour effectuer cette tâche, on dispose de paramètres linguistiques:

- un dictionnaire de préfixes, de bases, de suffixes et de désinences qui permet, par identification, la décomposition d'une partie contigüe de la chaîne d'entrée.
- une grammaire qui doit infirmer ou confirmer la concaténation de ces éléments pour constituer le segment cherché.

Ces paramètres sont décrits par la suite.

2/ Déterminer un certain nombre de renseignements linguistiques (catégories du mot et variables grammaticales) sur les chaînes fournies par la segmentation, c'est-à-dire effectuer une transduction.

Il s'agit de donner toutes les interprétations de toutes les décompositions d'un segment. Cette transduction est réalisée par la grammaire qui vérifie la segmentation d'un "mot" étant donné que la plupart des éléments constituant ce mot sont porteurs d'information.

2.2. Description des paramètres linguistiques

Comme nous l'avons vu précédemment, les paramètres linguistiques sont le dictionnaire et la grammaire.

2.2.1. Dictionnaire

Les critères retenus pour l'organisation du dictionnaire sont les suivants:

- fréquence des éléments,
- recherche de la plus longue superposition possible entre l'occurrence et un élément du dictionnaire (technique de "Longest Match") et obtention rapide des éléments plus courts,
- le blanc n'est plus un séparateur, mais un caractère comme les autres,
- temps de recherche minimum et volume de mémoire raisonnable.

Avant de décrire l'organisation du dictionnaire, on rappelle quelques définitions.

Définitions

- Soit un ensemble V fini appelé vocabulaire contenant tous les caractères:

$V = \{\_, A, B, C, \ldots, ;, \ldots, 0, \ldots 9\}$

où $\_$ désigne le blanc.

- On appelle chaîne (ou élément) sur V, une suite finie de symboles $x_1\ x_2 \ldots x_n$ avec $x_i \in V$, $\forall i \in [1,n]$

- On désigne par V^*, l'ensemble de toutes les chaînes que l'on peut construire sur V, par V^+ l'ensemble $V^* - \{\Lambda\}$ où Λ est la chaîne vide.

- On appelle dictionnaire (D), tout sous-ensemble de V^+.

- LONGUEUR (α)

C'est une application de V^* dans N (ensemble des entiers naturels)

si $\alpha = \Lambda$ alors LONGUEUR(α) = 0

sinon si $\alpha = a_i\phi$ alors LONGUEUR(α)= 1 + LONGUEUR(ϕ)

$\forall a_i \in V$ et $\forall_\phi \in V^*$

- PREFIXE(α,β) $\alpha, \beta \in V^*$

C'est une application de $V^* \times V^*$ dans V^*

Soient $a,b \in V$, ϕ, λ , $\mu \in V^*$

si $\alpha = a\lambda$ et $\beta = b\lambda$ alors PREFIXE(α,β) = Λ

si $\alpha = \phi\lambda$ et $\beta = \phi\mu$ alors PREFIXE(α,β) = ϕPREFIXE(λ,μ)

- Relation inférieure ("<") entre deux chaînes non vides (ordre alphabétique)

Soient $\alpha, \beta \in V^+$, avec $\alpha \neq \beta$,

a_1 $a_2 \in V$,

$\lambda, \phi, \mu \in V^*$

si $\alpha = \phi a_1 \lambda$ et $\beta = \phi a_2 \mu$ avec $a_1 \neq a_2$

$\alpha < \beta$ si et seulement si $a_1 < a_2$ sinon $\beta < \alpha$

si $\beta = \alpha a_2 \lambda$ alors $\alpha < \beta$

si $\alpha = \beta a_1 \lambda$ alors $\beta < \alpha$

c'est une relation d'ordre strict.

- Relation "C" (contenue) sur D

Soient $\alpha,\beta \in V^+$

α C β si et seulement si

1/ PREFIXE(α,β) = α

2/ LONGUEUR(α) < LONGUEUR(β)

- Relation "< & $\neg$C" (inférieure et non contenue)

Soient $\alpha,\beta \in V^+$

$\alpha <\&\neg C\ \beta$ si et seulement si:

1/ $\alpha < \beta$

2/ PREFIXE(α,β) $\neq \alpha$

- Ensemble des Fils d'un élément β

Soit $\beta \in D$,

FILS(β) = $\{\alpha \in D / \alpha C \beta\}$

- Ensemble des Frères d'un élément β

Soit $\alpha \in D$ tel que $\alpha \notin$ FILS(β) $\forall \beta \in D$

FRERE(α) = $\{\beta \in D / \alpha <\&\neg C\ \beta\}$

- Fils direct

Soient α et $\beta \in D$

α est fils direct de β si et seulement si:

1/ $\alpha \in FILS(\beta)$

2/ Il n'existe pas d'élément $\gamma \in FILS(\beta)$ tel que $\alpha \in FILS(\gamma)$

- Frère direct

Soient α et $\beta \in D$ tel que $\alpha \notin FILS(\gamma)\ \forall\ \gamma \in D$,

β est frère direct de α si et seulement si:

1/ $\beta \in FRERE(\alpha)$

2/ Il n'existe pas d'élément $\delta \in FRERE(\alpha)$ tel que $\beta \in FRERE(\delta)$

- Elément dominant

Tout élément α de D tel qu'il n'existe pas d'éléments $\gamma \in D$ tels que $\alpha \in FILS(\gamma)$.

Exemple:

Soit D = {␣␣␣,PARTIEL,PARTIES,PARTIE,PART,PARTOUT,ZZZZ}

FILS(PARTIEL) =FILS(PARTIES) = {PARTIE,PART}

FILS(PARTOUT) = FILS(PARTIE) = {PART}

PART est fils direct de PARTIE et de PARTOUT

PARTIE est fils direct de PARTIES et de PARTIEL

PARTIES est frère direct de PARTIEL

PARTOUT est frère direct de PARTIES

Organisation

Le dictionnaire D est composé:

- d'un arbre binaire ordonné par ordre alphabétique (D1) qui permettra l'accès à une sous-liste,

- d'une liste chaînée par ordre alphabétique (D2) tenant compte des relations frère-direct et fils-direct composée de p sous-listes.

Un élément du dictionnaire contient 4 types d'informations:

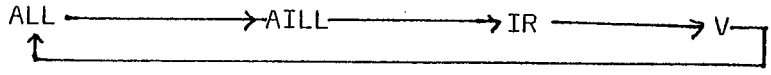
a/ un identificateur: il est constitué d'une chaîne de caractères quelconques, y compris le blanc, d'une longueur maximale actuelle de 24. Le caractère "/" étant utilisé comme délimiteur de chaîne, il ne peut être contenu dans celle-ci.

b/ des pointeurs: il existe trois types différents de pointeurs.

- des pointeurs réalisant le chaînage physique: tous les éléments sont chaînés entre eux, ce qui permet une identification optimale. Le nombre de ces pointeurs est fixe.
L'algorithme de recherche fournissant toujours l'adresse de l'élément le plus long contenu dans la chaîne donnée, les pointeurs physiques permettent d'accéder aux éléments fils et aux éléments frères.

- des pointeurs réalisant le chaînage logique: ces pointeurs assurent un chaînage circulaire entre plusieurs éléments"frères" ou "fils" qui appartiennent à un même ensemble logique dans une application linguistique. La génération morphologique est réalisée à l'aide de ces pointeurs.

Exemple: chaînage circulaire des diverses bases du verbe aller:



La génération morphologique du verbe aller donnera les résultats suivants:

ALLons, ALLez ...
AILLes, ...
IRa, IRons, ...
Va, Vont, ...

D'autres familles peuvent être imaginées telles que les synonymes pour l'application documentaire, ou les chaînes de traduction pour l'adaptation du système à la traduction.

Le nombre de pointeurs logiques est un paramètre du programme, il est constant pour une application donnée.

- un ensemble de trois pointeurs faisant référence aux trois ensembles de modèles habituels (morphologique, syntaxique, sémantique).

c/ des renseignements

- la longueur de l'élement (car les chaînes ont une longueur variable),
- trois indicateurs représentés chacun par un bit, utilisés pour indiquer si:
  - l'élément est situé dans l'arbre,
  - l'élément est un fils,
  - l'élément est une forme indécoupable.
- un indicateur supprimant la génération de solutions parasites qui pourraient être obtenues à partir de chaînes plus courtes.

d/ des compteurs incrémentés lors de l'identification d'une chaîne et utilisés par le programme de réorganisation du dictionnaire.

Environnement du dictionnaire

L'appel au dictionnaire permet soit son extension, soit sa réorganisation. En ce qui concerne l'extension du dictionnaire, un petit langage a été défini pour introduire, supprimer, remplacer, ou interroger un élément du dictionnaire. Celui-ci est décrit dans [COUGD].

Nous indiquons ici, la commande utilisée pour introduire un élément dans le dictionnaire.

/CHAINE {(*)} / MODELE / {CHAINAGE/}

Les éléments entre crochets sont facultatifs.

Cette commande permet de mettre en évidence les attributs d'une chaîne de caractères:

- le MODELE morphologique auquel la chaîne fait obligatoirement référence,
- l'indicateur de chaîne indécoupable: (*),
- le ou les CHAINAGES circulaires avec d'autres bases.

2.2.2. Grammaire

La grammaire définie est une grammaire à validations et saturations [COU1], équivalente à une grammaire d'états finis, mais qui permet une écriture plus aisée et plus condensée que celles-ci. Les définitions des validations et des saturations sont données dans la suite.

Comme nous l'avons déjà vu, le rôle de la grammaire est double:

- contrôler la concaténation des différents constituants du mot. Ce contrôle est effectué à l'aide des validations, saturations et du code morphologique.
- effectuer la transduction des variables fournies par le modèle et la règle.

Des opérations ont été définies pour réaliser le contrôle et la transduction. Une règle de grammaire est composée de deux parties, l'une est utilisée pour la reconnaissance, l'autre pour obtenir des renseignements linguistiques.

Informations linguistiques

On distingue deux sortes d'informations linguistiques:

- la catégorie qui ne peut prendre qu'une seule valeur,
- la variable grammaticale qui est constituée d'un ensemble de valeurs.

On appelle type, une information qui ne peut prendre qu'une seule valeur et variable, une information composée d'un ensemble de valeurs.

Deux types ont été définis:

- la classe lexicale (CL) qui peut être modifiée au cours de la dérivation,
- la classe de base notée CB.

Exemple:

DIVISION qui se décompose en DIVIS+ION a pour classe lexicale "substantif" et pour classe de base "VERB" si l'on a fait dériver le substantif de la base verbale "DIVIS".

Les opérations applicables à ces types sont:

- transmettre la valeur d'un type d'un état à un autre,
- créer une valeur,
- accéder à la valeur définie au niveau du modèle.

Un type particulier, appelé code dérivation (CD) a été défini. Il permet au cours de la dérivation, de produire une chaîne de caractères. Ce n'est pas exactement un type par le fait que le code dérivation n'est pas une valeur, mais une concaténation de valeurs.

Le CD est utilisé soit pour indiquer le cheminement, soit pour effectuer une traduction de chaîne à chaîne.

Par convention, les symboles "G" et "D" signifient respectivement gauche et droite (ou dictionnaire).

Les opérations sur le CD sont les suivantes:

- transmission d'un état à un autre: CD := CD(G)
- création d'une valeur: CD := valeur
- accès à la valeur définie par le modèle: CD := CD(D)
- concaténation: CD := CD(G) || CD(D).

Les opérations définies sur les variables sont:

- transmettre des valeurs d'un état à un autre,
- imposer un ensemble de valeurs,
- accéder aux valeurs définies au niveau du modèle,
- ajouter ou retrancher une valeur ou un ensemble de valeurs,
- combiner les quatre opérations précédentes.

Modèles

Les modèles sont les représentants d'une classe dont tous les éléments ont le même comportement morphologique (par exemple, les verbes du premier groupe).

Ils réalisent l'interface entre le dictionnaire et la grammaire. En particulier, ils contiennent des informations permettant de faire la liaison avec les règles de grammaire, des renseignements linguistiques et également des validations et des saturations afin de limiter le nombre de règles. De ce fait, pour contrôler le cheminement, un calcul supplémentaire réalise l'union des validations contenues dans une règle et un modèle ainsi que l'union des saturations.

Code morphologique

Il sert à définir un autre système de validation et est utilisé en particulier dans les langues naturelles pour faire référence à un système de désinences. Son introduction vient du fait suivant: soit par exemple la base "CENTR". Nous pouvons obtenir la conjugaison du verbe CENTRER ou bien un adjectif (CENTRAL) ou un substantif (CENTRE ou CENTRALISATION) ou la conjugaison du verbe CENTRALISER etc... Afin de simplifier les validations et les saturations, le code morphologique qui peut être modifié au cours de la dérivation, permet de définir à chaque pas le système de désinences adéquat. En reprenant l'exemple "CENTRE", cette base aura comme valeur de CM le système de désinences correspondant à la conjugaison des verbes du premier groupe et au substantif commun et comme validation la règle permettant de concaténer le suffixe "AL". Par contre, la règle qui permet le suffixe "AL" aura comme valeur de CM le système de désinence des adjectifs et comme validation, une règle permettant de concaténer le suffixe IS etc... le CM ayant été défini comme un type, on peut le transmettre d'un état à un autre, ce qui est très intéressant au niveau des applications.

Règles

Une règle de grammaire est définie par:

- des informations linguistiques,
- des validations, des saturations et des indicateurs qui permettent de contrôler la concaténation des constituants. Ces indicateurs sont l'indicateur FIM(fin de mot) qui indique à l'automate qu'il doit s'arrêter (le blanc n'étant plus considéré comme un séparateur) et l'indicateur ND dont l'utilisation est précisée plus loin.

Le résultat de l'application d'une règle d'état fini conduit à calculer un nouvel ensemble de validations et saturations ou état courant de l'automate et à transducter des propriétés à partir des propriétés de l'état précédent et de celles fournies par le modèle et la règle.

Réalisation du contrôle

Le contrôle est effectué à l'aide des validations, des saturations et du code morphologique (CM), qui sont exprimées sous forme de listes de règles.

Les validations et les saturations sont utilisées pour le contrôle de la dérivation alors que le "CM" est utilisé pour faire référence à un système de désinences. Le système de désinences peut se modifier au cours de la dérivation et a donc été distingué des validations.

Le calcul des validations et saturations est automatique. Le principe est le suivant:
Après application d'une règle, les validations sont redéfinies, alors que pour les saturations, elles se transmettent d'états en états en effectuant l'union entre les saturations précédentes et celles définies par la règle et le modèle.

Soient VAL_I et SAT_I les validations et saturations à l'étape I de la dérivation.

Soient VAL_R, VAL_M, SAT_R, SAT_M, les validations et les saturations de la règle R et du modèle M. A l'étape I+1 de la dérivation, après application de la règle R faisant appel au modèle M, on obtient:

$$VAL_{I+1} = (VAL_R \cup VAL_M) \cap \overline{SAT}_{I+1}$$

$$SAT_{I+1} = (SAT_I \cup SAT_R \cup SAT_M)$$

Par convention, l'état initial de l'automate est défini par la liste des règles contenues dans VAL00.

Si le modèle sélectionné est "M+1" alors la liste des règles applicables à l'étape I+1 de la dérivation avec le modèle "M+1" est égale à:

$$REG_{M+1} \cap VAL_{I+1}$$

où REG_{M+1} est l'ensemble défini par le mot-clé REG du modèle M+1.

Si à l'étape I+1 de la dérivation, la liste des règles applicables avec le modèle "M+1" est vide et que la règle "R" ne contient pas l'indicateur "ND" (non désinence), alors la liste des règles applicables esr égale à:

$$REG_{M+1} \cap CM_R$$

où CM_R est le code morphologique défini par l'application de la règle R et du modèle M+1.

Afin d'accélérer l'algorithme, l'indicateur ND présent dans une règle interdit l'utilisation du code morphologique porté par celle-ci. En effet, le "CM" ne doit intervenir qu'à la fin du mot et se transmet au cours du cheminement.

Exemple d'utilisation

Un compilateur incrémentiel permet la création ou la mise à jour de la grammaire. La syntaxe des déclarations (variables, types, codes morphologiques, validations, saturations) ainsi que la syntaxe de l'écriture des règles et des modèles, sont décrites dans [COUGD].

Soient VALOO = (RIB, ...)
CM12 = (PR4, SU1, ...)
VAL1 = (E1,E8)

Considérons les mots FINIR, FINIRONS, FINIREZ ...
DIRE, DIRONS, DIREZ /..
On peut admettre les découpages suivants:

FINI + R + { ONS␣ / EZ␣ / ␣ }

DI + R + { ONS␣ / EZ␣ / E␣ }

Cela signifie qu'après application de la règle permettant la concaténation du suffixe "R", il faut autoriser:

- <u>pour la base FINI</u> : les désinences "ONS␣", "EZ␣" du futur et la désinence "␣" de l'infinitif.
- <u>pour la base DI</u> : les désinences "ONS␣", "EZ␣" du futur et la désinence "E␣" de l'infinitif.

L'utilisation des saturations permet de résoudre ce problème: après application de la règle autorisant le suffixe "R", on valide les règles autorisant "ONS␣", "EZ␣", "␣" et "E␣", mais par contre, au niveau de la base FINI, on interdit la règle du "E␣" et au niveau de la base DI on interdit la règle de "␣".

D'où:

les modèles

/FINI/ : REG := (RIB) ; CL := VERB ; CB := BVRB ; VAL := (...,FU5,...) ;
SAT := (NF2).

/DI/ : REG := (RIB) ; CL := VERB ; CB := BVRB ; VAL := (...,FU5,...) ;
SAT := (NF1).

/R/ : REG := (FU5) ; VAL := ( ) ; SAT := ( ).

/␣/ : REG := (NF1,PR7,E1) ; ... ; VAL := ( ) ; SAT := ( ).

/E␣/ : REG := (SU1,PR1,RO3,NF2) ; ... ; VAL := ( ) ; SAT := ( ).

/ONS␣/ : REG := (PR2,RO5,FU7,SU4) ; VAR := UNO+PLU ; VAL := ( ); SAT := ( ).

/EZ␣/ : REG := (PR2,RO5,FU7,SU4) ; VAR := DUO+PLU ; VAL := ( ); SAT := ( ).

les règles

FU5 : ... ; VAL := (FU7,FU8,NF1,NF2) ; SAT := ( ).

FU7 : ... ; VAR := VAR(D) + FUT + IND ; VAL := ( ) ; SAT := ( ) ; FIM.

NF1 : CL := INFI ; CB := CB(G) ; VAL := ( ) ; SAT := ( ) ; FIM.

NF2 : CL := INFI ; CB := CB(G) ; VAL := ( ) ; SAT := ( ) ; FIM.

RIB : CL := CL(D) ; CB := CB(D) ; CM := CM(D) ; VAR := VAR(D) ; VAL := ( );
SAT := ( ).

Analyse de "FINIR␣"

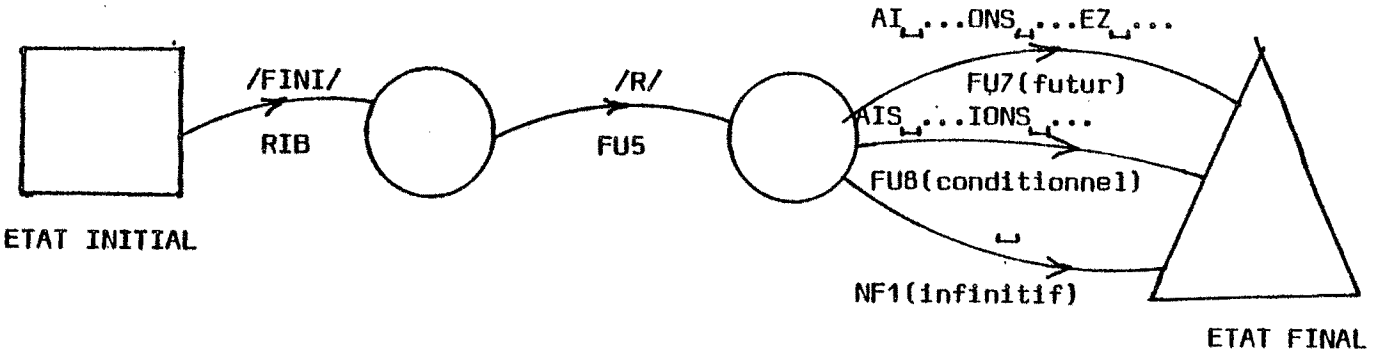
Après avoir appliqué la règle RIB, on a:

VAL := (...,FU5,...) et SAT := NF2.

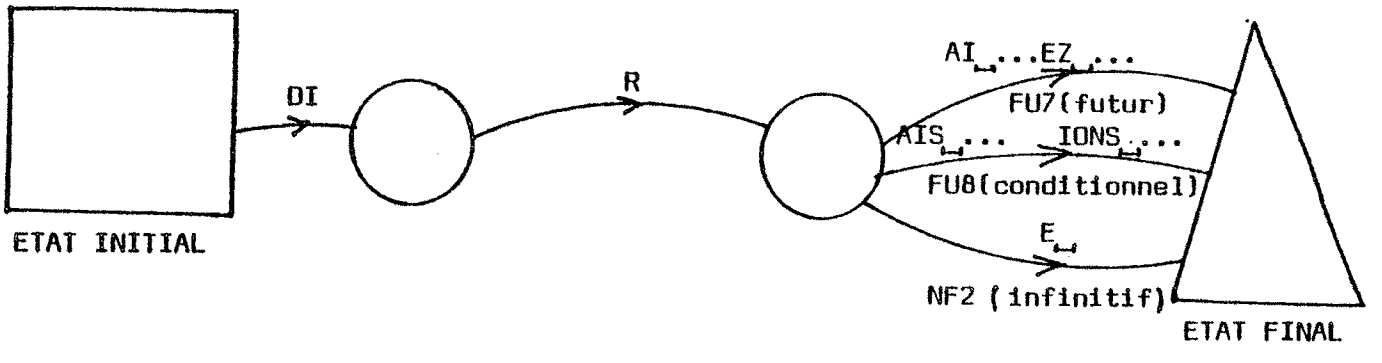
L'identification de "R" permet l'analyse de "FINIR" avec

VAL := (FU7,FU8,NF1,NF2) et SAT := NF2.

NF2 ayant été saturé par la base FINI, les validations de l'état suivant sont égales à (FU7, FU8, NF1), ce qui permet l'analyse de "FINIR␣" mais interdit la reconnaissance de "FINIRE␣".



Pour DIRE, on obtient le diagramme suivant:



2.3. Editeur lexicographique

Pour permettre l'exécution conversationnelle de la morphologie, avec possibilité de modifier les informations linguistiques même en cours d'exécution, un éditeur lexicographique a été défini. Celui-ci est décrit dans le chapitre suivant.

II - ADAPTATION DU SYSTEME PIAF A LA TRADUCTION

La traduction d'une instruction écrite dans un certain langage de programmation (langage source) en une instruction écrite dans un autre langage de programmation (langage cible) de la même famille que le langage source peut être réalisée par l'analyse morphologique d'une langue naturelle moyennant la création d'un dictionnaire et d'une grammaire adéquats. En effet:

- la segmentation de l'instruction effectue la reconnaissance des éléments du langage source ;
- la transduction des résultats fournit la traduction dans le langage cible de cette instruction.

La conception générale des modules constituant le système PIAF nous a permis d'utiliser cette analogie pour construire, à partir de ce système, un traducteur (le traducteur PIAFTRAD) de programmes écrits dans des langages semi-évolués. [CHAS1]

Les principales modifications sont les suivantes:

. comme nous l'avons déjà indiqué, seul le transducteur général d'états finis a été utilisé. L'adaptation de celui-ci à la traduction repose principalement sur l'utilisation de la technique de retour-arrière pour la transduction des résultats ;

. les paramètres linguistiques (dictionnaire et grammaire) ont été modifiés ;

. des programmes spécifiques à la traduction ont été créés (interprétation des résultats fournis par le transducteur, traitement des déclarations, enchaînement des modules.)

Dans cette partie, nous décrivons l'adaptation des modules du système PIAF, c'est-à-dire:

- adaptation des deux phases de l'analyseur morphologique (segmentation et transduction) ;

- modification des paramètres linguistiques.

Puis nous donnons des exemples de traduction en LP80, d'une instruction PL360.

La création des modules spécifiques à la traduction font l'objet du chapitre suivant: fonctionnement du traducteur PIAFTRAD.

1 - Utilisation de l'analyseur morphologique pour la traduction

1.1. Segmentation

Etant donné une instruction $\alpha_1\alpha_2 \ldots\alpha_n$ ($\forall i \in [1,n]$, $\alpha i \in V$) on doit déterminer des mots ou groupes de mots m_i ($m_i \in V^+$) tels que:

$$m_1 m_2 \ldots m_k = \alpha_1\alpha_2 \ldots \alpha_n \quad \text{avec} \quad k \leq n$$

La détermination d'un m_i est la reconnaissance de:

- un symbole de base ou un groupe de symboles de base du langage,
- une valeur (constante, chaîne de caractères),
- un identificateur.

Pour ce faire, on doit donc disposer de:

- un dictionnaire contenant les symboles de base du langage, les constantes et les identificateurs,
- une grammaire qui doit confirmer ou infirmer la concaténation de ces éléments. Cette grammaire doit contenir les règles de traduction des instructions PL360.

Le symbole terminal de fin d'instruction est le point virgule. L'organisation du dictionnaire permet d'utiliser la technique du "LONGEST MATCH" pour la recherche d'un élément.

En général, la décomposition d'un m_i est unique. En effet:

- si m_i est un identificateur ou une valeur, c'est la plus longue chaîne du dictionnaire obtenue par identification qui convient et non une sous-chaîne.

- si m_i est un symbole de base, deux possibilités se présentent:

. soit la plus longue chaîne obtenue par identification a conduit à un résultat; c'est celle-ci qui convient et on ne cherchera pas à la segmenter en sous-chaînes ;

. soit celle-ci n'a conduit à aucun résultat ; le découpage de m_i intervient alors pour trouver une sous-chaîne qui soit validée et qui permette d'atteindre un état final.

Exemple:

Soit le dictionnaire organisé de la manière suivante:

```
/;/ → /(/ → /)␣;/ → /:=/ → /=/ → /IDETAB/ → /R3/ → /R5/
          ↓       ↓              ↓
         /)/     /:/           /IDE/
```

- L'instruction "R3 := IDETAB;" est segmentée de manière unique:

R3 := IDETAB ;

En effet, l'identificateur IDETAB ayant été reconnu, on ne cherche pas à identifier la chaîne plus courte IDE. D'autre part, le symbole de base ":=" ayant conduit à un état final, on ne recommence pas l'analyse avec le symbole plus court ":".

- L'instruction "R3 := IDETAB(R5);" est segmentée de la manière suivante:

R3 := IDETAB (R5) ;

l'élément ")␣;" est identifié dans le dictionnaire. Mais l'essai avec cet élément se révélant infructueux, on reprend l'analyse avec l'élément plus court ")" qui conduit à un état final. On valide le symbole ")" plutôt que ")␣;" pour pouvoir traiter les instructions suivantes avec les mêmes règles:

R3 := IDETAB (R5) + ...

R3 := IDETAB (R5) ØR ...

1.2. Transduction

Pour chaque m_i déterminé, il ne s'agit pas de donner toutes les interprétations de toutes les décompositions de ce m_i, mais la chaîne fournie par les applications

de toutes les règles de tous les modèles sur lesquels ce m_i est indexé, lorsque celles-ci aboutissent à un résultat. Dans ce cas, chaque m_i ne fournit pas des renseignements linguistiques, mais une "chaîne de transduction" (élément de V^+). La concaténation de ces éléments donne la traduction de l'instruction.

Cette transduction est réalisée par la grammaire, en utilisant la technique de retour-arrière.

Nous présentons, d'abord, cette technique fréquemment utilisée pour la recherche d'une ou plusieurs solutions dans les problèmes combinatoires représentables sous forme d'arborescences, puis indiquons l'utilisation de celle-ci pour la traduction.

La technique de retour-arrière (ou back-track)[FLOYD, GOBA, NIFARE, VEI] consiste choisir une solution possible et à revenir en arrière, en cas d'achec, pour essayer la solution suivante. C'est un problème de recherche à travers un ensemble fini, qui peut être formalisé de la manière suivante:

- soit n ensembles ordonnés U_1, U_2, ...U_n.

Le but de ce problème est de construire un vecteur $A = (a_1, a_2, ..., a_n)$ où $a_i \in U_i$ $(i \in [1,n])$ et satisfaisant à un ensemble de conditions ou de contraintes.

Supposons que les (k-1) premiers composants du vecteur ont été trouvés.

$A = (a_1, a_2, ..., a_{k-1}, ? ...)$.

L'ensemble des contraintes limite les choix du composant suivant (a_k) à un sous-ensemble S_k de U_k.

- si S_k est non-vide, on choisit un élément de S_k et on poursuit la recherche avec S_{k+1}, etc...

- si S_k est vide, on revient en arrière, à l'étape précédente, pour rejeter le $(k-1)^{ème}$ élément de a_{k-1} de A et choisir un nouvel élément a_{k-1} dans S_{k-1}. Compte tenu de ce nouveau choix pour a_{k-1} et des conditions du problème, l'ensemble S_k peut ne pas être vide. On choisit un élément a_k dans S_k.

- si aucun choix n'est possible pour a_{k-1}, lors du retour-arrière, on revient à nouveau en arrière, à l'étape k-2 pour choisir un nouvel élément a_{k-2} de S_{k-2} etc...

La technique de retour-arrière habituelle ne conserve pas les résultats ou choix qui ne conduisent pas à une solution finale. Cette technique est utilisée dans l'analyse morphologique d'une phrase par le système PIAF.

En ce qui concerne la traduction PL360-LP80, on peut être amené à engendrer plusieurs instructions LP80 pour la traduction d'une seule instruction PL360. Ces instructions LP80 sont engendrées séquentiellement par parcours successifs de l'arbre de dérivation.

L'adaptation du système PIAF à la traduction utilise une technique dérivée de la technique de retour-arrière, de deux façons différentes:

- tant qu'une partie de la solution ou la solution complète n'a pas été trouvée, il y a retour-arrière sans conservation des résultats ;

- dès qu'on a trouvé une partie de la solution, il y a retour-arrière avec conservation des résultats afin de concaténer celle-ci avec les solutions éventuellement engendrées par d'autres parcours.

Par analogie, on appellera également "retour-arrière" cette technique dans la suite.

Pour conserver les résultats, on utilise l'indicateur FIM (fin-de-mot) défini dans le système PIAF, pour indiquer à l'automate qu'il doit s'arrêter (le blanc n'étant plus un séparateur de mots). L'utilisation de FIM, dans l'analyse d'une instruction, permet de conserver les résultats en ce point, et de provoquer le retour-arrière de l'analyseur. FIM est donc utilisé comme point d'arrêt dans l'instruction et pas uniquement comme fin d'instruction.

On utilise l'indicateur FIM (fin-de-mot) défini dans le système PIAF, pour indiquer à l'automate qu'il doit s'arrêter (le blanc n'étant plus un séparateur de mots). L'utilisation de FIM, dans l'analyse d'une instruction, permet de conserver les résultats en ce point,et de provoquer le retour-arrière de l'analyseur. FIM est donc utilisé comme point d'arrêt dans l'instruction et pas uniquement comme fin d'instruction.

La traduction d'une instruction peut se formuler de la manière suivante:
Soit $U_1=U_2=\ldots=U_n=U=$ {ensemble des règles de la grammaire}
Construire un vecteur $A=(a_1,a_2,\ldots,a_n)$ où $a_i \in U$, satisfaisant aux conditions et contraintes établies par la grammaire de traduction et produire simultanément un vecteur de traduction $C = (c_1,c_2,\ldots,c_n)$ où $c_i \in V^+$ (c_i pouvant être la chaîne vide).

Soit S_k l'ensemble des règles applicables à l'étape k-1 de la dérivation. Comme on l'a déjà indiqué, la liste des règles applicables à l'étape k-1 de la dérivation est:

- $REG_m \cap VAL_{k-1}$ si cette intersection n'est pas vide
- $REG_m \cap CM_r$ sinon

(où m et r sont le modèle et la règle respectivement sélectionnés).

De plus, un élément pouvant être indexé dans le dictionnaire, suivant plusieurs modèles dans un certain ordre donné, chaque modèle p_i détermine donc un ensemble $S_{k,i}$ de règles apllicables.
S_k est alors considéré comme l'union des sous-ensembles $S_{k,i}$.

Le principe de la traduction d'une instruction, est le suivant:

- recherche de chaque modèle p_i sur lequel est indexé ce segment,
- détermination de l'ensemble $S_{k,i}$ correspondant à chaque p_i.

- si $S_{k,i}$ est non-vide, choix d'une règle $a_{k,i}$ de $S_{k,i}$. Si $a_{k,i}$ conduit à un état marqué par FIM (point d'arrêt dans l'instruction), on concatène la chaîne de traduction c_k portée par la règle $a_{k,i}$ au vecteur $C=(c_1,c_2,\ldots,c_{k-1})$ et on conserve ce résultat, sinon il y a retour-arrière avec non-conservation des éléments de C concaténés depuis le dernier état marqué par FIM.

- si $S_{k,i}$ est vide, détermination de $S_{k,i+1}$ et poursuite de l'analyse avec $S_{k,i+1}$.

Lorsque tous les $S_{k,i}$ ont été utilisés et si aucune règle n'a conduit à un état marqué par FIM, alors s'il existe un segment plus court, l'analyse reprend avec ce nouveau segment.

Donc le retour-arrière sans conservation des résultats permet d'éliminer les choix qui ne conduisent pas à un état final, tandis que le retour-arrière avec conservation des résultats permet de concaténer des chaînes de traduction relatives à un élément et au type de l'instruction.

D'autre part, les chaînes de traduction doivent être concaténées dans un certain ordre. C'est la raison pour laquelle l'écrivain des paramètres linguistiques pour la traduction (dictionnaire et grammaire) est conduit à indexer dans un certain ordre, un même élément sur plusieurs modèles, ce qui permet d'établir une hiérarchie parmi les règles applicables (détermination de $S_{k,1}$, puis de $S_{k,2}$ etc...) et par conséquent d'établir un ordre d'écriture pour les chaînes de traduction C_k.

D'après ce qui précède, le principe de la transduction des résultats pour une instruction peut être considéré comme l'écriture ordonnée de chaînes de traduction portées par les éléments du dictionnaire ou les règles de la grammaire, obtenues par les techniques de transduction et de retour-arrière.

Un exemple illustrant ces techniques de transduction est décrit à la fin de ce chapitre.

2 - Modification des paramètres linguistiques

2.1. Dictionnaire

Les éléments indexés dans le dictionnaire sont:

- les symboles de base du langage,
- les valeurs (constantes et chaînes de caractères),
- les identificateurs.

Parmi les informations décrivant un élément du dictionnaire, les renseignements et les pointeurs sont utilisés différemment:

- le chaînage physique FILS-DIRECT est utilisé pour indexer un identificateur sur plusieurs modèles et dans un certain ordre. Ceci permet de construire la chaîne de traduction par la technique de retour-arrière.

- le chaînage logique circulaire permet de:

. mettre en correspondance un élément du langage PL360 indexé dans le dictionnaire et sa traduction en LP80.

Exemple: L'identificateur MVC en PL360 est traduit par l'identificateur MOVE en LP80. Le chaînage logique circulaire assure cette correspondance.
Remarque : L'identificateur MOVE est indexé dans le dictionnaire sur un modèle (le modèle BIDON) qui n'intervient pas dans l'analyse des instructions.

. mettre en correspondance un élément (ou une famille d'éléments) et une chaîne de caractères que l'on veut substituer à cet élément (ou à un élément de la famille) dans la traduction.

L'élément du chaînage circulaire est:

- soit un symbole de base ou un groupe de symboles de bases appartenant déjà au dictionnaire ;
- soit un élément n'appartenant pas au dictionnaire ; celui-ci est alors créé et indexé sur un modèle arbitraire, le modèle 'BIDON .

- indicateurs

Les indicateurs de situation de l'élément (arbre, fils) ne sont pas modifiés.
L'indicateur de découpage n'est pas utilisé, puisque la décomposition d'un segment m_i est unique.

Deux indicateurs ont été définis:

- un indicateur de chaînage circulaire: il est représenté symboliquement par l'indicateur S.

Exemple: /MVC(S)/MVC/MOVE/
L'identificateur MVC en PL360 est indexé sur le modèle MVC et chaîné circulairement avec l'identificateur MOVE.

- un indicateur (l'indicateur D) utilisé pour le traitement des déclarations dont nous préciserons l'emploi ultérieurement.

2.2. Grammaire

Les informations linguistiques sont utilisées de façon différente en vue de la production d'une chaîne de traduction.

2.2.1. Utilisation du code dérivation (CD)

La classe lexicale (CL) et la classe de base (CB), de même que les variables ne sont pas utilisées par le traducteur.

En revanche, l'utilisation du type particulier CD permet au cours de la dérivation de produire une chaîne de traduction formée de la concaténation des chaînes portées par chaque m_i.

Les opérations sur le CD dans une règle s'interprètent de la manière suivante:

- transmission de la chaîne de traduction courante à l'état suivant: CD := CD(G) ;
- accès à la chaîne définie par le modèle: CD := CD(D) ;
- création d'une valeur de CD: CD := chaîne ;
- concaténation :

  CD := CD(G)||CD(D)

  CD := CD(G)||chaîne.

D'autre part, on a défini deux nouvelles opérations autorisées uniquement dans un modèle qui constituent un accès à l'élément indexé sur ce modèle dans le dictionnaire:

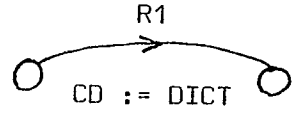
- la première opération fait référence à la chaîne de caractères de l'élément. On la note par CD := DICT.
- la deuxième opération fait référence au chaînage logique circulaire de l'élément. On la note par CD := SYNO.

Exemples

. Les identificateurs de registres R0,R1,...,R9 sont traduits en LP80 par les mêmes identificateurs.
R0,R1,...,R9 sont indexés dans le dictionnaire suivant le modèle REGK de la grammaire défini de la manière suivante:
modèle: /REGK/ ... CD := DICT

dictionnaire /R0/REGK/
/R1/REGK/
etc...

Lors de l'analyse de l'identificateur R1, le code-dérivation du modèle REGK sur lequel R1 est indexé, fait référence à la chaîne de caractères de l'élément R1.



le CD fait référence à la chaîne de caractères "R1"

. Soit le modèle OPDEC sur lequel sont indexés les opérateurs de décalage (SHRL,SHLL, etc...) du langage PL360, dans le dictionnaire.

Les opérateurs SHRL,SHLL, etc... sont traduits respectivement en LP80 par les opérateurs SRLS,SLLS,etc... On met en relation les opérateurs PL360 et leurs correspondants en LP80 par l'intermédiaire du chaînage circulaire:

dictionnaire: /SHRL(S)/OPDEC/SRLS/
/SHLL(S)/OPDEC/SLLS/

On définit le modèle OPDEC dans la grammaire de la manière suivante:
/OPDEC/ : ... CD := SYNØ ...

Lorsqu'on identifie le segment SHRL dans une instruction, le code dérivation correspondant au modèle OPDEC sur lequel il est indexé, fait référence à la chaîne "SRLS" obtenue par le chaînage circulaire de SHRL.

2.2.2. Règles et modèles

Les règles sont constituées par une liste de validations, une liste de saturations, de code morphologique (CM), le code-dérivation et des indicateurs (FIM et ND).

L'utilisation de l'indicateur ND est expliquée ultérieurement dans le fonctionnement du traducteur.

Exemples:

RIB : CD := CD(D) ; CM := CM(D) ; VAL := (BLA).
RB11 : CD := CD(G)||CD(D) ; CM := CM(D) ; VAL := (BLA) ; SAT := (BINA)

Les modèles sont constitués par une liste de règles, une liste de validations et saturations, le code morphologique (CM), le code dérivation (CD).

Nous rappelons que les deux opérations permises sur le code dérivation sont CD := DICT (référence à l'élément) et CD := SYNO (référence au chaînage circulaire).

Exemples:

/FOR/ : REG := (RIB) ; CD := DICT ; CM := 3 ;
VAL := (RG1) ; SAT := (NBR4)

où RIB, RG1 et NBR4 sont des noms de règles.

CM := 3 fait référence au CM3 qui est défini parmi les codes morphologiques.

2.2.3. Code morphologique, validations, saturations

Contrairement à l'utilisation du code morphologique dans le traitement des langues où il intervient en fin d'analyse, dans la traduction celui-ci est défini pour un type d'instruction (instruction conditionnelle, choix, d'affectation etc...) et permet d'appliquer à chaque pas les règles correspondant à cette instruction.

Le code morphologique peut être transmis d'un état à un autre, tandis que les validations ne se transmettent pas. Les validations sont utilisées pour éviter l'application des règles fournies par le code morphologique.

Exemple:

Instruction conditionnelle (IF, WHILE): Le modèle sur lequel est indexé l'identificateur IF caractérisant le type d'instruction détermine par son code morphologique, l'ensemble des règles à appliquer à l'instruction.

Or, l'instruction (1): IF R5 <=1 THEN... est traduite par la même instruction en LP80, tandis que l'instruction (2): IF<= THEN est traduite en LP80, par IF > = THEN... (modification du code condition) dans certains cas (cf. IIème partie)

La même règle ne peut pas être utilisée pour la traduction de l'opérateur de relation.

Pour résoudre ce problème, on range dans les validations du modèle IF, la règle qui permet de traduire l'opérateur de relation dans l'instruction (2) et dans le code morphologique de ce modèle, la règle permettant de traduire cet opérateur dans l'instruction (1). Ce choix est imposé par le fait que les validations ne se transmettent pas (présence de l'identificateur R3 entre IF et <= dans l'instruction (1)).

Le calcul des validations et des saturations est automatique et s'effectue de la même façon que dans le système PIAF.

Les saturations interdisent l'emploi de certaines règles contenues dans les validations, compte tenu des saturations portées par le modèle, la règle et les saturations provenant de l'état précédent.

L'état initial de l'automate est défini par la liste des règles contenues dans VAL00.

On rappelle que l'indicateur "ND" n'est pas utilisé pour interdire l'utilisation du code morphologique porté par une règle, mais intervient dans le fonctionnement du traducteur.

2.2.4. Exemples

Ces exemples ont été choisis pour illustrer les techniques de retour-arrière et le fonctionnement de la grammaire. Certains détails ne pouvant être expliqués que dans le fonctionnement du traducteur, cet exemple figure ici uniquement à titre indicatif et ne peut donc être utilisé pour traduire réellement cette instruction.

1/ Principe de la transduction: traduction de l'instruction PL360: IC(R5,WTAB);

En LP80, pour des raisons données dans la deuxième partie, cette instruction se traduit par la séquence d'instructions suivantes:

```
R2 := @WTAB  SLLS 2; R5 := BYTE0(R2);
```

L'enchaînement de ces deux instructions est spécifique à l'instruction IC qui doit également valider des opérandes de type différent. C'est la raison pour laquelle on choisit d'indexer l'identificateur IC sur au moins deux modèles permettant d'établir un ordre dans l'application des règles.

D'autre part, de même que la racine d'un mot dans l'analyse d'une langue indique les règles à appliquer à ce mot, de même le nom de l'instruction

doit indiquer pour toute instruction les règles à appliquer (par exemple, IC, := etc...)

Dans ce cas, pour l'instruction IC, cet identificateur est indexé sur deux modèles, pour les raisons suivantes:

- le premier modèle permet l'analyse de l'instruction IC avec un opérande de type mot, demi-mot, ou de type B1(=MEM(R1)) ;

- le deuxième modèle permet soit l'analyse de IC avec un opérande de type octet, soit l'obtention de la deuxième instruction de la traduction, dans le cas de l'analyse de IC faite dans le cas précédent.

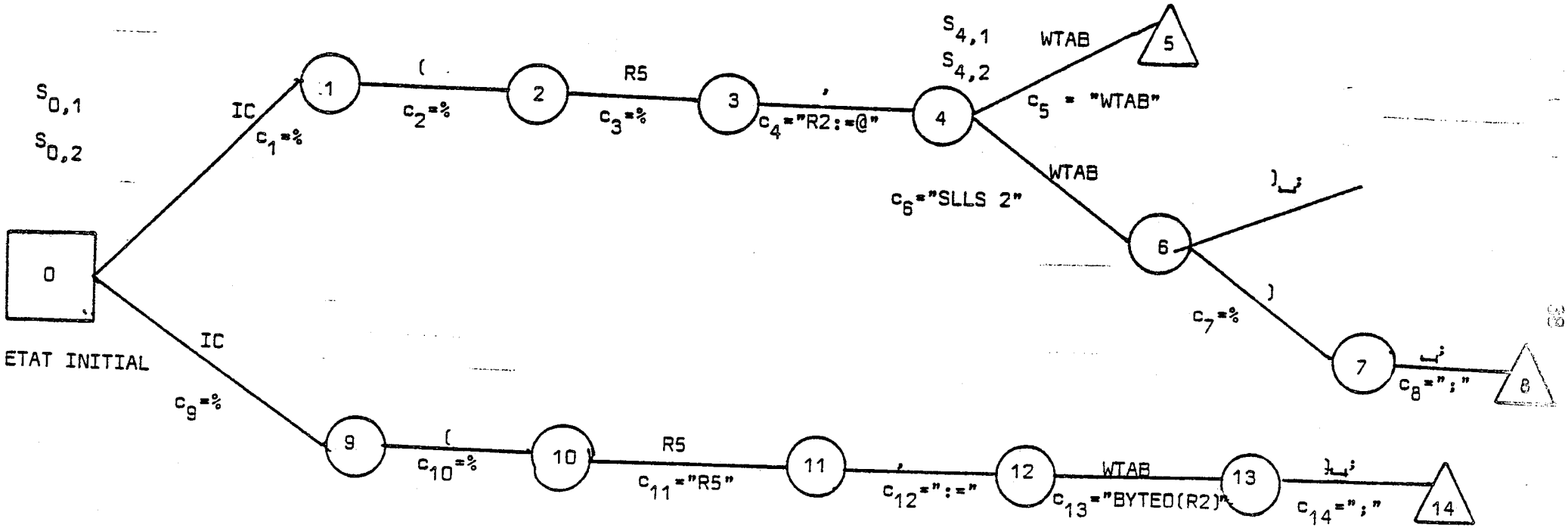
- le premier chemin (1) fourni par l'application des règles prises dans l'ensemble $S_{0,1}$ relatif au premier modèle m_1 conduit à un embranchement à l'étape 4. Le vecteur de traduction obtenu à cette étape-là est $C = (c_1, c_2, c_3, c_4)$.
On détermine l'ensemble $S_{4,1}$ qui contient une seule règle conduisant à un état final. Cette solution est donc conservée avant le retour-arrière et fournit le vecteur $C = (c_1, c_2, c_3, c_4, c_5)$. On détermine ensuite l'ensemble $S_{4,2}$ qui contient aussi une règle unique.

A l'étape 6 de la dérivation, la segmentation identifie l'élément $)_{\sqcup}$ du dictionnaire. L'ensemble S_6 étant vide pour le modèle de ce segment, on cherche une solution avec un segment plus court. Il existe une règle qui conduit à un état final. Le vecteur de traduction obtenu est:

$C = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$

- le deuxième chemin (2) fourni par l'application des règles $S_{0,2}$ relatif à m_2 conduit à un état final et donne le vecteur de traduction $C = (c_1, \ldots, c_{14})$ qui correspond à la chaîne de traduction obtenue par concaténation des chaînes de chaque c_i

R2 := @WTAB SLLS 2 ; R5 := BYTED(R2) ;



c_i=% indique la chaîne vide, par définition.

2/ Application des règles

Nous préférons ici, expliquer la traduction de l'instruction PL360 SET(DTERM); plutôt que celle de l'exemple précédent, afin de montrer l'utilisation des saturations.

Cette instruction est traduite par l'instruction LP80:

DTERM1 := TRUE

DTERM désigne un octet et DTERM1 est la variable boolèenne déclarée en LP80 comme synonyme de DTERM: voir traduction de l'instruction SET dans la IIème partie).

Soient

VALOO = (RIB1,...

CM2 = (BOO1,PG1)

les règles

```
RIB1  :  CD := CD(G) ;          CM := CM(D)
PG1   :  CD := CD(G) ;          CM := CM(G)
BOO1  :  CD := CD(G)||CD(D) ; CM := CM(G) ; VAL := (PST1,PST2) -
PST1  :  CD := CD(G)||" := TRUE;" ; FIM .
PST2  : CD  := CD(G)||" := FALSE" ; FIM.
```

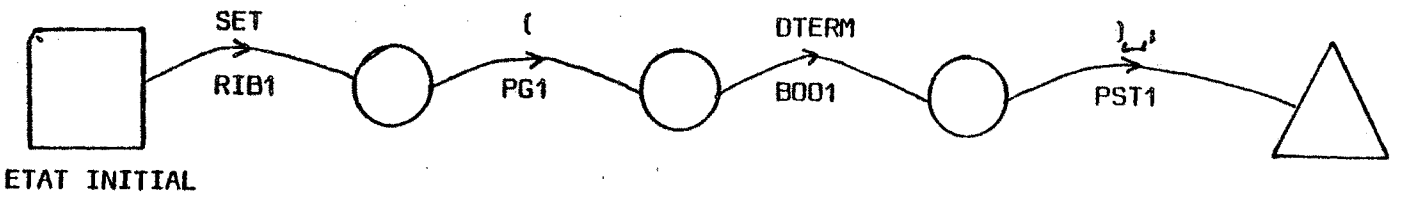
les modèles

```
/SET/       : REG := (RIB1) ; CM := 2 ; SAT := (PST2)
/RESET/     : REG := (RIB1) ; CM := 2 ; SAT := (PST1)
/(/         : REG := (PG1,...) ; CD := DICT ...
/)␣;/       : REG := (PST1,PST2,...) ; CD := DICT ; ...
/BOOLEEN/   : REG := (BOO1,...) ; CD := SYNO ; ...
```

le dictionnaire

```
/SET/SET/
/RESET/RESET/
/(/(/
/)␣;/)␣;/
/DTERM(S)/BOOLEEN/DTERM1/
```

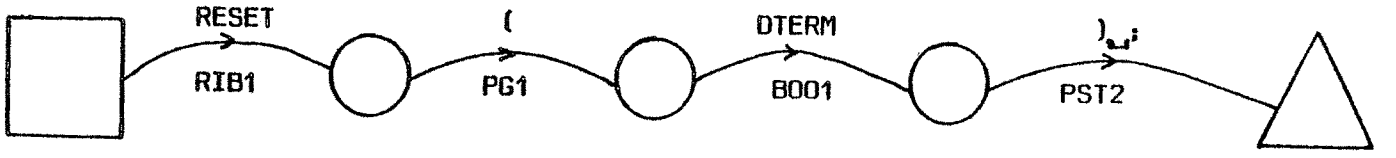
DTERM1 est la variable booléenne synonyme de l'octet DTERM.



ETAT INITIAL

CHAINE DE TRADUCTION

DTERM1 :=TRUE ;



ETAT INITIAL

CHAINE DE TRADUCTION

DTERM1 :=FALSE ;

- Par identification avec la chaîne d'entrée et les éléments du dictionnaire, on détermine "SET" dont le modèle est SET.

Comme VALOO = (RIB1,...) et $REG_{/SET/}$=(RIB1), on sélectionne la règle RIB1 (par VALOO ∩ REG).

Cette règle impose:

CD := CD(G); A l'état initial, la chaîne de traduction est vide. D'autre part l'opération indique la transmission du CD de l'état précédent à l'état courant.

CM := CM(D) Le code morphologique défini par SET est CM2 := (PG1,BOO1).

Le calcul des validations et des saturations fournit dans cet état:

VAL := ( ) et SAT := (PST2)

- Après avoir analysé SET, la phase d'identification permet d'obtenir "(" dont le modèle est (

L'intersection des validations de l'état précédent et des règles est vide. On procède alors à l'intersection du code morphologique CM2 = (BOO1,PG1) et de $REG_{/(/}$ = (PG1) qui permet de sélectionner PG1. On a alors:

CD := CD(G); La chaîne de traduction est toujours vide.

CM := CM(G) Le code morphologique de l'état précédent est transmis à l'état courant donc CM2.

Le calcul des validations et saturations fournit VAL := ( ) et SAT := (PST2).

- On identifie le segment suivant |DTERM| qui est indexé suivant le modèle BOOLEEN.

L'intersection des validations de l'état précédent avec les règles du modèle BOOLEEN permet de sélectionner la règle BOO1 qui impose:

CD := CD(G)||CD(D) On concatène à la chaîne de traduction de l'état précédent (ici, c'est la chaîne vide), la chaîne obtenue par accès au modèle. L'opération sur le CD dans le modèle faisant référence au chaînage synonyme de l'élément dans le dictionnaire (par CD := SYNO) la chaîne de traduction sera "DTERM1".

CM := CM(G) Le code morphologique est toujours CM2.

Le calcul des validations et des saturations fournit:

SAT_{I+1} = (PST2)

et $VAL_{I+1} = (VAL_M \cup VAL_R) \cap \overline{SAT}_{I+1} = (PST1,PST2) \cap \overline{PST2} = PST1$

car VAL_M = (PST1,PST2) et VAL_R = ( ).

. On identifie ensuite le segment ")␣;" dont le modèle est)␣;
L'intersection des validations de l'état précédent et des règles du modèle /)u;/ fournit la règle PST1 qui impose:

CD := CD(G)||:= TRUE ;	Concaténation de la chaîne courante avec la valeur ":=TRUE;" La chaîne de traduction devient DTERM1 := TRUE ;
FIM	Indicateur marquant l'état final. Il n'y a pas de règles permettant le retour-arrière, donc la traduction de l'instruction est terminée.

Remarque:

L'instruction RESET(DTERM) ; en PL360 est traduite par l'instruction DTERM1 := FALSE ; en LP80. L'analyse est commune à celle de l'instruction SET, jusqu'au modèle)␣;
Pour l'instruction RESET, c'est la règle PST2 qui est appliquée et qui permet d'obtenir la chaîne ":=FALSE;". Pour cela, PST1 (et non PST2) est saturée dans le modèle RESET.

2.2.5. Utilité du retour-arrière

On peut se demander si le retour-arrière avec conservation des résultats est indispensable et quels seraient les moyens de l'éviter.

Reprenons l'exemple précédent de l'instruction IC (R5,WTAB) ; et plus généralement considérons l'instruction IC en PL360. D'après les options choisies pour la traduction de PL360 en LP80 (voir IIième partie) cette instructions peut être traduite en LP80, en cinq instructions différentes suivant le type de l'opérande.

Le tableau ci-dessous résume ces différents cas:

TYPE DE L'OPERANDE	INSTRUCTION PL360	INSTRUCTION LP80 1er chemin	INSTRUCTION LP80 2° chemin
(1) MOT	IC(R5,WTAB) ;	R2:=@WTAB SLLS 2 ;	R5:=BYTEO(R2) ;
(2) DEMI-MOT	IC(R5,STAB) ;	R2:=@STAB SLLS 1 ;	R5:=BYTEO(R2) ;
(3) Bi ou MEM(Ri)	IC(R5,B7);	R2:=R7 ;	R5:=BYTEO(R2) ;
(4) OCTET	IC(R5,BTAB) ;		R5:=BTAB ;
(5) op IMMEDIAT	IC(R5,"␣") ;		R5:="␣" ;

Le retour-arrière avec conservation des résultats dans le traitement de l'instruction IC possède les deux avantages suivants:

- recherche du contexte pour les instructions de type (1),(2),(3) ;
- séparation de l'analyse en deux traitements.

Le retour-arrière utilisé pour la recherche du contexte peut effectivement être évité. Pour cela, il suffit d'utiliser le type CL (cf. chapitre 1) pour conserver le contexte (ici le registre R5) et le transmettre d'état en état, jusqu'à l'état final. Il ne reste plus qu'à écrire la deuxième instruction qui dépend de l'instruction IC, à l'aide du contexte transmis par le CL, ce qui permet d'obtenir l'instruction R5 := BYTEO(R2);

Le retour-arrière utilisé pour séparer l'analyse en deux traitements peut être évité, moyennant la complication de l'écriture des règles de la grammaire et l'augmentation de leur nombre.

En effet, cette méthode permet:

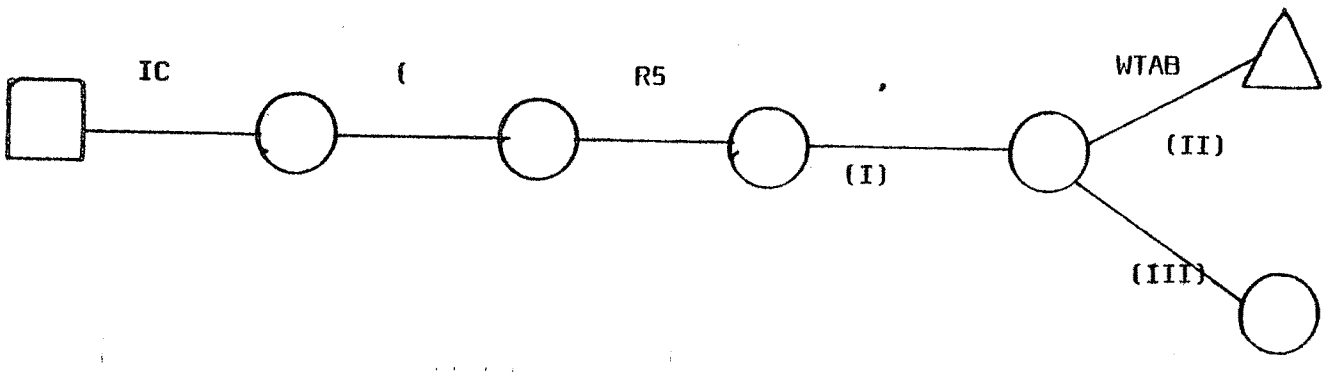
1/ La mise en facteur des règles au niveau de l'analyse

Soient les instructions PL360 suivantes et leur traduction correspondante:

INSTRUCTION PL360	INSTRUCTION LP80
(1) IC(R5,WTAB) ;	R2:=@WTAB SLLS 2 ; R5:=BYTE0(R2);
(2) LH(R5,WTAB) ;	R2:=@WTAB SLLS 1 ; R5:=SHORT0(R2) ;
(3) R5:=@WTAB ;	R5:=@WTAB SLLS 2 ;

On remarque donc, que si l'on applique à l'opérande WTAB de (1) une règle qui fournit la chaîne "R2:=@WTAB SLLS 2" relative à cet opérande, cette règle ne pourra pas être utilisée par les instructions (2) et (3). Il parait plus intéressant d'appliquer trois règles différentes dont certaines pourront être utilisées dans l'analyse des autres instructions.

Soient les règles (I), (II), (III) permettant l'analyse de WTAB dans l'instruction (1).

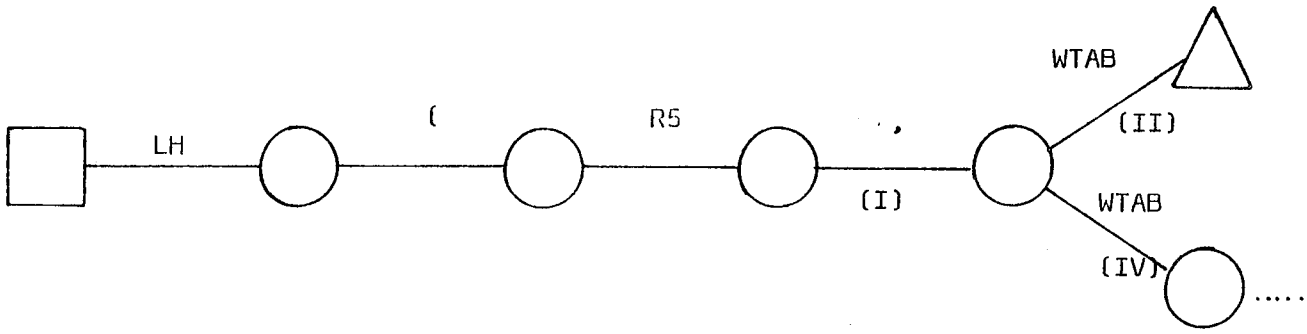


(I) fournit la chaîne de traduction "R2:=@"

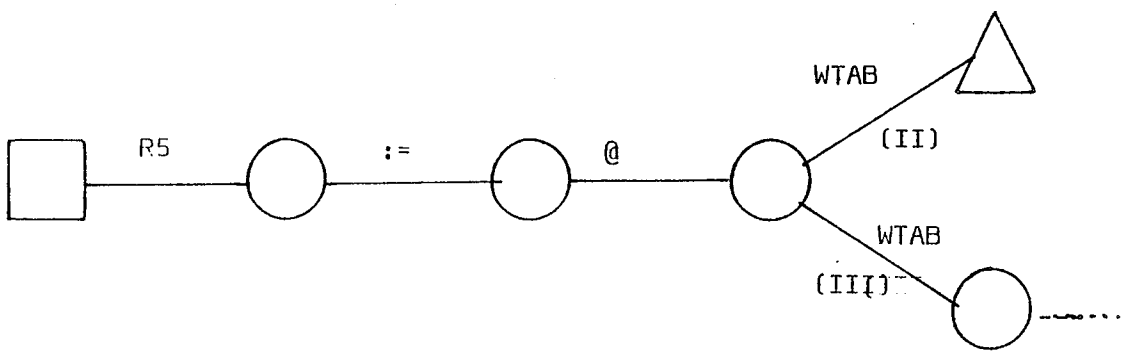
(II) fournit la chaîne de traduction "WTAB"

(III) fournit la chaîne de traduction "SLLS 2"

Pour les autres instructions, on a les schémas suivants:



où (IV) fournit la chaîne de traduction "SLLS 1"



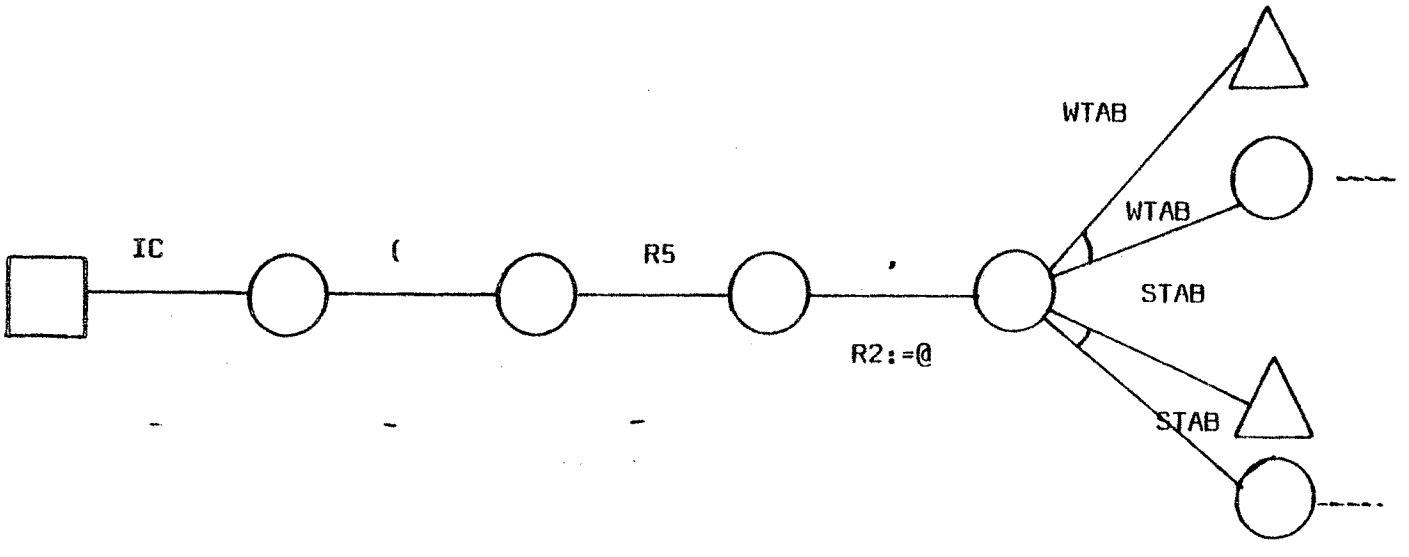
2/ La mise en facteur des règles au niveau de la transduction des résultats

Soit l'instruction IC et sa tradd tion correspondante dans les cinq cas précédemment cités.

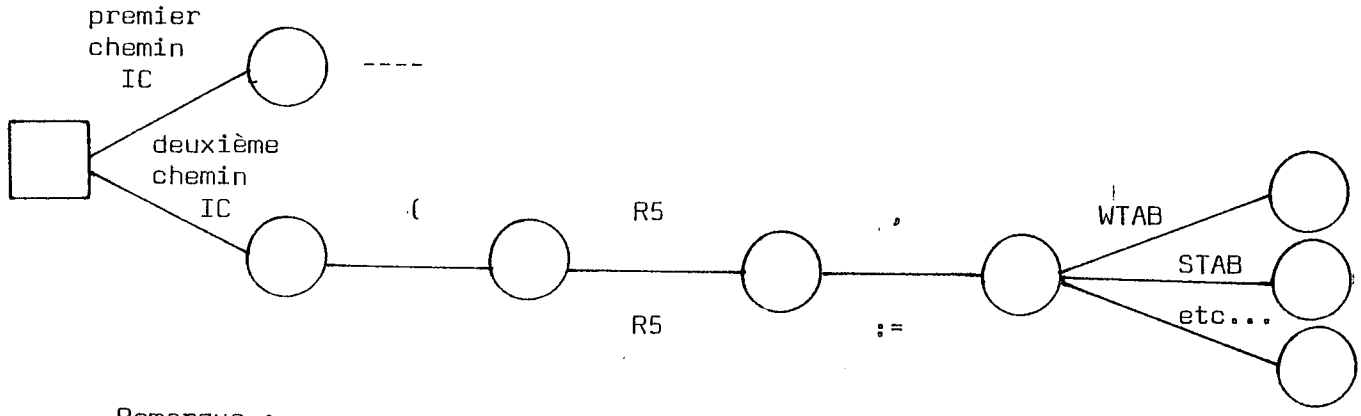
On constate que la première instruction LP80 contient une partie commune lorsque le deuxième opérande est de type mot ou;demi-mot:

```
R2 := @WTAB SLLS 2 ;
R2 := @STAB SLLS 2 ;
```

Donc la chaîne de traduction obtenue jusqu'à l'analyse de l'opérande est identique pour les deux instructions, ce qui fournit le schéma suivant:

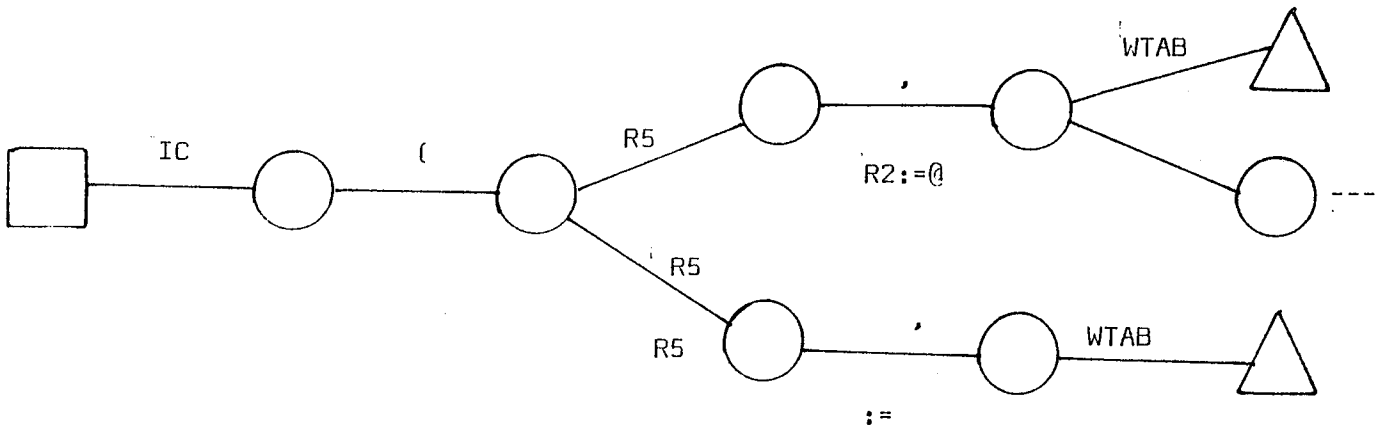


De même, dans les instructions LP80 obtenues par le deuxième chemin de l'instruction IC, on peut mettre en facteur la chaîne de traduction "R5:=".



Remarque :

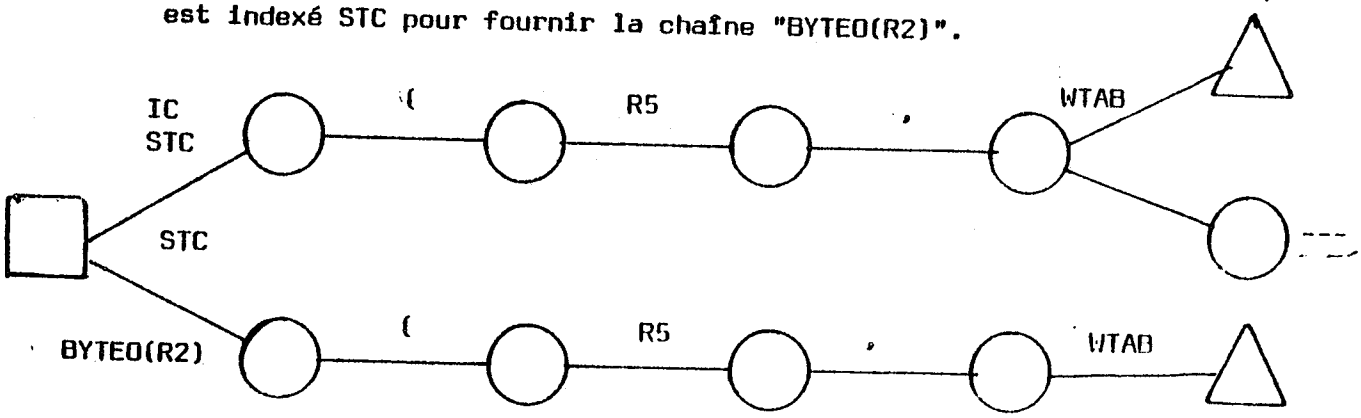
Cette mise en facteur peut également être faite au cours du premier chemin, à condition de provoquer le retour-arrière au niveau du registre R5, pour pouvoir traduire la séquence d'instructions.



On a préféré provoquer le retour-arrière au niveau du modèle IC pour deux raisons:

- l'indexation des quinze registres prédéclarés en PL360 sur deux modèles fournit quinze éléments supplémentaires dans le dictionnaire, tandis que l'indexation de l'identificateur IC ne fournit qu'un élément supplémentaire.

- l'instruction PL360,@STC(R5,WTAB); est traduite en LP80 par la séquence: R2:=@WTAB SLLS 2 ; BYTE0(R2):=R5;
Le premier chemin de IC est commun au premier chemin de STC et fournit la première instruction LP80. On utilise le deuxième modèle sur lequel est indexé STC pour fournir la chaîne "BYTE0(R2)".



Le retour-arrière au niveau du registre ne peut pas fournir "BYTE0(R2)". Donc l'identificateur STC doit être indexé pour au moins deux modèles et par souci de cohérence, on agit de même pour l'identificateur IC.

La solution consistant à utiliser la technique de retour-arrière avec conservation des résultats possède donc le double avantage de minimiser le nombre des règles de la grammaire et de simplifier l'écriture de celles-ci.

III - FONCTIONNEMENT DU TRADUCTEUR

1 - Introduction

Tout programme écrit dans un langage PL360 (ou plus généralement dans tout langage de type "PL") est constitué de déclarations et d'instructions.

Le traitement des déclarations consiste à indexer les données ainsi déclarées et à traduire celles-ci dans le langage cible , tandis que le traitement des instructions consiste seulement à traduire celles-ci dans le langage LP80. Aussi avons-nous choisi de séparer ces deux traitements et d'appliquer un module particulier du traducteur aux déclarations.

D'autre part, bien que la grammaire d'un langage de programmation soit bien définie (par rapport aux langues naturelles), il est difficile de réaliser un modèle rigoureux de traduction. Nous avons largement utilisé l'interactivité du système PIAF qui permet de laisser le contrôle à l'utilisateur et d'apporter les renseignements nécessaires au processus de traduction. La traduction des instructions se déroule en deux étapes:

- contrôle et modification éventuelle des instructions PL360 du programme source ;
- traduction de ces instructions en instructions LP80 qui constituent le programme cible.

Ces options ont été choisies pour faciliter la traduction mais elles ne sont pas impératives et d'autres perspectives sont données dans la suite (voir Troisième Partie, chapitre II).

Chaque phase utilise le transducteur général d'états finis. Au cours de la première phase, du traitement, les instructions PL360 constituant le fichier source sont analysées par le transducteur et transférées, avec ou sans modification, dans le fichier intermédiaire. Ce premier passage, permettant d'obtenir des instructions adéquates à la grammaire de traduction, constitue le module "d'adéquation" du traducteur.

Au cours de la deuxième phase du traitement, les instructions du fichier intermédiaire sont analysées et traduites en instructions LP80 rangées dans le fichier cible. Ce deuxième passage du transducteur constitue le module de traduction.

Le traducteur est donc constitué de trois modules:

- le module d'indexation des déclarations,
- le module d'adéquation des instructions,
- le module de traduction des instructions.

L'utilisateur fournit les paramètres nécessaires à l'enchaînement séquentiel de ces trois modules indépendants.

Dans ce chapitre, nous décrivons successivement ces trois modules.

2 - TRAITEMENT DES DECLARATIONS

Après avoir présenté le problème, nous indiquons la forme des déclarations et le fonctionnement du module d'indexation.

2.1. - Présentation du problème

Les déclarations servent à préciser certaines propriétés des identificateurs utilisés dans un programme:

1/ leur nature: variables (simple, tableau), constantes, étiquettes, procédures, échanges d'entrées-sorties,

2/ leur type : octet, mot, double-mot, entier, réel.

Les déclarations ont lieu uniquement en-tête de bloc. La portée d'un identificateur est le bloc où il a été déclaré.

A l'intérieur d'un même bloc, il n'y a pas d'ambiguité pour les déclarations, contrairement aux langages de type ALGOL. Mais par contre, deux identificateurs de même nom et de type différent peuvent coexister dans des procédures de même niveau.

La décomposition d'un problème en traitements, caractérisés par leurs données et leurs instructions, conduit à répartir les données en:

- données communes à plusieurs traitements, constituant des blocs externes à ces traitements ;
- données locales à chaque traitement, accessibles uniquement à l'intérieur de ce traitement.

D'autre part, nous avons vu que le dictionnaire doit contenir:

- les symboles de base du langage,
- les valeurs,
- les identificateurs ou déclarations rencontrées dans le programme à traduire.

Les symboles de base sont spécifiques à la traduction PL360-LP80 tandis que les valeurs et les identificateurs sont propres à chaque programme PL360 à traduire, sauf les identificateurs déclarés dans des blocs de données communes.

Ces considérations obligent à avoir deux dictionnaires:

- un dictionnaire de base, contenant les symboles de base du langage et les données communes à plusieurs traitements, lorsqu'elles existent.

- un dictionnaire de traduction, spécifique à chaque programme, construit à partir du dictionnaire de base, en ajoutant les données locales à ce programme. Celui-ci n'est plus utilisé, une fois que le programme a été traduit.

Deux traitements doivent donc s'appliquer aux déclarations d'un programme PL360:

- indexation de ces données dans le dictionnaire pour former le dictionnaire de traduction,
- traduction dans le langage LP80.

Les déclarations fournies par l'utilisateur sont indexées automatiquement par un module particulier du traducteur. En revanche, la traduction des déclarations en LP80, ne peut pas se faire de manière automatique, car l'implantation des données est différente sur chaque machine (voir chapitre Adressage et implantation des données). L'utilisateur traduit lui-même les déclarations PL360 en LP80.

L'indexation automatique des déclarations présente les deux avantages suivants:

- l'utilisateur n'a pas besoin de connaître la grammaire et le dictionnaire. Il contrôle cependant les déclarations puisqu'il les fournit au module d'indexation.

- l'introduction des identificateurs est très rapide. En effet, on a choisi d'indexer dans le dictionnaire , les types de déclaration (octet-mot-demi-mot), suivant plusieurs bases, permettant d'établir une certaine hiérarchie dans les règles à appliquer (voir chapitre: Adaptation du système PIAF). L'indexation manuelle serait une opération fastidieuse.

2- 2 - Description du module

2 -2.1. Types de déclarations:

Les identificateurs sont indexés, dans le dictionnaire, en fonction seulement de leurs types (octet, mot, demi-mot, double-mot). La nature d'un identificateur (tableau ou variable simple) n'intervient que dans la réservation de place en mémoire. D'autre part, les procédures et étiquettes n'étant pas déclarées préalablement, elles ne sont donc pas traitées par le module d'indexation.

On distingue deux sortes de déclarations, du point de vue de l'indexation:

a/ Les déclarations de type simple (octet, mot, double-mot, etc...)
Ces types existent dans les deux langages PL360 et LP80. [WISU,PEQU]
Du fait de la distinction entre adresses de mot, de demi-mot et d'octets, sur IRIS 80, la traduction d'un identificateur en LP80, nécessite dans la plupart des cas, plusieurs chaînes de traduction.

Exemple: L'instruction PL360,(1): IC(R5,WTAB) ; est traduite par la séquence d'instructions

R2 := @WTAB SLLS 2 ; R5 := BYTE0(R2 ;

ceci, parce que l'opérande WTAB est de type mot et l'instruction IC de type octet, ce qui oblige à convertir l'adresse-mot de WTAB en adresse d'octet (cf. Deuxième Partie). Les deux chaînes de traduction "WTAB" et "SLLS 2" sont spécifiques au type de l'opérande WTAB.
En revanche l'instruction (2): R5 := WTAB ; est traduite en LP80 par la même instruction (2). Ici, seule la chaîne de traduction "WTAB" est spécifique à l'opérande.

Afin de faire correspondre ces chaînes au type de l'opérande, on indexe le type "mot" sur plusieurs modèles, permettant d'établir un ordre parmi les règles à appliquer.
En effet l'ordre est important dans (1) puisque il faut obtenir "WTAB" "SLLS 2" et non "SLLS 2" "WTAB".

On a défini des modèles particuliers auxquels se réfèrent les types différents (mot, demi-mot, octet, double-mot, etc...) définis dans le langage PL360. Ceux-ci ont été indexés, une fois pour toutes, dans le dictionnaire de base, lorsque celui-ci et la grammaire ont été définis pour la traduction.

Exemple: Indexation du type LOGICAL (mot en PL360) dans le dictionnaire.

```
        /LOGICAL (S) / LOGINT / WORD /
1 -     / WORD / BIDON /
2 -     / WORD / WARRAY1 /
3 -     / WORD (S) / WARRAY2 / SLLS 2 /
4 -     / WORD / WARRAY3 /
```

On a ajouté un niveau intermédiaire (/WORD/BIDON/) pour avoir un algorithme d'indexation plus cohérent, la recherche des modèles s'effectuant sur les chaînages FILS de l'élément intermédiaire.

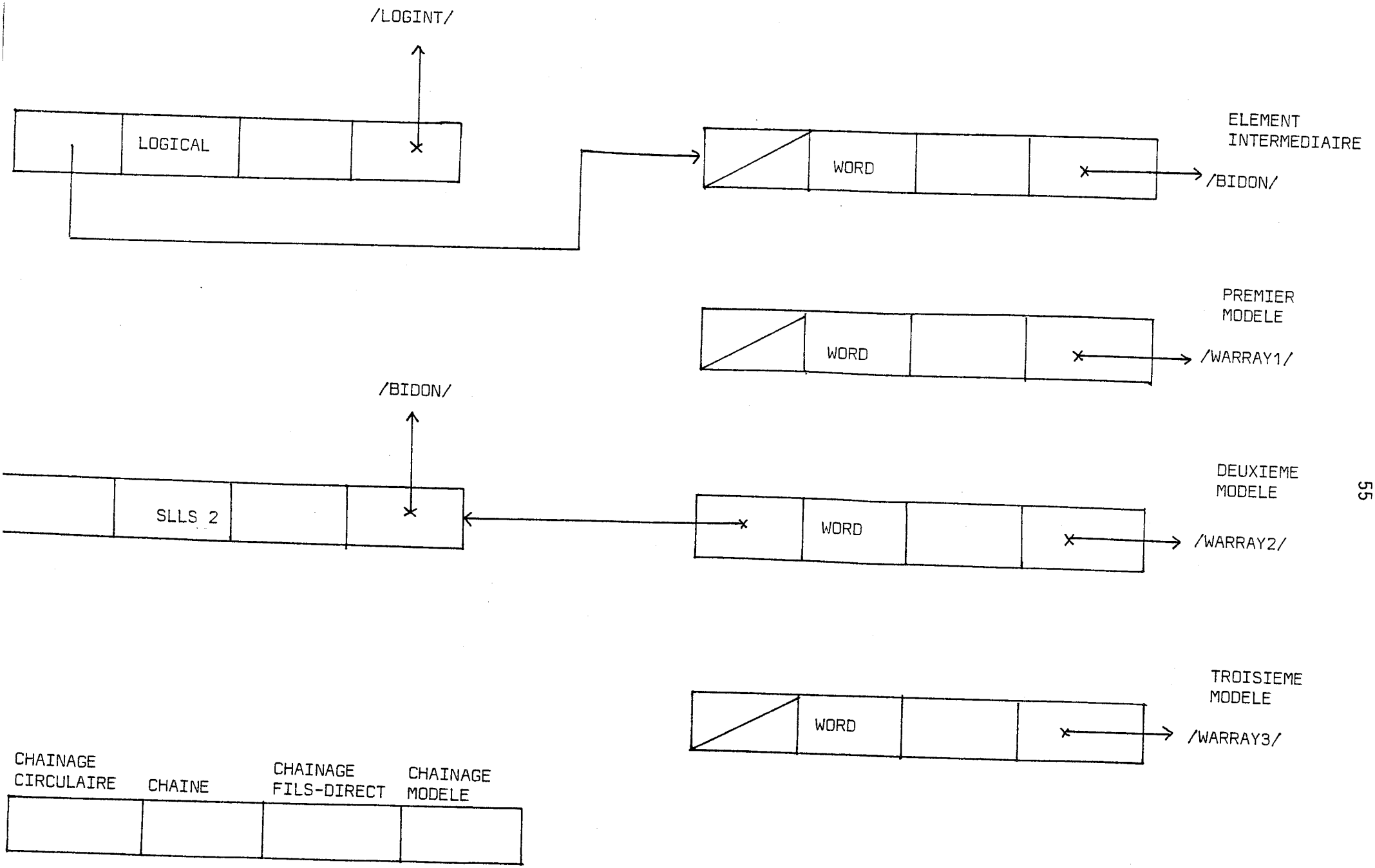
Représentons un élément du dictionnaire, par les quatre informations:

- chaînage circulaire de l'élément,
- chaîne de caractères constituant l'élément,
- chaînage -FILS-DIRECT de l'élément,
- un pointeur sur le modèle auquel se réfère l'élément dans la grammaire.

Le traitement de la déclaration LOGICAL WTAB ; provoque l'indexation de l'identificateur WTAB dans le dictionnaire sur les trois modèles définis pour le type mot (ceux-ci, ainsi que les attributs correspondants, sont indiqués par la liste des fils-direct de l'élément intermédiaire).

```
/ WTAB / WARRAY1 /
/ WTAB(S) / WARRAY2 / SLLS 2 /
/ WTAB / WARRAY3 /
```

Remarque: La présence des trois modèles peut se justifier de la manière suivante les modèles sont définis dans la grammaire de la façon suivante:


/LOGINT/
LOGICAL
WORD
ELEMENT
INTERMEDIAIRE
/BIDON/
WORD
PREMIER
MODELE
/WARRAY1/
/BIDON/
SLLS 2
WORD
DEUXIEME
MODELE
/WARRAY2/
WORD
TROISIEME
MODELE
/WARRAY3/
CHAINAGE
CIRCULAIRE
CHAINE
CHAINAGE
FILS-DIRECT
CHAINAGE
MODELE

```
/ WARRAY1 /:  REG := (--) ; CD := DICT ; CM := 31 ;
/ WARRAY2 /:  REG := (--) ; CD := SYNØ ; CD := 49
/ WARRAY3//:  REG := (--) ; CD := DICT ; CM := 5
```

Le premier modèle fait référence à la chaîne constituant l'élément (c'est-à-dire WTAB), tandis que le deuxième modèle fait référence à l'élément obtenu par chaînage circulaire (c'est-à-dire "SLLS 2").
Le troisième modèle permet de définir (par le CM) un ensemble de règles applicables différent des ensembles définis par les deux autres modèles. Il est surtout utilisé dans les instructions d'affectation d'une base à une mémoire.

b/ Les déclarations de type synonyme

On désigne ainsi les déclarations intervenant dans des structures de données comme synonymes d'éléments de cette structure.

Bien que ces déclarations possèdent le même type que les déclarations précédemment traitées (mot, demi-mot, octet), elles constituent néanmoins une classe distincte, pour les raisons d'implantation de données. En effet, toute adresse d'octets peut être nommée sur IBM360, tandis que sur IRIS 80 on ne peut désigner que des frontières de mot.

Exemple:
Cet exemple met en parallèle la description d'une structure de données déclarée en PL 360, et sa traduction dans le langage LP80.

Seules les entités situées à une adresse-mot (c'est-à-dire multiple de 4, donc ADINF, ADEGAL, RARB) apparaissent explicitement dans la traduction LP80 de cette structure de données. Les autres entités sont considérées comme éléments d'un tableau d'octets ou de demi-mots ayant la même adresse que le tableau décrivant cette structure (ici le tableau MOTARB). Ceci revient donc à déclarer un octet et un demi-mot synonymes du tableau MOTARB et à se référer ensuite à un élément de ce tableau.

Dans cette perspective, les éléments ADSUP, LFOR, FORME n'apparaissant pas dans la traduction de la structure, sont traduits respectivement par

SMOTARB(R2 := 1), BMOTARB(R2 := 9) et BMOTARB(R2 := 10).

Remarque:

On aurait pu traduire ADSUP par ADINF(R2 := 1), puisque ADINF est, comme SMOTARB, synonyme de MOTARB. On a préféré garder la cohérence de la structure.

Le demi-mot ADSUP, et les octets LFOR et FORME, sont des éléments de déclaration de type synonyme: ils ont la même fonction que les éléments de type simple correspondants (octet, demi-mot) dans une instruction, mais sont traduits différemment. L'élément de déclaration de type synonyme est mis en relation avec sa traduction par l'intermédiaire du chaînage circulaire.

C'est l'utilisateur qui indique la chaîne de traduction correspondant à un élément de type synonyme, lors de l'indexation de celui-ci. Pour distinguer ces éléments des éléments de type simple, il désigne les octets de type synonyme par le type "BYTS" et les demi-mots par le type "SHTS".

Exemple:

Déclaration faite par l'utilisateur pour indexer l'octet LFOR et le demi-mot ADSUP de type synonyme, à l'aide du module d'indexation.

```
BYTS  LFOR = BMOTARB(R2 := 9) ;
SHTS  ADSUP= SMOTARB(R2 := 1) ;
```

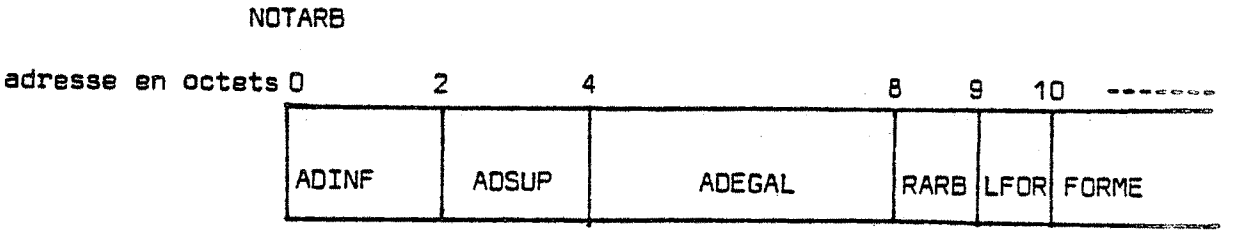
La syntaxe de ces déclarations particulières est décrite dans le paragraphe suivant.

Les types BYTS et SHTS sont indexés dans le dictionnaire de la même manière que les types simples (présence d'un élément intermédiaire et liste des modèles indiquée par la liste des FILS-DIRECT). Lorsqu'un des modèles fait référence à cet élément, il faut un moyen d'indiquer qu'il s'agit d'un élément de type synonyme qui est traduit par la chaîne de caractères indiquée dans la déclaration. Pour cela, on positionne un indicateur (D) dans l'élément du dictionnaire se référant à ce modèle.

STRUCTURE DE DONNEES PL 360

```
ARRAY 10 LOGICAL MOTARB ;
      SHORT INTEGER ADINF SYN MOTARB ,
                    ADSUP SYN MOTARB(2) ;
      LOGICAL ADEGAL SYN MOTARB(4) ;
           BYTE RARB SYN MOTARB(8) ,
                LFOR SYN MOTARB(9) ,
                FORME SYN MOTARB(10) ,
```

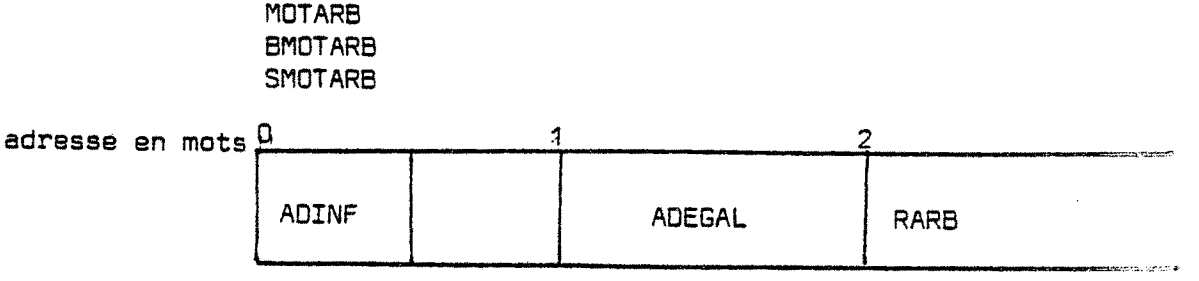
REPRESENTATION EN MEMOIRE SUR IBM 360



TRADUCTION DE CETTE STRUCTURE EN LP80

```
ARRAY 10 WORD MOTARB ;
BYTE BMOTARB SYN MOTARB ;
SHORT SMOTARB SYN MOTARB ;
SHORT ADINF SYN MOTARB ;
WORD ADEGAL SYN MOTARB(1) ;
BYTE RARB   SYN MOTARB(2) ,
```

REPRESENTATION EN MEMOIRE SUR IRIS 80



Exemple: Indexation de LFOR dans le dictionnaire.

Le type BYTS est indexé dans le dictionnaire de la manière suivante:

/ BYTS(S) / BYTE-SYNONYME / BYTE1 /
/ BYTE1 / BIDON /
/ BYTE1(D) /BSYN1 /
/ BYTE1(D) / BSYN2 /
/ BYTE1(S) / BARRAY3 / SRLS 2 /

Les modèles / BSYN1 / , / BSYN2 / , / BARRAY3 / sont définis dans la grammaire par:

/ BSYN1 / : ... CD := SYNO ; ...
/ BSYN2 / : ... CD := SYNO ; ...
/ BARRAY3 / : ... CD := SYNO ; ...

L'indicateur D indique qu'il faut créer dans le dictionnaire un élément représentant la chaîne indiquée dans la déclaration (cet élément sera indexé sur le modèle BIDON) et qu'il faudra mettre celui-ci comme chaînage circulaire. Cet indicateur a été mis une fois pour toutes, lors de la définition des modèles auxquels se réfèrent les types synonymes et n'est utilisé nulle part ailleurs. Il sert uniquement de test des déclarations synonymes et n'est par conséquent, jamais manipulé par l'utilisateur.
LFOR sera indexé ainsi:

/ BMOTARB(R2 := 9) / BIDON /
/ LFOR(S) / BSYN1 / BMOTARB(R2 := 9) /
/ LFOR(S) / BSYN2 / BMOTARB(R2 := 9) /
/ LFOR(S) / BARRAY3 / SRLS 2/

Lorsqu'on fera référence au CD du modèle / BSYN1 / (ou / BSYN2 /) c'est BMOTARB(R2 := 9) qui constituera la chaîne de traduction.

Remarque: Les modèles sur lesquels est indexé un "byte" (type simple) sont constitués de la même façon que les modèles sur lesquels est indexé un

"byts" (type synonyme) sauf l'opération sur le CD. Pour un modèle de byte, si le CD fait référence à l'élément lui-même, le CD du modèle correspondant de type synonyme fera référence au chaînage circulaire.

2.2.2. Syntaxe des déclarations

<déclaration> ::=<type simple> <liste de déclarations simples> ;|
<type synonyme> <liste de déclarations synonymes> ;
<liste de déclarations simples> ::= <identificateurs> |
<identificateur> <liste de déclarations simples>
<liste de déclarations synonymes> ::= <élément de déclaration synonyme> |
<élément de déclaration synonyme> <liste de déclarations synonymes>

<élément de déclaration synonyme> ::= <identificateur> = <synonyme>(R2:=<nombre>)
<synonyme> ::= <identificateur>
<type simple> ::= <logical>|<short integer>|<byte>|<long real>|<equate>
<type synonyme> ::= <byts>|<shts>

2.2.3. Algorithme de traitement des déclarations

Le principe de l'algorithme est le suivant:

- recherche de l'élément intermédiaire,
- s'il s'agit d'une donnée de type synonyme, indexation dans le dictionnaire de la traduction figurant dans la déclaration,
- parcours de la liste des fils-direct de même longueur que cet élément et indexation de la donnée suivant chaque modèle compte tenu de ses attributs.

Tous les éléments dominants sont implantés dans le dictionnaire de la manière suivante:

$$c_i \; l_i \; b_i \; fsd_i \; r_i \; m_i \quad attribut_i$$

où c_i = chaînage circulaire sur l'élément synonyme
l_i = longueur de l'élément b_i
frd_i = pointeur sur le frère-direct de b_i
fsd_i = pointeur sur le fils-direct de b_i
r_i = indicateur de b_i
$attribut_i$ = liste de pointeurs sur les renseignements linguistiques de b_i, parmi lesquels figure le modèle.

On sépare r_i des $attribut_i$ pour la compréhension de l'algorithme.

Tous les éléments-fils sont implantés de la manière suivante:

$c_i\ l_i\ fsd_i\ r_i m_i\ attribut_i$

Cet algorithme, de même que les algorithmes exprimés dans la suite, est donné sous forme de schémas de programme [CØVØ] dans un langage similaire à un langage de programmation de haut niveau.

DEFINITION DES STRUCTURES DE DONNEES ET DES PRIMITIVES

. Soit NOEUD un élément dominant

NOEUD: c_i l_i b_i frd_i fsd_i r_i m_i $attribut_i$

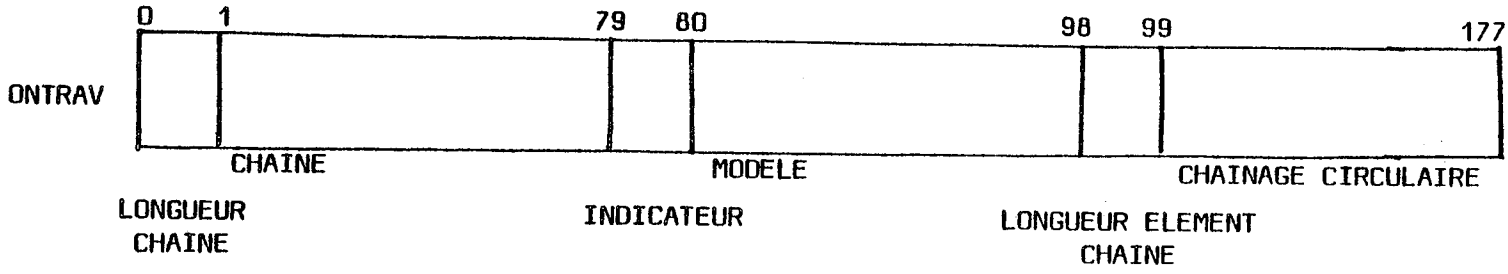
On dispose de primitives d'accès à cet élément qui sont:

CHAINAGE (NOEUD) = c_i
LONGUEUR (NOEUD) = l_i
CHAINE (NOEUD) = b_i
FRERE DIRECT (NOEUD) = frd_i
FILS DIRECT (NOEUD) = fsd_i
INDIC (NOEUD) = r_i
MODELE (NOEUD) = m_i

. Lors de l'entrée d'une commande frappée par l'utilisateur sous la forme

/ CHAINE(INDIC) / MODELE /CHAINAGE /

(où INDIC désigne soit S soit D), ces informations sont rangées dans les différents champs d'une variable structurée appelée ZONTRAV et définie de la manière suivante:



Cette variable permet de:

- vérifier la cohérence des informations (existence du modèle et du chaînage circulaire),
- construire un élément du type NOEUD décrit précédemment en effectuant la mise à jour du dictionnaire.

LIGNE désigne le buffer d'entrée-sorties dans lequel l'utilisateur transmet les déclarations au module d'indexation. Ce buffer se présente comme une file séquentielle de 177 caractères. Après l'application du transducteur sur la déclaration, on est positionné sur le caractère (blanc) précédant le premier identificateur.

Exemple:

L'utilisateur désire indexer les trois mots WTAB1, WTAB2 et WTAB3 ;
La déclaration est la suivante:

LOGICAL WTAB1, WTAB2, WTAB3 ;

L'application du transducteur donne tous les renseignements correspondants à "LOGICAL" (qui est indexé dans le dictionnaire) mais s'interrompt sur WTAB1 qui est inconnu.

La file séquentielle LIGNE se présente alors de la manière suivante:

␣WTAB1, WTAB2, WTAB3 ;

On dispose des actions suivantes sur cette file:

PRENDREC (LIGNE, CARAC) : cette action fournit dans CARAC, le caractère courant de la file.

PRENDRE (LIGNE, IDENT): cette action isole l'identificateur courant dans IDENT (les séparateurs entre les identificateurs sont les symboles "␣", ",", ";", "=").

On dispose également des prédicats suivants:

DERNIER(LIGNE) : ce prédicat prend la valeur VRAI lorsqu'on rencontre le symbole ";" dans LIGNE.

EGAL (LIGNE) : ce prédicat prend la valeur VRAI lorsqu'on rencontre le symbole "=" dans LIGNE.

On dispose des actions suivantes sur ZONTRAV:

RANGECHAINE (CHAINE): range la chaîne ainsi que sa longueur dans les champs correspondants (octets 0 à 78) de la variable ZONTRAV.

RANGINDIC (INDIC): range les indicateurs dans l'octet correspondant (79) de la variable ZONTRAV.

RANGEMOD (MODELE): range la chaîne de caractères représentant le modèle dans le champ correspondant (octets 80 à 97) de la variable ZONTRAV.

RANGETRAD (CHAINE): range la chaîne caractérisant le chaînage circulaire ainsi que sa longueur dans les champs correspondants (octets 98 à 177) de la variable ZONTRAV.

On dispose également des primitives suivantes:

ECRIRE ("MESSAGE"): écriture de la chaîne MESSAGE.

EXTENS (VECTEUR, COMPTEUR): réalise l'indexation dans le dictionnaire de l'élément décrit dans VECTEUR.
COMPTEUR désigne le numéro du modèle lorsque l'élément doit être indexé sur plusieurs modèles.

RAZ (VECTEUR, INDICE): remise à zéro du tableau VECTEUR à partir de INDICE.

On dispose aussi du prédicat:

RECHSYNONYME (ADRELEM, NOEUD): prend la valeur VRAI lorsqu'on trouve le dernier élément du chaînage circulaire de l'élément dont l'adresse est ADRELEM. L'adresse de cet élément est rangée dans NOEUD.

```
ACTION  DECLARATION (ENTREES : TYPE,LIGNE)
  SI  RECHSYNONYME (TYPE,NOEUD1)
  ALORS CØ : NOEUD1 = POINTEUR SUR L'ELEMENT INTERMEDIAIRE ;
     PRENDEC (LIGNE,CARAC) ;
     TANTQUE NON DERNIER (LIGNE)
     FAIRE
        CØ : SAISIE DE L'IDENTIFICATEUR DANS LIGNE ;
        PRENDRE (LIGNE,IDENT) ;
        PRENDREC(LIGNE,CARC) ;
      SI  EGAL(LIGNE)
      ALORS CØ: ELEMENT DE DECLARATION SYNONYME,INDEXATION DE LA CHAINE DE
                TRADUCTION ;
          PRENDRE(LIGNE,TRAD) ;
          RANGECHAINE (TRAD) ;
          RANGEMOD (MODELE(NOEUD1))
          EXTENS (ZONTRAV,1) ;
          RAZ (ZONTRAV,0) ;
      FINSI
```

```
        CPTBASE ← 1
        NOEUD2 ← FILS DIRECT(NOEUD1)
        TANTQUE    NOEUD2¬=0
        FAIRE  CØ : PARCOURS DE LA LISTE DES FILS-DIRECT ;
          SI LONGUEUR(NOEUD1) = LONGUEUR(NOEUD2)
          ALORS
            RANGEMOD(MODELE (NOEUD2)) ;
            SI  INDIC (NOEUD2) = D
            ALORS
                RANGETRAD (TRAD)
                RANGINDIC (INDIC(NOEUD2))
            FINSI
            SI  INDIC(NOEUD2)=S ET RECHSYNONYME(NOEUD2,SYN)
            ALORS
                RANGETRAD(FORME(SYN))
                RANGINDIC(INDIC(NOEUD2))
            FINSI
           EXTENS(ZONTRAV,CPTBASE) ;
           CPTBASE ← CPTBASE+1
           RAZ(ZONTRAV,79)
           NOEUD1 ← NOEUD2
           NOEUD2 ← FILS DIRECT(NOEUD1)
          SINON
                NOEUD2 ← 0
          FINSI
        FINFAIRE
        RAZ(ZONTRAN,0)
        PRENDREC(LIGNE, CARAC) ;
    FINFAIRE
  SINON
      ECRIRE("DECLARATION INCORRECTE")
  FINSI
FINACTION
```

3 - MODULE D'ADEQUATION

Le but de ce module est la création d'un fichier intermédiaire constitué d'instructions pouvant être traduites par la grammaire de traduction PL360 - LP80.

Le principe de ce module est le suivant:
Le transducteur analyse chaque instruction qui constitue le programme source:

- si le transducteur n'est pas interrompu, l'instruction est transférée automatiquement dans le fichier intermédiaire.
- si le transducteur est interrompu, le contrôle est passé à l'utilisateur pour modifier l'instruction ou apporter les renseignements faisant défaut. Celui-ci peut choisir ensuite entre tansférer lui-même l'instruction dans le fichier intermédiaire ou demander de reprendre l'analyse de l'instruction.

L'arrêt du transducteur sur l'analyse d'une instruction, peut provenir soit:

- du dictionnaire: le mot sur lequel s'est arrêté le transducteur n'existe pas dans le dictionnaire et celui-ci est incapable de segmenter l'instruction. L'utilisateur ajoute le mot dans le dictionnaire et relance l'analyse, c'est le cas des étiquettes qui ne sont pas déclarées dans un programme PL360 et des variables booléennes qui sont distinctes des octets dans le langage LP80.
- de la grammaire qui ne peut pas confirmer la concaténation des éléments constituant l'instruction. L'utilisateur modifie l'instruction pour qu'elle devienne conforme aux paramètres linguistiques et la transfère dans le fichier intermédiaire. C'est le cas d'opérandes immédiats intervenant dans les instructions de manipulation de caractères, des bases etc... (cf. Deuxième Partie).

L'indicateur ND dont nous avons signalé l'existence dans le chapitre précédent, est utilisé dans les règles de la grammaire à la place de l'indicateur FIM, lorsqu'une instruction comporte un point d'arrêt, mais que l'analyse de cette instruction n'est pas terminée.

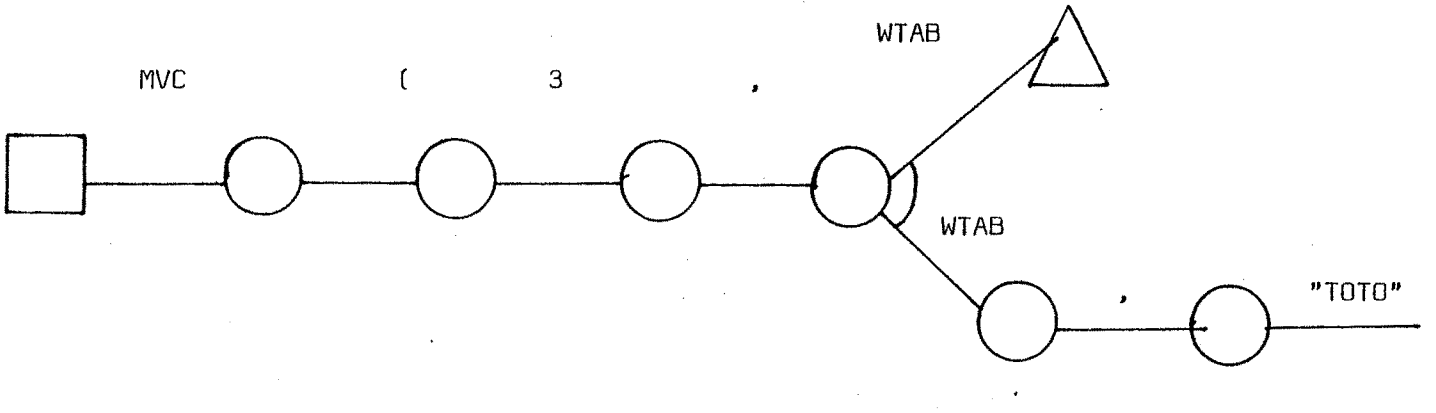
Exemple: L'instruction PL360 MVC(3,WTAB,"TOTO") se traduit en LP80 par l'instruction MOVE(R2,3, @WTAB SLL 2, @STR01) ; (cf. Deuxième Partie).

- @WTAB SLLS 2 indique que l'adresse mot de WTAB est transformée en adresses d'octets
- @STRO1 : l'instruction MOVE n'autorise pas les opérandes immédiats comme l'instruction MVC en PL360. Il faut donner l'adresse d'une chaîne d'octets déclarée de la manière suivante:

```
ARRAY 4 BYTE STRO1 = "TOTO".
```

Pour traduire "WTAB" par "@WTAB SLLS 2", on utilise la technique de retour-arrière avec conservation des résultats étudiés dans le chapitre II de la Première Partie.

Or l'analyseur ne pouvant analyser l'opérande suivant "TOTO", s'interrompt sur cet identificateur et laisse le contrôle à l'utilisateur pour qu'il remplace "TOTO" par @STRO1.



Le premier chemin conduisant à un état final ne doit pourtant pas être considéré comme correct et recopié dans le fichier intermédiaire puisque l'analyse est ensuite interrompue sur "TOTO". On utilise donc l'indicateur "ND" dont la présence indique un point d'arrêt dans une instruction mais interdit la recopie partielle de cette instruction.

Deux éditeurs permettent à l'utilisateur de résoudre ces problèmes au cours du premier passage du traducteur:

- l'éditeur lexicographique du système PIAF,
- l'éditeur d'adéquation contenant les commandes propres au module d'adéquation.

3-1 - Editeur lexicographique

On trouve dans [COUGD] le détail de toutes les commandes accessibles. L'éditeur lexicographique permet l'utilisation en mode conversationnel du transducteur général d'états finis, c'est-à-dire:

- l'analyse d'un programme en pas à pas ou continu,
- l'accès aux paramètres linguistiques en cours ou non d'exécution,
- la correction de l'instruction en cours d'exécution,
- la mise à jour du dictionnaire soit directement, soit par l'intermédiaire d'un système d'équivalences permettant l'obtention du résultat désiré,
- l'impression au terminal ou à l'imprimante, des paramètres linguistiques et de l'instruction d'entrée,
- l'accès à l'éditeur particulier de la grammaire et des modèles (voir diagramme de GRAM).

Le diagramme de l'éditeur lexicographique montre l'enchaînement des commandes. Celles-ci sont décrites dans les pages qui suivent.

Pour les états de l'éditeur, le système répond un caractère différent pour:
- matérialiser l'état dans lequel on se trouve,
- signaler à l'utilisateur qu'il est en attente d'une réponse.

3-1.1. Commandes accessibles sous l'état "$"

Les commandes accessibles sout cet état sont:

la commande DECLARATION : Cette commande permet l'accès au module d'indexation des déclarations. La syntaxe des déclarations que présente l'utilisateur a été décrite précédemment.

la commande TRADUCTION : Cette commande permet l'accès aux deux modules d'adéquation et de traduction. Une question est posée à l'utilisateur pour déterminer quel module il demande (PREMIER PASSAGE ou DEUXIEME PASSAGE). Les deux modules effectuent l'analyse d'un programme en continu.

la commande INTERRO : Cette commande effectue l'analyse et la traduction d'une instruction frappée au terminal. Elle n'est utilisée que pour la mise au point et les tests pour la traduction. Par conséquent, seul le module de traduction est utilisé, l'instruction étant supposée conforme à la grammaire de traduction.

la commande DICT : Cette commande place l'utilisateur dans l'environnement dictionnaire.

la commande GRAM : Cette commande donne accès à l'éditeur de la grammaire et des modèles.

Les commandes accessibles dans ces deux environnements sont décrites plus loin.

$FIN : Cette commande permet de sortir de l'environnement du traducteur et de se retrouver sous CMS.

3-1.2. Commandes accessibles sous l'état ">"

On se trouve dans cet état
- soit après avoir lancé la commande INTERRO,
- soit après la traduction d'une instruction, en pas à pas. Le système est en attente de définition d'une nouvelle instruction.

L'utilisateur définit une instruction délimitée par un Retour Chariot (R.C.). Si l'analyse est interrompue, on passe dans l'état "-".

Les commandes accessibles sous cet état sont:
- la définition d'une nouvelle instruction,
- la commande $FIN qui permet de revenir sous l'état "$".

3-1.3. Commandes accessibles sous l'état "-" (attente de commandes)

On se trouve dans cet état quand l'analyse n'a pu être réalisée.

L'utilisateur dispose des commandes suivantes, dans cet état:
- ? : Cette commande, accessible dans tous les états de l'éditeur, permet d'obtenir la liste de toutes les commandes disponibles dans l'état où on se trouve.

- TEXTE : Cette commande a pour effet d'imprimer le segment en cours d'analyse en précisant quel est le mot qui n'a pas produit de résultat.

- ORTHO : Cette commande n'est utilisée que pour la traduction d'instructions pas à pas. En effet, l'utilisateur peut faire des fautes de frappe, lorsqu'il définit une instruction. La commande ORTHO permet de modifier le texte en cours, en faisant passer l'utilisateur dans l'environnement modification du texte d'entrée.
  Pour un programme analysé de manière continue, on considère qu'il est correct. La commande ORTHO n'est donc pas utilisée.

- SAUV : Cette commande permet de sauver les fichiers du dictionnaire et de la grammaire, pour conserver les modifications faites au cours de l'analyse.

Les commandes SAUV, ORTHO, TEXTE, permettent de rester dans le même état "-" (attente de commandes).

- SUITE : Cette commande a pour effet de sauter la ligne d'entrée. Une ligne d'entrée peut être composée de plusieurs instructions. On se trouve alors dans l'état ">" (analyse des instructions).

- MORP : Cette commande a pour objet de rélancer l'analyse au début de l'instruction, compte tenu des indications ajoutées dans le dictionnaire ou des modifications apportées à l'instruction.

- R.C. : Cette commande a pour objet de sauter l'analyse des mots de l'instruction qui restaient à analyser et de passer à l'analyse de l'instruction suivante.

La différence entre les commandes SUITE et R.C. est que:

- R.C. saute l'instruction en cours et lance l'analyse de l'instruction suivante,
- SUITE saute les instructions constituant la ligne du fichier d'entrée et lance l'analyse de la ligne suivante.

Les commandes SUITE, MORP, R.C. font passer de l'état "-" (attente de commandes)à l'état ">" (analyse des instructions).

GRAM : Accès à l'éditeur de la grammaire et des modèles. Les commandes sont précisées dans la suite.

DICT : Les commandes sont celles utilisées pour l'extension du dictionnaire seulement, c'est-à-dire: indexation, suppression ou interrogation d'une base.

3-1.4. Commandes accessibles dans l'environnement du dictionnaire

L'appel au dictionnaire permet soit:

- son extension,
- sa réorganisation en fonction des fréquence d'utilisation des éléments,
- l'obtention d'une liste:
  - . à l'imprimante de tous les éléments du dictionnaire (LISTIMP),
  - . au terminal de tous les éléments du dictionnaire (LISTERM),
  - . à l'imprimante des éléments constituant l'arbre (ARBRE).

En ce qui concerne la réorganisation et les listes, le programme demande à l'utilisateur les paramètres et les données et prévient celui-ci en cas de traitement irréversible(effacement du dictionnaire), de traitement impossible ou de paramètre erroné.

En ce qui concerne l'extension du dictionnaire, un petit langage permet alors les extensions, suppressions, remplacements et interrogations du dictionnaire.

Extension: /CHAINE{(S)} /MODELE/ CHAINAGE/
Suppression: ¬/CHAINE/
Interrogation ou Environnement d'une base: ?/CHAINE/

3- 1.5. Commandes accessibles dans l'environnement de la grammaire

Les commandes accessibles dans cet environnement sont les suivantes:

DECLARATIONS : Cette commande fait passer l'utilisateur dans l'état "-" où il pourra définir, modifier ou lister les déclarations des variables et des types.

DIAGRAMME DE L'EDITEUR LEXICOGRAPHIQUE

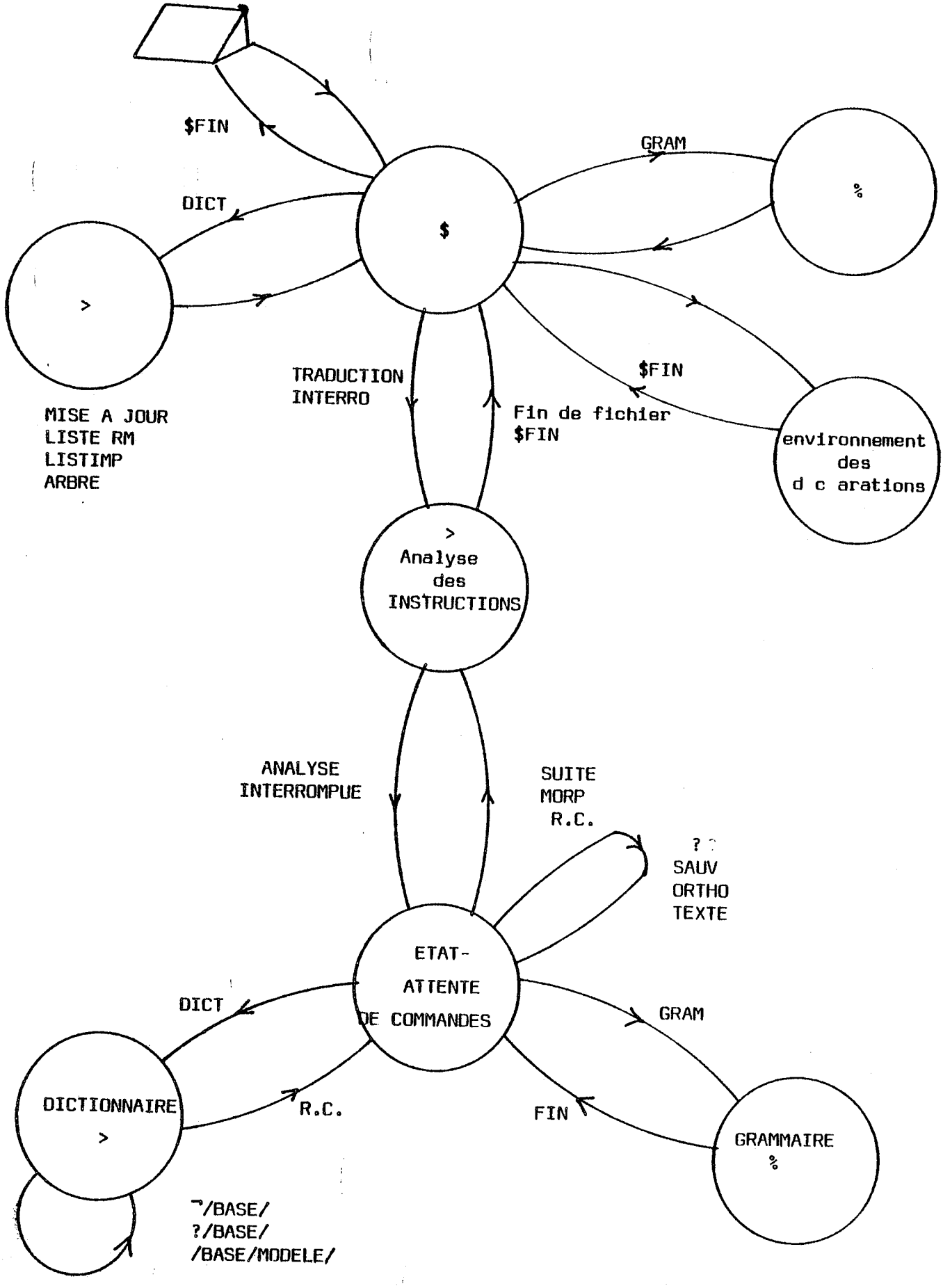
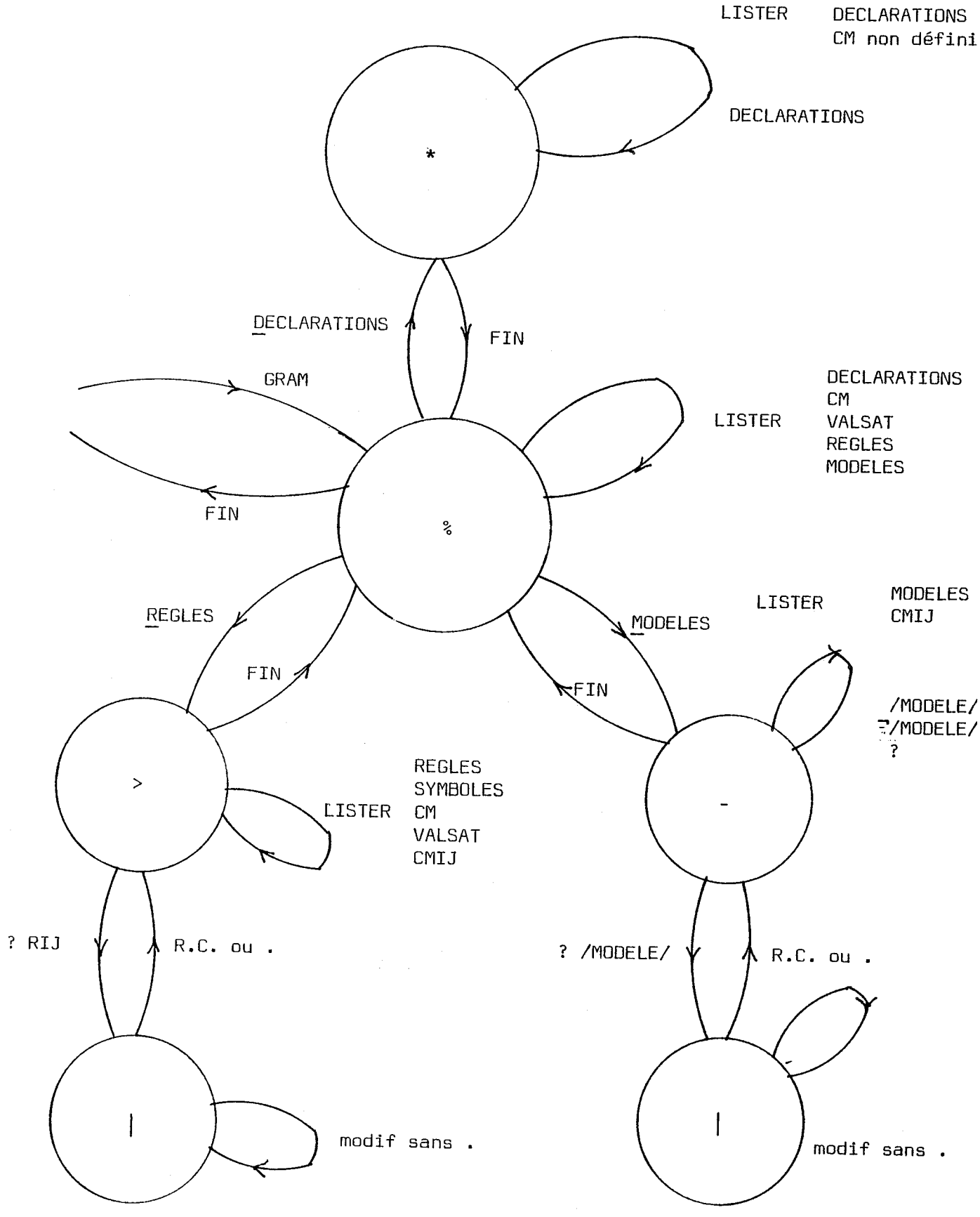


DIAGRAMME DE GRAM



REGLES : Cette commande fait passer l'utilisateur dans l'état ">" dans lequel ont été regroupées les déclarations de validations (initiales ou non), de codes morphologiques (CM), de saturations et l'écriture des règles de la grammaire.

MODELES : Cette commande place l'utilisateur dans l'état "-" qui permet la définition, la modification ou la suppression d'un modèle morphologique.

3-2 - Editeur d'adéquation

L'éditeur d'adéquation permet la création, en mode conversationnel du fichier intermédiaire de la traduction, à l'aide des opérations suivantes:

- analyse des instructions pas à pas, au cours d'un programme analysé en continu (SET, RESET).
- insertion d'une ou plusieurs instructions supplémentaires (INPUT),
- remplacement ou modification de l'instruction analysée (REPL, CHANGE),
- transfert d'une instruction du fichier source dans le fichier intermédiaire sans modification (WRITE),
- sauvegarde du fichier intermédiaire (SAVE).

Etat d'attente de commandes: ("-")
On se trouve dans cet état:

- soit par interruption du transducteur sur une instruction qu'il ne peut pas analyser,
- soit après une instruction, dans le cas de l'analyse pas à pas provoquée par la commande SET.

Les commandes accessibles dans cet état sont les suivantes:

Commande CHANGE : CHANGE/Chaîne1/Chaîne2/

Cette commande a pour effet de remplacer l'occurrence de chaîne1 dans l'instruction par chaîne2. L'instruction peut être modifiée plusieurs fois par la commande CHANGE, qui n'écrit pas l'instruction dans le fichier intermédiaire.

Commande WRITE :

Cette commande a pour effet d'écrire l'instruction qui vient d'être analysée, dans le fichier intermédiaire. La commande CHANGE suivie de la commande WRITE provoque la modification et le rangement de l'instruction dans le fichier intermédiaire.

Commande SAVE :

Cette commande permet de sauvegarder la version actuelle du fichier intermédiaire et de reprendre l'écriture dans ce fichier à cet endroit.

Commande ? :

Cette commande permet d'obtenir la liste de toutes les commandes disponibles dans l'état "-" de l'éditeur.

Commande REPL :

Cette commande place l'utilisateur dans l'état "#" (attente d'une instruction). Elle permet à l'utilisateur de remplacer l'instruction en cours d'analyse, par une instruction de son choix, dans le fichier intermédiaire. L'utilisateur sort de l'état "#" pour revenir à l'état "-" après le retour chariot délimitant l'instruction formulée.

Commande INPUT :

Cette commande a pour effet d'introduire une ou plusieurs instructions en séquence dans le fichier intermédiaire. Elle place l'utilisateur dans l'état "%" (écriture d'instructions). Pour sortir de cet état, l'utilisateur dispose de la commande %FIN qui le fait revenir dans l'état "-" (attente de commandes).

Commande MORP :

Cette commande permet de relancer l'analyse de l'instruction du fichier source.

Commande R.C. (retour chariot):

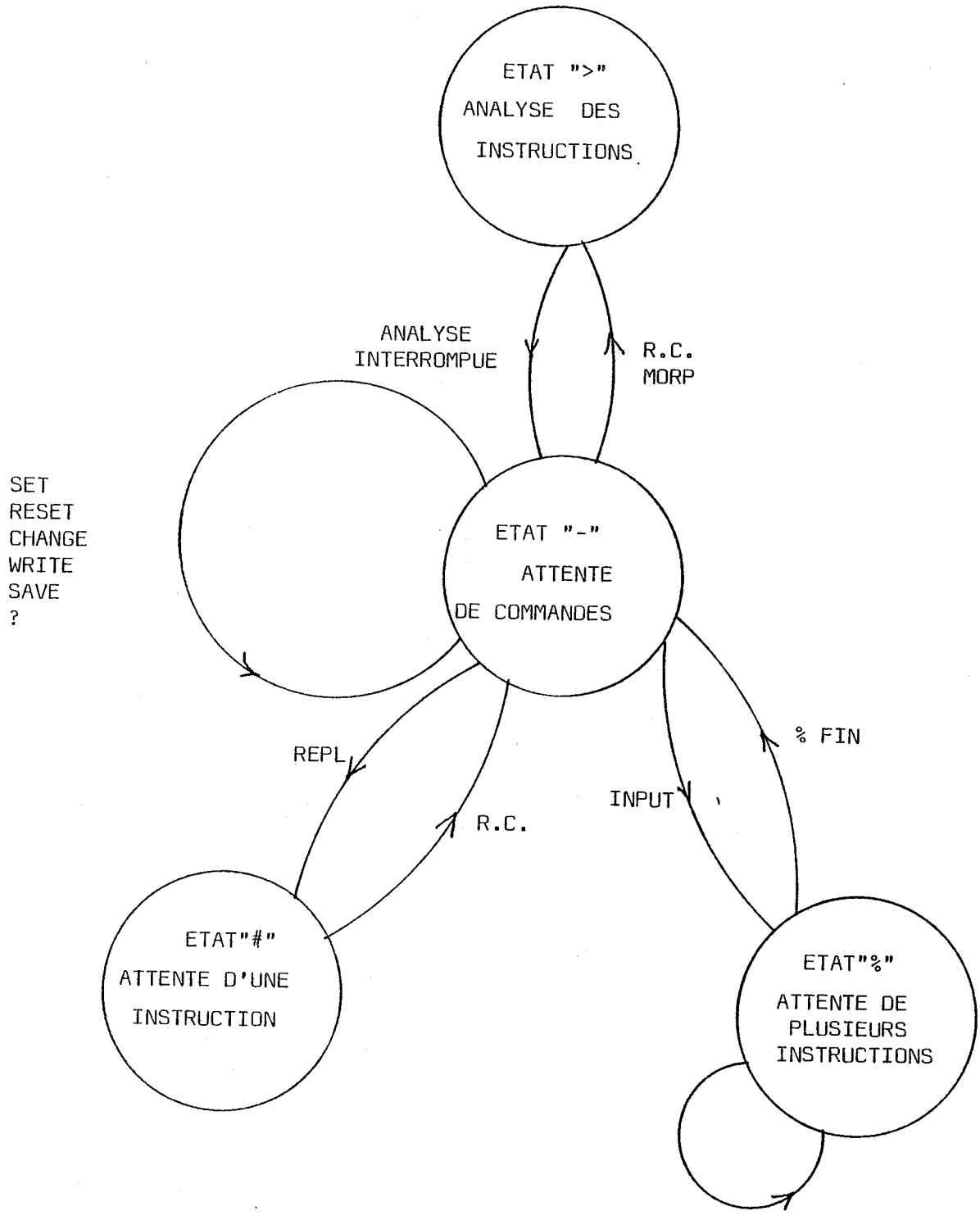
Cette commande a pour effet de sauter l'analyse des mots de l'instruction restants et de passer à l'analyse de l'instruction suivante.

Commandes SET et RESET:

Un programme (fichier source) étant analysé en continu, le transducteur ne s'interrompt que sur les instructions dont l'analyse est impossible. La commande SET a été définie pour provoquer l'arrêt du transducteur après l'analyse de chaque instruction, même si elle est correcte et donner le contrôle à l'utilisateur dans l'état "-". Elle permet à l'utilisateur de contrôler ou modifier des instructions syntaxiquement correctes. L'analyse des instructions se poursuit pas à pas, tant que l'effet de la commande SET n'a pas été annulé par la commande RESET.

Les commandes SET, RESET, CHANGE, WRITE et ? laissent l'utilisateur dans l'état "-" (attente de commandes).

DIAGRAMME DE L'EDITEUR D'ADEQUATION



4 - MODULE DE TRADUCTION (Deuxième passage du tranducteur)

Le but de ce module est:

- soit l'analyse des instructions constituant le fichier intermédiaire fourni par le module d'adéquation et la production d'un fichier résultat constitué par les instructions traduites en LP80,

- soit l'analyse et la traduction de l'instruction envoyée au terminal. Dans ce cas, l'instruction est mise directement sous la forme convenable et il n'y a pas besoin d'appliquer le module d'adéquation.

Le principe de ce module est le suivant:

- l'analyse de chaque instruction PL360 (rendue conforme à la grammaire de traduction) est effectuée par le transducteur général d'états finis qui permet d'obtenir, par transduction, la chaîne formée de la concaténation des codes de dérivation (CD).

- le décodage de la chaîne des codes de dérivation (CD) est effectué par un programme d'interprétation des résultats ou interpréteur, qui fournit ainsi, la forme explicite de la traduction en LP80 de l'instruction. Cette opération permet d'écrire les résultats obtenus par le transducteur.

Ce programme d'interprétation des résultats possède les caractéristiques suivantes:

1/ il est constitué de traitements propres à la traduction d'instructions PL360 en instructions LP80. Il est indépendant du module principal et peut être modifié lorsqu'on envisage la traduction dans d'autres langages de programmation ;

2/ il est interactif: l'utilisateur donne les indications que l'interpréteur ne peut pas trouver lui-même, parce qu'elles sont relatives à des instructions postérieures ou antérieures et qui ne peuvent être données ni par programme, ni par le transducteur (contrairement à l'analyse syntaxique).

Exemples:

a/ Pour traduire en LP80, l'instruction IC (registre, mémoire), l'interpréteur doit savoir si le registre a été remis à zéro précédemment (cf. Deuxième Partie).

Résoudre ce problème par programme, conduit à tenir à jour un tableau d'indicateurs de remise à zéro de tous les registres, au fur et à mesure des instructions. Il est préférable d'utiliser l'interaction de l'interpréteur, dans ce cas.

b/ Pour traduire en LP80, l'instruction Choix, l'interpréteur doit connaître le nombre de choix de l'instruction. Les indications relatives à des instructions postérieures ne peuvent être connues par programme.

Le fonctionnement du programme d'interprétation est basé sur la recherche et le décodage du code-dérivation.

Le tranducteur général d'états finis fournit un tableau dans lequel est rangé chaque segment et les informations qui lui sont relatives, parmi lesquelles se trouve le code dérivation. L'instruction LP80 est obtenue par concaténation des chaînes de traduction résultant du décodage de chaque code dérivation.

Comme on l'a déjà vu, le code dérivation indique, soit:

- une base du dictionnaire,
- un élément synonyme,
- une valeur.

Les valeurs, par définition, sont des chaînes de quatre caractères (au maximum) qui ne permettent pas, dans la plupart des cas, d'indiquer dans sa totalité, la chaîne constituant la traduction de cet élément. Un tableau mettant en correspondance la valeur symbolique du code dérivation et la traduction, est défini dans le programme d'interprétation.
Les valeurs sont soit:

. des valeurs contenues dans le tableau et pour lesquelles il suffit de prendre la traduction correspondante:
Exemple: CD := DVL2 ; fournit la chaîne de traduction "SLLS 2"

. des valeurs indiquant la chaîne de traduction directement (car elle est inférieure à 4 caractères)

Exemple: CD :=)

. des valeurs propres à chaque problème de traduction, demandant un traitement particulier,

Exemple: CD := QU01

Suivant l'indication du code dérivation, la chaîne de traduction est obtenue de cinq manières différentes:

- le code dérivation est un indicateur de traitement. La chaîne de traduction est obtenue soit par réponse de l'utilisateur, soit par traitement.
- la chaîne de traduction est obtenue directement dans le code dérivation (valeur immédiate),
- le code dérivation se réfère à une valeur du tableau. La chaîne de traduction est prélevée dans ce tableau,
- le code dérivation fait référence au dictionnaire. La chaîne de traduction est fournie par le dictionnaire,
- le code dérivation fait référence au synonyme. La chaîne de traduction est la base synonyme de l'élément dans le dictionnaire.

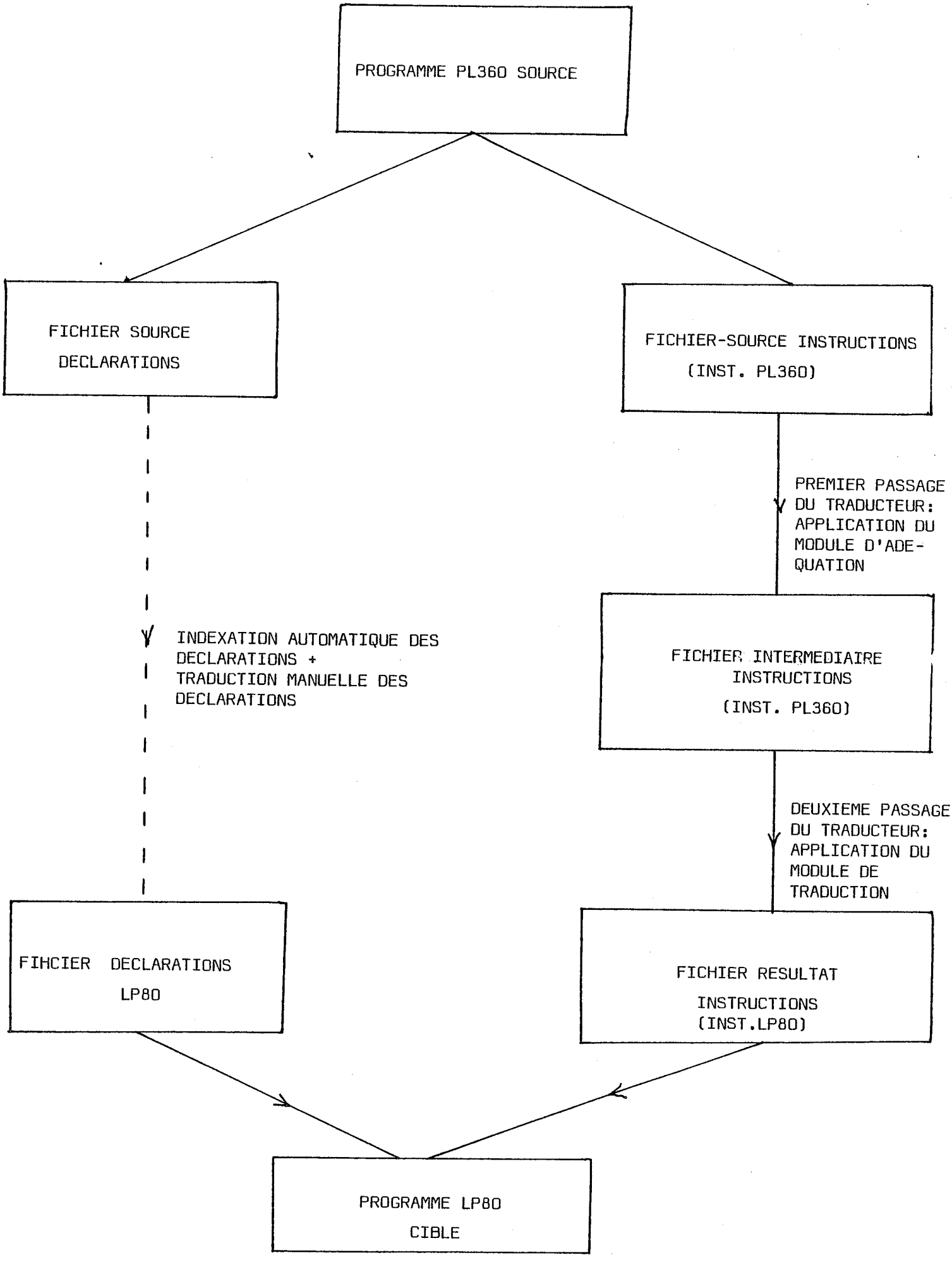
Les problèmes exigeant un traitement spécial de l'interpréteur, sont traitées en détail dans le chapitre suivant. Ce sont principalement:

. les commentaires,
. les nombres (calcul d'un diviseur de 4, de 2, etc...),
. les fonctions logiques de connexion,
. certaines instructions (test, insertion de caractères, choix),
. certains caractères spéciaux: par exemple, le point d'interrogation utilisé pour l'interrogation des modèles et des bases, ne peut être indexé dans le dictionnaire.

N.B.- La séparation de la traduction en deux passages n'est pas indépendante mais elle est préférable pour des raisons de clarté et de modification du programme source.

Les deux schémas suivants illustrent d'une part, l'articulation entre les différents composants du traducteur PIAFTRAD et d'autre part le fonctionnement en trois étapes de celui-ci.

SCHEMA DE LA TRADUCTION D'UN PROGRAMME PL360 EN UN PROGRAMME LP80



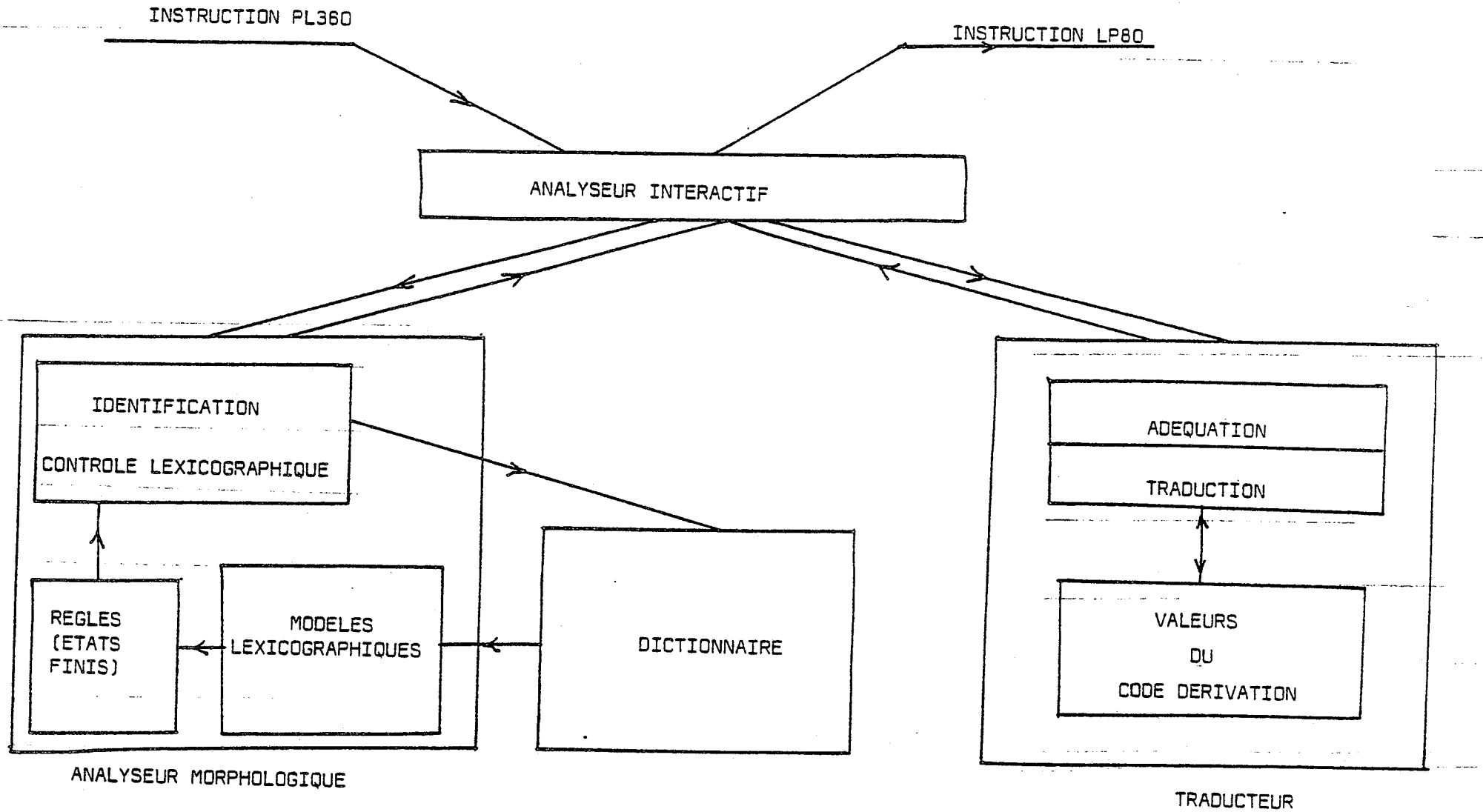


Schéma du traducteur

DEUXIEME PARTIE

PROBLEMES LIES A LA TRADUCTION

Les problèmes posés par la traduction d'un langage de programmation dans un autre langage de la même famille, se répartissent en deux catégories:

- problèmes liés à la structure de la machine, chacune possédant une organisation, qui lui est propre. Ceux-ci font l'objet de la première partie.

- problèmes liés à la définition du langage, traités dans la seconde partie.

Nous indiquons, d'abord, dans un chapitre préliminaire, certaines notations utilisées dans la suite.

Dans le premier chapitre, nous comparons, d'abord, brièvement les caractéristiques de chaque ordinateur, et développons ensuite les problèmes d'adressage et d'utilisation de registres, qui sont les plus importants, ainsi que les solutions adoptées pour résoudre ces problèmes.

Le deuxième chapitre est construit suivant la définition syntaxique du langage PL360. Elle met en évidence les problèmes rencontrés au fur et à mesure de l'élaboration de la grammaire de traduction et définit un sous-ensemble de langage LP80, image du langage PL360.

0 - DEFINITIONS ET NOTATIONS DE BASE

Nous présentons ici un sous-ensemble de la syntaxe du langage PL360 utilisé dans la description des instructions et des fonctions. (En particulier, nous n'avons pas conservé les relations entre les types).

a/ Entités syntaxiques

```
<nombre entier> ::= <chiffre> |
                    <nombre entier> <chiffre>
<registre>      ::= <identificateur de registre>
<mémoire>       ::= <identificateur de mémoire>|
                    <identificateur de mémoire> (<index>)
<index>         ::= <nombre entier>|
                    <registre entier>|
                    <registre entier> + <nombre entier>|
                    <registre entier> - <nombre entier>
<valeur octet>  ::= "<caractère>"|
                    <valeur hexadécimale>
<valeur hexadécimale>::= #<chiffre hexadécimal>|
                         <valeur hexadécimale><chiffre hexadécimal>
<chiffre hexadécimal>::= <chiffre>|A|B|C|D|E|F
<chiffre>            ::= 0|1|..|9
<lettre>             ::= A|B|..|Z
<opérande immédiat>  ::= <valeur octet>|
                         <nombre entier>
<primaire> ::= <valeur octet>|<valeur hexadécimale>|
               <nombre entier>|<registre>|<mémoire>

<identificateur d'adresse> ::= <mémoire>

<identificateur d'adressage direct> ::= B1|B2|B3|.. B13|B14
```

<caractère> désigne

- soit un chiffre
- soit une lettre
- soit un symbole de base du langage: "(", ",", ")" etc...

b/ Notations utilisées

Dans les exemples illustrant le traitement des fonctions, on désigne par:

IDB	un octet
IDS	un demi-mot
IDW	un mot
BTAB	un tableau d'octets
STAB	un tableau de demi-mots
WTAB	un tableau de mots
DTERM	un booléen.

I - PROBLEMES LIES A LA STRUCTURE DE LA MACHINE

1 - PRESENTATION COMPAREE DE L'IBM360 et de l'IRIS80 [I360, IR80]

Les deux ordinateurs IRIS80 et IBM360 possèdent une structure de base identique qui comprend:

- une mémoire centrale,
- une unité centrale (l'IRIS80 peut en comporter deux),
- des unités d'échange directes ou multiplexées,
- des unités de liaison,
- des unités périphériques.

Le système IRIS80 peut comporter une ou deux unités centrales. Cela permet de constituer un système "partageable" et/ou en multitraitement qui accroît la puissance de traitement du système.

1-1 mémoire principale

. IBM360

- La capacité théorique de la mémoire centrale est environ 16000K (2^{24}) octets.
- L'unité d'information de base est l'octet (groupe de 8 bits).
- Les octets peuvent être traités séparément ou groupés en zones:
  - un demi-mot: zone de 2 octets consécutifs dont l'adresse est un multiple de 2.
  - un mot : zone de 4 octets consécutifs dont l'adresse est un multiple de 4.
  - un double mot: zone de 8 octets consécutifs dont l'adresse est un multiple de 8.

Pour ces trois zones, le bit de gauche contient le signe.

- L'adresse d'une zone ou d'un groupe d'octets est spécifiée par l'adresse de l'octet le plus à gauche.

. IRIS80

- La capacité de la mémoire centrale peut varier de 128 K octets à 4096 K octets.

- Le système est basé sur une structure de mots de 32 bits. Des multiples (double-mot) et des sous-multiples (octet, demi-mot) sont aussi accessibles.

- L'IRIS80 dispose de 3 modes d'adressage A, B et C mais seuls les modes B et C permettent l'utilisation du biprocesseur. Le mode C diffère du mode B par l'utilisation d'un dispositif d'adressage étendu.

1.2. unité centrale

1.2.1. Registres

. IBM360: l'Unité Centrale dispose de:

- 16 registres généraux implantés soit dans une unité de mémoire locale; soit dans une zone à part de la mémoire principale.

- 4 registres virgule flottante.

. IRIS80: l'Unité Centrale dispose de 24 registres, implantés en mémoire aux adresses 0 à 23. Ils se substituent aux mots d'adresse réelle 0 à 23 de l'espace virtuel.

1.2.2. Instructions

Les opérations arithmétiques et logiques des deux machines sont réparties en 4 classes:

- arithmétique à virgule fixe,
- arithmétique décimale,
- arithmétique à virgule flottante,
- opérations logiques.

Ces classes diffèrent par le format des données qu'elles utilisent, par les registres spécifiés et par les opérations réalisées.

Les instructions IRIS80 et les instructions IBM360 diffèrent souvent, en ce qui concerne leur fonctionnement. Si on considère la traduction dans le sens IBM360 vers IRIS80, on constate qu'un certain nombre d'instructions IBM360 ne possèdent pas d'instructions équivalentes sur IRIS80, pouvant être utilisées dans des conditions d'emploi similaires à l'IBM360 (instructions de traitement de caractères, de conversion, etc...).

1.2.3. Mot d'état du programme

Le mot d'état programme (PSW sur IBM360, PSD sur IRIS80) contient toutes les informations qui sont nécessaires pour l'exécution correcte du programme, c'est-à-dire:

- des informations sur le déroulement de certaines instructions,
- des masques pour autoriser ou inhiber certains modes de fonctionnement,
- des adresses (compteur ordinal, pointeur de la table des segments).

Le PSW occupe deux mots sur l'IBM360, tandis que le PSD occupe quatre mots (3 mots cadrés sur frontière de double-mot) sur l'IRIS80, en mode C.

On ne décrit ci-dessous que les différences concernant le code condition de chaque machine, les autres différences n'intervenant pas dans les problèmes de traduction.

1.2.4. Code condition

Sur l'ordinateur IBM360, le code condition occupe les positions 34 à 35 du PSW. Les 2 bits fournissent les valeurs possibles 0, 1, 2 et 3. La valeur prise par le code condition n'a de signification que pour l'opération qui lui a donné cette valeur.

Sur l'IRIS80, les indicateurs de condition représentant le code condition occupent les positions 0 à 3 du PSD. Ils sont référencés par IC1, IC2, IC3 et IC4. IC3 et IC4 sont utilisés pour caractériser le résultat d'une opération (rôle analogue à celui des 2 bits représentant le code condition sur IBM360), tandis que IC1 et IC2 sont positionnés pour enregistrer certains incidents (par exemple, IC1 indique la détection d'un code de chiffre ou d'un code de signe incorrect et IC2 le débordement, dans les instructions d'arithmétique décimale).

1.3. unités d'entrées-sorties et unités de contrôle

Les opérations d'entrées-sorties permettent le transfert d'informations entre la mémoire principale et une unité d'entrées-sorties (imprimante, lecteur de carte). Le travail de l'unité d'entrées-sorties est commandé par une unité de contrôle.

Les instructions d'entrées-sorties sont dépendantes de chaque machine, et les conditions d'application sont différentes. C'est la raison pour laquelle, de telles instructions ne peuvent être traduites automatiquement, et doivent, en général, être programmées directement par l'utilisateur.

2- ADRESSAGE ET IMPLANTATION DES DONNEES

2.1. Adressage sur IBM360

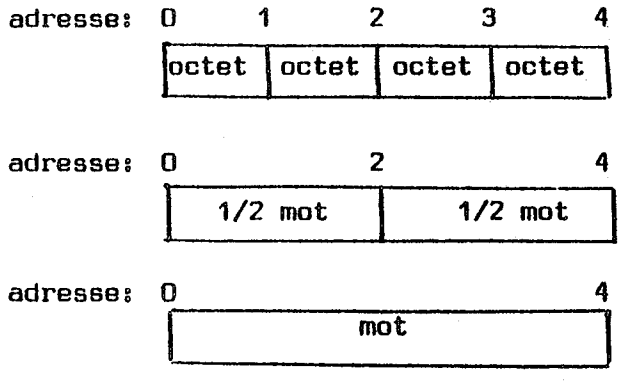
2.1.1. rangement des informations

L'IBM360 possède une organisation basée sur des octets.

Les emplacements des octets en mémoire sont numérotés en séquence à partir de 0. Des zones de longueur fixe, telles que demi-mots et doubles-mots doivent être placées à une adresse spécifique pour cette unité d'information.

Une adresse est dite spécifique pour une unité d'information lorsque son adresse mémoire est un multiple du nombre d'octets contenus dans cette unité (l'adresse d'un demi-mot est un multiple de 2, etc...).

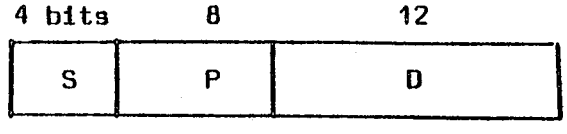
Les zones variables n'ont aucune contrainte vis-à-vis des adresses mais peuvent commencer à n'importe quelle adresse d'octet.



2.1.2. génération d'adresse

La mémoire réelle du calculateur est divisée en blocs de 4096 octets (4K) appelés pages. Sa capacité théorique maximale est (2^{24}) octets soit 16000 K.

Le passage de l'adresse virtuelle à l'adresse réelle est réalisé par le dispositif de translation dynamique des adresses. Une adresse virtuelle d'octets est codée sur 24 bits et possède la structure suivante:



S = numéro de segment
P = numéro de page
D = déplacement

La mémoire virtuelle est donc constituée au plus de 16 segments (2^4) de 256 pages (2^8), chacune comportant 4096 octets.

Du point de vue de l'adressage, les facteurs peuvent être groupés en 3 classes:

- facteurs explicitement adressés dans la mémoire principale,
- facteurs immédiats placés dans le corps de l'instruction,
- facteurs placés dans des registres généraux.

L'adresse utilisée pour se référer à la mémoire principale est engendrée à partir des trois nombres binaires suivants:

- l'adresse de base,
- l'index,
- le déplacement.

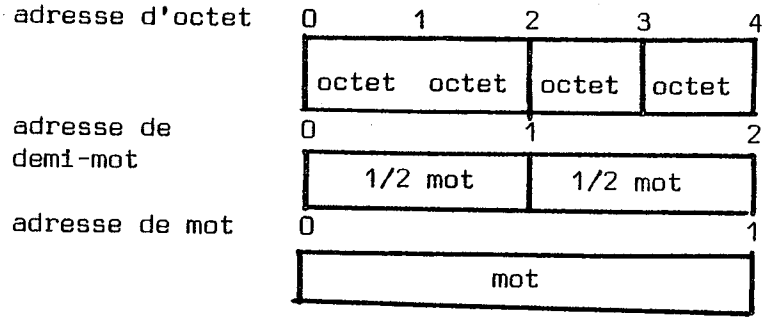
2.2. Adressage sur IRIS 80

2 .2.1. Rangement des informations

Le système IRIS 80 a une organisation basée sur un mot de 32 bits et il est pourvu de facilités pour traiter des informations d'octet, de demi-mot ou de double-mot.

Les bits sont numérotés de 0 à N, de gauche à droite, c'est-à-dire que le bit de poids le plus fort a toujours le numéro 0.

Un octet, un demi-mot, un mot sont toujours placés à une adresse de mot. Un double-mot est placé à une frontière paire de mots. Il s'ensuit qu'une déclaration d'octet (BYTE) ou de demi-mot (SHORT), réserve toujours un mot, une déclaration de double-mot (LONG ou LONGREAL) deux ou trois mots.

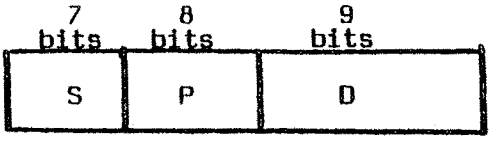


2.2.2. Génération d'adresse

En mode C, l'IRIS 80 peut utiliser un système de traduction dynamique des adresses et il faut distinguer l'adressage de la mémoire réelle et celui de la mémoire virtuelle.

La mémoire réelle a une capacité maximale de 4096 K octets. Elle est constituée de 2048 pages réelles, chacune comportant 512 mots soit 2048 octets.

Une adresse virtuelle de mot est codée sur 24 bits et possède la structure suivante:



S = numéro de segment
P = numéro de page
D = déplacement dans la page

L'espace d'adressage virtuel se décompose donc en 128 (2^7) segments de 256 (2^8) pages, chaque page ayant 512 mots ou 2048 octets.

L'adresse effective d'un élément de la mémoire principale est obtenue de la même façon que sur IBM 360, c'est-à-dire à partir des 3 nombres suivants: base, index, déplacement.

Certaines instructions opèrent sur des octets, d'autres sur des demi-mots, la plupart sur des mots, et quelques unes sur des double-mots.

Sauf pour les opérations sur chaîne d'octets, l'adresse effective avant indexation est toujours une adresse de mot.

L'indexation a un double rôle:

- faciliter l'adressage d'éléments de tableaux à une dimension,
- autoriser l'adressage des octets, demi-mots et double-mots, alors que la machine a une structure de mots.

La nature de l'opérande mémoire d'une instruction est déterminée par son code opération. Lors de l'opération d'addition du contenu du registre d'index, celui-ci est multiplié par 1/4, 1/2, 1 ou 2 pour les opérations portant respectivement sur cet octet, demi-mot, mot et double-mot. Ainsi le contenu d'un registre d'index permet toujours de référencer un quelconque élément d'une liste quelle que soit sa nature.

2.3. Mécanisme d'adressage choisi pour la traduction

2.3.1. Représentation générale des informations

A chaque déclaration de type simple en PL 360 (octet, demi-mot, mot etc...) correspond une déclaration similaire en LP80. La place en mémoire est réservée différemment, mais cela ne pose pas de problèmes lorsque l'élément ne fait pas partie d'une structure particulière de données.

En revanche, il se pose un problème, lorsque:

- il s'agit d'une structure de données figée ou un enregistrement (par exemple les fichiers de données constituant le dictionnaire et la grammaire qui doivent être transportés sur l'IRIS80).
- certains éléments (octet ou demi-mot) sont décrits dans cette structure par synonymie.

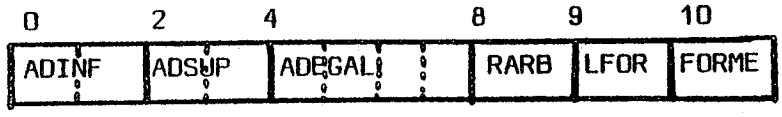
En effet, toute adresse d'octets peut être nommée par synonymie sur IBM 360, tandis que sur IRIS 80, on ne peut désigner que des frontières de mot. La solution consiste à passer par l'intermédiaire d'un tableau d'octets pour désigner un octet non cadré sur une frontière de mot, dans cette structure, ou un tableau de demi-mots pour les demi-mots.

Exemple:

Soit la structure de données déclarée de la façon suivante, en PL 360:

```
ARRAY 10 LOGICAL MOTARB ;
SHORT INTEGER ADINF SYN MOTARB ,
              ADSUP SYN MOTARB (2) ;
LOGICAL ADEGAL SYN MOTARB (4) ;
BYTE RARB SYN MOTARB (8) ,
     LFOR SYN MOTARB (9) ,
     FORME SYN MOTARB (10) ;
```

Représentation en mémoire sur IBM 360 de cette structure:

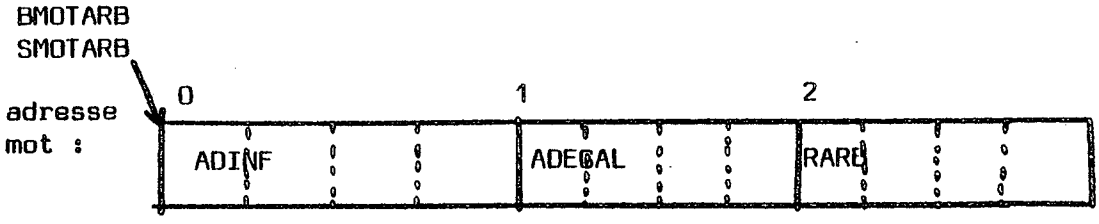


Seules les entités situées à une adresse-mot (c'est-à-dire multiple de 4 donc ADINF, ADEGAL, RARB) apparaîtront explicitement dans la traduction en LP 80 de cette structure de données.

Les autres entités sont considérées comme éléments d'un tableau d'octets ou de demi-mots (suivant leur nature) ayant la même adresse que le tableau décrivant cette structure. Ceci revient donc à déclarer un octet et un demi-mot synonymes du tableau MOTARB et à se référer ensuite à un élément de ce tableau.

Traduction et représentation en mémoire de la structure précédente sur IRIS80:

```
ARRAY 10 WORD MOTARB ;
BYTE BMOTARB SYN MOTARB ;
SHORT SMOTARB SYN MOTARB ;
SHORT ADINF SYN MOTARB ;
WORD ADEGAL SYN MOTARB (1) ;
BYTE RARB SYN MOTARB (2) ;
```



Dans cette perspective, LFOR est traduit par BMOTARB (R2:=9) et donc indexé dans DICT, sous ce nom-là. Il n'apparait pas dans la déclaration de la structure de données en LP80.

2.3.2. Adressage de la mémoire

La différence essentielle entre l'adressage de chaque machine, réside dans le fait que l'organisation de l'IRIS 80 est basée sur une structure de mots (32 bits) tandis que l'ordinateur IBM 360 possède un adressage à octets.

Compte tenu de cette différence, deux solutions se présentent pour traduire le mécanisme d'adressage:

1/ Conserver les adresses absolues sans modification suivant l'opérande référencé. On distinguera donc dans la version traduite en LP 80, des adresses de mot, ainsi que des adresse de demi-mots, d'octets et de double-mots.

En ce qui concerne les instructions utilisant un adressage relatif (base + déplacement), le traducteur doit recourir à l'aide de l'utilisateur pour préciser sur quelle entité (mot, demi-mot, octet ou compteur) porte le déplacement.

Cette solution présente l'avantage de garder la structure caractéristique de l'IRIS 80 ("machine à mots"), mais possède l'inconvénient de poser un trop grand nombre de questions à l'utilisateur et d'exiger une parfaite connaissance du programme en tout point.

2/ Convertir toutes les adresses absolues de mot, demi-mot et double-mot en adresses d'octet (au moyen de décalages), c'est-à-dire conserver la structure de base de l'IBM 360.

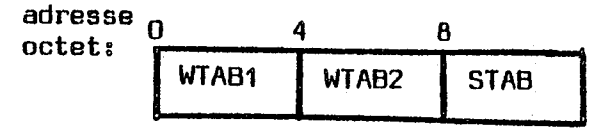
Exemple:

Soient les déclarations suivantes dans les deux langages et leur représentation sur la machine correspondante:

PL 360

```
LOGICAL WTAB1, WTAB2 ;
  SHORT INTEGER STAB ;
```

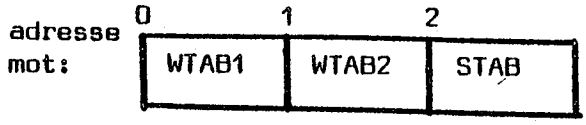
Représentation en mémoire sur IBM 360



LP 80

```
WORD WTAB1, WTAB2 ;
  SHORT STAB ;
```

Représentation en mémoire sur IRIS 80



L'instruction PL 360 d'affectation R5:=@WTAB2; se traduira en LP 80 par l'instruction R5:=@WTAB2 SLLS 2 ;

Cette solution oblige à modifier les déplacements et le contenu du registre d'index dans la traduction des instructions utilisant l'indexation, mais ceci peut être fait automatiquement par le traducteur, puisque la nature de l'opérande apparaît dans l'instruction

Exemple: instruction PL360

```
R4 := WTAB(R5+4) ;
```

instruction LP80

```
R4 := WTAB(R2 := R5+4 SRLS 2);
```

R5+4 est considérée comme une valeur exprimée en octets. Pour référencer le $(R5+4)^{e}$ mot du tableau WTAB, il faut donc diviser (R5+4) par 4, pour obtenir un déplacement exprimé en mots.

En revanche, une instruction comportant un adressage relatif pourra être traduite automatiquement:

Exemple: instruction PL360

R5 := R5+4 ;

instruction traduite en LP80

R5 := R5 + 4 ;

Remarque:

En mode C, l'Unité Centrale dispose d'un jeu de 8 registres de base qui permettent une translation dynamique des adresses. Ceux-ci doivent obligatoirement contenir une adresse-mot, afin de pouvoir référencer les éléments dans l'emplacement qu'ils déterminent (cf. traitement des bases).

Cette solution présente le gros avantage de laisser faire le traducteur sans exiger de connaissances de l'utilisateur. Néanmoins elle alourdit nettement la traduction (décalages dans les instructions) et complique quelque peu la grammaire.

3 - CORRESPONDANCE ET UTILISATION DES REGISTRES

3.1. Description des registres de l'IBM 360

L'Unité Centrale de l'IBM 360 comporte 16 registres généraux pour les facteurs en virgule fixe et 4 registres virgule flottante pour les facteurs en virgule flottante.

Les registres généraux peuvent être utilisés comme registres d'index dans les opérations arithmétiques sur des adresses et les indexages, et comme accumulateurs dans les opérations arithmétiques en virgule fixe et les opérations logiques. Les registres ont une capacité de un mot (ou 32 bits).

Pour quelques opérations, deux registres adjacents peuvent être couplés, permettant ainsi une capacité de deux mots. Le registre indiqué dans

l'instruction contient les bits d'ordre le plus élevé du facteur et doit avoir une adresse paire, tandis que le registre couplé contient les bits d'ordre le moins élevé du facteur et doit avoir l'adresse impaire immédiatement supérieure.

Quatre registres sont disponibles pour les opérations en virgule flottante. Ces registres sont formés de deux mots (64 bits) et peuvent contenir un facteur en virgule flottante, soit court (1 mot), soit long (2 mots). Les registres virgule flottante sont identifiés par les numéros 0, 2, 4, 6.

Dans le langage PL 360, les identificateurs prédéclarés utilisés pour désigner les registres sont:

R0, R1, ..., R15 : (registres traitant des valeurs de type entier ou logique)
F0, F2, F4, F6 : (registres traitant des valeurs de type réel)
F01, F23, F45, F67: (registres traitant des valeurs de type réel long).

3.2. Description des registres de l'IRIS 80

L'Unité Centrale de l'IRIS 80 dispose de 24 registres de 32 bits numérotés de 0 à 23. Ces registres apparaissent comme implantés en mémoire basse aux adresse mémoires de 0 à 23.

Ces 24 registres ont les fonctions spécialisées suivantes:

- Les 16 registres généraux (notés de 0 à 15) sont utilisés pour les opérations en arithmétique virgule fixe et flottante, les opérations logiques, décalages et comparaisons. On remarque que les mêmes registres sont employés en arithmétique virgule fixe et en arithmétique virgule flottante, contrairement à l'IBM 360.
- Les 7 registres notés de 1 à 7 sont employés également comme registres d'index.

- Les 4 registres notés de 12 à 15 servent également de totalisateur aux opérations décimales.

- Le registre noté 16 est utilisé comme compteur de longueur pour les instructions sur chaîne de caractères.

- Les 7 registres notés de 17 à 23 sont utilisés comme registres de base pour l'adressage (en mode C).

Dans le langage LP 80, les identificateurs prédéclarés utilisés pour désigner ces registres sont les suivants:

R0, ..., R9,RA,RB,RC,RD,RE,RF : pour les registres traitant des valeurs entières ou logiques,

R0_1,R2_3,...RE_F : pour les registres utilisés pour l'arithmétique flottante

F0, F1, ... FF : pour les registres traitant des valeurs de type réel

F0_1,F2_3,...,FE_F : pour les registres traitant des valeurs de type réel lon

B1, ... B7 : registres de base.

3.3. Correspondance entre les registres de chaque machine

3.3.1. Registres virgule fixe

D'après l'utilisation des 16 registres généraux sur chaque machine, il est naturel d'établir la correspondance suivante entre:

- les registres notés R0 à R9 en PL360 avec les registres R0 à R9 en LP80,
- les registres notés R10 à R15 en PL 360 avec les registres notés RA à RF en LP 80.

Cependant certains registres sont utilisés d'une façon particulière dans les deux langages, ce qui modifie cette correspondance.

Dans tout programme écrit en langage PL360, les registres R13 et R15, ont par convention, les utilisations suivantes:

- R13 est la base des données du programme,
- R15 est la base des instructions du programme.

Ces registres sont réservés à cette seule utilisation.
Sur l'IRIS 80, dans les programmes écrits en langage LP80, ce sont les registres de base B2 et B1 qui adressent respectivement les données et les instructions du programme. Ces registres sont distincts des registres généraux.

La correspondance ainsi établie entre les registres R13 et B2 d'une part, R15 et B1 d'autre part, laissent donc les registres RD et RF libres, dans la perspective de la traduction d'un programme PL 360 en un programme LP 80. Ceux-ci peuvent donc être utilisés comme registres de travail.

Cependant, les besoins de la traduction imposent:

- un registre de travail utilisé comme registre d'index (ces registres d'index doivent être pris entre R1 et R7).
- un registre de travail pour stocker des résultats intermédiaires.

D'autre part, en LP 80, certaines instructions (transfert et comparaison de caractères) exigent un couple de registres de travail contigus. On choisit arbitrairement le couple (R2, R3) en LP 80.

Les registres R2 et R3 en PL360 se traduiraient donc par les registres libres RD et RF. Mais pour des raisons de couplage, concernant certaines opérations (division, multiplication) nécessitant une extension, il est préférable de traduire R2 et R3 par un couple de registres contigus,soit par exemple le couple (RE,RF). Par conséquent, R14 (dans le langage PL 360) est mis en correspondance avec RD (dans le langage LP 80).

PL360	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
LP 80	R0	R1	RE	RF	R4	R5	R6	R7	R8	R9	RA	RB	RC	B2	RD	B1

Grâce à la correspondance entre les registres (R13, R15) et (B2, B1), deux registres R2 et R3 sont récupérés pour être utilisés soit:

- comme registres d'index,
- comme couple de registres de travail (instructions MOVE,COMP)

3.3.2. Registres virgule flottante

Puisque tous les registres généraux peuvent être utilisés sur l'IRIS 80 pour les opérations arithmétiques virgule flottante, il y a donc une redondance par rapport à l'IBM 360, qui ne fournit que quatre registres flottants. Ceux-ci correspondent donc à 4 des registres généraux de l'IRIS 80.

Cependant puisqu'il n'y a pas de registres flottants particuliers sur l'IRIS 80, il convient de ne pas écraser le registre général correspondant, lors de la traduction d'une instruction d'artihmétique flottante. C'est à l'utilisateur d'éviter ce problème qui ne peut être résolu par le système de traduction.

La correspondance est donc la suivante:

PL360	F0	F2	F4	F6
LP 80	F0	F2	F4	F6

PL360	F01	F23	F45	F67
LP 80	F01	F23	F45	F67

II - PROBLEMES LIES AU LANGAGE

1 - VALEURS

1.1. Nombres entiers ou flottants

Les nombres entiers ou flottants sont formés de chiffres allant de 0 à 9. Ceux-ci sont indexés dans le dictionnaire suivant un modèle unique. Les règles qui s'appliquent à ces chiffres, se valident elles-mêmes, pour l'analyse d'un nombre.

1.2. Chaînes de caractères et valeurs hexadécimales

Une chaîne de caractères est une suite éventuellement vide de caractères placés entre deux guillemets.

Une valeur hexadécimale est une suite variant de un octet à un double-mot de caractères hexadécimaux (0, 1, ..., 9, ou A, B, C, D, E, F) précédée du signe dièse (#).

Ces deux entités sont traitées de la même façon par le dictionnaire et la grammaire. Par la suite, on désigne par "chaîne" l'une ou l'autre de ces entités.

On distingue trois sortes d'instructions PL 360 utilisant une chaîne:

- les instructions de traitement d'un caractère (IC, MVI, CLI, etc...) utilisant une valeur octet (chaîne occupant un octet),
- les instructions de chargement d'un registre par une chaîne. La longueur de la chaîne varie de 1 à 8 octets suivant le type du registre,
- les instructions de traitement de caractères opérant sur des chaînes de longueur supérieure à un octet (MVC, CLC).

En ce qui concerne les deux premières catégories, il existe une instruction LP 80 de chargement d'un registre par un octet, tandis que les instructions

LP 80 de transfert et de comparaison de plusieurs caractères n'admettent pas d'opérandes immédiats.

On distingue deux traitements différents suivant le type de l'instruction.

1.2.1. Chaînes utilisées comme valeur octet ou dans des instructions de chargement de registres

On a le choix entre deux solutions:

- soit indexer chaque caractère alphanumérique dans le dictionnaire de même que pour les nombres. Chaque règle applicable à un caractère doit se valider elle-même. Mais il existe des caractères spéciaux parmi les caractères alphanumériques,(tels que"(","(",";")qui ont une autre signification dans les instructions. Différencier les deux rôles de ces caractères complique considérablement la grammaire. De même pour les valeurs hexadécimales formées non pas de caractères spéciaux, mais de chiffres et de lettres. Cette solution a été abandonnée.

- soit indexer dans le dictionnaire, la chaîne dans sa totalité au fur et à mesure de leur rencontre.

Les chaînes de caractères ou les valeurs hexadécimales sont indexées suivant le même modèle /DPALFA/. Au cours du premier passage, l'analyse de l'instruction s'arrête sur les guillemets ou le caractère dièse, qui ne sont pas déclarés dans le dictionnaire. Après indexation manuelle de la chaîne dans le dictionnaire, l'analyse peut se poursuivre.

1.2.2. Chaînes de longueur variable intervenant dans des instructions transfert et de mouvement de caractères

Il n'existe pas d'instructions LP80 de transfert et de comparaison de caractères admettant des opérandes immédiats de longueur supérieure à quatre caractères.

Au cours du premier passage, l'analyse de l'instruction s'arrête sur les guillemets ou le caractère dièse. L'utilisateur remplace la chaîne par un identificateur de son choix dans l'instruction MVC ou CLC.
En principe, une notation similaire à celle utilisée pour les lignes d'un fichier PL360 est adoptée pour nommer ces identificateurs: trois premières lettres du nom du fichier suivies d'un compteur de deux chiffres.

L'utilisateur a alors le choix entre deux alternatives:

- soit indexer manuellement l'identificateur suivant le modèle adéquat, dans le dictionnaire, lors de l'interruption, et reprendre l'analyse.

- soit l'indexer automatiquement à la fin de l'analyse du programme, en appelant le module correspondant, en même temps que tous les autres identificateurs ajoutés.

Exemple: MVC (2, REPONSE, "ABC") ;

```
A partir de : "ABC"
Analyse impossible: (ORTHO,DICT,REPL...)
> REPL
 MVC(2, REPONSE,LIRO5) ;
>
```

On suppose que le programma s'appelle LIRDOC et que "ABC" est la 5ème chaîne rencontrée dans le programme. LIRO5 sera indexé sur le tableau octet.

2 - DECLARATIONS

Les déclarations servent à associer des identificateurs aux entités utilisées dans le programme, à attribuer certaines propriétés permanentes à ces entités (notamment le type) et à déterminer leur taille. La portée des déclarations est liée à la structure de blocs.

Dans ce paragraphe, on examine les diverses correspondances existant entre les déclarations des deux langages.

2.1 - Déclaration de registres

En PL360, comme en LP80, les registres utilisés correspondent aux registres généraux ou flottants, existant respectivement sur l'IBM360 et sur l'IRIS80. Ces registres sont prédéclarés.

Les problèmes liés à l'utilisation et la correspondance des registres en vue de la traduction PL360-LP80, sont développés dans le chapitre Correspondance et utilisation des registres.

2.2- Déclaration de mémoires

a/ Une déclaration de mémoire introduit des identificateurs et leur associe des mémoires de type spécifié.
Les types sont les mêmes dans les deux langages (octet, entier court, entier, réel, réel long) mais réserve de la place en mémoire différemment (cf. chapitre Adressage et implantation des données).

b/ La déclaration de synonymie permet de donner des noms différents à un même élément de mémoire ou a un même registre, dans les deux langages.

Il existe un certain nombre de variables prédéclarées par synonymie. Celles-ci permettent de faire un adressage direct de variables quelconques.

En PL360, ces variables sont déclarées pour chaque registre, comme une variable de type mot.

Ce sont les variables suivantes:

```
integer MEM syn 0 ;
integer B1 syn MEM(R1), B2 syn MEM(R2), ...
        ....
        B12 syn MEM(R12), B13 syn MEM (R13);
```

En LP80, il n'y a pas une variable correspondant à chaque registre mais une variable prédéclarée synonyme de l'adresse 0 pour chaque type. En utilisant un des registres comme index, elles permettent d'adresser directement une variable quelconque.

Ce sont les variables suivantes:

```
byte  BYTE0  syn 0 ;
short SHORT0 syn 0 ;
word  WORD0  syn 0 ;
long  LONG0  syn 0 ;
```

La traduction de l'identificateur B1 du langage PL360, diffère donc en LP80, suivant le type de l'instruction utilisant B1.

Exemple:

- Instruction PL360 opérant sur des octets: IC(R4,B1).
  B1 est traduit en LP80 par BYTE0(R1)
- Instruction PL360 opérant sur des demi-mots: LH(R4,B1).
  B1 est traduit en LP80 par SHORT0(R1)
- Instruction d'affectation opérant sur un mot: R4:=B1 ;
  B1 est traduit en LP80 par WORD0(R1).

Ceci oblige à définir des règles différentes pour chaque type d'opérations et alourdit sensiblement la grammaire de traduction.

c/ Déclarations de tableaux: En PL360, comme en LP80, les tableaux sont à une seule dimension. Un tableau n'est qu'une suite de variables d'un même type accessible au moyen d'un seul nom.

2.3 - Déclaration de constante

La déclaration de constante permet d'associer un nombre entier à un identificateur. Elle ne provoque aucune réservation mémoire. La syntaxe est identique dans les deux langages.

2.4 - Déclaration de fonction assembleur

La déclaration de fonction sert:

- à donner un nom à cette fonction,
- à associer à ce nom l'instruction désirée du langage d'assemblage,
- à donner les différents champs des opérandes de la fonction.

Un certain nombre de fonctions correspondant aux instructions de l'assembleur 360 les plus utilisées, sont prédéclarées. Certaines d'entre elles (en particulier les instructions de traitement de caractères: XC, NC, TM...) ne possèdent pas d'analogues dans le langage LP80. Il faut donc:

- soit les traduire par une autre instruction,
- soit définir des procédures de service.

2.5 - Déclaration de segmentation

Une déclaration de segmentation indique au compilateur que le registre spécifié dans cette déclaration est à utiliser comme registre de base dans le calcul des adresse des éléments qui seront déclarés dans le bloc où la déclaration de segmentation apparaît.

Dans les deux langages, il existe la possibilité de déclarer plusieurs sortes de segments de données (zone de données globales, externes, communes ou fictives).

2.6 - Déclaration de procédure

La déclaration de procédure permet de donner un nom (ou identificateur de procédure) à une instruction appelée aussi corps de procédure.

Dans la déclaration de procédure en PL360, un registre de retour est spécifié Celui-ci (en général R14) contient l'adresse de retour suivant l'appel de la procédure. Il est sauvegardé à l'entrée de chaque procédure et restauré à la sortie.

En LP80, la syntaxe de déclaration procédure est identique. Le registre de retour est:

- soit le registre RD lorsque R14 est le registre de retour dans le programme PL360 (cf. chapitre Correspondance et utilisation des registres),
- soit le registre correspondant.

Les déclarations de procédure ne figurent pas parmi les déclarations des données, mais parmi les instructions du programme. Les identificateurs de procédure ne sont, par conséquent, pas indexés automatiquement avec les autres identificateurs.

Deux solutions existent pour indexer les identificateurs de procédure:

1/ indexation manuelle: lorsque l'analyseur rencontre une déclaration de procédure, il s'arrête sur l'identificateur, au cours du premier passage, afin de permettre à l'utilisateur de l'indexer dans le dictionnaire. Cette solution ralentit le premier passage.

2/ indexation automatique par programme: chaque instruction analysée est comparée avec une déclaration de procédure. Lorsqu'une telle déclaration a été identifiée, l'identificateur suivant est indexé sur le modèle de procédure. Cette solution, présentant l'avantage de ne plus provoquer d'arrêt de l'analyseur sur une déclaration procédure, semble préférable à la solution précédente.

2.7 - Autres déclarations

a/ étiquettes:

Dans un programme LP80, toute utilisation d'une étiquette doit être précédée d'une déclaration de cette étiquette.
Il y a deux types d'étiquettes:

- LABEL ou étiquettes internes, connues seulement du segment courant,
- ENTRY ou étiquettes externes, connues du segment courant et des segments externes.

En PL360, comme dans la plupart des autres langages, les étiquettes sont utilisées sans aucune déclaration préalable. L'utilisateur doit donc ajouter dans la zone des données du programme LP80, les déclarations des étiquettes, rencontrées au cours du deuxième passage par l'analyseur.

b/ variables booléennes:

En LP80, il existe une déclaration spécifique pour les variables booléennes. Une déclaration de variable booléenne (type "LOGICAL") introduit les identificateur correspondantset leur associe un octet de mémoire. L'initialisation d'une variable booléenne permet de réserver la zone mémoire correspondante avec les valeurs TRUE (#F0) et FALSE (#00). Une variable booléenne diffère d'un octet par son utilisation dans une expression conditionnelle.

En PL360, il n'y a pas de différence entre les variables booléennes et les octets. Une variable booléenne est un octet prenant , entre autres, les valeurs TRUE (#FF) et FALSE (#00) et ne nécessitant pas une déclaration particulière. Un octet peut être utilisé comme variable booléenne et testé dans les instructions conditionnelles.

Ceci oblige donc à déclarer en LP80, une variable booléenne synonyme de chaque octet destiné à être utilisé comme variable booléenne dans le programme PL360. Suivant le rôle joué dans le programme, un octet peut être traduit différemment.

Exemple:

déclaration PL360: byte DTERM ;

déclaration LP80 correspondante: byte DTERM ;

Logical DTERM1 SYN DTERM ;

	instruction PL360	instruction LP80 correspondante
Utilisation de DTERM comme booléen	SET(DTERM) ;	DTERM1 := TRUE ;
Utilisation de DTERM comme octet	IC(R1,DTERM) ;	R1 := DTERM ;

3- INSTRUCTIONS

3- 1. INSTRUCTIONS DE CONTROLE

On désigne par instruction de contrôle, les instructions qui permettent de contrôler le déroulement du programme.

Ces instructions sont, en PL360:

- l'instruction choix (CASE),
- les instructions conditionnelles (IF et WHILE),
- l'instruction pour (FOR).

Le langage LP80 dispose, entre autres, des mêmes instructions de contrôle, mais celles-ci ont une syntaxe différente (excepté l'instruction FOR) et dans certains cas, posent des problèmes de traduction qui ne peuvent être résolus qu'avec l'aide de l'utilisateur. L'instruction FOR, ne posant pas de problèmes, de traduction, n'est pas traitée dans la suite.

3.1.1. Instruction choix

En PL360, comme en LP80, une instruction choix se décompose en:

- une proposition choix,
- une suite de "choix" formée d'instructions.

La description de l'instruction dans la syntaxe des deux langages met en évidence les problèmes posés pour la traduction:

Syntaxe PL360:

```
<instruction choix>   ::=  <suite de choix> END
<suite de choix>      ::=  <proposition choix> BEGIN
                           <suite de choix> <instruction> ;
<proposition choix>   ::=  CASE <registre entier> OF
```

Le sous-ensemble de la syntaxe LP80 de l'instruction choix, image de la syntaxe précédente est:

<instruction choix> ::= <proposition choix>/<nombre entier> OF <suite de choix>
<suite de choix> ::= <instruction> | <suite de choix> OR <instruction> ;
<proposition choix> ::= CASE <registre entier>

La traduction d'une instruction choix PL360 consiste donc à:

- supprimer les mots-clés BEGIN et END encadrant la suite des choix en PL360. En effet en LP80, la fin de l'instruction choix est marquée seulement par un point virgule ,
- indiquer le nombre de choix dans la proposition Choix en LP80,
- remplacer les points virgules à la fin de chaque "choix", en PL360, par le mot-clé "OR" en LP80.

Ceci est impossible à réaliser sans intervention de l'utilisateur, puisque l'on utilise un transducteur général d'états finis qui ne tient pas compte de l'analyse des instructions précédentes contrairement à l'analyse syntaxique.

Exemple:

Instruction choix en PL360

```
CASE  R3  OF
BEGIN
      R2 := 1 ;
      BEGIN
            R2 := 2 ;
            R2 := 3 ;
      END ;
      R2 := 4 ;
END ;
```

Traduction en LP80

```
CASE R3/3  OF
      R2 := 1       OR
      BEGIN
            R2 := 2 ;
            R2 := 3 ;
      END  OR
      R2 := 4 ;
```

Le problème de traduction est résolu à l'aide des commandes SET et RESET de l'éditeur ainsi que des autres commandes de celui-ci.

Lors de l'analyse d'une instruction choix, au cours du premier passage, l'utilisateur indique, par la commande SET, qu'il demande le contrôle après le traitement de chaque instruction, afin de pouvoir modifier celle-ci en conséquence. A la fin de l'instruction CASE, il signale par la commande RESET, que l'analyse peut reprendre d'une façon continue.

L'instruction choix PL360, ainsi modifiée, pourra être traduite automatiquement au cours du deuxième passage.

3.1.2. Instructions conditionnelles: IF et WHILE

Ces instructions sont définies dans les deux langages par:

<proposition IF> ::= IF <condition composée> THEN <instruction>|
IF<condition composée> THEN<instruction> ELSE <instruction>
<proposition WHILE> ::= WHILE <condition composée> DO <instruction>
<condition composée> ::= <condition>|
<condition> AND <condition>|
<condition> OR <condition>

Le problème posé par ces instructions est la traduction en LP80, de l'entité syntaxique <condition>.

En PL360, <condition> est définie par:

<condition> ::= <registre><relation><primaire>|
<identificateur d'octet>|
¬<identificateur d'octet>|
<relation>|
OVERFLOW

3.1.2.1. <identificateur d'octet> ou ¬<identificateur d'octet>

Les variables booléennes sont représentées par les valeurs hexadécimales #FF et #00, en PL360 et par les valeurs hexadécimales #F0 et #00 en LP80. Dans la mesure où les booléens sont affectés par l'intermédiaire des fonctions SET et RESET en PL360, par les valeurs TRUE et FALSE en LP80, ceux-ci peuvent être testés de la même manière dans les deux langages, même s'ils ne contiennent pas la même valeur hexadécimale.

Exemple:

PL360	LP80
SET(DTERM) ;	DTERM1 := TRUE ;
...	
IF DTERM THEN ...	IF DTERM1 THEN ...
(DTERM contient #FF)	(DTERM1 contient #F0)

3.1.2.2. <registre> <relation> <primaire>

Une condition qui est une relation entre un registre et un primaire est vérifiée si et seulement si la relation entre la valeur courante du registre et le terme est vraie. Dans ce cas les opérateurs de relation sont traduits de la même façon dans les deux langages.

Comparaison des valeurs de chaque code condition en fonction des opérateurs de relation:

opérateur de relation prédéclaré	IRIS 80 IC1	IC2	IC3	IC4	IBM 360
=	-	-	0	0	00
<	-	-	0	1	01
>	-	-	1	0	10
<=	-	-	0	-	0-
>=	-	-	-	0	-0

3.1.2.3. <relation> et overflow

Cette condition est vérifiée si et seulement si le code condition contient la valeur exprimée par l'opérateur de relation.

Du point de vue de la traduction, on distinge trois sortes d'instructions PL360,qui positionnent le code condition afin qu'il soit analysé dans l'instruction conditionnelle suivante:

a/ instruction de comparaison d'un caractère: CLI (I2, mémoire).
Cette instruction est traduite par un chargement dans un registre de travail R3 de l'octet situé à l'adresse spécifiée. La condition intervenant dans l'instruction suivante, ne se traduira pas par l'opérateur équivalent, mais par la comparaison de ce registre avec l'opérande immédiat cité dans l'instruction CLI. En effet, cette instruction ne possède pas d'instruction équivalente en LP80, à part la fonction composée COMP que l'on préfère éviter lorsque la chaîne de caractères ne comporte qu'un seul octet.

Exemple:

PL360 :	LP80 :
CLI ("*",IDB) ;	R3 := IDB ;
IF = THEN	IF R3="*" THEN

L'opérande immédiat est conservé lors de l'instruction précédente et transmis par programme lors de la traduction de la condition (l'utilisation du transducteur général d'états finis ne permet pas de se référer à l'instruction précédente)

b/ instruction de comparaison de mémoire à mémoire: CLC
Cette instruction possède une instruction équivalente COMP en LP80, mais le code condition n'est pas positionné de la même façon par ces deux instructions.

Le tableau suivant montre le positionnement du code condition par les deux instructions:

signification	COMP(R2,1,1er facteur,2°facteur) IC1 IC2 IC3 IC4	CLC(1,1°facteur, 2° facteur)
1er fact = 2° fact	- - 0 0	0 0
1er fact < 2° fact	- - 1 0	0 1
1er fact > 2° fact	- - 0 1	1 0

Dans ce cas, lors de l'analyse d'une condition qui est un opérateur de relation ,

celui-ci est modifié par programme (les opérateurs de relation interviennent également entre un registre et un primaire et sont donc déjà traduits d'une certaine façon dans le dictionnaire).

Exemple: Soient BTAB1 et BTAB2 2 tableaux d'octets.

PL360		LP80
CLC (3,BTAB1, BTAB2)	⇒	COMP(R2,3,@BTAB1,@BTAB2)
IF <= THEN		IF >= THEN

c/ instructions de traitement de caractères: TM, XC, OC , NC, XI, OI, NI
Ces instructions sont traduites par des procédures externes (voir chapitre: Fonctions) qui positionnent le registre R2 en retour, suivant le résultat de l'opération logique effectuée. C'est donc le contenu de ce registre qui est testé dans la condition en LP80. On a conservé, dans la mesure du possible, les valeurs du code condition en PL360.

Le tableau suivant indique la correspondance entre le code condition positionné après une instruction TM ou une opération logique et la valeur contenue dans le registre R2.

signification pour l'instruction TM	signification pour les opérations logiques	opérateur de relation PL360	traduction LP80
les bits sélectés sont tous des 0	le résultat est nul	=	R2=0
les bits sélectés sont des 0 et des 1	le résultat n'est pas nul	¬=	R2=0
les bits sélectés sont tous des 1		OVERFLOW	R2=3

Exemple:

PL360		LP80
TM(2,IDB) ;	⇒	traduction de TM
IF = THEN ...		IF R2=0 THEN ...

Les conditions définies par <opérateur de relation> sont traduites différemment suivant l'instruction précédente. Lorsque l'analyseur rencontre une instruction de type CLC, CLI ou de traitements de caractères, une variable booléenne est positionnée. L'instruction conditionnelle est traduite selon les valeurs de cette variable.

3.2. FONCTIONS

Le langage PL360 étant un langage proche du niveau assembleur, il est possible de déclarer comme fonction dans un programme, toutes les instructions du langage d'assemblage de l'IBM360. Parmi celles-ci, il existe un certain nombre de fonctions prédéclarées correspondant aux instructions 360 les plus utilisées. En général, celles-ci constituent un jeu varié et suffisant de sorte qu'il est rare que le programmeur ait recours à d'autres instructions de l'assembleur.

Le langage LP80 possède également cette caractéristique propre à cette famille des langages de programmation. Mais les conditions d'application de ces fonctions sont souvent plus complexes et peu pratiques. Un certain nombre de fonctions PL360 (en général des fonctions opérant sur des octets) doit donc être traduit par des séquences de une ou plusieurs instructions, voire même par une procédure externe, lorsque la traduction devient trop complexe.

Selon la classification adoptée dans le manuel de l'IBM360[ ], on répartit les diverses fonctions PL360 dans les catégories suivantes:
- opérations logiques (MVI, MVC,CLI,CLC,XNOI,XNOC,IC,STC,SLDL,SRDL),
- arithmétique décimale (PACK,UNPK),
- arithmétique à virgule fixe (LH,STH,LM,STM,SRDA,SLDA,CVB,CVD),
- instruction EXECUTE (EX).

3.2.1. Fonctions logiques

Les principaux problèmes posés par la traduction de ces fonctions sont les suivants:

- L'IRIS 80 étant essentiellement une machine à structure "mot", il n'existe pas d'instructions de traitement logique de caractères en mémoire. Il faut donc simuler le comportement de certaines instructions.

- Les facteurs de ces fonctions sont généralement traités sous la forme d'octets. Dans certains cas les 4 bits de droite ou de gauche d'un octet sont traités séparément, ou bien les facteurs sont décalés bit par bit. Sur l'IRIS80, la distinction entre adresses de mots,de demi-mots et d'octets oblige à transformer celles-ci en adresses d'octets.

- Certains facteurs sont introduits à partir d'instructions. Ces opérandes immédiats ne sont en général pas admis par les instructions correspondantes en LP80.

Dans cette partie, nous distinguons:

- les fonctions de traitement de caractères: TM, XC, OC, NC, XI, NI, OI qui posent les plus gros problèmes de traduction.
- les autres fonctions: IC, STC, MVI, CLI, MVC, CLC,SRDL , SLDL dont le traitement est moins complexe.

3.2.1.1. Fonctions de tests et de connexions

Ces fonctions se répartissent en deux classes, suivant le format des données:

- format de longueur variable: ce sont les fonctions XC, OC,et NC s'effectuant de mémoire à mémoire sur une longueur spécifiée(<=256 octets) ;
- format limité à un seul octet. Ce sont la fonction TM, d'une part, et XI, OI, NI, d'autre part, pour lesquelles les données sont introduites à partir de l'instruction.

Du point de vue syntaxique, les fonctions XC, OC et NC ont la même définition. Pour simplifier l'écriture, on désigne par l'identificateur XNOC l'une quelconque de ces trois fonctions. De même pour XI, OI et NI qui sont désignées par XNOI.

a/ problèmes de traduction

Contrairement à d'autres fonctions assembleur logiques (telles que MVI, CLI, IC qui sont étudiées plus loin) dont la traduction est constituée au maximum de trois instructions LP80, celles-ci exigent plus de complexité.

En effet, les facteurs sur lesquels opèrent ces fonctions sont pris en mémoire ou introduits à partir de l'instruction elle-même. Or, en LP80, il n'y a pas de fonctions logiques équivalentes, mais seulement des opérateurs logiques, ce qui oblige à utiliser des registres de travail pour le chargement et le rangement de l'opérande. L'exemple suivant met en évidence ces problèmes:

Exemple: Fonction OI(I2,d1(B1))

Cette fonction effectue la somme logique des bits de l'opérande immédiat I2 et de l'octet situé à l'adresse d1(B1). Le résultat est placé à cette adresse et le code condition est positionné en conséquence.

Pour traduire,en LP80, une telle fonction, il faut:
- charger dans un registre de travail (R2 ou R3), l'opérande immédiat ;
- faire la somme logique dans ce registre avec l'octet situé à l'adresse d1(B1);
- ranger le registre à cette adresse ;
- positionner le code condition (celui-ci est effectivement positionné après la somme logique, mais est modifié par l'instruction de rangement).

La fonction OI opérant sur un octet, il faut éventuellement transformer l'adresse du deuxième facteur en adresse d'octet. Ceci conduit à la mobilisation de plusieurs registres (les deux registres de travail, ne sont, en général, plus suffisants) et à l'augmentation du nombre d'instructions constituant la traduction de cette fonction.

Le fonctionnement des autres instructions (de type XNOC ou TM) est plus complexe. Pour la fonction TM, il n'existe pas d'opérateur équivalent dans le langage LP80.

La solution choisie pour résoudre ces problèmes est la définition de procédures équivalentes. Celles-ci, communes à tous les programmes, sont rangées avec les procédures systèmes, dans le fichier externe des procédures de service.

b/ traitement de ces fonctions

La syntaxe de ces fonctions en PL360 est la suivante:

```
{TM  }(<opérande immédiat>, <mémoire>)
{XNOI}(<opérande immédiat>, <mémoire>)

XNOC(<nombre entier>, <mémoire>, <mémoire>)
```

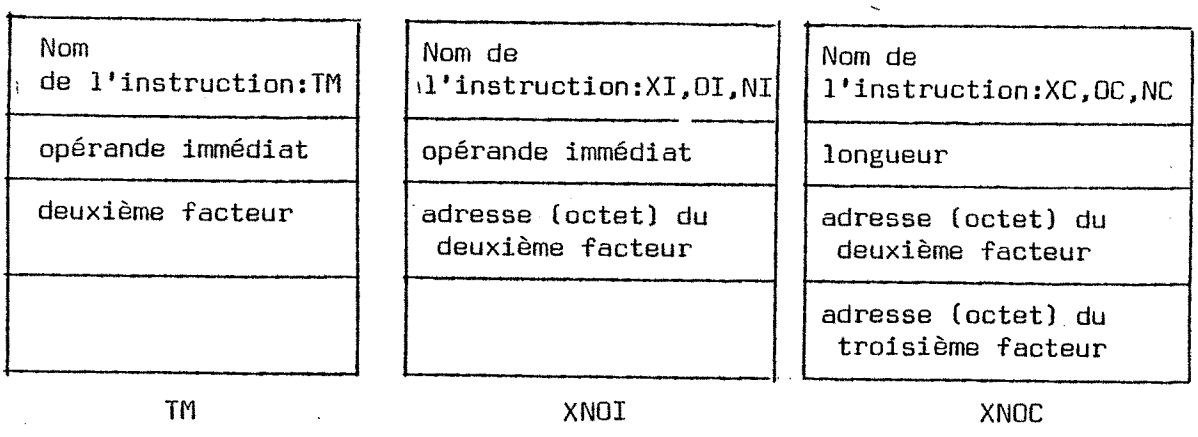
Ces fonctions présentent les caractéristiques suivantes:

- elles ne sont pas indexables par un registre ;
- les fonctions XNOC et XNOI s'effectuent avec modification d'octets. Les adresses des zones modifiées doivent donc être passées en paramètres aux procédures externes correspondantes ;
- la fonction TM opère sur un octet, et celui-ci n'est pas modifié. Il peut donc être directement passé en paramètre (au lieu de son adresse) ;
- toutes les fonctions positionnent le code condition. Compte tenu des sauvegardes et restaurations effectuées à l'entrée et à la sortie des procédures externes, le code condition doit être passé comme paramètre de retour par l'intermédiaire d'un registre de travail (R2).

Le passage des paramètres et la définition syntaxique de ces fonctions dans la grammaire de traduction, sont établies d'après ces considérations.

. passage des paramètres

Les paramètres sont transmis dans un tableau de quatre mots commun à toutes les procédures externes et aux procédures systèmes. Le contenu de ce tableau est le suivant:



La procédure "INTERFACE" appelle la procédure externe correspondante au nom de l'instruction passée en paramètre. Au retour, le registre R2 est positionné avec les mêmes valeurs que le code condition (0,1,2,3).

. représentation intermédiaire de ces fonctions

C'est la représentation acceptée par la grammaire de traduction.

La représentation de TM est inchangée.

Pour les fonctions XNOI et XNOC, on distingue deux sortes d'identificateurs de mémoire:

- les identificateurs standard prédéclarés en PL360 (B1 à B15) qui indiquent implicitement l'adresse des facteurs. Ils sont directement passés en paramètre ;
- les identificateurs de type mot, demi-mot ou octet déclarés dans le programme. Ceux-ci indique le facteur lui-même et non l'adresse de ces facteurs. Pour les passer comme paramètres, il faut ajouter le caractère "@" devant ces identificateurs.

Les fonctions XNOI et XNOC ont la structure suivante:

XNOI (<opérande immédiat>, <facteur de fonction>)

XNOC (<longueur>, <facteur de fonction> , <facteur de fonction>)

avec <facteur de fonction> ::= @<identificateur d'adresse>|
<identificateur d'adressage direct>

. fonctionnement du traducteur

Lors du premier passage, l'analyseur s'interrompt sur les instructions non conformes à la grammaire, c'est-à-dire les identificateurs d'adresse pour les fonctions XNOC et XNOI. L'utilisateur doit ajouter le caractère d'adressage pour que l'analyse reprenne.

Les autres fonctions logiques sont dans l'ensemble, moins complexes. Nous nous contentons d'indiquer brièvement leur fonctionnement en PL360 et les problèmes posés par la traduction. Il convient de se reporter à la grammaire, pour plus de détails.

3.2.1.2. Fonctions IC-STC

Syntaxe PL360

{IC / STC} (<registre> , <mémoire>)

Fonctionnement

IC : l'octet pris en mémoire est introduit dans les positions de bit 24 à 31 du registre spécifié. Les bits restants du registre sont inchangés.

STC : les positions de bits 24 à 31 du registre spécifié sont placés à l'adresse-mémoire indiquée.

Traduction

En LP80, les instructions d'affectation entre registre et mémoire sont aussi utilisées pour des opérandes de type octet (l'instruction IRIS 80 correspondante au type de l'opérande est engendrée).

Lorsque l'opérande n'est pas effectivement un octet (mais un mot ou un demi-mot), son adresse doit être transformée en adresse d'octet et l'octet peut être chargé

ou rangé par l'intermédiaire de l'identificateur standard BYTE0 qui provoque un adressage direct.

Exemples:

. IC (R4,IDW) ; où IDW est un identificateur de type mot

Traduction : R2:= @IDW SLLS 2 ; - chargement de l'adresse octet dans R2

R4:= BYTE0(R2) - chargement dans R4 de l'octet situé à cette adresse.

. IC (R4,IDB) où IDB est un identificateur de type octet

Traduction : R4:= IDB ;

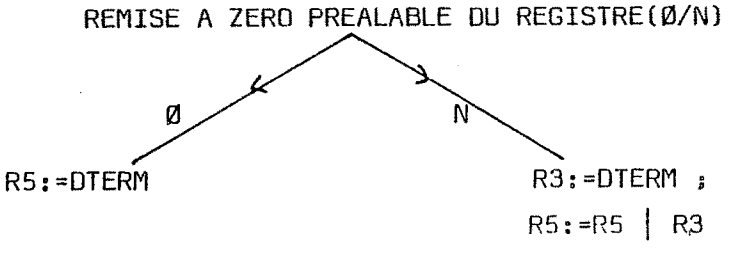
D'autre part, l'instruction IC, en PL360, est souvent utilisée pour ranger plusieurs octets dans un registre, par décalages successifs. Or l'instruction d'affectation obtenue en LP80 par traduction, remet à zéro les trois octets de gauche du registre. Ce problème ne peut être résolu que par l'intervention de l'utilisateur.

Au cours du deuxième passage, le traducteur demande à l'utilisateur:

- si les bits restants du registre ont été remis à zéro préalablement (la traduction se fait alors par chargement dans le registre),
- ou s'ils ont été utilisés pour ranger l'information (la traduction se fait par le chargement de l'octet dans un registre de travail suivi de l'union logique de celui-ci avec le registre spécifié).

Exemple:

IC(R5,DTERM) ;



3.2.1.3. Fonctions MVI, CLI

Syntaxe PL360

$$\left\{ \begin{array}{l} \text{MVI} \\ \text{CLI} \end{array} \right\} \quad (\langle\text{opérande immédiat}\rangle \ , \ \langle\text{mémoire}\rangle)$$

Fonctionnement

MVI : rangement de l'opérande immédiat à l'adresse indiquée,

CLI : comparaison de l'opérande immédiat avec l'octet situé à l'adresse indiquée et positionnement du code condition. L'opérande immédiat est nécessairement un octet.

Traduction

. instruction MVI

- chargement du registre de travail R3 avec l'octet désigné par l'opérande immédiat ;
- transformation de l'adresse du facteur en adresse d'octet ;
- rangement du registre R3 à cette adresse.

Exemple:

```
MVI ("$",IDB); => R3 := "$" ;  IDB:=R3 ;
MVI ("$",IDW); => R3 := "$" ;
                  R2 := @IDW SLLS 2;
                  BYTE0(R2) := R3 ;
```

. instruction CLI

- transformation de l'adresse du facteur en adresse d'octet ;
- chargement dans le registre R3 de l'octet situé à cette adresse.

L'instruction CLI est toujours suivie d'une instruction de test. C'est le contenu du registre R3 qui est comparé avec l'opérande immédiat, lors de l'analyse de cette instruction (voir instructions de test).

Exemple:

PL360	LP80
CLI ("$",IDB) ;	R3 := IDB ;
IF = THEN...	IF R3 = "$" THEN...

3.2.1.4. Fonctions MVC, CLC

Syntaxe PL360

$$\left\{ \begin{array}{l} \text{MVC} \\ \text{CLC} \end{array} \right\} \quad (\langle\text{nombre entier}\rangle,\langle\text{mémoire}\rangle\ ,\ \langle\text{mémoire}\rangle)$$

Fonctionnement

- transfert ou comparaison d'une zone d'octets de longueur spécifiée de mémoire à mémoire ;
- positionnement du code condition pour l'instruction CLC.

Traduction

En LP80, il existe des fonctions composées (MOVE et COMP) qui ont pour but de simplifier l'emploi des instructions de manipulation de caractères(MBS,CBS,TBS).

Les principales différences avec les fonctions PL360 correspondantes sont les suivantes:

- utilisation d'un couple de registres pair-impair pour le fonctionnement de l'instruction: on utilisera donc (R2, R3) ;
- les adresses des facteurs sont les adresses explicites d'octets et non les identificateurs eux-mêmes ;
- la longueur spécifiée est la longueur effective (soit 1) tandis qu'en PL360 c'est (1-1) qui est indiqué: ce problème ne peut être résolu que par programme ;
- pas de possibilité d'opérande immédiat comme deuxième facteur: celui-ci doit donc être déclaré comme tableau d'octets et remplacé par l'adresse de son identificateur.

En ce qui concerne l'instruction de comparaison COMP le positionnement du

code condition n'est pas le même en LP80. Ceci se traduit par des opérateurs de relation différents (voir Chapitre Instructions conditionnelles).

Exemple:

. MVC(5,BTAB(3),WTAB) ;	MOVE(R2,6,@BTAB+3, @WTAB SLLS 2) ;
. MVC(3,BTAB,"TOTO") ;	MOVE(R2,4,@BTAB,@STR01) ,
	avec ARRAY 4 BYTE STR01 = "TOTO" ;
. CLC (5,BTAB,"TOTO") ;	COMP(R2,4,@BTAB,@STR01) ;

3.2.1.5. Fonctions SLDL, SRDL

Syntaxe PL360

{ SLDL / SRDL } (<registre>,<nombre entier>)

Fonctionnement

Décalage logique double à gauche (SLDL) ou à droite (SRDL). <registre> spécifie une paire de registres pair-impair et doit contenir une adresse de registre pair.

Traduction

En LP80, il existe les opérateurs logiques équivalents SLLD et SRLD. Il n'y a pas de problème de traduction, puisque l'opération s'effectue dans le couple de registres.

Exemple:

PL360		LP80
SLDL(R4,15) ;	=>	R4 := R4 SLLD 15 ;

3.2.2. Arithmétique décimale

Fonctions PACK-UNPK

Syntaxe PL360

{ PACK / UNPK } (<nombre entier>,<nombre entier>,<mémoire>,<mémoire>)
l1 l2 d1(B1) d2(B2)

Fonctionnement

PACK (resp. UNPK): Le second facteur de longueur 12 octets situé à l'adresse d2(B2) est transformé de la forme dilatée (resp. condensée) en forme condensée (resp. dilatée) et le résultat est placé dans la zone indiquée par le premier facteur, sur 11 octets.

Traduction

L'IRIS80 fournit des instructions PACK et UNPK qui ont le même but (conversion de nombres décimaux du format dilaté en format condensé). Mais celles-ci s'appliquent dans des conditions différentes:

- elles utilisent les quatre registres RC à RF du groupe des registres généraux sous la forme d'un registre unique de 16 octets appelé totalisateur décimal ;

- une seule longueur est précisée dans ces instructions. C'est la longueur en octets du nombre décimal en format condensé à obtenir dans le totalisateur décimal pour PACK, en format dilaté contenu dans le totalisateur décimal pour UNPK.

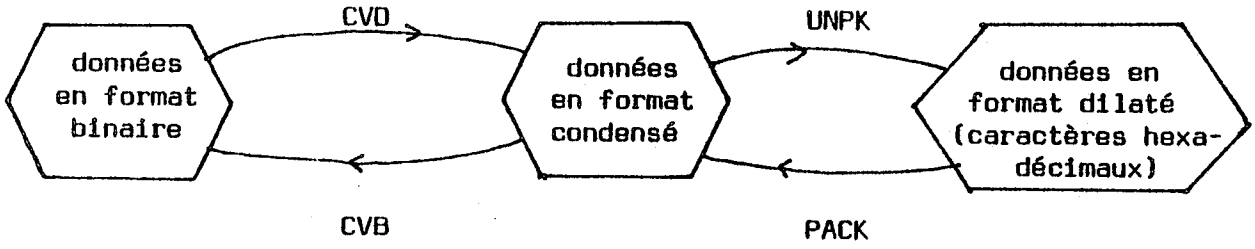
- un nombre décimal condensé doit occuper un nombre entier d'octets consécutifs. Dans le totalisateur décimal, l'instruction PACK ajoute un nombre suffisant de 0 en poids fort pour obtenir un nombre décimal représenté sur 16 octets. Ces considérations font qu'il est pratiquement impossible de traduire les instructions PACK et UNPK de l'IBM360, par les instructions correspondantes sur l'IRIS80.

Une deuxième possibilité est donc de simuler le fonctionnement de ces instructions sur l'IRIS80. Pour cela,on remarque que:

- dans des programmes de traitement de textes et de documentation (cadre dans lequel nous nous situons), PACK et UNPK ne sont utilisés que pour la conversion de données hexadécimales en données binaires et vice-versa.

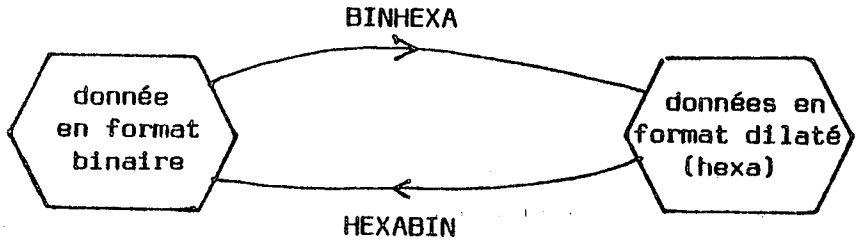
- ces instructions ne sont pas fréquemment utilisées. (Dans le système PIAF, par exemple, elles n'apparaissent qu'une fois: PACK dans LIGNELOG, UNPK dans ECRILOG).

En fait, la conversion des données n'est pas réalisée par la seule fonction PACK (ou UNPK) mais par le couple d'instructions PACK + CVB (ou CVD + UNPK), comme l'indique le schéma suivant:



Les fonctions (prédéclarées en PL360) CVB et CVD font partie de l'arithmétique à virgule fixe. L'IRIS80 dispose également d'instructions de conversion (conversion par addition et par soustraction) mais leur emploi est différent et se fait par référence à une table de conversion. De plus, les remarques précédentes s'appliquant également à ces fonctions, on choisit de traduire l'ensemble des instructions CVD + UNPK et PACK + CVB, par deux procédures de conversion locales. Lorsqu'on rencontre ces instructions dans un programme, il suffit donc de:

- remplacer ces instructions par l'appel à la procédure correspondante,
- ajouter au programme LP80, la déclaration de cette procédure.



Ces procédures sont définies de telle façon qu'elles respectent le positionnement des paramètres (en Entrée et en Sortie), qui sont utilisés par les fonctions PACK+CVB et CVD+UNPK.

3.2.3. Arithmétique à virgule fixe

3.2.3.1. fonctions LH-STH

Syntaxe PL360

$$\left\{\begin{array}{c}\text{LH}\\ \text{STH}\end{array}\right\}\ (\text{<registre>},\text{<mémoire>})$$

Fonctionnement

LH : Le demi-mot indiqué par le second facteur est étendu pour former un mot entier par projection de la valeur du bit signe dans les 16 positions d'ordre le plus élevé et placé dans le registre spécifié.

STH : Les 16 bits d'ordre inférieur du registre général spécifié sont transportés sans changement dans les positions de mémoire du second facteur.

Traduction

Comme pour les fonctions IC et STC (qui ont la même syntaxe) ces fonctions sont traduites par l'instruction d'affectation LP80 et ne font pas appel à une fonction prédéclarée (l'instruction assembleur IRIS80 est engendrée suivant le type de l'opérande).

Les problèmes posés par la traduction de LH et STH sont les mêmes que ceux posés par IC et STC, c'est-à-dire:

- si l'opérande possède le type mot ou octet, son adresse est transformée en adresse de demi-mot par décalage. Le demi-mot situé à cette adresse sera chargé par l'intermédiaire de l'identificateur standard SHORTO. Si l'opérande est déclaré comme demi-mot, il sera directement chargé par l'affectation dans le registre. Ces considérations restent valables pour la fonction STH.

- les 16 bits restants du registre ne sont pas affectés par l'instruction LH, ce qui est moins important que pour IC. En effet LH n'est pas utilisée en PL360 pour ranger deux demi-mots consécutivement dans un registre, mais seulement pour des opérations isolées de chargement de demi-mots dans un registre.

Exemple:

```
PL360:  LH (R4,IDW) ;
  LP80:     R2:=@IDW SLLS 1 ;   - chargement de l'adresse demi-mot dans R2 ;
            R4:=SHORTO(R2) ;    - chargement dans R4 du demi-mot situé à cette
                                  adresse.
PL360:  LH (R4,IDS) ;
  LP80:     R4:=IDS ;
```

3.2.3.2. fonction LM-STM

Syntaxe PL360

$$\left\{\begin{array}{l}\text{STM}\\ \text{LM}\end{array}\right\} \quad (\langle\text{registre}\rangle_{(1)},\langle\text{registre}\rangle_{(2)},\langle\text{mémoire}\rangle_{(3)})$$

Fonctionnement

LM (resp. STM) : L'ensemble des registres généraux commençant au registre spécifié par le premier facteur et finissant au registre spécifié par le deuxième facteur est chargé (resp. placé) à partir de la zone de mémoire désignée par l'adresse du troisième facteur.

Traduction

De même que pour les fonctions MVC et CLC, il existe, en LP80, des fonctions composées LOAD et STORE, ayant pour but de faciliter l'emploi des instructions de rangement et de chargement de registres multiples.

Les différences avec les fonctions LM et STM en PL 360 sont les suivantes:

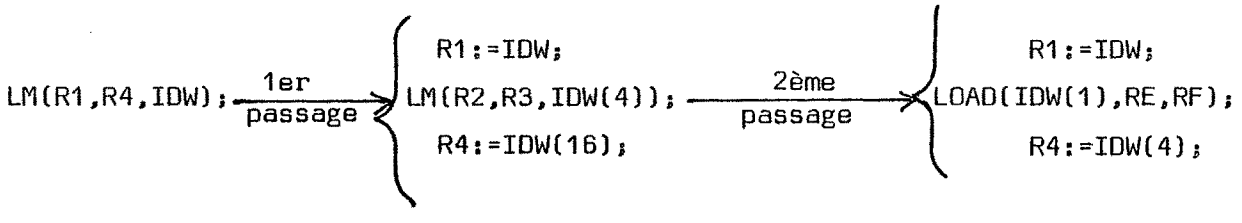
- l'adresse de la mémoire spécifiée doit être une adresse-mot. Lorsque c'est une adresse de demi-mots ou d'octets (tableau d'octets ou identificateurs standard B1, ..., B13), celle-ci doit être convertie en adresse de mots ;
- l'ordre des facteurs dans la syntaxe est inversé (ce qui nécessite le retour arrière de l'analyseur).

De plus, ces instructions utilisent un ensemble de registres dont seulement, le premier et le dernier sont spécifiés. Or la correspondance PL360-LP80 des registres a été définie de telle façon que, entre deux registres spécifiés, on ne trouve pas forcément les mêmes éléments.

Soit l'instruction LM (R1,R4,IDW) en PL360.

Ce sont les registres R1, R2, R3, R4 qui sont spécifiés par l'instruction. Or, le couple (R2,R3) est en correspondance avec le couple (RE,RF) du langage LP80, et par conséquent ce sont ces deux registres qui doivent être mentionnés dans la traduction de cette instruction.

La traduction par les fonctions STORE et LOAD doit être faite avec beaucoup de précautions. C'est pourquoi l'initiative est laissée à l'utilisateur, pendant le premier passage, pour controler ou modifier ces instructions, ou même éventuellement ajouter des instructions dans le fichier intermédiaire. Celui-ci modifiera l'instruction précédente en la décomposant en trois instructions qui pourront être traduites au deuxième passage.



Fonctions CVB-CVD (se reporter aux fonctions PACK-UNPK)

3.2.3.3. Fonctions SRDA-SLDA

Ces fonctions ont la même syntaxe et le même fonctionnement que les fonctions de décalage logiques SLDL et SRDL, la seule différence étant que le facteur contenu dans le couple de registres spécifié est considéré comme un nombre entier de 63 bits avec un signe. Le langage LP80 possède les opérateurs arithmétiques de décalage équivalents SLAD et SRAD. La traduction est donc identique à celle des fonctions SLDL et SRDL.

3.2.4. Instruction EXECUTE (EX)

Syntaxe PL360

EX (<registre>,<indicateur de fonction>)

Fonctionnement

L'instruction unique située à l'adresse de transfert est modifiée par le contenu du registre général spécifié, et l'instruction résultante est exécutée.

Traduction

Il existe, sur l'IRIS 80, une instruction EXU qui provoque l'exécution d'une instruction secondaire située à l'adresse effective contenue dans l'instruction.

En fait, l'emploi de la fonction EX en PL360 est restreint aux deux instructions MVC et CLC de transfert et de comparaison de caractères dans lesquelles la longueur spécifiée ne peut être qu'une valeur connue à l'exécution. Or ce problème est résolu en LP80, car les instructions MOVE et COMP ont la possibilité d'indiquer comme longueur un registre dont le contenu ne sera connu qu'à l'exécution.

Exemple:

PL360

```
EX(R4,MVC(0,BTAB,BTAB(1)));
```

LP80

```
MOVE(R2,R4+1,@BTAB,@BTAB+1);
```

4 - ADRESSAGE ET SEGMENTATION DES DONNEES

Le mécanisme d'adressage de l'IBM360, de même que le fonctionnement en mode C de l'IRIS80, sont tels que les instructions ne peuvent contenir des adresses que sous forme relative par rapport à la valeur d'une base.

La portée de l'adressage d'une base est limitée à 4 K octets en PL360 et 64 K octets en LP80. La segmentation du programme source PL360 peut donc être conservée en LP80.

Compte tenu de la limitation de l'adressage, il existe dans les deux langages:

- la possibilité de décomposer un programme en plusieurs parties correspondant généralement aux différentes phases d'un traitement ;
- la possibilité de segmenter les données.

4.1. Segmentation des programmes

Un programme PL360 ou LP80 peut se segmenter en:

- un segment principal,
- des segments secondaires.

- Un segment principal est une suite d'instructions pouvant référencer des données locales ou des données déclarées dans un bloc de données externes. Un segment principal peut appeler des procédures externes déclarées en tête du segment principal.

Correspondance entre les registres de base:

	PL360	LP80
Base du code	R15	B1
Base des données	R13	B2

En LP80, un segment principal se termine par un appel moniteur sous SIRIS8.

- Un segment secondaire est une procédure externe qui peut être appelé de la même façon qu'une procédure interne, soit dans un segment principal, soit dans un segment secondaire. L'appel d'un segment externe est identique à l'appel d'une procédure externe sauf que le compilateur LP80 engendre une séquence d'instructions permettant la restauration des registres de base.

Correspondance entre les caractéristiques d'un segment secondaire:

	PL360	LP80
Base du code	R15	B1
Base des données	R13	B2
Registre de retour	registre spécifié dans la déclaration	registre correspondant

4.2. Segmentation des données

4.2.1. Déclaration

La déclaration de blocs de données en PL360, comme en LP80, permet:

- soit le partage entre différents segments d'un même ensemble de données,
- soit la définition de blocs de données fictives permettant la gestion de structures à réalisations multiples.

En PL360 et en LP80, il existe trois sortes de déclarations de données:

Correspondance entre les déclarations:

	PL360	LP80
Données globales	segment base <registre>	globaldata . <identificateur><base
Données communes	common base <registre>	commondata <identificateur><base
Données fictives	dummy base <registre>	dummydata <identificateur><base>

Une zone de données globales permet de partager un ensemble de données entre différents segments. Elle doit être déclarée dans chaque segment qui utilise tout ou partie des données ainsi déclarées.

Une zone de données communes permet de déclarer un ensemble de données dans un ensemble logique. Les données ainsi déclarées peuvent être partagées entre plusieurs segments, mais contrairement à la zone de données globales, les données doivent être déclarées dans un autre segment, par une zone de données fictives.

La déclaration des blocs de données en LP80, impose:

- de nommer chaque zone de données au moyen d'un identificateur quelconque,
- de définir un registre de base.

4.2.2. Définitions et utilisation des registres de base

Dans le langage PL360, les registres de base n'étant pas différenciés des registres généraux, c'est donc un registre choisi entre R0 et R14 (R15 étant le registre de base du programme) qui permet l'adressage de ces données. Tandis que, dans le langage LP80, il existe des registres de base particuliers (notés B1 à B7).

Ceux-ci ne peuvent être traités comme des registres généraux, pour deux raisons:

- toutes les instructions utilisant des registres généraux ne sont pas autorisées avec des registres de base, en LP80 ;

- pour pouvoir adresser une donnée appartenant à un bloc défini à l'aide d'une zone de données globales ou fictives, le registre de base doit contenir une adresse de mot.

L'utilisation dans le programme PL360, d'un registre tantôt comme registre général, tantôt comme registre de base, empêche la traduction systématique du registre. Il revient à l'utilisateur de modifier l'instruction en cours d'analyse, en spécifiant le rôle du registre, pendant le premier passage.

Exemple:

Programme PL360	Programme LP80
→ déclarations de la zone de données	
DUMMY BASE R8 ;	DUMMYDATA ZONE B3
ARRAY NCHAIN LOGICAL CHAINCLAS ;	ARRAY NCHAIN WORD CHAINCLAS ;
ARRAY 10 LOGICAL MOTCLAS ;	ARRAY 10 WORD MOTCLAS ;
LOGICAL ADMAI SYN MOTCLAS ;	WORD ADMAI SYN MOTCLAS
→ utilisation (dans un même programme)	
utilisation de R8 comme base:	
R8 := RACINE ;	B3 := R3 := RACINE SRLS 2 ;
R6 := ADMAI ;	R6 := ADMAI ;
→ utilisation de R8 comme registre:	
R8 := 5 ;	R8 := 5 ;
R5 := R8 ;	R5 := R8 ;

- L'utilisateur choisit le registre de base B3 comme base correspondante de R8 dans le programme LP80. R8 étant déclaré comme registre de base dans le programme PL360, il supprime cet identificateur du dictionnaire.

- A la rencontre de R8 dans une instruction, il choisit entre modifier R8 en B3 (pour l'utilisation en tant que registre de base) ou le laisser inchangé (pour l'utilisation en tant que registre général).

- Les registres de base comme les registres généraux sont indexés une fois pour toutes, dans le dictionnaire. Pour éviter la confusion avec les identificateurs B3, B4 (=MEMO(R4)) pouvant exister dans le programme source, on a recours à un identificateur intermédiaire

exemple: indexation de B3 dans DICT

/ BASE3 (S) / BASE1 / B3 /

/ BASE3 (S) / BASE2 / B3 /

5 - PROCEDURES SYSTEMES

Elles se répartissent en trois groupes:

- des procédures d'entrées-sorties (écriture et lecture à la console ou à l'imprimante, etc...),
- des procédures de traitement de fichiers (ouverture, fermeture, lecture, écriture, etc...),
- des procédures de gestion de la mémoire (libération, allocation).

Ces procédures sont déclarées explicitement au début du programme comme procédures externes. Ces procédures exigent des paramètres en entrée et en sortie, passés par l'intermédiaire de certains registres (en général R0 et R1). Toutes ces procédures supposent que le registre R13 contienne l'adresse d'une zone de 18 mots dans laquelle le contenu des registres est sauvegardé et restauré.

En LP80, la plupart des macro-instructions de SIRIS8 sont inclues dans le compilateur LP80. Ces macro-instructions étant spécifiques à l'IRIS80, le positionnement des paramètres avant et après l'appel d'une macro-instruction, n'est pas forcément identique à celui d'une procédure système PL360, réalisant la même fonction.

De plus en LP80, certaines macro-instructions peuvent nécessiter la création de données en protection écriture. Si l'on veut utiliser les performances de l'accompagnateur LP80, d'une part, et par souci de clarté d'autre part, il convient d'isoler ces macro-instructions dans un programme externe, protégé en écriture, appelé "programme de service".

Pour chaque procédure système PL360, on définit sous la forme de point d'entrée dans ce programme externe une procédure LP80 utilisant la macro-instruction correspondante, suivant le schéma suivant:

- sauvegarde des registres généraux,
- positionnement des paramètres pour l'appel de la macro-instruction,
- appel de la macro-instruction,
- positionnement des paramètres restitués par la macro-instruction,
- restauration des registres généraux, à l'exception de ceux qui sont positionnés comme paramètres de sortie.

La sauvegarde et la restauration des registres généraux à l'entrée et à la sortie de chaque procédure de service sont imposées par la modification de ceux-ci, lors de l'appel de la macro-instruction.

Exemple: Traduction de la procédure système TYPE

La procédure système TYPE qui réalise l'écriture d'un message à la console, correspond à la macro-instruction :TYPE dans le langage LP80.

• positionnement des paramètres en PL360

R0 contient l'adresse de début du message à envoyer,
R1 contient le nombre d'octets du message à imprimer.
Le nombre total d'octets du message ne doit pas excéder 132 octets.
Aucun paramètre n'est restitué en sortie.

• positionnement des paramètres en LP80

:TYPE (MESS,adr)

adr est l'adresse-mot du message à envoyer. Ce message se présente sous la forme d'une suite de caractères, dont le premier octet donne le nombre d'octets à imprimer.
Le nombre total d'octets à imprimer ne doit pas dépasser 80.

• schéma de programme décrivant la procédure de service LP80, créée pour la traduction

On définit dans le programme de service, un tableau de 132 octets utilisé

comme buffer intermédiaire, déclaré de la façon suivante:

```
ARRAY 132 BYTE BREPONSE ;
WORD REPONSE SYN BREPONSE ;
```

On dispose des actions suivantes:

STORE(ADR,RO,RF):	sauvegarde des registres généraux à l'adresse ADR.
LOAD(ADR,RO,RF) :	restauration des registres généraux.
TRANSFERT(NB,REG):	transfert de NB (adresse octets) dans la zone BREPONSE, à partir du deuxième octet et rangement de NB dans le premier octet.

Avant l'appel de cette procédure dans un programme quelconque, RO contient l'adresse d'un tableau à imprimer et R1, le nombre d'octets à imprimer, conformément au positionnement des paramètres en PL360.

```
ENTRY TYPE ;
   SI R1 ≠ 0  ALORS
      |        STORE(ADR,RO,RF) ;
      |        SI R1 >80  ALORS
      |           |     TRANSFERT (80,RO) ;
      |           |     :TYPE (MESS,@REPONSE) ;
      |           |     R1 := R1-80 ;
      |           |     RO := RO + 80
      |        FINSI
      |        TRANSFERT (R1,RO) ;
      |        :TYPE (MESS,@REPONSE) ;
      |        LOAD(ADR,RO,RF) ;
   FINSI
```

La solution d'isoler les procédures systèmes dans un programme externe de service présente les deux avantages suivants:

- séparer les instructions spécifiques à chaque machine, des instructions indépendantes de la machine,
- laisser inchangé l'appel des procédures systèmes, le positionnement et la restitution des paramètres étant identiques.

Cette solution a néanmoins l'inconvénient de nécessiter une interface entre le programme où apparaissent l'appel de procédures systèmes d'une part et d'autre part, les procédures de service.

En effet, dans chaque programme, lors de l'appel d'une procédure système, le contexte est modifié. Plutôt que de sauvegarder celui-ci à chaque appel et le restaurer ensuite, on définit une procédure interne, mais identique pour chaque programme.

Celle-ci réalise:

- la sauvegarde du contexte à l'appel et la restauration au retour de la procédure. Le contexte est constitué par les bases utilisées dans le programme et le registre RF (modifié par l'appel de la macro-instruction) ;
- l'aiguillage vers la procédure de service correspondante.

TROISIEME PARTIE

RESULTATS

1 - RESULTATS

Le traducteur PIAFTRAD ainsi réalisé a permis de transporter sur IRIS80 les programmes constituant le module d'analyse morphologique, du logiciel PIAFDOC résidant sur IBM360.

Les critères retenus pour étudier les performances du traducteur sont les suivantes:

- comparaison du nombre d'instructions constituant chaque programme source et le programme cible correspondant,
- comparaison de la place occupée en mémoire par chaque programme source sur IBM360 et le programme cible correspondant sur IRIS80,
- temps t mis pour traduire chaque programme. Ce temps est obtenu de la manière suivante:

$$t = t_{CP} + t_{CMS} + t_R$$

avec

t_{CP} = temps de calcul

t_{CMS} = temps d'interaction

t_R = temps de réflexion de l'utilisateur.

A titre indicatif, on indique également le temps de calcul pour la traduction de chaque programme. Celui-ci est indiqué en secondes (les mesures ont été arrondies à l'unité supérieure ou inférieure).

Ces mesures ont été prélevées sur la traduction de quinze des dix-sept programmes constituant le module d'analyse morphologique, ainsi que sur les deux autres programmes CHDIGRAM et STDIGRAM qui ont été programmés directement sur l'IRIS80. En effet, ces programmes qui effectuent le chargement (CHDIGRAM) et la sauvegarde (STDIGRAM) des fichiers du dictionnaire et de la grammaire, contiennent essentiellement des instructions d'entrées-sorties et de traitement de fichiers. (On rappelle que ces instructions dépendant de la structure de chaque machine, ne peuvent être traduites de façon automatique).

Les mesures concernant CHDIGRAM et STDIGRAM ne sont donc pas relatives au traducteur PIAFTRAD, mais ont été utilisées pour comparer les totaux (nombre d'instructions et place en mémoire).

	nbre d'instr. PL360	nbre d'instr. LP80	PLACE MEMOIRE (OCTETS) DECL	INSTR.	TOTAL	PLACE MEMOIRE (OCTETS) DECL.	INSTR.	TOTAL	TEMPS DE TRADUCTION $t_{cp}+t_{cms}+t_r$	t_{cp} (secondes)
?IAFDOC	1100	1300	14400	7600	22000	15800	9300	25100	4 heures	70
[DEMORF	90	100	5200	530	5730	5290	910	6200	60 mn	6
[DGEN	600	1000	570	3340	3910	660	6310	6970	3 heures	35
?HOIGRAM	450	480	360	2610	2970	1130	2800	3930		
?FILTRE	50	70	260	360	620	320	680	1000	30 mn	3
.IRDOC	230	340	340	1030	1370	510	2550	3060	1 heure	14
.IGNELOG	200	210	730	1320	2050	800	1680	2480	60 mn	12
TDIGRAM	230	290	470	1180	1650	1100	2180	3280		
ICTIO	100	100	670	690	1360	770	1090	1860	60 mn	6
XTENS	720	990	900	4020	4920	960	6720	7680	3 heures	44
ODFRGL	90	100	210	600	810	280	920	1200	60 mn	6
RTHO	60	80	580	390	970	630	820	1450	30 mn	4
GEN	90	170	260	460	720	310	1400	1710	60 mn	6
CRILOG	140	140	930	780	1710	970	1100	2020	1 heure	8
ISTDICT	220	360	930	1120	2050	1070	3250	4320	2 heures	14
ISTARBRE	90	130	480	580	1060	560	1220	1780	2 heures	6
EORDICT	400	580	3750	4000	7750	4440	4640	9080	3 heures	24
TAL	4860	6360			60850			83110		

Ces mesures suscitent les remarques suivantes:

1/ Les déclarations LP80 sont plus importantes que les déclarations PL360.

Ceci s'explique par:

- la déclaration de tableaux supplémentaires de sauvegarde, spécifiques à LP80 (par exemple, sauvegarde des registres de bases),
- la déclaration de chaînes de caractères utilisés à la place des opérandes immédiats dans les instructions MOVE et COMP.

Exemple: Dans le programme PIAFDOC, 37 chaînes de caractères notées PIA01 à PIA37 sont définies de cette manière.

- la déclaration de tableaux de 800 caractères utilisés comme tampons pour le traitement des fichiers dictionnaire et grammaire...

2/ Le traducteur PIADTRAD produit une dilatation des instructions et par conséquent de la place mémoire occupée. Ceci s'explique, principalement, par:

- la traduction des fonctions logiques (XC, NC, OC, XI, NI, OI, TM) en plusieurs instructions LP80,
- l'utilisation des fonctions MOVE et COMP pour traduire les instructions PL360, MVC et CLC, ce qui ne produit qu'une instruction mais occupe une place mémoire plus importante,
- la traduction des instructions de traitement de base en au moins deux instructions,
- la conversion de toutes les adresses en adresses d'octets,
- l'utilisation de procédures réalisant l'interface entre les procédures services et les programmes.

N.B.- La différence en ce qui concerne le nombre d'instructions et la place mémoire des deux programmes STDIGRAM est assez importante bien que STDIGRAM n'ait pas été traduit par PIAFTRAD. Cette différence s'explique par l'insertion d'une procédure permettant la copie de fichiers, car il n'est pas possible d'appeler les procédures systèmes correspondantes, à partir d'un programme.

Un deuxième tableau met en évidence les points cités précedemment, à l'aide de quelques exemples:

INSTRUCTION PL360	TRADUCTION LP80	nombre d'instruct. LP80	PLACE MEMOIRE IBM360 (octets)	PLACE MEMOIRE IRIS8 (octets)
R5 := @WTAB ;	R5 := @WTAB SLLS 2 ;	1	4	8
WTAB := R12 ;	WTAB := R3 := B4 SLLS 2 ;	1	4	12
MVC(2,BTAB,WTAB) ;	MOVE(R2,3,@BTAB,@WTAB SLLS 2 ;	1	6	24
XC(79,WTAB,WTAB) ;	R3 := "XC" ; XX := R3 ; R3 := 79 ; XX(1) := R3 ; R3 := @WTAB SLLS 2; XX(2) := R3 ; R3 := @WTAB SLLS 2; XX(3) := R3 ; R2 := @XX ; INTERFACE ;	10	6	48
TM(1,RENS) ;	R3 := "TM" ; XX := R3 ; R3 := 1 ; XX(1) := R3 ; R3 := BMOTCLAS(R2 := 14);XX(2):=R3 ; R2:=@XX ; INTERFACE ;	8	4	36

Dans cet exemple, WTAB et BTAB désignent respectivement un tableau de mots et d'octets. RENS est un octet déclaré de la faqon suivante en LP80:

```
ARRAY 10 WORD MOTCLAS ;
BYTE RENS SYN MOTCLAS (14) ;
```

Comme on ne peut pas adresser une frontière d'octet en LP80, RENS est traduit par BMOTARB(R2:=14); (cf. Deuxième partie).
Le registre R12 dans l'instruction WTAB:=R12; est utilisé comme registre de base.

3/ Le temps mis pour traduire un programme varie donc en fonction du nombre d'instructions et des traitements à effectuer sur ce programme (en particulier, traitement des bases, indexation des déclarations). Par exemple, le programme LISTARBR comporte peu d'instructions (90), mais effectue des opérations à l'aide de registres de bases, ce qui ralentit le fonctionnement du traducteur.

Les mesures concernant le temps de traduction sont données à titre indicatif, la traduction des programmes ayant été réalisée une seule fois. On constate que le temps de calcul est de l'ordre de 6 secondes pour la traduction de 100 instructions.

Ce temps peut être amélioré, en adoptant certaines mesures relatives au langage source, ou en diminuant l'interaction du traducteur. Ces deux points sont développés dans la suite.

II - PERSPECTIVES D'OPTIMISATION

1 - Relatives au langage

Compte tenu de la dilatation importante créée par la traduction des instructions, on peut envisager certaines mesures intervenant dans le langage PL360, qui permettraient d'optimiser la traduction.

Parmi ces mesures on peut citer les suivantes:

- ne pas déclarer d'éléments synonymes de frontières d'octets et éviter l'utilisation des octets dans un programme, lorsque c'est possible ;

- utiliser des registres de base uniquement comme base à l'intérieur d'un programme,

- utiliser les opérateurs PL360, AND, OR et XOR qui possèdent un équivalent en LP80, au lieu d'utiliser les fonctions XI, NI, OI,

- ne pas utiliser la fonction XC pour remettre à zéro un tableau de caractères et plus généralement réaliser l'intersection, l'union ou l'exclusion à l'aide des opérateurs AND, OR et XOR et de boucles, plutôt que d'utiliser les fonctions logiques XC, OC, NC,

- utiliser des registres de travail pour la comparaison ou le transfert de chaînes comportant au plus quatre caractères.

Ces considérations conduisent à exhiber un langage intermédiaire, sous-ensemble du langage PL360, tenant compte des particularités du langage LP80.

2 - Relatives aux systèmes

L'objectif que nous nous étions fixés en réalisant le traducteur PIAFTRAD était de pouvoir traduire rapidement les programmes du logiciel PIAFDOC, à la demande du BNIST (Bureau National de l'Industrie des Sciences Techniques). Compte tenu de ces impératifs de temps, certaines phases de traitement qui auraient pu être réalisées par programme, ont été laissées au contrôle de l'utilisateur. Nous indiquons ici les principales modifications pouvant être envisagées pour réduire l'interaction, ainsi que les problèmes posés par ces modifications et quelques indications pour les résoudre.

2 - 1. Automatisation totale du traitement des déclarations

Actuellement, le module d'indexation permet d'indexer automatiquement les déclarations fournies par l'utilisateur, mais la constitution du fichier des déclarations LP80 est laissée à la charge de l'utilisateur.

Il est néanmoins possible de traiter de façon automatique les déclarations, c'est-à-dire, d'une part, d'indexer les éléments ainsi déclarés et d'autre part, de traduire les déclarations.

Les principaux problèmes posés par ce traitement sont:

a/ traitement des données synonymes

Un élément (demi-mot ou octet) déclaré comme synonyme d'une frontière d'octets dans une structure de données PL360 ne peut être traduit en LP80 que par l'intermédiaire d'un tableau de demi-mots ou d'octets synonyme de la structure de données (cf. Deuxième partie).

Lorsqu'il rencontre un tel élément, le programme doit donc:

- reconnaître qu'il s'agit d'une donnée de type synonyme,
- fournir un "synonyme" soit par convention, soit par question à l'utilisateur,
- ne pas traduire cette déclaration en LP80,
- insérer dans le fichier déclaration, l'élément utilisé comme synonyme de la structure.

b/ traduction des déclarations

La syntaxe des déclarations PL360, différant de la syntaxe des déclarations en LP80 (en ce qui concerne les initialisations, déclarations de base, etc...) les règles s'appliquant à l'analyse des déclarations compliquent la grammaire de traduction.

Pour résoudre ce problème, on peut envisager la création d'une grammaire propre au traitement des déclarations (indexation et traduction). Lorsque le traitement est terminé, celle-ci est remplacée par la grammaire et le dictionnaire spécifiques à la traduction des instructions.

c/ simulation de la gestion des registres de base

Les déclarations de sections de données utilisant un registre de base ne sont pas traduites automatiquement, actuellement, pour laisser à l'utilisateur le choix du registre de base (B3, B4, B5, B6, B7) dans la traduction LP80.

Cette opération peut être réalisée automatiquement en simulant la gestion des registres. En particulier, il suffit de savoir quels sont les registres de base

déjà utilisés. Il arrive que le programme source PL360 utilise plus de cinq registres de base qui est le nombre maximal de bases disponibles en LP80. Il est difficile de résoudre ce problème automatiquement, puisqu'il faut avoir une vue d'ensemble du programme et connaître l'utilisation des registres de base dans celui-ci.

Remarque:
L'indexation et la traduction automatiques des déclarations permettraient de résoudre le problème d'ambiguité posé par la déclaration d'identificateurs de même nom et de type différent dans des blocs de même niveau. Si le module d'indexation découvre que l'identificateur à indexer est déjà dans le dictionnaire mais a été déclaré avec un type différent, il modifiera celui-ci afin de l'indexer convenablement.

2 -2. Réalisation d'un seul module de traduction

La coexistence des deux modules (adéquation et traduction) est justifiée principalement par le traitement des registres de base et l'indexation d'éléments dans le dictionnaire que l'utilisateur réalise à l'aide du module d'indexation.

En fait, un seul module fonctionnant en deux étapes (contrôle et traduction) peut être réalisé, l'enchaînement des deux étapes étant effectué de façon automatique.

Pour cela, il faut modifier certains traitements:

- le traitement des instructions MVC et CLC avec opérandes immédiats, peut être réalisé automatiquement: le système remplace l'opérande immédiat par un nom interne et indexe cet élément dans le dictionnaire.

- l'ambiguité relative à l'utilisation des registres de base ne peut pas être résolue automatiquement. Néanmoins, l'indexation des registres effectuée par l'utilisateur entre les deux passages, peut être faite automatiquement par le traducteur qui doit:

. supprimer du dictionnaire les registres correspondants afin de provoquer l'arrêt du transducteur sur la base,

. indexer ces registres dans le dictionnaire entre les deux passages (cf. Deuxième partie).

N.B. - L'ambiguité de l'utilisation des registres de base peut être supprimée dans le programme source, si les registres définis comme bases dans le programme source, ne sont effectivement utilisés qu'en tant que bases dans le programme.

CONCLUSION

La création et la mise au point du traducteur PIAFTRAD a nécessité six mois de travail pour une personne. Compte tenu de cet investissement, on peut se demander si la réécriture des programmes constituant PIAFDOC, directement sur IRIS80, n'aurait pas été plus rapide et si l'investissement peut se justifier réellement.

En ce qui concerne la réécriture des programmes, cette méthode aurait peut-être été plus rapide, quoique ce ne soit pas certain, compte tenu du fait qu'il aurait fallu mettre au point sur IRIS80, environ cinq mille instructions.

La réalisation de cette méthode de transport permet de constater que le transducteur général d'états finis est suffisant pour la traduction du langage PL360 dans le langage LP80 sauf pour certaines instructions (traitement des bases, ou instruction choix pour laquelle il suffit d'avoir un compteur pour résoudre le problème) qui obligent à connaître le contexte.

Néanmoins, le traducteur, outre le transport du logiciel PIAFDOC, possède des avantages incontestables:

- il permet de réaliser sur IRIS80, d'autres applications programmées en PL360, implantées sur IBM360. De plus, comme il n'est pas indispensable de connaître le langage LP80 et par conséquent l'ordinateur IRIS80, tout utilisateur peut transporter sur IRIS80 les programmes écrits en PL360, à l'aide du traducteur PIAFTRAD.
D'autre part, les programmes constituant le système PIAFDOC étant modifiés et mis à jour sur IBM360, le traducteur permet d'obtenir très rapidement les versions correspondantes sur IRIS80.

- il permettra de réaliser directement le transport sur HB64 de programmes écrits en PL360, lorsque les travaux de M.BENNETT [BENN] seront achevés.
En effet, ces travaux consistent à construire un compilateur du langage LP72 similaire au langage LP80 en code machine HB64.

- des langages de type "PL" étant disponibles sur d'autres machines (PL11 sur PDP11, PL16 sur T1600, etc...), on peut envisager l'extension de ce traducteur vers d'autres matériels.

D'autre part, pour répondre à l'argument souvent répété selon lequel il est préférable de réécrire les programmes parce que ceux-ci peuvent être améliorés simultanément, on peut dire que le traducteur n'offre certainement pas une version optimisée mais permet d'obtenir rapidement une version qui peut être disponible tandis qu'une autre version peut être réécrite simultanément.

BIBLIOGRAPHIE

[ANSI] American National Standard Institute
FORTRAN - 1966

[BENN] BENNETT M.
Note interne CII-HB

[CHAS] CHASSAGNE C.
"Contribution à la transportabilité des programmes: une méthode de traduction assistée"
Rappport d'avancement de contrat - mars 1978

[CHASQ] CHASSAGNE C. - QUINT V.
"Exploitation de logiciel réparti sur deux ordinateurs: transport de programmes écrits en langage semi-évolué et utilisation d'un réseau"
Congrès AFCET, PARIS, novembre 1978

[COUGD] COURTIN J, DUJARDIN D., GRANDJEAN E.
"Editeur lexicographique pour les langues naturelles"
Document interne - Grenoble 1976

[COUD] COURTIN J., DUJARDIN D.
"Paramètres linguistiques de la morphologie française dans le système PIAF"
Document interne, GRENOBLE, 1976

[COU1] COURTIN J.
"Algorithmes pour le traitement interactif des langues naturelles"
Thèse d'Etat - GRENOBLE, octobre 1977

[COU2] COURTIN J.
"Organisation d'un dictionnaire pour l'analyse morphologique"
Séminaire de théorie des automates et de traitement automatique des langues - GRENOBLE - février 1973

[COVO] COURTIN J., VOIRON J.
"Modèle pour l'apprentissage de la programmation"
RAIRO - Informatique Vol. 11, n°1, 1977

[FLOYD] FLOYD
"Non deterministic algorithms"
JACM 14, octobre 1967

[FOR1] Brochure FORTRAN IBM 360
Note interne IBM

[FOR2] Brochure FORTRAN IRIS50
Note interne CII-HB

[GOBA] GOLOMB S. et BAUMERT L.
"Backtrack programming"
JACM 12(1965), pp. 516/524

[GRAN] GRANDJEAN E.
"Application d'un système de traitement de langues naturelles pour l'indexation automatique"
avril 1978

[IR80] CII - Ordinateur IRIS 80
Manuel d'utilisation

[I360] IBM - Ordinateur IBM360
Principles of operations

[NIFARE] NIEVERGELT - FARRAR - REINGOLD
"Computer approaches to mathematical problems"
Prentice Hall

[PEQU] PEQUIGNOT M.
"LP80 - Metteur au point"
Manuel d'utilisation

[POOL] POOLE P.C.
"Portable and adaptable compilers"
Advanced course on software engineering - 1974

[POWA] POOLE P.C., WAITE W.M.
"Portability and adaptability"
Advanced course on software engineering - 1973

[VEIL] VEILLON G.
"Algorithmes combinatoires"
Cours d'Université de Grenoble - 1975

[WAIT] WAITE W.M.
"The mobile programming system: STAGE2"
CACM 13 - July 1970

[WISU] WIRTH - SUTY
"Le langage PL360"
Janvier 1971.

ANNEXES

ANNEXE 1

GRAMMAIRE DE TRADUCTION PL360 - LP80

```
/P:=CM,CL,CB,CD;
1:=1,2,3,4,5,6,7,8,9,0,10,11,14,12,13,15,16,17,18,19,20,21,22,23,24,25,26,27,
),31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,
5,57,58,59,60,61,62,63,64,65,28,29;
D:=DICT,SYNO,=1,=0,<,<=,=,¬=,>,>=,AND,OR,INTG,STRG,CCMM,DO,ASS,DIV2,DIV4,INTS,
ITW,SET,RSET,:=R1,:=,);,+,POIN,Q001,):=,PFPV,ADRS,MUL2,MUL4,Q002,Q003,Q004,
/A2,DVA4,DVL2,DVL4,OCT,DEMI,R1:=,$MOT,,$MTA,),:=RF,%,(RE,$(RE,@,INTM,VGAD,$VAD,
.D,SRLD,SLAD,SRAD,|,XOR,TEST,$(R2,(R2,:=R2,R2:=,:=R3,+1,SEQU,TRUE,FALS,R3,
PVV,$DEF,QUST,$COP,$RST,$SET,(,PVRG,QUO1,QUO2,PL1A,PL1V,DVP4,ASSR,=*,TRAD,DVE4,
SE,E,L,PKNT,OF,SPEC,COND,TRU1,TRU2,FAL1,FAL2;
G11:CD:=CD(G)||$(R2;VAL:=();SAT:=();ND.
M1:CD:=CD(G)||CD(D);CM:=CM(G);/AL:=(PG11);SAT:=().
G24:CD:=CD(G)||+1;CM:=CM(G);VAL:=();SAT:=().
G6:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(VG24);SAT:=().
.M2:CD:=CD(G)||%;CM:=CM(G);VAL:=(PG1);SAT:=().
FP2:CD:=CD(G)||DVE4;VAL:=();SAT:=();FIM.
G0:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
IC:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
KT:CD:=CD(G)||PKNT;CM:=CM(G);VAL:=();SAT:=().
TER:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
ST1:CD:=CD(G)||TRU1;VAL:=();SAT:=();FIM.
GB8:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
EG:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(BLA);SAT:=().
G2:CD:=CD(G)||PVRG;CM:=CM(G);VAL:=(WD03,NBR6);SAT:=().
ST2:CD:=CD(G)||FAL1;VAL:=();SAT:=();FIM.
)00:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
400:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
FF:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(NBD);SAT:=().
FB:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
)9:CD:=CD(G)||$SET;CM:=CM(G);VAL:=();SAT:=().
/00:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
/FL:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
.A:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
TOP:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
)02:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
)01:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(PST1,PST2,PBL3,PBL4);SAT:=().
SLR:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
R00:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
)MM:CD:=CD(G)||COMM;VAL:=(BLA);SAT:=();FIM.
BCD:CD:=CD(G)||COND;CM:=CM(D);VAL:=();SAT:=().
D10:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(VRG1,VRG2);SAT:=().
INA:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(NBD);SAT:=().
BP3:CD:=CD(G)||);CM:=CM(G);VAL:=();SAT:=().
RCD:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
DTS:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(POCS);SAT:=().
NA1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBD);SAT:=().
IL:CM:=CM(G);VAL:=();SAT:=().
```

```
SHO9:CD:=CD(G)||MUL2;CM:=CM(G);VAL:=(PGB);SAT:=().
NBR1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
LOGI:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBD);SAT:=().
DECA:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
BYO9:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(PGB);SAT:=().
NBD:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(BLA,NBD);SAT:=().
VG16:CD:=CD(G)||:=;VAL:=();SAT:=();FIM.
VG17:CD:=CD(G)||SLLD;CM:=CM(G);VAL:=(BLA);SAT:=().
PG1:CD:=CD(G)||%;CM:=CM(G);VAL:=(RG1);SAT:=().
PAS:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
REAL:CD:=CD(G)||L;CM:=CM(G);VAL:=();SAT:=().
DXPT:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
CDTW:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(POCW);SAT:=(PMD4).
RGS:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(BINS,PAD2,PLD2);SAT:=().
RGW:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(BINW,PLD4,PMD4,PWD4);SAT:=().
UNA2:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBR4);SAT:=().
SYNO:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
RIB3:CD:=CD(G)||%;CM:=CM(D);VAL:=();SAT:=().
ALF3:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
LONG:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
NBOW:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBOW,PLD4,PMD4,PWD4);SAT:=().
NBOS:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBOS,PAD2,PLD2);SAT:=().
NBRW:CD:=CD(G)||INTW;CM:=CM(G);VAL:=(NBRW,PD3,PD4);SAT:=().
BINS:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBOS);SAT:=().
BINW:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBOW);SAT:=().
POCS:CD:=CD(G)||ASS;CM:=CM(G);VAL:=(RGS,NBS);SAT:=().
POCW:CD:=CD(G)||ASS;CM:=CM(G);VAL:=(RGW,NBRW);SAT:=().
POCB:CD:=CD(G)||ASS;CM:=CM(G);VAL:=();SAT:=().
PG4:CD:=CD(G)||:=;CM:=CM(G);VAL:=(BLA);SAT:=().
BYO8:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
ABS8:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
ADR:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
AB31:CD:=CD(G)||%;CM:=CM(D);VAL:=();SAT:=().
RG1:CD:=CD(G)||%;CM:=CM(G);VAL:=(BLA);SAT:=().
VG1:CD:=CD(G)||ADRS;CM:=CM(G);VAL:=(BLA);SAT:=().
INIT:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
SX:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
BYO1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(PLD2);SAT:=().
AB10:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
AB20:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
RBAF:CD:=CD(G)||R3;CM:=CM(D);VAL:=();SAT:=().
PD7:CD:=CD(G)||%;CM:=CM(G);VAL:=(BLA);SAT:=().
WDO1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=();ND.
WDO2:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(PGB);SAT:=().
AFF1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
PTVG:CD:=CD(G)||POIN;CM:=CM(G);VAL:=();SAT:=();FIM.
BAS3:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
VG18:CD:=CD(G)||SRLD;CM:=CM(G);VAL:=(BLA);SAT:=().
PD8:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
ABS2:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(PGB);SAT:=().
VG19:CD:=CD(G)||SLAD;CM:=CM(G);VAL:=(BLA);SAT:=().
GDP3:CD:=CD(G)||SPEC;CM:=CM(G);VAL:=();SAT:=().
BYO3:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
VG20:CD:=CD(G)||SRAD;CM:=CM(G);VAL:=(BLA);SAT:=().
RG5:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(VG17,VG18,VG19,VG20);SAT:=().
VG21:CD:=CD(G)|||;CM:=CM(G);VAL:=();SAT:=().
WSBM:CD:=CD(G)||:=R3;CM:=CM(G);VAL:=(PTVG);SAT:=().
PD1:CD:=CD(G)||%;CM:=CM(G);VAL:=(BLA);SAT:=().
WSM1:CD:=CD(G)||OCT;VAL:=();SAT:=();FIM.
RBS5:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=(PLD4,PLD2,PD4,PMD4).
VG22:CD:=CD(G)||AND;CM:=CM(G);VAL:=();SAT:=().
WDO4:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
BEG:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
SHO1:CD:=CD(G)||CD(D);VAL:=();SAT:=();ND.
RIB1:CD:=CD(G)||%;CM:=CM(D);VAL:=(BLA);SAT:=()
```

```
P1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
P2:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
C1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
TH:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
I1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
I2:CD:=CD(G)||=*;CM:=CM(D);VAL:=();SAT:=().
10:CD:=CD(G)||R3;CM:=CM(D);VAL:=();SAT:=(STOP).
11:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=(BINA).
55:CD:=CD(G)||R2:=;VAL:=();SAT:=();FIM.
D2:CD:=CD(G)||NUL2;CM:=CM(D);VAL:=(PGB);SAT:=().
B:CD:=CD(G)||+;CM:=CM(G);VAL:=(NBD);SAT:=().
N3:CD:=CD(G)||PDTN;VAL:=();SAT:=();FIM.
D3:CD:=CD(G)||DCT;VAL:=(PTVG);SAT:=().
2:CD:=CD(G)||%;CM:=CM(D);VAL:=(PD3);SAT:=().
3:CD:=CD(G)||R2:=;CM:=CM(G);VAL:=();SAT:=().
D5:CD:=CD(G)||PDTN;VAL:=();SAT:=();FIM.
2:CD:=CD(G)||:=;CM:=CM(G);VAL:=(BLA);SAT:=().
D4:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
12:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
08:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
3:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
G1:CD:=CD(G)||TRAD;CM:=CM(G);VAL:=();SAT:=().
7:CD:=CD(G)||R3;CM:=CM(G);VAL:=();SAT:=().
8:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(STOP,BINA);SAT:=().
P4:CD:=CD(G)||TRAD;VAL:=();SAT:=();FIM.
06:CD:=CD(G)||DEMI;VAL:=(PTVG);SAT:=().
S:CD:=CD(G)||INTS;CM:=CM(G);VAL:=(PDD5,PD3,NBS,PDD1,PDD2);SAT:=().
R1:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(SH09,BY09,WD01,WD04,SH01,BSYN);SAT:=().
9:CD:=CD(G)||QUI2;CM:=CM(G);VAL:=();SAT:=().
AD:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
D4:CD:=CD(G)||DVL4;CM:=CM(G);VAL:=(BLA);SAT:=().
D2:CD:=CD(G)||DVA2;CM:=CM(G);VAL:=(BLA);SAT:=().
D2:CD:=CD(G)||DVL2;CM:=CM(G);VAL:=(PBP1,PBP2,PBL1,PBL2);SAT:=().
5:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
03:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
06:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(PBP5);SAT:=().
4:CD:=CD(G)||);VAL:=();SAT:=();FIM.
26:CD:=CD(G)||%;CM:=CM(D);VAL:=();SAT:=().
L4:CD:=CD(G)||FAL2;VAL:=();SAT:=();FIM.
B3:CD:=CD(G)||MUL4;CM:=CM(D);VAL:=();SAT:=().
6:CD:=CD(G)||%;CM:=CM(G);VAL:=(BLA);SAT:=().
G1:CD:=CD(G)||PL1A;CM:=CM(G);VAL:=(SH09,BY09,WD01,WD04,SH01);SAT:=().
08:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
5:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
S1:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=(BY08).
G2:CD:=CD(G)||PL1V;CM:=CM(G);VAL:=(AB22);SAT:=().
8:CD:=CD(G)||PFPV;VAL:=();SAT:=();FIM.
B1:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=();FIM.
3:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
7:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(RG3);SAT:=().
6:CD:=CD(G)||:=R2;CM:=CM(G);VAL:=(BLA);SAT:=().
S7:CD:=CD(G)||$MOT;VAL:=();SAT:=();FIM.
S2:CD:=CD(G)||%;CM:=CM(D);VAL:=();SAT:=(ABS8,WD08,SH08).
S3:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=(SH08).
S4:CD:=CD(G);CM:=CM(D);VAL:=();SAT:=(RGB8,BY08,ABS8,WD08).
B2:CD:=CD(G)||NUL4;CM:=CM(G);VAL:=();SAT:=().
9:CD:=CD(G)||PDTN;CM:=CM(D);VAL:=(RGB8,BY08,ABS8,WD08,SH08);SAT:=().
10:CD:=CD(G)||PDTN;VAL:=();SAT:=();ND.
12:CD:=CD(G)||R2:=;CM:=CM(D);VAL:=(ABS2,RGB3,RGB1);SAT:=().
F3:CD:=CD(G)||ASSR;CM:=CM(G);VAL:=();SAT:=().
D4:CD:=CD(G)||DVL4;VAL:=();SAT:=();FIM.
8:CD:=CD(G)||ASSR;CM:=CM(G);VAL:=(BLA);SAT:=().
D4:CD:=CD(G)||DVA4;CM:=CM(G);VAL:=(BLA);SAT:=().
40:CD:=CD(G)||DVA4;CM:=CM(G);VAL:=(PG9);SAT:=().
```

```
BIN2:CD:=CD(G)||%;CM:=CM(G);VAL:=(NBR2);SAT:=().
RG4:CD:=CD(G)||%;CM:=CM(G);VAL:=(PD1,BIN2);SAT:=().
RB40:CD:=CD(G)||DVA4;CM:=CM(D);VAL:=(PG9);SAT:=().
PDD1:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
PDD2:CD:=CD(G)||);VAL:=();SAT:=();FIM.
NUL1:CM:=CM(G);VAL:=();SAT:=().
PG10:CD:=CD(G)||$(R2;CM:=CM(G);VAL:=(WD10,NBR3);SAT:=().
NBR3:CD:=CD(G)||INTM;CM:=CM(G);VAL:=();SAT:=().
VG14:CD:=CD(G)||$VAD;CM:=CM(G);VAL:=(SH09,BY09,WD01,WD04,SH01,BSYN);SAT:=().
VG15:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(AB22);SAT:=().
NUL2:CM:=CM(G);VAL:=();SAT:=().
NUL3:CM:=CM(G);VAL:=();SAT:=().
NUL4:CM:=CM(G);VAL:=();SAT:=().
NUL5:CM:=CM(G);VAL:=();SAT:=().
NUL6:CM:=CM(G);VAL:=();SAT:=().
NUL7:CM:=CM(G);VAL:=();SAT:=().
NUL8:CM:=CM(G);VAL:=();SAT:=().
PDD3:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
STP1:CD:=CD(G)||DVP4;VAL:=();SAT:=();FIM.
PBP1:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
PBP2:CD:=CD(G)||PFPV;VAL:=();SAT:=();FIM.
RBS8:CD:=CD(G)||CD(D);VAL:=(AFF4);SAT:=().
CORD:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
TOCT:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(BOO2);SAT:=().
COMP:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBR4);SAT:=().
NBR4:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=(NBR4);SAT:=().
TCC:CD:=CD(G)||TEST;CM:=CM(G);VAL:=();SAT:=().
AFF4:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
PDD6:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
NBR6:CD:=CD(G)||%;CM:=CM(G);VAL:=(NBR6);SAT:=().
AFF5:CD:=CD(G)||ASSR;CM:=CM(D);VAL:=();SAT:=().
BSYN:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(PGB);SAT:=().
PBP5:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
DECL:CD:=CD(G)||DF;CM:=CM(G);VAL:=();SAT:=();FIM.
NBRO:CD:=CD(G)||CD(D);CM:=CM(D);VAL:=();SAT:=().
BIN:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=(NBD);SAT:=().
PBL1:CD:=CD(G)||ELSE;VAL:=();SAT:=();FIM.
RBFN:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
AB22:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
PG14:CD:=CD(G)||$PVV;CM:=CM(G);VAL:=();SAT:=().
PBL2:CD:=CD(G)||CD(D);VAL:=();SAT:=();FIM.
SY03:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
PPB2:CD:=CD(G)||DVL2;CM:=CM(G);VAL:=();SAT:=().
CTOO:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
RBSH:CD:=CD(G)||$DEF;CM:=CM(D);VAL:=();SAT:=().
NBW:CD:=CD(G)||INTW;CM:=CM(G);VAL:=();SAT:=().
DCL:CD:=CD(G)||%;CM:=CM(G);VAL:=();SAT:=().
HALT:CD:=CD(G)||%;VAL:=();SAT:=();FIM.
VG5:CD:=CD(G)||CD(D);CM:=CM(G);VAL:=();SAT:=().
WD05:CD:=CD(G)||MUL2;VAL:=(PBP1,PBL1);SAT:=().
RBS7:CD:=CD(G)||R3;CM:=CM(D);VAL:=(POCW);SAT:=(PD3,PLC4,PD4,PMD4,PWD4).
PPB1:CD:=CD(G)||DVL4;CM:=CM(G);VAL:=();SAT:=().
VALOO:=(RTB,COMM,RBCD,RTB3,AB31,AB10,RBAF,RBS5,RTB1,RBS1,RBS2,RBS3,RBS4,RB40,
RBS8,RBFN,RBSH,RBS7)
CM1:=(REG,BLA,STOP,CDTS,CDTW,UNA2,AB31,AB10,BAS3,GOP3,GOP1,DOTH,AB40,CORD,TOCT,
COMP,BIN)
CM2:=(BLA,BOO1,PG1)
CM3:=(AFF,BLA,VG8)
CM4:=(WDOO,BLA,NBD,PG4,VG10)
CM5:=(BLA,STOP,POCW,AFF5)
CM6:=(BLA,STOP,NBD,PD1,PGB,PDD1,VG14,VG15,PBP2,PBL2)
CM8:=(FIC,REG,BLA,STOP,CDTS,UNA1,LOGI,DECA,PAS,CDTW,ADR,AB31,SX,AB10,GOP3,GOP1,
AB40,BIN)
CM9:=(BLA,PAS,DOTH)
```

```
11:=(BLA,STOP)
14:=(BLA,STOP,TRAP)
12:=(REG,BLA,STOP,SYO3,VG5)
13:=(BLA,PD2)
15:=(PKT,BLA,STOP,UNA1,NBD,REAL,LONG,AFF1,BIN)
16:=(BLA,PG1,PD5,BIN2,PDD6,NBR6)
18:=(REG,BLA,VG16,PG1)
19:=(BLA,STOP,NBD,PG1,RG5,PD1)
20:=(REG,BLA,DECL,BIN)
21:=(WDOO,BLA,NBRW,RG1,VG6,PG5,ABS7,PBP5)
22:=(REG,BLA,PG1,VG7)
23:=(RTB,BLA,AB31,AB10,RBS5,RTB1,RBS2,RBS3,RBS4,RB40,RBFN,RBSH)
24:=(REG,BLA,PRCD,PRC1,PD3,PG5,HALT)
25:=(BLA,STOP,NBD,RG1,ABS5,PGB,PLD4,VG6,PBP1,PBL1,AB22)
26:=(BLA,NBR1,GOP3,GOP1,PVG1,PBP4,ADR1,RGB1,RGB2,BIN,AB22,CTOO)
27:=(BLA)
30:=(BLA,STOP,PG1,VG1,WDO1,WDO2,PD1,SHO1,SHO2,VG12)
31:=(REG,BLA,STOP,NBD,PD1,BIN)
32:=(BLA,PTVG,GOP3,BYO3,GOP1,WSO3,VG2,RG9,PBP1,PBL1)
33:=(REG,BLA,STOP,NBD,PDCB,PD2,PDD1,BIN)
34:=(BLA,PG1,RG1,ABS2,VG3)
35:=(REG,BLA,STOP,NBD,VG1,BYO1,WDO1,WDO4,PGB,BIN,PPB2,WDO5)
36:=(REG,BLA,STOP,NBD,PGB,VG3,PLD2,RGB1,RGB2,BIN,AB22,PPB2)
37:=(REG,BLA,PG1,VG2,SHO4,WBO6)
38:=(AFF,BLA,STOP,POCS,PBP1,PBL1)
39:=(RTB,BLA,RTB3,AB10,RBAF,RBS5,RTB1,RBS1,RBS2,RBS3,RBS4,RB40,RBFN,RBSH,RBS7)
40:=(BLA,PG1,VG1,WDO1,WDO2,SHO1,SHO2,VG26)
41:=(BLA,PG1,BYO6,VG6,VG12)
42:=(REG,BLA,PG4,VG9)
43:=(REG,BLA,NBD,PDCB,PDD2,BIN)
44:=(BLA,CDTS,PDCS,VG6,PBP5)
45:=(BLA,PDCS,PD4,PBP5)
46:=(BLA,PG1,GOP2,BYO6,VG6,NBR6)
47:=(BLA,PG1,VG1,WDO1,WDO2,SHO1,GOP2,SHO2,VG12,NBR6)
48:=(BLA,NBD,GOP3,GOP1,VG9,PG8)
49:=(AFF,BLA,PDCW)
50:=(REG,BLA,NBD,PD8,PGB,BIN)
51:=(REG,BLA,STOP,PG1,PD1,PRC1)
52:=(REG,BLA,PG1,BYO3,GOP2,WSO3,VG2,NBR2)
53:=(BLA,PG1,VG1,WDO1,WDO2,PD1,SHO1,GOP2,SHO2,VG12,NBR2)
54:=(CLM1,PG1,RG1,VG6)
55:=(RG6,CLM2,PTTR,STOP,PBP3,PD1,PGB,VG14,VG15,NBR6)
56:=(WDOO,SHOO,BYOO,BLA,NBR1,GOP3,GOP1,PVG1,PBP4,AB22)
57:=(WDOO,SHOO,BYOO,BLA,STOP,BSTR,NBD,SYNO,INIT,GOP3,GOP1,VG5)
58:=(STP2,REG,BLA,CDTS,DECA,CDTW,ADR1,RGB1,RGB2,AFF3,STP1,NBRO,BIN,CTOO)
59:=(BLA,STOP,BINA,RGB1,RGB2)
61:=(REG,BLA,STOP,BINA,SHO9,LOGI,DECA,BYO9,NBD,PAS,WDO1,PDL,WDO4,SHO1,PGB,
SYN)
62:=(BLA,VG6,DCL,HALT)
63:=(BLA,STOP,PRC1,VG5)
65:=(NUL5)
28:=(NUL2)
29:=(WDOO,SHOO,BYOO,BLA,STOP,PD3,PG5,NBW)
IF /:CD:=DICT;VAL:=();SAT:=().
RGK/:REG:=(RG6,REG,RTB,RGS,RGW,RG1,RG5,RG7,RG8,RG9,RG3,RG4,RBS8);CD:=DICT;
4:=3;VAL:=(BLA);SAT:=().
 /:REG:=(BLA);CD:=DICT;VAL:=(BLA);SAT:=().
<=/:CD:=DICT;VAL:=(BLA);SAT:=().
AND /:REG:=(LOGI,CORD);CD:=DICT;CM:=1;VAL:=(BOO2);SAT:=().
OR /:REG:=(LOGI,CORD);CD:=SYNO;CM:=1;VAL:=(BOO2);SAT:=().
WHILE /:CD:=SYNO;VAL:=();SAT:=().
DCTET/:REG:=(TOCT);CD:=DICT;VAL:=();SAT:=().
-/:REG:=(TOCT);CD:=DICT;VAL:=();SAT:=().
;/:REG:=(STOP,STP1,HALT);CD:=DICT;VAL:=();SAT:=().
```

```
/</:CD:=DICT;VAL:=(BLA);SAT:=().
/=/:CD:=DICT;VAL:=(BLA);SAT:=().
/¬=/:CD:=DICT;VAL:=(BLA);SAT:=().
/>/:CD:=DICT;VAL:=(BLA);SAT:=().
/>=/:CD:=DICT;VAL:=(BLA);SAT:=().
/ID/:VAL:=(BLA);SAT:=().
/BEGIN /:VAL:=();SAT:=().
/BIDON/:REG:=(DCL);VAL:=();SAT:=().
/"/:CD:=DICT;VAL:=();SAT:=().
/CAR/:REG:=(NBR1,NBD,NBOW,NBOS,NBRW,NBS,NBR2,NBR3,NBR4,NBR6,NBW);CD:=DICT;
CM:=10;VAL:=();SAT:=().
/COMMENT/:REG:=(COMM);VAL:=();SAT:=().
/:=/:REG:=(AFF,AFF1,AFF3);CD:=DICT;CM:=8;VAL:=();SAT:=(PLD4,PLD2,PD4).
/OPBIN/:REG:=(BINA,BINS,BINW,BIN3,BIN2,BIN);CD:=DICT;CM:=8;VAL:=(BLA);SAT:=().
/?/:VAL:=(BLA);SAT:=().
/(/:REG:=(PG11,PG0,PG2,PG1,POCS,POCW,POCB,PG4,PTVG,PGB,PVG1,PG5,PG8,PG9,PG10,
PG14);CD:=DICT;CM:=13;VAL:=();SAT:=().
/)/:REG:=(PD9,PD7,PTVG,PD8,PD1,PD2,PD3,PLD4,PAD2,PLD2,PD5,PD4,PD6,PMD4,PWD4);
CD:=DICT;CM:=8;VAL:=(BLA);SAT:=().
/XOR /:REG:=(LOGI);CD:=DICT;VAL:=();SAT:=().
/MEM/:CD:=SYNO;VAL:=(BLA);SAT:=().
/OPDEC/:REG:=(DECA);CD:=SYNO;VAL:=(REG,NBD);SAT:=().
/FOR /:REG:=(RIB);CD:=DICT;CM:=3;VAL:=(REG);SAT:=(NBR4).
/STEP /:REG:=(PAS);CD:=DICT;CM:=9;VAL:=(WD00,NBD);SAT:=().
/UNTIL /:REG:=(PAS);CD:=DICT;CM:=9;VAL:=(REG,WD00,SH00,BY00,NBD);SAT:=().
/STAB/:VAL:=();SAT:=().
/STC3/:REG:=(RBS1,RBS2);CD:=SYNO;CM:=42;VAL:=(BLA);SAT:=().
/MEMO3/:REG:=(AB31,WSBM);CD:=SYNO;CM:=50;VAL:=();SAT:=().
/))) ;/:REG:=(PTER);CD:=SYNO;VAL:=();SAT:=().
/SET/:REG:=(RIB1);CM:=2;VAL:=();SAT:=(PST2,PBL4).
/MVI1/:REG:=(RIB1);CM:=46;VAL:=();SAT:=().
/MVI2/:REG:=(RIB3);CM:=47;VAL:=(BLA);SAT:=().
/,/:REG:=(VG24,PKT,VG16,VG17,VG1,VG18,VG19,VG20,VG21,VG22,VG23,VG3,VG2,PVG1,
VG6,VRG1,VRG2,VG8,VG7,VG9,VG10,VG12,VG14,VG15,VG5);CD:=DICT;CM:=27;VAL:=();
SAT:=().
/MVI3/:REG:=(RBS1,RBS2);CD:=SYNO;CM:=48;VAL:=(BLA);SAT:=().
/CLC/:REG:=(CLM1,CLM2,RBCD);CD:=SYNO;CM:=6;VAL:=(PG10);SAT:=().
/@/:REG:=(ADR,ADR1);CD:=DICT;CM:=61;VAL:=(BLA);SAT:=().
/STH2/:REG:=(RIB1);CM:=44;VAL:=(PG1);SAT:=(PDD5,PD3,PLD2,PDD1).
/STH3/:REG:=(RBS3,RBS4);CD:=SYNO;CM:=42;VAL:=(BLA);SAT:=(PD2,PAD2,PMD4,PWD4).
/SARRAY3/:REG:=(WSBM);CD:=DICT;CM:=45;VAL:=();SAT:=().
/MEMO4/:REG:=(AB40,RB40);CM:=3;VAL:=();SAT:=().
/MVC/:REG:=(CLM1,CLM2,RIB);CD:=SYNO;CM:=6;VAL:=(PG10);SAT:=().
/STC1/:REG:=(RIB1);CM:=40;VAL:=(BLA);SAT:=(PAD2,PMD4,PWD4).
/STC2/:REG:=(RIB1);CM:=41;VAL:=(BLA);SAT:=().
/SLDL1/:REG:=(RIB1);CM:=18;VAL:=();SAT:=(RG1).
/SLDL2/:REG:=(RIB1);CM:=19;VAL:=();SAT:=(RG1,VG18,VG19,VG20).
/STM1/:REG:=(RIB);CD:=SYNO;CM:=21;VAL:=(BLA);SAT:=(PD3).
/STM2/:REG:=(RIB);CD:=SYNO;CM:=22;VAL:=(BLA);SAT:=(RG1).
/STMO/:REG:=(RIB1);CM:=25;VAL:=(PG1);SAT:=().
/IC1/:REG:=(RIB1);CM:=30;VAL:=();SAT:=(PAD2,PMD4,PWD4).
/IC2/:REG:=(RIB1);CM:=34;VAL:=();SAT:=().
/IC3/:REG:=(RIB1);CM:=32;VAL:=(PG1);SAT:=(RG1,PD1).
/LH1/:REG:=(RIB1);CM:=35;VAL:=(PG1);SAT:=().
/LH2/:REG:=(RIB1);CM:=36;VAL:=(PG1);SAT:=(ABS2,PGB,PAD2,PMD4,PWD4,PDD1,PDD2,
PBP2,PBL2).
/LH3/:REG:=(RIB1);CM:=37;VAL:=();SAT:=(RG1,PDD5,PAD2,PMD4,PWD4,PDD2,PBP1,PBL1).
/WARRAY1/:REG:=(WD00,WD10,CDTW,WD01,RBS5,WS03,WD08,WB06,WD03);CD:=DICT;CM:=49;
VAL:=();SAT:=().
/WARRAY2/:REG:=(WD02,WSM1,WD04,RBS7);CD:=SYNO;CM:=31;VAL:=();SAT:=().
/WARRAY3/:REG:=(WSBM,RBS5,WD05);CD:=DICT;CM:=5;VAL:=();SAT:=().
/SARRAY1/:REG:=(SH00,CDTS,RBS5,SH01,WS03,SH04,SH08);CD:=DICT;CM:=38;VAL:=();
SAT:=().
/SARRAY2/:REG:=(SH09,WSM1,SH02);CM:=31;VAL:=();SAT:=().
/BARRAY1/:REG:=(BY00,BY09,BY08,BY01,BY03,WB06);CD:=DICT;CM:=33;VAL:=();SAT:=()
```

```
AKRAY3/:REG:=(WSNM);CD:=SYNO;CM:=20;VAL:=();SAT:=().
EMO1/:REG:=(ABS0,ABS5,WS03,WB06,ABS7);CD:=SYNO;VAL:=();SAT:=().
EMO2/:REG:=(ABS8,AB20,ABS2,WSM1,AB22);CD:=SYNO;CM:=31;VAL:=();SAT:=().
RDL2/:REG:=(RTB1);CM:=19;VAL:=();SAT:=(VG17,RG1,VG19,VG20).
LDA2/:REG:=(RTB1);CM:=19;VAL:=();SAT:=(VG17,RG1,VG18,VG20).
RDA2/:REG:=(RTB1);CM:=19;VAL:=();SAT:=(VG17,RG1,VG18,VG19).
ND/:REG:=(RTB);CD:=DICT;CM:=11;VAL:=();SAT:=().
TRING/:CD:=DICT;VAL:=();SAT:=().
ULL/:REG:=(RTB);CD:=DICT;VAL:=(STOP);SAT:=().
VERFLOW/:REG:=(OVFL);CD:=SYNO;VAL:=();SAT:=().
OOLEEN/:REG:=(BOO2,BOO1);CD:=SYNO;VAL:=();SAT:=().
LT1/:REG:=(RTB1);CM:=53;VAL:=();SAT:=().
RRAY/:REG:=(RTB);CD:=DICT;CM:=57;VAL:=();SAT:=().
EGMENT/:REG:=(RTB1);VAL:=(PROO);SAT:=().
EG1/:REG:=(RG6,REG,RTB,RGS,RGW,RG1,RG5,RG7,RG8,RG9,RG3,RG4,RBS8);CD:=SYNO;
:=3;VAL:=(BLA);SAT:=().
ROC/:REG:=(RTB,PRC1);CD:=DICT;CM:=11;VAL:=();SAT:=().
ROCEDURE/:REG:=(RTB,PROO,PRCD);CD:=DICT;CM:=51;VAL:=();SAT:=(RG1).
EGO/:CD:=DICT;VAL:=();SAT:=().
ESET/:REG:=(RTB1);CM:=2;VAL:=();SAT:=(PST1,PBL3).
HIF/:REG:=(RTB);CD:=DICT;CM:=1;VAL:=(OVFL,BOO2,TCC);SAT:=(PLD4,PLD2,PD4).
OTHEN/:REG:=(DOTH);CD:=DICT;CM:=23;VAL:=();SAT:=().
PREL/:REG:=(INIT,COMP,TCC,DCL);CD:=DICT;VAL:=();SAT:=().
) /:REG:=(PBP3,PD2,PLD3,PPB2,PPB1);CD:=DICT;VAL:=();SAT:=().
 ;/:REG:=(PST1,PST2,PBP4,PD4,PBP1,PBP2,PBP5);CD:=SYNO;VAL:=();SAT:=().
) ;/:REG:=(PDD5,PD4,PDD1,PDD2,PDD6);CD:=SYNO;VAL:=();SAT:=().
L12/:REG:=(RBCD);CD:=SYNO;CM:=52;VAL:=(PG1);SAT:=(RG1,PD1).
EGIN/:REG:=(BEG,RBEN);CD:=DICT;VAL:=();SAT:=().
LSE/:REG:=(STP2,STOP);CD:=DICT;VAL:=();SAT:=().
ABEL/:REG:=(RTB);CD:=DICT;CM:=63;VAL:=();SAT:=().
NOC/:REG:=(RTB);CD:=SYNO;CM:=26;VAL:=();SAT:=().
SYN4/:REG:=(BSYN);CD:=SYNO;VAL:=();SAT:=().
HSYN1/:REG:=(CDTS,RBS5,SH01,WS03,SH04,SH08,DCL);CD:=SYNO;CM:=38;VAL:=();
T:=().
HSYN3/:REG:=(WSBM);CD:=SYNO;CM:=45;VAL:=();SAT:=().
QUATE/:VAL:=();SAT:=().
NCLUDE/:VAL:=();SAT:=().
X1/:REG:=(RTB1);CM:=54;VAL:=();SAT:=(PG10).
X2/:REG:=(RTB1);CM:=55;VAL:=(PG1);SAT:=(RG1,PG10).
ARO/:REG:=(NBRO);CD:=DICT;CM:=11;VAL:=();SAT:=().
M/:REG:=(RTB);CD:=SYNO;CM:=56;VAL:=();SAT:=().
OTO/:REG:=(RTB);CD:=DICT;CM:=14;VAL:=();SAT:=().
TI/:REG:=(RTB,PRC1,TRAD);CD:=DICT;CM:=39;VAL:=(DXPT);SAT:=().
OGINT/:REG:=(RTB,BSLR);CD:=SYNO;CM:=57;VAL:=();SAT:=().
YTECHAR/:REG:=(RTB,BSLR);CD:=SYNO;CM:=57;VAL:=();SAT:=().
HORT/:REG:=(BSLR,RBSH);CM:=57;VAL:=();SAT:=().
ONGREAL/:REG:=(RTB,BSLR);CD:=DICT;CM:=57;VAL:=();SAT:=().
SYN1/:REG:=(BYOO,BY08,BY01,BY03,WB06);CD:=SYNO;CM:=33;VAL:=();SAT:=().
SYN2/:REG:=(BY06);CD:=SYNO;CM:=43;VAL:=();SAT:=().
YNONYME/:REG:=(RTB1);CM:=62;VAL:=(DCL);SAT:=().
ONST/:REG:=(WDOO,WD10,WD03,CTOO);CD:=DICT;VAL:=();SAT:=().
ONSTANT/:CD:=SYNO;VAL:=();SAT:=().
1) /:REG:=(PTER);CD:=SYNO;VAL:=();SAT:=().
LOBAL/:REG:=(RTB);CD:=SYNO;CM:=24;VAL:=();SAT:=().
XTERNAL/:REG:=(RTB);CD:=DICT;CM:=24;VAL:=(PROO);SAT:=().
FLOT/:REG:=(RTB);CD:=DICT;CM:=15;VAL:=();SAT:=().
/:CD:=DICT;VAL:=();SAT:=().
IPUN/:REG:=(UNA1,UNA2);CD:=SYNO;VAL:=();SAT:=().
ASE/:REG:=(RTB);CD:=SYNO;CM:=20;VAL:=();SAT:=().
 ELSE/:REG:=(PBL3,PBL4,PBL1,PBL2);CD:=DICT;VAL:=();SAT:=().
YN1/:REG:=(SYNO,SYC3);CD:=DICT;CM:=29;VAL:=();SAT:=().
IR/:REG:=(STOP);CD:=SYNO;VAL:=();SAT:=().
IF/:REG:=(DECL);CD:=DICT;VAL:=();SAT:=().
IPALFA/:REG:=(GDP1,GDP2);CD:=DICT;VAL:=();SAT:=().
```

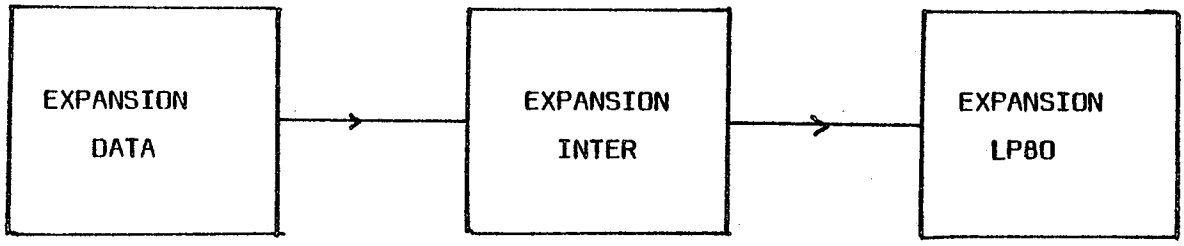
```
/BASE2/:REG:=(RGB3,RGB2);CD:=SYNO;CM:=31;VAL:=(BLA);SAT:=().
/BASE3/:REG:=(BAS3);CD:=SYNO;VAL:=();SAT:=().
/AFFECT/:REG:=(AFF4,AFF5);CD:=DICT;CM:=59;VAL:=();SAT:=().
/INTREG/:REG:=(RIB);CD:=SYNO;CM:=12;VAL:=();SAT:=().
/ALFAREEL/:REG:=(REAL);CD:=DICT;VAL:=();SAT:=().
/ALFALONG/:REG:=(LONG);CD:=DICT;VAL:=();SAT:=().
/ALFACOURT/:REG:=(SX);CD:=DICT;VAL:=();SAT:=().
/EXPOSANT/:REG:=(LONG);CD:=SYNO;VAL:=();SAT:=().
/OPHEX/:REG:=(GOP1,GOP2);CD:=SYNO;VAL:=();SAT:=().
/OPSPEC/:REG:=(GOP3,GOP2);CD:=DICT;VAL:=();SAT:=().
/:/:REG:=(DXPT);CD:=DICT;VAL:=();SAT:=().
```

ANNEXE 2

Exemple : Traduction de la procédure EXPANSION

La procédure EXPANSION est rangée dans le fichier de même nom et de type "DATA".

- Le module d'indexation permet de ranger dans le dictionnaire les déclarations locales (SREG, MCT, MKETENDU).
- Le module de traduction opère en deux passages et créé successivement les fichiers de type "INTER" et de type LP80.



N.B.- L'utilisateur devra rajouter dans les déclarations locales à la procédure EXPANSION du fichier LP80, l'étiquette FIN qui doit obligatoirement être déclarée dans le langage LP80.

PREMIER PASSAGE

```
plaftrad
EXECUTION BEGINS...
ACTIVATION PROGRAMME ? (GRAM,DICT,DECLARATION,TRADUCTION,INTERRO)
$
decl
>
logical sreg,mct,mketendu;
>
$fin
$
traduction
DONNEES ? (TERMINAL OU NOM DE FICHIER)
%
expansion
TYPE DU FICHIER?
>
data
PREMIER PASSAGE(1) OU DEUXIEME PASSAGE(2)
>
1
IMPRESSION DES DECOUPAGES?(O/N)
>
n
RESULTATS?(TERMINAL OU NOM DE FICHIER)
>
expansion
TYPE DU FICHIER?
>
inter
A PARTIR DE :PROCEDURE EXPANSION (R14) ;
-
dict
>
/expansion/proc/
>

-
morp
A PARTIR DE :IF R4=R0 THEN GOTO FIN ;
-
dict
>
/fin/etl/
>

-
morp
$
```

DEUXIEME PASSAGE

```
traduction
DONNEES ? (TERMINAL OU NOM DE FICHIER)
%
expansion
TYPE DU FICHIER?
>
inter
PREMIER PASSAGE(1) OU DEUXIEME PASSAGE(2)
>
2
RESULTATS?(TERMINAL OU NOM DE FICHIER)
>
expansion
TYPE DU FICHIER?
>
lp80
REMISE A ZERO PREALABLE DU REGISTRE?(O/N)
>
o
REMISE A ZERO PREALABLE DU REGISTRE?(O/N)
>
o
00000006    FINIS
00000006    FINIS
$
$fin
FILE NOT FOUND
R; T=3.66/6.11 11:35:34
```

EXPANSION DATA

```
PROCEDURE EXPANSION (R14);
BEGIN
      ARRAY 16 LOGICAL SREG;
      LOGICAL MCT;
      ARRAY 8 LOGICAL MKETENDU;
      IF R0¬=0 THEN
      BEGIN
            STM(R0,R15,SREG);
            R1:=R0+3;R0:=0;
            R2:=R0;R3:=R0;R4:=R0;R7:=5;
            R8:=R0;R9:=R0;IC(R4,B1);
            MVC(3,MCT,B1(1));
            R3:=MCT;
            R2:=R0;
            FOR R6:=R0 STEP 1 UNTIL 31 DO
            BEGIN
                  SLDL(R2,1);
                  IF R2¬=R0 THEN
                  BEGIN
                        IC(R8,B1(R7));
                        STC(R8,MKETENDU(R9));
                        R7:=R7+1;R2:=R0;R4:=R4-1;
                        IF R4=R0 THEN GOTO FIN;
                  END;
                  R9:=R9+1;
            END;
            FIN:LM(R0,R15,SREG);
      END;
      R1:=@MKETENDU;
END;
```

EXPANSION INTER

```
PROCEDURE EXPANSION (R14) ;
BEGIN
ARRAY 16 LOGICAL SREG ;
LOGICAL MCT ;
ARRAY 8 LOGICAL MKETENDU ;
IF R0¬=0 THEN BEGIN
STM(R0,R15,SREG) ;
R1:=R0+3 ;
R0:=0 ;
R2:=R0 ;
R3:=R0 ;
R4:=R0 ;
R7:=5 ;
R8:=R0 ;
R9:=R0 ;
IC(R4,B1) ;
MVC(3,MCT,B1(1)) ;
R3:=MCT ;
R2:=R0 ;
FOR R6:=R0 STEP 1 UNTIL 31 DO BEGIN
SLDL(R2,1) ;
IF R2¬=R0 THEN BEGIN
IC(R8,B1(R7)) ;
STC(R8,MKETENDU(R9)) ;
R7:=R7+1 ;
R2:=R0 ;
R4:=R4-1 ;
IF R4=R0 THEN GOTO FIN ;
END ;
R9:=R9+1 ;
END ;
FIN:LM(R0,R15,SREG) ;
END ;
R1:=@MKETENDU ;
END ;
```

EXPANSION LP80

```
PROCEDURE EXPANSION  RD  ;
BEGIN
ARRAY  16 WORD SREG  ;
WORD MCT  ;
ARRAY  8 WORD MKETENDU  ;
IF R0 ¬= 0 THEN BEGIN
STORE (SREG , R0 ,RF );
R1 := R0 +3 ;
R0 := 0 ;
RE := R0  ;
RF := R0  ;
R4 := R0  ;
R7 := 5 ;
R8 := R0  ;
R9 := R0  ;
R2:= R1  ;R4 := BYTE0(R2);
MOVE (R2,4,@MCT SLLS 2 ,R1 + 1);
RF := MCT  ;
RE := R0  ;
FOR R6 := R0  STEP 1 UNTIL 31 DO BEGIN
RE := RE SLLD 1 ;
IF RE ¬= R0  THEN BEGIN
R2:= R1 + R7  ;R8 := BYTE0(R2);
R2:=@MKETENDU SLLS 2 + R9  ;BYTE0(R2) := R8 ;
R7 := R7 +1 ;
RE := R0  ;
R4 := R4 -1 ;
IF R4 =R0  THEN GOTO FIN  ;
END ;
R9 := R9 +1 ;
END ;
FIN :LOAD (SREG , R0 ,RF );
END ;
R1 := @MKETENDU SLLS 2  ;
END ;
```

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de Messieurs :

- J. COURTIN, Maître de Conférences à l'Université des Sciences Sociales de GRENOBLE

- M. BENNETT, Ingénieur au Centre Scientifique CII de - GRENOBLE -

Madame Claudine CHASSAGNE née HUGUES

est autorisée à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Génie Informatique".

Grenoble, le 10 Janvier 1979
Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique