MASTER EN SCIENCES INFORMATIQUES

Un système d'aide à la programmation descendante

Lepage, Isabelle; Pauwen, Thérèse

Award date:
1983

Awarding institution:
Universite de Namur

Facultés Universitaires N.D. de la Paix - NAMUR

INSTITUT D'INFORMATIQUE

UN SYSTEME D'AIDE

A LA

PROGRAMMATION DESCENDANTE .

ISABELLE LEPAGE

THERESE PAUWEN

Mémoire présenté
en vue de l'obtention du grade de
LICENCIE ET MAITRE EN INFORMATIQUE

année académique 1982-1983

Nous tenons à remercier Monsieur A. VAN LAMSWEERDE d'avoir accepté de diriger ce travail. Ses critiques et ses conseils nous ont considérablement aidés dans la réalisation de ce mémoire.

Nous tenons également à remercier Messieurs P.N. ROBILLARD et R. PLAMONDON pour leur accueil lors du stage réalisé à l'Ecole Polytechnique de MONTREAL.

PLAN

TABLE DES MATIERES

C H A P I T R E 1 :

INTRODUCTION

1. INTRODUCTION

Depuis un certain nombre d'années, le coût du logiciel occupe une part croissante dans le coût total d'un système informatique et cette part dépasse celle du matériel. Ce que l'on qualifie de " crise du logiciel " trouve ses origines dans la faible productivité des programmeurs tout au long du precessus de développement d'un logiciel et dans la qualité insuffisante du logiciel produit.

Actuellement, un effort important est réalisé pour trouver des solutions à la production économique de logiciel fiable. Une des solutions consiste en la mise au point et l'application d'un ensemble d'outils automatisés et spécialisés, appelés " environnement logiciel ". Cet environnement permet d'assister l'analyste ou le programmeur au cours des différentes étapes du cycle de vie d'un projet : définition des besoins, spécification, conception, codage, validation, documentation et maintenance (VLA, 82).

Dans cette optique ce travail présente le système d'aide à la programmation Schémacode, développé à l'Ecole Polytechnique de MONTREAL. Ce système offre une aide au codage et à la documentation de programmes. Nous décrirons le codeur PASCAL que nous avons réalisé à cette Ecole Polytechnique de MONTREAL. Celui-ci a été intégré au logiciel Schémacode. Nous ferons ensuite l'ébauche d'un nouveau codeur PASCAL pour un sytème alternatif à Schémacode et nous le décrirons.

Après cette introduction, nous présenterons une classification de la plupart des outils logiciels existants.
Nous décrirons ensuite deux systèmes d'aide à la programmation : les systèmes MENTOR et ADELE.

Ce chapitre nous permettra de mieux comprendre ce que peut recouvrir le concept de système d'aide à la programmation.

Le troisième chapitre de ce travail consiste en un exposé du produit réalisé : un codeur PASCAL. Nous effectuerons tout d'abord une présentation succincte de Schémacode, de ses objectifs, de même que ceux du codeur PASCAL.
Nous présenterons ensuite l'environnement logiciel existant en explicitant un scénario d'utilisation du système, en définissant le Pseudo-langage associé à Schémacode, ainsi qu'en décrivant les outils existants dans cet environnement.
La troisème section de ce chapitre expose les techniques possibles et la solution retenue pour générer du code PASCAL à partir d'un programme Schémacode.
La section suivante introduit le codeur PASCAL en traitant tout d'abord le cas particulier des déclarations.
Ensuite, nous exposerons les règles de traduction pour le codage automatique et nous aborderons l'analyse des erreurs qui peuvent survenir au cours du processus de traduction.
Dans la cinquième section, nous décrirons l'architecture du codeur PASCAL. Après l'analyse de la structure des données en entrée et en sortie, nous exposerons la structure globale des traitements. Nous justifierons ces traitements avant de préciser la décomposition physique de ceux-ci.
Nous spécifierons ensuite les différentes routines du codeur et nous présenterons le procédé de construction qui leur est associé. Nous clotûrerons ce chapitre en exposant les différentes techniques de test appliquées lors de la réalisation de ce codeur.

Le quatrième chapitre de ce travail consiste en une évaluation critique du système Schémacode, en termes de l'objectif qu'il poursuit, de son Pseudo-langage et sa représentation interne, et enfin des performances de ce système. Nous évaluerons, ensuite, le codeur PASCAL réalisé.

Dans le cinquième chapitre, nous ferons l'ébauche d'un système alternatif à Schémacode. Ce système résulte des critiques effectuées au chapitre précédant.

Nous décrirons tout d'abord un nouveau pseudo-langage dont nous préciserons la représentation interne. Celle-ci sera sous forme arborescente. Nous envisagerons également l'intégration d'un éditeur syntaxique à ce système.
Nous essayerons de monter qu'une définition précise du pseudo-langage et sa représentation interne arborescente, ont facilité la réalisation du nouveau codeur PASCAL.

A la section suivante, nous présenterons l'architecture de ce nouveau codeur PASCAL.
Nous spécifierons, pour terminer, les différents modules et nous décrirons leur construction.

En annexe, nous présenterons un exemple complet de traduction affectuée par le codeur PASCAL intégré à Schémacode.
Nous y inclurons également le manuel d'utilisateur de ce codeur PASCAL.

Pour ne pas surcharger la partie principale de ce travail; nous avons aussi porté en annexes les programmes du codeur PASCAL intégré à Schémacode, de même que ceux du codeur associé au système alternatif.
Nous y avons également porté la définition du nouveau pseudo-langage.

C H A P I T R E 2 :

LES OUTILS D'AIDE A LA PROGRAMMATION

2. LES OUTILS D'AIDE A LA PROGRAMMATION

2.1. CLASSIFICATION

Nous allons donner un aperçu des principales catégories dans lesquelles se retrouvent la plupart des outils logiciels existants. A chacune de ces catégories correspond une étape du cycle de vie d'un projet.

(1) Outils d'aide à la définition des besoins

L'objectif de cette étape est d'obtenir un modèle de l'application à développer (données, traitements, contraintes) Ce modèle doit constituer le support initial de communication entre analyste et client; il est donc essentiel qu'il soit aisément communicable.

(2) Outils d'aide à la spécification

L'objectif de cette étape est d'obtenir une définition précise, complète et cohérente du système à développer. Cette définition est à la base du contrat liant le producteur du système au client.

(3) Outils d'aide à la conception

La phase de conception comporte deux aspects : d'une part la conception d'une architecture logicielle et d'autre part la construction des différents programmes au sein de cette architecture. On peut identifier deux types d'outils d'aide à la conception de programmes :

- Les environnements permettant à l'analyste programmeur de formuler ses programmes sous forme abstraite;
- Les environnements permettant de réduire la difficulté de programmation, en encourageant l'utilisatio d'un maximum d'outils offerts par l'environnement.

(4) Outils d'aide au codage

Parmi ce type d'outils, on trouve :

- Des langages mieux adaptés à un type particulier d'applications;
- Des éditeurs et formateurs guidés par la syntaxe. En plus des fonctions habituelles d'un éditeur orienté-ligne, un éditeur guidé par la syntaxe vérifie que les instructions introduites par l'utilisateur obéissent bien aux règles syntaxiques du langage. Il travaille généralement en compagnie d'un formateur, qui utilise également la connaissance de la syntaxe pour structurer le texte du programme en assurant une indentation et des sauts de ligne adéquats;
- Des préprocesseurs et générateurs de code. Les préprocesseurs sont des programmes qui génèrent des programmes dans un langage standardisé (COBOL, FORTRAN, PASCAL) à partir de programmes rédigés dans un langage de plus haut niveau.

(5) Outils d'aide à la validation

Il y a deux façons de valider un logiciel : le tester d'est-à-dire essayer de faire apparaître un maximum d'erreurs, ou le vérifier c'est-à-dire essayer de démontrer l'absence d'erreurs. Il existe par conséquent deux classes d'outil de validation : des outils d'aide aux tests et des outils d'aide à la vérification.

(6) Outils d'aide à la documentation et à la maintenance

Parmi ce type d'outils on trouve :

- Les outils permettant de préserver la cohérence d'une spécification, lors d'une modification de celle-ci;

- Les outils permettant de générer une documentation de programme adéquate en cours de conception de celui-ci;
- Les outils permettant de gérer la documentation de grands logiciels, d'en assurer la cohérence en contrôlant les versions multiples, sources et objets d'un même module;
- Les outils qui aident la production d'une documentation expliquant et justifiant les différents choix et hypothèses qui ont été faits au cours de la conception du système.

Les bénéfices de la mise au point de tels outils en terme d'accroissement de la productivité et de la qualité du produit final peuvent être substantiels. Des produits intermédiaires (spécification, architecture logicielle, conception,code, plans de tests) peuvent être obtenus de façon standardisée grâce aux outils. Ceci permet aux analystes programmeurs de consacrer leur temps et leur compétance aux points critiques de leur programme et d'éliminer ainsi les erreurs matérielles dues à l'inattention. Ces produits intermédiaires permettent une communication plus aisée au sein de l'équipe du projet et peuvent constituer une documentation standardisée et systématique pour toutes les étapes du cycle de vie du projet. Cette documentation facilitera une maintenance ultérieure du produit réalisé.

La tendance actuelle est de pouvoir intégrer des outils complémentaires et compatibles en un ensemble cohérent qui couvre la totalité du cycle de vie; cet ensemble cohérent constituerait un environnement logiciel (VLA, 82).

2.2. LE SYSTEME MENTOR.

Le premier système d'aide à la programmation que nous allons présenter est le système MENTOR de manipulation de programmes PASCAL (DON, 79), (DON, 80).

Le système MENTOR est issu d'un projet qui a débuté à l'INRIA (Institut National de Recherche en Informatique et en Automatique, FRANCE) en 1974. Le système MENTOR n'était à l'origine qu'un éditeur de programmes dirigé par la syntaxe du langage mais dont le langage de commande (MENTOL) est un véritable langage de programmation. A côté de l'outil initial, MENTOL a permis et permet de développer un " environnement de programmation ". Celui-ci est composé d'outils adaptables aux besoins de l'utilisateur et qui se sont révélés utiles dans la normalisation, la documentation, la mise au point, la transformation et la portabilité des programmes. Nous allons examiner les types d'outils d'aide à la programmation, que le système MENTOR propose.

(1) Outils d'aide à la normalisation d'un programme

La normalisation de la présentation d'un programme a une double utilité :

- Elle rend les programmes plus lisibles;
- Elle permet d'organiser de façon systématique et permanente toutes les informations d'un programme. Cela rend plus facile la vérification de certaines propriétés du programme.

Dans MENTOR, un décompilateur prend en charge le formatage du programme. Les autres formes de normalisation plus complexes, par exemple, la simplification des programmes ou la modification des conventions d'un programme pour qu'il respecte les conventions imposées par le compilateur sont assurées par des précédures MENTOL.

(2) **Outils d'aide à la documentation**

La documentation peut être développée à deux niveaux :

- La documentation locale : elle se présente sous forme de commentaires (textes, assertions ...) qui précisent les propriétés d'un programme.

- La documentation globale : elle se présente sous forme de texte, tables au représentation particulière d'un programme. Elle décrit les propriétés d'un programme dans son ensemble et les interactions de ses divers composants.

De plus, on distingue la documentation formelle (assertions, tables) organisée pour être utilisée mécaniquement, de la documentation informelle (texte). Sous MENTOR, on formalise au maximum la documentation, pour faciliter, grâce à des procédures MENTOL, la mécanisation de sa construction et de son utilisation.

(3) Outils d'aide à la mise au point d'un programme

MENTOR peut être utilisé pour mettre au point des programmes en les modifiant mécaniquement ou en y insérant des tests, des assertions ou des instructions de trace.

Exemples :
- Dans le but de localier les boucles infinies d'un programme, on écrira une procédure MENTOL qui insère une instruction de trace dans chaque boucle;
- Une procédure MENTOL permet d'insérer dans le programme des instructions de comptage, des appels de chaque procédure, afin d'obtenir des profils d'exécution.

(4) Outils d'aide à la transformation d'un programme

MENTOR dispose d'outils permettant de transformer un programme. Par exemple, on peut changer les appels de fonctions en des appels de procédure.
Notons, qu'il est plus facile de certifier le programme de transformation que de reprendre la certification du programme transformé. C'est une raison pour laquelle il est intéressant de développer des outils mécaniques de transformation.

(5) Outils d'aide au transport d'un programme

Le transport des programmes d'un compilateur à l'autre peut créer des problèmes. En effet, on trouve toujours certaines différences entre les diverses implémentations d'un même langage. Ce problème est en fait une application de transformation de programmes prise en charge par MENTOR.

Le transport comporte deux phases :

- Déterminer les éléments qui doivent être modifiés;
- Utiliser une procédure MENTOL qui transforme ces éléments en préservant leur sémantique et en utilisant des construction qui sont équivalentes dans les deux langages.

2.3. LE SYSTEME ADELE

Le système ADELE (MOS, 82), mis au point dans les laboratoires IMAG (GRENOBLE), offre un ensemble d'outils d'aide à la programmation ou " environnement de programmation " destiné à une petite communauté de chercheurs pour développer des applications complexes.

En ce qui concerne l'écriture et la mise au point d'un programme, l'objectif essentiel d'ADELE est de raccourcir le cycle édition, compilation, exécution. En effet, le plus souvent l'utilisateur doit encore se contenter pour la construction d'un programme d'outils disparates et appliquer manuellement l'enchaînement classique édition de texte, compilation, édition de liens, exécution. Les erreurs détectées à la compilation, à l'édition de liens ou encore à l'exécution impliquent dans tous les cas des retours en arrière fastidieux, qui freinent la productivité du programmeur et qui sont générateurs de nouvelles erreurs.

Pour raccourcir le cycle de développement d'un module, l'atelier ADELE dispose d'un éditeur syntaxique qui remplace la phase " édition - compilation ". La représentation interne sous-jacente à ce type d'éditeur, pour un programme PASCAL, est l'arbre syntaxique associé au programme.

Une phase d'interprétation permet une mise au point plus facile du programme. L'interpréteur accepte en entrée la représentation interne d'un programme syntaxiquement correct, et l'interprète jusqu'à ce qu'il rencontre soit une erreur (sémantique ou d'exécution) soit une instruction de mise au point. En cas d'erreur, l'interpréteur rend le contrôle à l'utilisateur qui édite son programme afin de le corriger.
Les instructions de mise au point constituent une extension au langage PASCAL.
Parmi celles-ci, on trouve :

- Le point d'arrêt

  C'est une instruction associée à un point précis du programme. L'interpréteur évalue la condition qui lui est associée et s'arrête si elle n'est pas vérifiée.

- L'assertion

  C'est une condition à évaluer à chaque étape de l'interprétation du sous-arbre qu'elle englobe. On peut ainsi demander à l'interpréteur de s'arrêter dès que la condition n'est plus vérifiée.

Nous trouvons dans l'atelier logiciel ADELE trois notions qui peuvent être considérées comme des besoins essentiels dans les " environnements de programmation ".

(1) L'utilisateur d'une base de données où sont conservés sous diverses formes (source, objet) et dans leurs multiples versions, les différents modules d'un projet.
(2) Un éditeur syntaxique.

(3) Une documentation qui permet d'assurer la cohérence de grands logiciels en contrôlant les multiples versions sources et objets d'un même module.

Actuellement, le système ADELE est en cours de réalisation :

- La représentation interne d'un programme PASCAL a été spécifiée;
- L'éditeur syntaxique a été expérimenté;
- La base de données, chargée de concerver les multiples versions de chaque module est implémentée.

C H A P I T R E 3 :

LE PRODUIT REALISE

3. LE PRODUIT REALISE.

3.1 PRESENTATION ET OBJECTIFS.

3.1.1 DEFINITION SUCCINCTE DE SCHEMACODE.

Schémacode est un logiciel d'aide à la programmation : il permet la mise au point de programmes de façon interactive. L'édition d'un programme se fait par raffinements successifs : partant d'un but l'utilisateur se définit des sous-buts et chacun d'eux peut encore contenir des sous-buts. Le processus continue tant qu'un sous-but nécessite une décomposition; finalement, chaque sous-but est décrit en terme d'instructions abstraites du pseudo-langage de Schémacode.

Schémacode fournit plusieurs types d'aides dans ce processus :

1. Insertion de documentation automatique : lors de la définition d'un sous-but, Schémacode crée automatiquement un raffinement dont le premier commentaire est l'énoncé du sous-but.
2. Edition des structures de contrôle sous forme graphique : lors de la description d'un but ou sous-but, les structures de contrôle séquentielles, conditionnelles et répétitives sont représentées à l'écran sous forme graphique suggestive; de là les expressions "développer un programme en pseudocode schématique" et "structures SPC" où Pseudocode schématique est abrégé en "SPC".
3. Analyse sommaire de la syntaxe :
   Schémacode vérifie qu'une structure répétitive contient au moins une condition de sortie et un traitement. Dans les structures conditionnelles, Schémacode refuse l'introduction d'une alternative "ELSE" s 'il n'y a pas d'alternative "THEN" et vérifie que chaque alternative contient au moins un traitement.

4. Lorsque le programme est au point en pseudocode schématique, un codeur, partie intégrante de Schémacode, effectue la "traduction" de ce programme dans le langage cible (FORTRAN ou PASCAL).

3.1.2 OBJECTIFS DE SCHEMACODE.

Schémacode a été conçu à l'Ecole Polytechnique de Montréal pour aider l'utilisateur à écrire des programmes de façon structurée et pour favoriser leur documentation.

1. Ecrire des programmes de façon structurée :
   Il faut distinguer "programmation structurée" de "codage structuré" : un programme est structuré s'il reflète la structure du problème; un code est structuré si les primitives de codage utilisées sont faciles à contrôler.
   Schémacode est un outil d'aide à ces deux niveaux : conception et codage.

   Il est un outil d'aide à la conception en ce sens qu'il assiste le programmeur dans la construction d'une version schématique abstraite de son programme en appuyant une démarche par raffinements successifs.

   Il est un outil d'aide au codage dans la mesure où il contient un codeur FORTRAN et un codeur PASCAL.
   Le code généré est conforme à la description schématique de l'algorithme.

2. Favoriser la documentation :
   Schémacode force l'utilisateur à documenter chaque raffinement, chaque étape de son algorithme et intègre automatiquement cette documentation de façon à fournir un produit communicable.

3.1.3 OBJECTIFS DU CODEUR PASCAL.

Nous distinguerons l'objectif des concepteurs de Schémacode de notre objectif.

L'objectif des concepteurs de Schémacode.

L'équipe qui développe le projet Schémacode l'a tout d'abord envisagé comme un outil d'aide à la programmation en FORTRAN. L'ajoût du codeur PASCAL constitue une suite logique : Schémacode est maintenant un outil d'aide à la programmation en FORTRAN et en PASCAL.

Notre objectif.

Notre objectif est de développer un outil d'aide à la programmation PASCAL et de l'intégrer dans un environnement existant. Cet outil est un codeur qui, à partir d'un programme développé en pseudocode schématique, génère un programme en code PASCAL; une analyse syntaxique sommaire est faite en amont au cours de l'édition du programme.

A côté de cela, il faut ajouter que jusqu'à présent le PASCAL est inutilisé à l'école Polytechnique de Montréal. Schémacode orienté PASCAL représente donc un outil pédagogique.

3.2 L'ENVIRONNEMENT LOGICIEL EXISTANT.

3.2.1 L'AIDE A LA PROGRAMMATION DESCENDANTE : SCENARIO D'UTILISATION DU SYSTEME.

L'utilisateur de Schémacode représente la structure logique de son programme à l'écran d'un terminal. Il utilise de façon interactive les trois structures suivantes : séquentielles conditionnelles et répétitives, en suivant une approche descendante par raffinements successifs.

Scénario d'interaction programmeur/système.

- Dans une première phase, l'utilisateur énonce les grandes étapes de résolution; pour chaque énoncé d'une étape, Schémacode crée automatiquement un raffinement dont le nom ou premier commentaire est l'énoncé introduit à la 1e phase. Ce nouveau raffinement est développé dans la seconde phase.

- Supposons qu'on ait terminé la phase i. A la phase (i + 1), l'utilisateur détaille chaque raffinement créé à la phase i; il énonce éventuellement les étapes pour résoudre un de ses sous-problèmes. Pour chacune de ces étapes, Schémacode crée donc un raffinement dont le nom est l'énoncé de l'étape; elle sera développée à la phase (i + 2).

- A la phase terminale, chaque problème a une solution simple en terme d'instructions abstraites du pseudo-langage de Schémacode.

Modélisation sous-jacente d'une démarche descendante de programmation.

De la présentation ci-dessus, on voit se dégager les concepts que nous utiliserons dans la suite de ce travail.

PROGRAMME.

Un programme est un ensemble de raffinements.

RAFFINEMENTS.

La définition d'un raffinement se fait à deux niveaux :
- sa création,
- son développement.

Lors de sa création, le raffinement ne comporte qu'un numéro et un commentaire narratif, le nom du raffinement.
Développé, le raffinement contient un ensemble d'instructions représentées sous forme de structures SPC; il contient toujours son nom et éventuellement des références à d'autres raffinements.

Le numéro d'un raffinement l'identifie; il est imposé par le système au moment de la création du raffinement : le raffinement (ijk) (le raffinement de numéro ijk) est le (ijk)ème créé dans le programme en cours de développement.
Le nom du raffinement est la première structure du raffinement; le numéro du raffinement ne fait pas partie de son nom.
Remarque : dans la suite de notre travail, nous abandonnerons la numérotation des raffinements sous forme (ijk), nous parlerons du Raff i, raffinement de numéro i, tout en sachant qu'un numéro de raffinement est formé de trois chiffres.

REFERENCE A UN RAFFINEMENT.

Sous ce vocable, on peut distinguer un objet et un mécanisme.

C'est le mécanisme de "référence à un raffinement" qui permet de développer un algorithme de façon descendante. Et

l'objet "référence à un raffinement" est l'énoncé d'une étape développée par ailleurs.

L'objet : la référence à un raffinement se compose d'un numéro de raffinement imposé par l'éditeur au moment de la création de la référence et d'un commentaire dit "opérationnel"; ce commentaire ne peut dépasser 72 caractères.

Le mécanisme : lorsqu'on écrit une référence à un raffinement, Schémacode crée un nouveau raffinement. Son numéro est celui imposé par l'éditeur, son nom est automatiquement le commentaire de la référence.

A l'exception du premier raffinement un raffinement ne peut être créé que par le mécanisme de "référence à un raffinement"; le premier raffinement est créé en même temps que le fichier destin à contenir le programme et porte le numéro zéro.

Ce mécanisme implique qu'un raffinement ne peut être créé qu'une seule fois par un autre raffinement; dans un programme, il n'existe donc qu'une seule référence à un raffinement donné.

RAFFINEMENT REFERENCE - RAFFINEMENT LIBRE.

Lorsqu'un raff i contient une référence à un raff j, on dit que le raff i "référence" le raff j.
Lorsqu'un raff i référence un raff j et qu'on efface la référence au raff j dans le raff i, on dit que le raff j est libre; il existe toujours mais n'est plus référencé par aucun raffinement du programme.

Si un raffinement libre référence d'autres raffinements, le raffinement libre et ceux qu'il référence forment un groupe isolé du reste du programme.

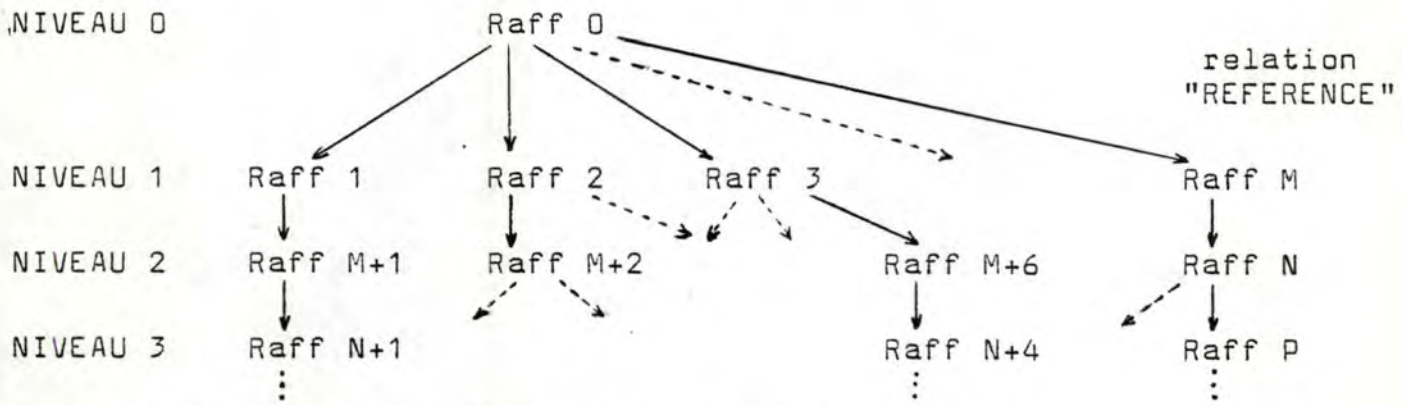
RATTACHEMENT D'UN RAFFINEMENT.

Pour qu'un raffinement libre fasse à nouveau partie du programme, il faut le "rattacher". Pour ce faire, il faut créer une nouvelle référence à ce raffinement dans un des raffinements du programme. Il suffit de donner le numéro de raffinement à rattacher, l'éditeur se charge de replacer le commentaire correspondant.

Il serait souhaitable que Schémacode permette qu'un raffinement soit référencé de plusieurs endroits du programme. Dans ce cas, un raffinement créé par "référence à un raffinement" pourrait être "rattaché" à différents endroits du programme.

ARBRES DES RAFFINEMENTS.

De la relation entre les différents raffinements, on peut déduire l'arbre des raffinements.



- Le niveau 0 ne reprend que le raff 0.
- Le niveau 1 reprend tous les raffinements référencés par le Raff 0.
- Le niveau i reprend tous les raffinements référencés par les différents raffinements du niveau (i - 1).

père d'un raffinement

Le raff i est le père du raff j, si le raff i contient une instruction de référence au raff j.

Cycle de référence entre raffinements

Les raff k_1, k_2, ... k_n constituent un cycle de référence entre raffinements si et seulement si

- $k_1 = k_n$,
- le raff k_j ($1 \leq j < n$) contient une instruction de référence au raff k_{j+1}.

3.2.2 LES LANGAGES SOURCE ET CIBLE DE SCHEMACODE.

Le langage CIBLE de Schémacode est le langage dans lequel on veut traduire l'algorithme développé grâce à Schémacode : FORTRAN ou PASCAL.

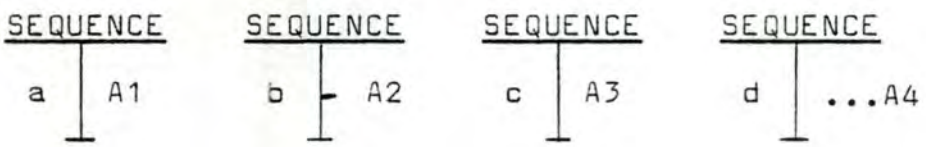
Le langage SOURCE, langage de description d'un programme, est en fait un mélange de deux langages :
- le langage cible, dont on doit se servir pour introduire les affectations, les appels de procédure externe, les déclarations et les expressions booléennes.

- le pseudocode schématique (SPC) permettant une représentation graphique des traitements : séquence, test, tests imbriqués, répétition, et des délimiteurs de début et fin de raffinement.

Grâce à ce pseudo-langage, il est en effet possible de représenter schématiquement l'ossature d'un programme à l'aide de trois types de structures : séquentielles, conditionnelles et répétitives.

(1) STRUCTURES DU PSEUDO-LANGAGE ET INSTRUCTIONS ASSOCIEES.

(1.1) Structure séquentielle.



Syntaxe.
Un élément Ai est une instruction ou une partie d'instruction :
soit une instruction du pseudo-langage : une référence à un raffinement,

soit une instruction ou partie d'instruction du langage cible :
- affectation,
- appel de procédure externe,
- commentaire.

Notation graphique.

- Un commentaire est précédé d'un tiret (ex. : A2 est un commentaire).
- La continuation d'une instruction à la ligne suivante est précédée de trois points (ex.: A4 est la continuation de A3); d'où l'on peut dire qu'un Ai est une instruction ou une partie d'instruction.

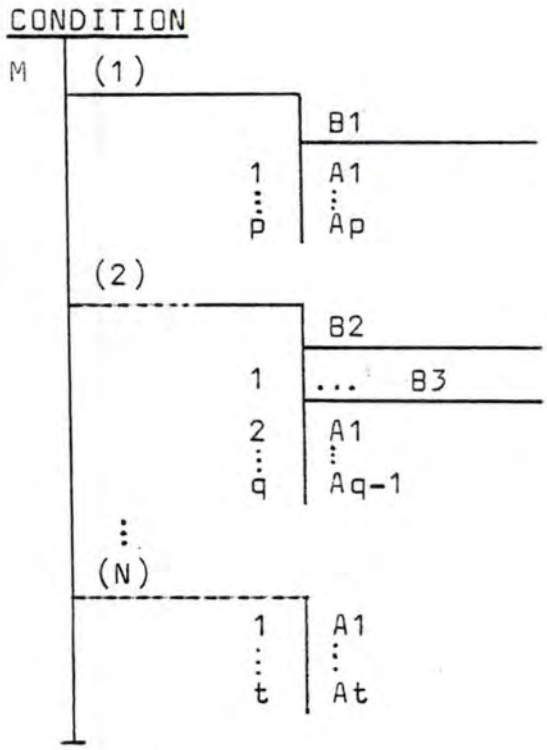
Sémantique.

Si Ai est une instruction, elle doit être exécutée.
Si Ai est une partie d'instruction, on ne peut pas définir de sémantique en dehors du contexte de cette partie d'instruction.

Numérotation.

Chaque élément Ai est précédé d'un numéro. Ce numéro est généré par l'éditeur de Schémacode à partir du début de chaque raffinement.

(1-2) Structure conditionnelle.



Syntaxe.

Un élément Bi est une expression booléenne ou une partie d'expression booléenne écrite en langage cible.
Un élément Ai a la même signification que dans les structures séquentielles :

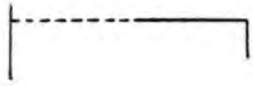
soit une instruction du pseudo-langage : une référence à un raffinement,

soit une instruction ou partie d'instruction du langage cible :
- affectation,
- appel de procédure externe,
- commentaire.

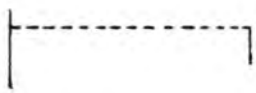
Schémacode impose certaines contraintes :
- une structure conditionnelle doit toujours commencer par une condition,
- chaque alternative doit contenir au moins un traitement : affectation, appel de procédure externe ou référence à un raffinement,
- une structure conditionnelle ne peut contenir qu'une seule alternative "ELSE",
- une alternative ne peut contenir deux conditions.

Notation graphique.

représente l'alternative "IF...THEN"



représente une alternative "ELSE IF...THEN



représente l'alternative "ELSE"

Les expressions booléennes ou parties d'expressions booléennes sont soulignées.

Sémantique.

L'exécution de la structure conditionnelle ci-dessus se déroule de la manière suivante :

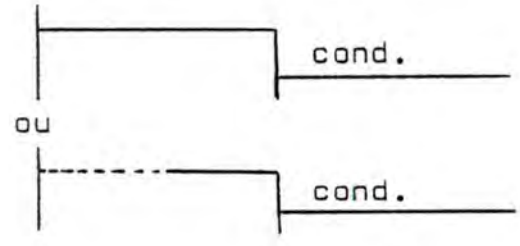
- on évalue B1, soit v1 sa valeur
  si v1 = vrai, on exécute A1...Ap et l'exécution de la conditionnelle est terminée,
  si v1 = faux, alors
  on évalue B2B3, soit v2 sa valeur
  si v2 = vrai, on exécute A1...Aq-1 et l'exécution de la conditionnelle est terminée
  si v2 = faux, on exécute A1...At et l'exécution de la conditionnelle est terminée.

Numérotation.

La numérotation est générée par l'éditeur de Schémacode.
La structure conditionnelle est la Même structure du raffinement. Chaque alternative de la conditionnelle porte un numéro de 1 à N dans la structure, N n'étant pas limité.
Dans chaque alternative, les éléments portent un numéro de 1 à p,q ou t, tous trois $\leq$ 9; le numéro 0 est implicitement réservé à l'expression booléenne ou à la première partie de l'expression.

Instructions de pseudo-langage associées à la structure conditionnelle.

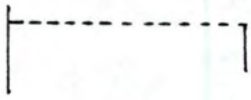
1. Condition.



syntaxe : expression booléenne en langage cible,
sémantique : sémantique PASCAL.

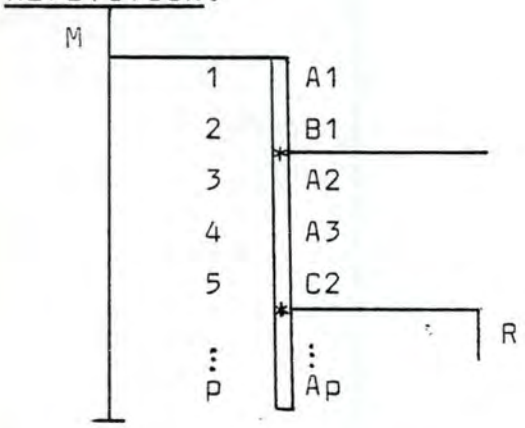
2. "Instruction "SINON".

Ce que nous appelons "instruction SINON" est en fait un délimiteur entre les premiers cas d'une structure conditionnelle et le dernier cas "ELSE".



(1.3)Structure répétitive.

REPETITION.



Syntaxe.

Un élément Bi est une expression booléenne ou partie d'expression booléenne écrite en langage cible.
Un élément Ci est une expression booléenne écrite en langage cible.
Un élément Ai a la même signification que dans les autres structures :

soit une instruction de pseudo-langage : une référence à un raffinement,
soit une instruction ou partie d'instruction du langage cible :

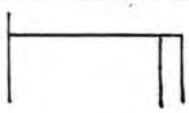
- affectation,
- appel de procédure externe,
- commentaire.

R est un numéro de raffinement (ijk) imposé par l'éditeur de Schémacode.

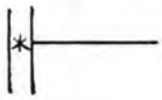
Schémacode impose certaines contraintes :

- une structure répétitive doit contenir au moins une condition de sortie de type Bi ou Ci,
- une structure répétitive doit contenir au moins un traitement : affectation, appel de procédure externe ou référence à un raffinement.

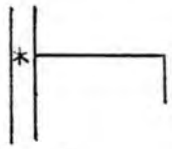
Notation graphique.



exprime qu'on est dans une structure répétitive.



représente une condition de sortie de la répétitive. L'expression booléennne est soulignée et peut continuer à la ligne suivante.



représente une condition de sortie de la répétitive avec référence à un raffinement; l'expression booléenne est soulignée et elle ne peut pas se prolonger à la ligne suivante.

Sémantique.

L'exécution de la structure répétitive ci-dessus s'effectue de la façon suivante :

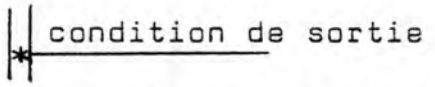
- on exécute A1
- on évalue B1, soit v1 sa valeur
  si v1 = vrai, l'exécution de la répétitive est terminée,
  si v2 = faux, alors
    - on exécute A2 et A3
    - on évalue C2, soit v2 sa valeur
    si v2 = vrai, on exécute le raffinement dont le numéro est R et l'exécution de la répétitive est terminée,
    si v2 = faux, on exécute les instructions suivantes jusque Ap, une exécution de la structure répétitive est terminée et on l'exécute à nouveau.

Numérotation.

La numérotation est générée par l'éditeur de Schémacode. La structure répétitive est la Mème du raffinement. Chaque élément Ai et chaque élément d'expression booléenne Bi ou Ci porte un numéro de 1 à P dans la répétitive ($P \leqslant 9$).

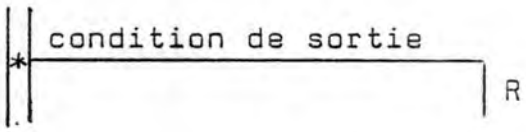
Instructions du pseudo-langage associées à la structure répétitive.

1. Condition de sortie.



sy : expression booléenne exprimée en langage cible,
se : sémantique PASCAL.

2. Condition de sortie avec référence à un raffinement.



sy : expression booléenne rédigée en langage cible, à laquelle l'éditeur de Schémacode associe un numéro de raffinement,
se : à l'exécution, on évalue l'expression booléenne selon les règles d'évaluation PASCAL.
Si le résultat de l'évaluation est "vrai", on exécute le raffinement dont le numéro est R.

A la section 3.2.1 nous avons défini une "référence à un raffinement" comme un commentaire "opérationnel" et un numéro de raffinement. Ici la condition de sortie remplit en même temps la fonction de commentaire, le raffinement de numéro R aura le texte de la condition comme nom.

(2.4)Commentaire narratif :

|-ceci est un commentaire

sy : texte libre

l'éditeur de Schémacode génère un tiret devant un commentaire narratif.

(2.5)Commentaire opérationnel :

|ceci est un commentaire opérationnel (ijk)

sy : le commentaire opérationnel est un texte libre d'au plus 72 caractères, énoncé d'une étape de résolution dans le programme en cours de développement.
L'éditeur de Schémacode associe un numéro de raffinement (ijk) à ce commentaire.

Nous avons défini l'association d'un commentaire opérationnel et d'un numéro de raffinement comme une "référence à un raffinement" (cfr. section 3.2.1.)

se : A l'exécution, le raffinement dont le numéro est associé au commentaire est exécuté.

(2.6)Instruction à coder en 1e colonne :

|<instruction à coder en 1e colonne

sy : instruction à coder en 1e colonne dans un programme FORTRAN

se : instruction destinée au compilateur FORTRAN.
Puisque nous nous situons dans un environnement PASCAL, cette instruction constitue une erreur.

(2.7)Continuation :

|... partie d'instruction

sy : partie d'instruction rédigée en langage cible : partie d'affectation, de déclaration ou d'appel de procédure externe. Il peut aussi s'agir de la continuation d'une condition, rédigée en langage cible ou d'un commentaire narratif.

se : à l'exécution, la "partie d'instruction" forme un tout avec la ligne précédente.

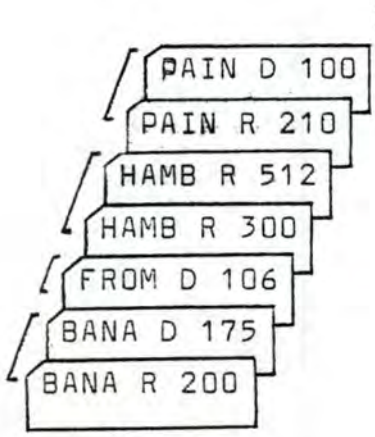
Remarque : l'instruction vide n'existe pas, Schémacode la refuse.

3.2.3 UN EXEMPLE.

Enoncé du problème :

Le département entrepôt de Mc Schem reçoit et distribue des articles. Chaque livraison reçue ou distribuée est enregistrée sur une carte perforée qui contient le nom de l'article, le type de l'opération (R pour réception, D pour distribution) et la quantité affectée. Ces cartes de transactions sont triées en ordre alphabétique pour éditer chaque semaine un rapport de gestion. Ce rapport doit identifier le changement net pour chaque article ainsi que le nombre d'articles affectés par les transactions.

ENTREE



SORTIE

RAPPORT D'INVENTAIRE

Articles	Changement net
----	----
----	----
----	----
----	----
----	----

Nombre de groupes modifiés =

RESOLUTION.

Le Raff 0 contient un bref rappel du problème à résoudre et une première décomposition de celui-ci.
En effet, après avoir déclaré les variables (Raff 1) et fait les initialisations (Raff 2) il faudra

imprimer l'en-tête du rapport de gestion (Raff 3)
imprimer le corps de celui-ci (Raff 4)
et imprimer le résumé (Raff 5).

L'impression du corps (Raff 4) consiste en l'impression d'une ligne par groupe; et ce, tant que le fichier contient des transactions. Nous supposons que ITEM contient la transaction courante.

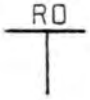
Pour pouvoir effectuer cette impression, il faut tout d'abord retenir le nom du nouveau groupe d'articles dans la variable "NOMARTICLE" et initialiser le changement net de l'article relatif à ce groupe à 0 (NETCHG). Ensuite, il faut calculer le changement net de ce groupe (Raff 6) puis effectuer l'impression de la ligne relative à ce groupe (Raff 7). Ensuite, le nombre de groupes rencontrés doit être augmenté de 1 (CMPTGR) et on recommence pour les groupes suivants.

Pour le calcul des changements nets de groupe (Raff 6), il faut additionner ou soustraire la quantité de chaque transaction appartenant à ce groupe. Pour cela, il faut calculer la quantité affectée par la transaction (Raff 8), mettre à jour le changement net de groupe (NETCHG) et ensuite lire une nouvelle transaction.

Le calcul de la quantité affectée de la transaction (Raff 8) dépend du type de la transaction (D ou R) si le type est D (distribution), la quantité est à soustraire; sinon elle est à additionner.

Les impressions de l'en-tête (Raff 3), du résumé (Raff 5) et d'une ligne du groupe (Raff 7) peuvent être détaillées. Les déclarations des variables (Raff 1) rappeleront la signification de chacune de celles-ci.
Enfin l'initialisation (Raff 2) effectue la première lecture et initialise le compteur de nombre de groupes rencontrés à 0.

Lorsqu'on a créé le fichier destiné à contenir le programme, on ne peut éditer que le Raff 0.
Le système génère le délimiteur de début de raffinement.



et au fur et à mesure que l'utilisateur introduit les éléments du raffinement, Schémacode génère également les numéros des éléments (de 1 à 18).

Lorsqu'on introduit une référence à un raffinement, Schémacode impose le numéro du raffinement créé (1 à 5).

A la fin de l'édition du Raff 0, Schémacode génère le délimiteur de fin de raffinement 

L'utilisateur peut alors choisir quel raffinement il va développer.

Cet exemple est tiré de (Ber, 80)

RAFFINEMENTS.

```
0.   R0
     1  -  PROGRAM MCSCHEM (DONNEES, OUTPUT);
     2  -  Programme de mise-à-jour de l'inventaire d'un entrepôt
     3  -  d'aliments. Cet entrepôt reçoit (R) ou distribue (D)
     4  -  des aliments. Chaque transaction est enregistrée sur
     5  -  une carte perforée contenant comme information :
     6  -        le nom de l'aliment
     7  -        le type de transaction
     8  -        la quantité affectée
     9  -  L'ensemble des cartes est trié en ordre alphabétique
    10  -  sur le nom des aliments.
    11  -  Le rapport de gestion est imprimé toutes les semaines.
    12  -  * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    13      DECLARATION (001)
    14      INITIALISATION (002)
    15      IMPRESSION EN-TETE (003)
    16      IMPRESSION CORPS (004)
    17      IMPRESSION RESUME (005)
    18      STOP
     R0
```

1. Lors de l'édition du Raff 1, Schémacode affiche directement à l'écran le délimiteur de début de raffinement et le nom du Raff 1 comme premier commentaire; l'utilisateur peut développer le raffinement.

R1

```
1   -   DECLARATION
2   -   -----------
3   -   NOMARTICLE : nom de l'article groupe courant
4   -   CMPTGR     : nombre de groupes rencontrés
5   -   NETCHG     : changement net du groupe courant
6   -   DONNEES    : nom du fichier des données
7   -   ITEM       : transaction courante
8   -   GROUPE     : nom de l'article de la transaction courante
9   -   RD         : type de la transaction
10  -   QTE        : quantité affectée de la transaction courante
11      TYPE
12        TRANSACTION = RECORD
13                      GROUPE : PACKED ARRAY  1..4  OF CHAR;
14                      RD : CHAR;
15                      QTE : INTEGER
16                      END;
17        FICHIN = FILE OF TRANSACTION;
18      VAR   DONNEES : FICHIN;
19            ITEM : TRANSACTION;
20            CMPTGR, NETCHG : INTEGER;
21            NOMARTICLE : PACKED ARRAY  1..4  OF CHAR;
```

R0

A la fin de l'édition du raffinement, le système génère le délimiteur de fin de raffinement $\frac{\perp}{R0}$ qui signifie que le Raff 1 a été créé au Raff 0.

2. L'édition des raffinements 2 et 3 se fait de la même manière que celle du Raff 1.

R2

```
1   -    INITIALISATION
2   -    --------------
3   -    initialisation du nombre de groupes rencontrés
4        CMPTGR : = 0;
5   -    lecture de la première transaction
6        RESET (DONNEES);
7        ITEM : = DONNEES↑ ;
```

R0

R3

```
1   -    IMPRESSION EN-TETE
2   -    ------------------
3        WRITELN; WRITELN ('' : 28, 'RAPPORT DE L "INVENTAIRE');
4        WRITELN ('' : 27, '*' : 25);
5        WRITELN;
6        WRITELN ('' : 10, 'ARTICLES', '' : 10, 'CHANGEMENT NET');
```

R0

3. L'édition du Raff 4 se fait de la même manière que celle des raffinements précédents.
Mais le problème à résoudre nécessite d'être raffiné, d'où les références aux raffinements 6 et 7.

```
R4
1   -  IMPRESSION DU CORPS
2   -  ------------------
3
                1   EOF
                *
                2      - initialisation du nouveau groupe
                3        NOMARTICLE : = ITEM.GROUPE;
                4        NETCHG : = 0;
                5        CALCUL CHANGEMENT NET DU GROUPE (006)
                6        IMPRESSION LIGNE DU GROUPE (007)
                7      - modification du nombre de groupes rencontrés
                8        CMPTGR : = CMPTGR + 1;
RO
```

```
R5
1   -  IMPRESSION RESUME
2   -  ----------------
3      WRITELN: WRITELN ('' : 4, '-' : 42);
4      WRITELN;
5      WRITELN ('' : 4, 'NOMBRE DE GROUPES MODIFIES= ',CMPTGR);
6      WRITELN;
RO
```

4. L'édition des derniers raffinements se fait toujours de la même manière. Remarquons que les délimiteurs de fins de raffinements représentent le numéro de raffinement d'où est référencé le raffinement délimité.

```
R6
1   - CALCUL CHANGEMENT NET DU GROUPE
2   - ------------------------------
3
          1   ITEM GROUPE    NOMARTICLE
              *
          2   CALCUL QUANTITE AFFECTEE (008)
          3   - Mettre à jour le changement net
          4     NETCHG : = NETCHG + ITEM.QTE;
          5   - Lecture d'une nouvelle transaction
          6     GET (DONNEES);
          7     ITEM : = DONNEES ↑ ;
R4
```

```
R7
1   -   IMPRESSION LIGNE DU GROUPE
2       WRITELN ('' : 12, ITEM.GROUPE, '' : 17, NETCHG);
R4
```

```
R8
1   -   CALCUL QUANTITE AFFECTEE
2       1
                          RD = 'D'
                    1     ITEM.QTE : = - ITEM.QTE;
R6
```

3.2.4 LES OUTILS EXISTANT DANS L'ENVIRONNEMENT SCHEMACODE.

3.2.4.1 Architecture générale de Schémacode.

Schémacode se compose de quatre modules :

1. l'EDITEUR :
Il permet d'introduire un programme et de le modifier; il transforme la représentation textuelle du programme en une représentation interne : le fichier SPC.
L'éditeur fait également une analyse syntaxique sur les structures SPC, comme nous l'avons expliqué à la section 3.1.1.

2. le MODULE D'IMPRESSION :
Il effectue le processus inverse de l'éditeur:à partir de la représentation interne d'un programme, il produit sur papier la représentation textuelle de l' algorithme.

3. le CODEUR FORTRAN :
Il traduit en code FORTRAN la représentation interne de l' algorithme développé grâce à Schémacode orienté FORTRAN.

4. le CODEUR PASCAL:
Il traduit en code PASCAL la représentation interne de l' algorithme développé grâce à Schémacode orienté PASCAL.

Le seul interface entre les différents modules est le fichier SPC : il est produit par l'éditeur et tous les autres modules travaillent à partir de ce fichier.
En fait, pendant l'édition, on travaille sur un fichier à accès direct; à la fin de l'édition, il est recopié dans le fichier SPC, fichier à accès séquentiel.

3.2.4.2 Structure interne du fichier SPC.

L'introduction d'un algorithme via l'éditeur de Schémacode produit le fichier SPC.

(1) Structure logique du fichier SPC.

- le fichier SPC est composé d'un ou plusieurs RAFFINEMENTS et au maximum 300 raffinements,
- un RAFFINEMENT est composé d'une ou plusieurs STRUCTURES,
- une STRUCTURE est de type séquentielle, conditionnelle ou répétitive. La représentation interne des différentes structures n'est pas homogène :
  - ° les structures conditionnelles et répétitives se composent de plusieurs INSTRUCTIONS,
  - ° une structure séquentielle se compose d'un seul ELEMENT,
- une INSTRUCTION se compose d'un ou plusieurs ELEMENTS,
- un ELEMENT correspond à un RECORD physique.

(2) Structure physique du fichier SPC.

Le fichier SPC est une suite de RECORDS de longueur fixe; on distingue deux types de records :

(2.1) Les cinquante premiers RECORDS.

Chacun d'eux est divisé en 6 articles et chaque article contient les renseignements décrits ci-dessous pour un des 300 raffinements du fichier SPC.

Le 1e record contient ces renseignements pour les raffinements 0 à 5,
le 2e record contient ces renseignements pour les raffinements 6 à 11,
et ainsi de suite,
le 50e record contient ces renseignements pour les raffinements 294 à 299.

L'article se rapportant au raff i a donc une place fixe.
Les articles ont la forme suivante :

PREM	DERN	NBSTRU	UTIL	ERR

PREM : adresse de la première structure du raffinement,
DERN : adresse de la dernière structure du raffinement,
NBSTRU : nombre de structures dans le raffinement,
UTIL : concerne l'utilisation du raffinement

vaut -1 si le raffinement n'existe pas,
vaut 0 si le raffinement est libre,
vaut 1 si le raffinement existe et est attaché au programme contenu dans le fichier,

ERR : concerne l'existence du raffinement

vaut False si le raffinement existe,
vaut True si le raffinement n'existe pas.

Pour l'exemple exposé à la section précédente (3.2.3) on aurait :
- l'article se rapportant au Raff 0

51	73	18	1	F

- l'article se rapportant à chaque raffinement au delà du 8e :

0	0	0	-1	T

Pour un raffinement libre, on aurait par exemple :

105	109	5	0	F

(2.2) <u>Tous les autres records du fichier</u> :

Ils sont en nombre illimité; nous allons les décrire et montrer ensuite comment une structure est représentée dans le fichier SPC grâce à ces records.

(2.2.1) Description.

Les records ont la forme suivante :

RAFF	STRUC	TYPE	TEXTE	POINTEUR

RAFF : numéro du raffinement que compose ce record.

STRUC : type de la structure que compose ce record
vaut S pour une structure séquentielle,
C pour le 1e élément d'une structure conditionnelle,
R pour le 1e élément d'une structure répétitive,
␣ "espace" pour tous les autres éléments d'une structure conditionnelle et d'une structure répétitive.

TYPE : type de l'instruction que contient ce record
vaut A pour une affectation,
D pour une déclaration,
P pour un appel de procédure externe,
T pour un commentaire narratif,
O pour un commentaire opérationnel,
X pour une instruction à coder en 1e colonne,
B pour une condition (expression booléenne) dans une structure conditionnelle,
E pour "l'instruction SINON",
S pour une condition de sortie d'une structure répétitive, avec ou sans référence à un raffinement,
- pour une continuation.

TEXTE : suite de 72 caractères, représente le texte de l'instruction; c'est ce que nous avons appelé ELEMENT dans la structure logique du fichier SPC.

POINTEUR : liste de 5 pointeurs :

* le premier et le second pointeurs ne sont pas utilisés,
* les pointeurs n° 3 et 4 sont utilisés dans les instructions qui référencent un raffinement, c'est-à-dire la condition de sortie avec raffinement d'une structure répétitive et le commentaire opérationnel :
  le pointeur numéro 3 contient le numéro du raffinement référencé,
  le pointeur numéro 4 contient l'adresse du 1e record du raffinement référencé,
* le pointeur numéro 5 contient l'adresse du record suivant :
  - le record contenant l'élément suivant de l'instruction courante,

  ou - le record contenant le 1e élément de l'instruction suivante dans une structure,

  ou - le record contenant le 1e élément de la structure suivante,

  ou - pour le dernier élément d'un raffinement, le pointeur numéro 5 référence le record contenant l'élément qui a créé ce raffinement (un commentaire opérationnel).
  Dans le cas d'un raffinement libre, l'élément qui a créé le raffinement n'existe plus; le pointeur numéro 5 du dernier record d'un raffinement libre vaut zéro.

(2.2.2) Représentation des structures dans le fichier SPC.

* La structure conditionnelle.

Le record contenant le 1e élément de la structure contient les valeurs suivantes :

RAFF : numéro du raffinement où figure la structure,
STRUC : 'C',
TYPE : 'B',
TEXTE : expression booléenne de la condition,
POINTEUR numéro 5 : référence au second record de la structure.

Ensuite pour chacun des records contenant les éléments qui composent la structure conditionnelle :

RAFF : reste inchangé,
STRUC : vaut 'espace',
TYPE : prend une des valeurs de l'ensemble suivant (A, D, P, T, O, X, B, E, -) tout en respectant la syntaxe des structures conditionnelles exposée à la section 3.2.2 (1.2),
TEXTE : le texte de l'instruction,
POINTEUR numéro 5 : adresse du record suivant, tel que défini plus haut.

* La structure répétitive.

Le record contenant le 1e élément de la structure contient les valeurs suivantes :

RAFF : numéro du raffinement où figure la structure,
STRUC : 'R',
TYPE : prend une des valeurs de l'ensemble suivant (A, D, P, T, O, X, S).

Pour chacun des records contenant les éléments qui composent la structure :

RAFF : reste inchangé,
STRUC : vaut 'espace',

TYPE : prend une des valeurs de l'ensemble (A, D, P, S, T, O, X, -) tout en respectant la syntaxe de la structure répétitive définie à la section 3.2.2 (1.3).

* La structure séquentielle.

Cette structure ne comprend jamais qu'un seul record contenant le seul élément de la structure :

RAFF : numéro du raffinement où figure la structure,
STRUC : 'S',
TYPE : prend une des valeurs de l'ensemble suivant : (A, D, P, T, O, X, -).

3.2.4.3. Les routines de Schémacode.

Certaines routines déjà écrites dans le logiciel Schémacode nous seront utiles pour le codeur. Ce sont les routines de lecture du fichier à accès direct sur lequel on travaille (cf section 3.2.4.1.) : VARTIC, LIT, LOCAL et des routines utilitaires telles que l'adressage de l'écran, l'affichage et la transormation d'un nombre entier en une chaîne de caractères : ADRE, FPUTC, ALPHA.

REMARQUE : VARTIC, LOCAL et LIT sont liées à la structure interne du ficheir SPC décrite à la section précédente.

(1) VARTIC

VARTIC est une routine d'accès à l'annuaire du fichier à accès direct (copie de fichier SPC), c'est-à-dire aux cinquante premières fiches.
A partir d'un numéro de raffinement, VARTIC lit l'article correspondant à ce raffinement.

En entrée, VARTIC a un numéro de raffinement, entier de 0 à 299.

En sortie, VARTIC fournit pour le raffinement demandé, la valeur des variables PREM, DERN, NBSTRU, UTIL et ERR décrites à la section précédente.

VARTIC appelle LOCAL,
est appelée par la routine CRENIV du codeur.

(2) LOCAL

LOCAL localise dans l'annuaire la position des renseignements relatifs à un raffinement.

En entrée, LOCAL a un numéro de raffinement, entier de 0 à 299.

En sortie, LOCAL fournit

- le numéro de la fiche (entier de 1 à 50) dont fait partie l'article relatif au raffinement demandé,

° l'indice de début de l'article dans la fiche, c'est-à-dire l'indice du premier renseignement.

LOCAL est appelée par VARTIC.

(3) LIT

LIT est une routine d'accès au fichier à accès direct (copie du fichier SPC).

En entrée, LIT a un numéro de fiche, entier plus grand que 50.

En sortie, LIT fournit pour la fiche demandée, la valeur des variables RAFF, STRUC, TYPE, TEXTE, et PNTR décrites à la section précédente.

LIT est appelée par des routines du codeur : CRENIV, TRTRAF et LECT.

(4) ADRE

ADRE est la routine qui positionne le curseur à l'écran au point voulu de coordonnées (x,y).

En entrée, ADRE a les coordonnées de la position désirée pour le curseur : entiers x et y.

En sortie, la position du curseur est (x,y).

ADRE est appelée par la routine IMPECR du codeur.

(5) FPUTC

FPUTC est la routine d'affichage ou d'impression d'une chaîne de caractères.

En entrée, FPUTC a
° une chaîne de caractères,
° le nombre de caractères dans la chaîne
° un code d'identification des caractères de contrôle à utiliser selon qu'on imprime ou qu'on affiche.

En sortie, la chaîne de caractères est affichée ou imprimée.

FPUTC est appelée par la routine IMPECR du codeur.

(6) ALPHA

ALPHA est une routine de conversion d'un nombre entier en une chaîne de caractères.

En entrée, ALPHA a
- l'entier à convertir,
- n, le nombre de caractères désiré dans le résultat.

En sortie, ALPHA fournit un tableau de caractères, de dimensions (1,n) dans lequel le nombre converti est cadré à droite.

ALPHA est appelée par des routines du codeur : TRTRAF, REPETS, COMOP et PTSOR.

3.2.4.4 Interface avec l'utilisateur : le langage de commande.

Schémacode possède 2 niveaux d'interface avec l'utilisateur : ils sont respectivement appelés MODE MONITEUR et MODE EDITEUR. Chacun d'eux affiche son identification dans le coin inférieur gauche de l'écran.
Cette identification est :: pour le mode moniteur,
.. pour le mode éditeur.

Le MODE MONITEUR est considéré comme le premier niveau d'interaction entre l'utilisateur et le programme. Il permet d'avoir accès aux composants du logiciel Schémacode : éditeur, module d'impression, codeur FORTRAN et codeur PASCAL.

Le MODE EDITEUR consiste en un second niveau d'interaction de l'utilisateur avec Schémacode. Il prend cours lorsque le moniteur a passé la main à l'éditeur, c'est-à-dire lorsque l'utilisateur demande au moniteur l'édition d'un raffinement. L'utilisateur ne peut éditer qu'un seul raffinement à la fois; il a donc une vue morcellée du programme.
Les commandes de l'éditeur permettent d'ajouter, supprimer ou modifier une "INSTRUCTION" ou une "STRUCTURE" dans le raffinement édité; elles ne peuvent être invoquées que si l'identificateur de mode (..) est présent à l'écran.

Voici le plan de ce qui est détaillé dans cette section :
(1) Langage de commande pour l'introduction de structures et d'intructions.
(1.1) Introduction de structures.
(1.1.1) Structure conditionnelle.
(1.1.2) Structure répétitive.
(1.1.3) Structure séquentielle.

(1.2) Introduction d'instructions.

(1.2.1) Affectation.

(1.2.2) Déclaration.

(1.2.3) Appel de procédure externe.

(1.2.4) Commentaire narratif.

(1.2.5) Commentaire opérationnel.

(1.2.6) Instruction à coder en 1e colonne.

(1.2.7) Condition.

(1.2.8) Continuation.

(1.2.9) Fin.

(2) Langage de commande pour le positionnement et la modification de structure ou d'instruction.

(2.1) Positionnement.

(2.2) Modification.

(3) Langage de commande d'effacement.

(3.1) Effacement simple.

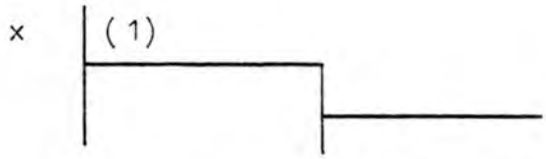
(3.2) Effacement d'un raffinement.

(1) Langage de commande pour l'introduction de structures et d'instructions.

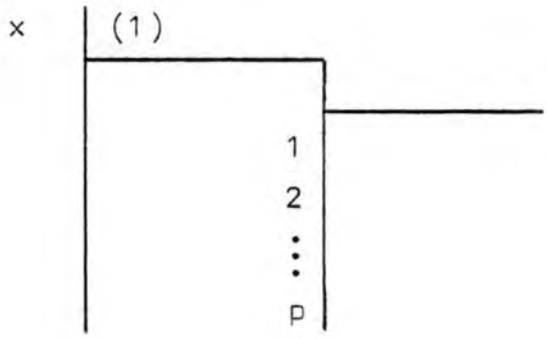
(1.1) Introduction de structures.

(1.1.1) Structure conditionnelle.

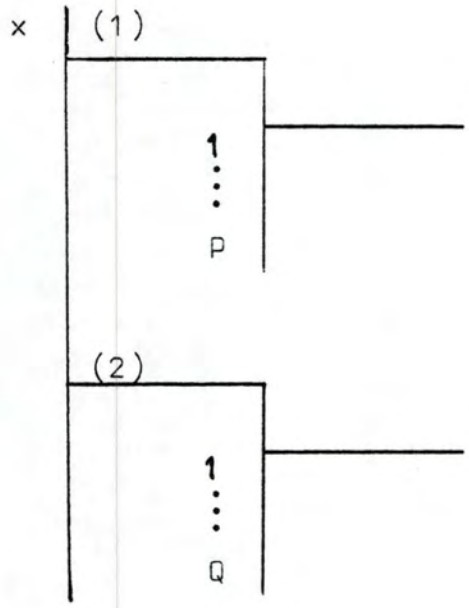
* L'introduction de la commande "C" en mode éditeur (..C) produit à l'écran une représentation graphique de la première alternative ("IF ... THEN") d'une conditionnelle, qu'on appellera aussi "cas" ou "branche".



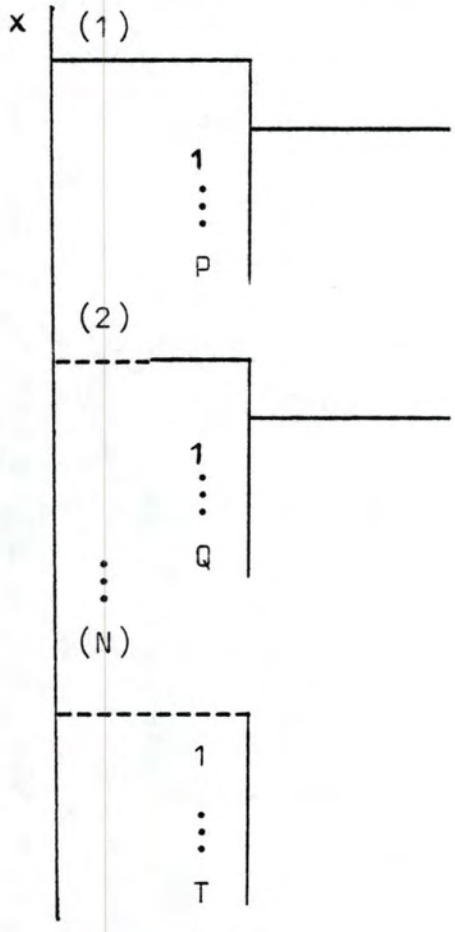
L'utilisateur doit alors introduire le texte de la condition booléenne en langage cible et les actions correspondant à cette alternative grâce aux commandes d'introduction d'instructions.



* L'introduction d'une nouvelle alternative "ELSE IF ... THEN" est amorcée par une nouvelle invocation de la commande "C", qui produit à l'écran une représentation graphique correspondante. La garniture de la branche "ELSE IF" est identique à celle de la branche "IF".



* La commande "O" (pour OTHER) produit à l'écran la dernière branche d'une conditionnelle excluant toutes les autres alternatives précédentes. La garniture de la branche "ELSE" est identique, à ceci près qu'il n'y a pas de condition.

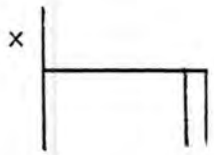


* La fin d'une structure conditionnelle est signalée par la commande "F".

* Limitations : Rappelons les deux limitations évoquées à la section 3.2.2 :
  - on ne peut pas imbriquer de structure conditionnelle ou répétitive,
  - il ne peut y avoir plus de 9 éléments par branche.

(1.1.2) Structure répétitive.

* L'introduction de la commande "R" en mode éditeur (.. R) produit à l'écran une représentation graphique de la structure répétitive.



L'utilisateur doit alors introduire dans un ordre quelconque :
- une ou plusieurs actions, grâce aux commandes d'introduction d'instructions,
- une ou plusieurs conditions de sortie de la répétitive. Deux types de conditions de sortie peuvent être introduites suivant la commande utilisée :

C permet d'introduire une condition simple, c'est-à-dire une expression booléenne exprimée dans le langage cible,

CR permet d'introduire une condition de sortie avec raffinement; la condition est une expression booléenne exprimée dans le langage cible. La commande "CR" a pour effet de créer automatiquement un raffinement associé au point de sortie; cela signifie qu'à l'exécution, si on sort de la boucle parce que cette condition est devenue vraie, on exécute le raffinement référencé par la condition de sortie avant de continuer le traitement normalement.

Relevons que la condition de sortie avec raffinement permet de programmer des exceptions; mais il est regrettable qu'on doive programmer les exceptions en repassant par le langage de commande.

* La fin d'une structure répétitive est signalée par la commande "F".

* Limitations : Rappelons les 2 limitations évoquées à la section 3.2.2. :

- On ne peut pas imbriquer de structure conditionnelle ou répétitive.
- Il ne peut y avoir plus de 9 éléments dans la répétitive.

(1.1.3) Structure séquentielle.

Lorsqu'on n'est ni dans une structure conditionnelle, ni dans une structure répétitive, par défaut on est dans une structure séquentielle; il n'y a pas de commande particulière pour éditer une structure séquentielle. L'utilisateur introduit les instructions grâce aux commandes d'introduction d'instructions.

(1.2) Introduction d'instructions.

(1.2.1) Affectation : ..A

Cette commande permet d'introduire une affectation écrite en langage cible; elle permet également d'introduire des instructions de lecture et d'écrite rédigées en langage cible.

(1.2.2) Déclaration : ..D

Cette commande permet d'introduire une déclaration formulée dans le langage cible.

(1.2.3) Appel de procédure externe : ..P

Cette commande permet d'introduire un appel de procédure ou de sous-routine externe au programme développé; cet appel doit être rédigé en langage cible.

(1.2.4) Commentaire narratif : ..T

Cette commande permet d'introduire un texte libre; il doit contenir au moins un caractère différent du caractère blanc " ".

(1.2.5) Commentaire opérationnel : .. G ou .. GN

Cette commande est celle qui donne la possibilité de reproduire le raisonnement descendant explicité par le programmeur sous forme de commentaires insérés aux points adéquats du programme; elle est donc utilisée lorsqu'on désire énoncer brièvement l'essentiel d'un traitement qui sera affiné ultérieurement.

G permet d'introduire une référence à un raffinement.
GN permet de rattacher le raffinement dont le numéro est N. (cf. section 3.2.1)

(1.2.6) Instruction à coder en 1e colonne : .. X

Cette commande permet d'introduire une instruction FORTRAN qui devra être codée en 1e colonne dans la traduction de l'algorithme Schémacode en FORTRAN.
Ce concept n'existe pas en PASCAL; la commande X ne devrait donc pas être utilisée dans Schémacode orienté PASCAL.

(1.2.7) Condition : .. C

Cette commande permet d'introduire une expression booléenne écrite en langage cible.

(1.2.8) Continuation : .. +

Cette commande permet d'introduire la suite d'une instruction introduite à la ligne précédente par l'une des commandes A, D, T, P ou C. C'est donc cette commande qui permet qu'une instruction se compose de plusieurs éléments.
Remarquons que le commentaire opérationnel ne peut être continué à la ligne suivante.

(1.2.9) Fin : .. F

Cette commande a deux fonctions selon le moment auquel on l'invoque :
- elle termine la construction d'une structure conditionnelle ou répétitive, auquel cas on reste dans le mode éditeur,
- elle termine l'exécution du mode éditeur et assure le retour au mode éditeur.

2. Langage de commande pour le positionnement et la modification de structure ou d'instruction.

(2.1) Positionnement.

Lors de l'édition d'un raffinement, on peut éditer :
* une structure du raffinement
  .. = M où M est le numéro de la structure par rapport au début du raffinement.
  Rappelons qu'une structure séquentielle se compose d'un seul élément.

* un élément d'une structure répétitive
  .. = M P où M est le numéro de la structure,
  P est le numéro de l'élément dans la structure.

* une branche (un cas) d'une structure conditionnelle
  .. = M N où M est le numéro de la structure,
  N est le numéro de la branche.

* un élément dans une structure conditionnelle

.. = M N P où M est le numéro de la structure,
N est le numéro de la branche,
P est le numéro de l'élément dans la branche.

(2.2) Modification.

Une fois positionné sur l'élément à modifier, la commande "S délimiteur chaîne 1 délimiteur chaîne 2" permet de remplacer "chaîne 1" par "chaîne 2" dans le texte de l'élément. Un délimiteur peut être un caractère quelconque qui n'existe dans aucune des 2 chaînes (*, /, + , $, ...)
Il y a cependant une exception : Schémacode refuse la modification du nom d'un raffinement. En effet, pour rester cohérent, il faut que le texte de la référence et le nom du raffinement restent identiques. Or, dans la représentation interne des raffinements, la liaison entre une référence à un raffinement et le nom du raffinement qu'elle crée est à sens unique. A partir du nom d'un raffinement, on ne sait pas retrouver automatiquement la référence à ce raffinement. Par contre, à partir de la référence à un raffinement (cf section 3.2.3.2 (2.2.1), on retrouve facilement le nom du raffinement. Pour modifier un nom de raffinement, on est donc obligé de modifier le texte de la référence qui crée le raffinement.

(3) Langage de commande d'effacement.

On utilise le mot "effacement" plutôt que le mot "suppression". En effet, lorsqu'on "enlève" un ou plusieurs éléments dans la représentation textuelle de l'algorithme, la place qu'ils occupaient dans la représentation interne reste vide, Schémacode ne compacte pas la représentation interne.

(3.1) Effacement simple ..E ou ..E M

Lors de l'édition d'un raffinement, on peut effacer n'importe quelle structure du raffinement à l'exception de son nom (le 1e commentaire narratif du raffinement). Par définition, un raffinement a toujours un nom, dès sa création (cf section 3.2.1).

..E M efface la structure de numéro M du raffinement

..E efface la structure courante du raffinement, c'est à dire la dernière structure éditée.

(3.2) Effacement d'un raffinement ER ou ER M

Parmi les structures du raffinement édité, il peut y avoir une référence à un raffinement.

.. ER M efface la référence à un raffinement qui est la Même structure du raffinement courant et efface en même temps le raffinement correspondant à cette référence.

.. ER efface la référence à un raffinement sur laquelle on s'est positionné au préalable et efface en même temps le raffinement correspondant à cette référence.

Remarque : l'utilisation de la commande "E" ou "EM" pour effacer une référence à un raffinement a un effet différent : le raffinement considéré devient libre (cf section 3.2.1).

3.3 TECHNIQUES DE GENERATION : POSSIBILITES ET CHOIX.

Nous avons envisagé deux types de techniques pour générer du code PASCAL à partir d'un programme en pseudocode schématique :

- soit le remplacement de texte,
- soit la génération de code structuré en niveaux.

Pour bien visualiser les différentes possibilités, nous avons traduit l'exemple présenté à la section 3.2.2 selon les techniques proposées. Voir annexe 1.

3.3.1 LE REMPLACEMENT DE TEXTE.

Le code correspondant à chaque raffinement est inséré dans la programme PASCAL à l'endroit où ce raffinement est référencé.

Avantage : on obtient ainsi un programme d'un seul tenant.

Inconvénients :

- la structure du raisonnement n'est pas reproduite dans le code généré,
- la maintenance risque d'être coûteuse : la correction du texte d'une instruction signifie une nouvelle traduction et une nouvelle compilation de la totalité du programme.

Le code généré pourrait être de la forme suivante :

```
PROGRAM nom du programme;
Déclaration des constantes
            des types
            des variables
```

```
A1
A2
(nom du raff  2)
  A3
  A4
  (nom du raff  4)
    A10
    A11
  A5
  A6
(nom du raff  3)
  A7
  A8
  (nom du raff  5)
    A12
    A13
  A9
FIN
```

où

- les Ai représentent les instructions PASCAL générées par le codeur, correspondant aux instructions et aux structures de l'algorithme en pseudocode schématique,
- les noms des raffinements sont mis en commentaire dans le programme PASCAL

et l'indentation est produite par le codeur.

3.3.2 GENERATION DE CODE STRUCTURE EN NIVEAUX.

(1) Comme indiqué dans la section 3.2.1 l'utilisation de Schémacode construit en fait, via l'éditeur, un arbre de raffinements. Dans cet arbre on peut identifier une structure en niveaux :

- le niveau 0 ne reprend que le raff 0,
- le niveau 1 reprend tous les raffinements référencés par le raff 0,
- le niveau i reprend tous les raffinements référencés par les différents raffinements du niveau (i - 1).

Si l'on veut obtenir du code PASCAL reflétant la philosophie de la décomposition par raffinements successifs, la technique du remplacement de texte n'est pas adéquate. Nous proposons donc une autre solution qui consiste à générer un code reflétant cette structure en niveaux.

Avantages:
- le code obtenu est plus proche de la philosophie de Schémacode qui se veut appuyer une démarche de résolution descendante par raffinements successifs,
- le regroupement des procédures en niveaux atténue le caractère morcellé du programme généré.

Inconvénients :
- le niveau d'un raffinement dépend de l'endroit d'où il est référencé et non pas de son importance logique,
- du fait que Schémacode interdit l'imbrication des structures, on est forcé de créer un nouveau raffinement dans lequel on développe la structure à imbriquer. On risque donc d'avoir des raffinements courts et nombreux et un certain morcellement du programme.

Deux alternatives sont possibles pour produire du code structuré en niveaux :

- soit du code PASCAL avec GOTO,
- soit du code PASCAL avec procédures.

(2) Structure du code en niveaux avec GOTO.

(2.1) Principe.

Au niveau i, la référence à un raffinement du niveau (i + 1) se traduit par un GOTO à l'étiquette de ce raffinement (étiquette d'entrée). A la fin du raffinement référencé, un GOTO renvoie à l'étiquette de l'instruction du niveau i suivant la référence au raffinement qui vient d'être détaillé. (étiquette de retour).

(2.2) La numérotation des étiquettes.

Etiquette WXYZ

où W = numéro du niveau où l'on va,
X = numéro du paragraphe où l'on va, c'est à dire numéro de raffinement dans le niveau,
Y = numéro de l'étiquette dans le paragraphe,
Z = 0 si c'est une étiquette d'entrée,
1 si c'est une étiquette de retour.

(2.3) Le code généré pourrait dès lors être de la forme suivante :

PROGRAM nom du programme;
Déclaration

Déclaration des constantes
des types
des variables

```
                Niveau 0
                §1 nom du_raff _0_
                  A1
                  A2
                  T nom du raff  2    (niveau 1, § 1)
                  GOTO étiq 1110
étiq 0111         T nom du raff  3    (niveau 1, § 2)
                  GOTO étiq 1210
étiq 0121         FIN

                Niveau 1
étiq 1110         § 1 nom du_raff _2_
                  A3
                  A4
                  T nom du raff  4    (niveau 2, § 1)
                  GOTO étiq 2110
étiq 1121         A5
                  A6
                  GOTO étiq 0111

étiq 1210         § 2 nom du_raff _3_
                  A7
                  A8
                  T nom du raff  5    (niveau 2, § 2)
                  GOTO étiq 2210
étiq 1221         A9
                  GOTO étiq 0121

                Niveau 2
étiq 2110         § 1 nom du_raff _4_
                  A10
                  A11
                  GOTO étiq 1121
étiq 2210         § 2 nom du_raff _5_
                  A12
                  A13
                  GOTO étiq 1221
```

Où

- les mots niveau i et § i sont du commentaire généré par le codeur,
- les Ai représentent les instructions PASCAL générées par le codeur , correspondant aux instructions et aux structures de l'algorithme écrit en pseudocode schématique,
- les T correspondent à du commentaire du pseudocode schématique,
- les commentaires entre parenthèses (niveau i, § j) sont générés par le codeur, ils signifient que le raffinement correspondant à la référence qui précède sera détaillé au niveau i, § j.

Suggestion pour la lisibilité :

L'emploi des GOTO pourrait être réservé à la gestion des aller-retour entre niveaux, ce qui implique que toute autre structure du pseudocode schématique soit traduite sans GOTO.

Avantage : l'avantage est celui évoqué pour un code structuré en niveaux : on reste plus proche de la philosophie de Schémacode.

Inconvénients: outre ceux inhérents au code structuré en niveaux (niveau arbitraire d'un raffinement, risque de morcellement du programme),

- on obtient un code peu esthétique,
- ce type de codage ne permet pas de développer un programme sur plusieurs unités de compilation séparées,
- et enfin, la maintenance risque d'être coûteuse.

(3) Structure de code en niveaux avec procédures.

(3.1) Organisation du code.

- Le Raff 0 devient le programme principal.
- Pour chaque raffinement :
  la référence à un raffinement devient un appel de procédure et le détail de ce raffinement référencé devient la déclaration de cette procédure.
- En PASCAL, la déclaration d'une procédure doit précéder tout appel à celle-ci. Or, dans la décomposition en niveaux, la déclaration des procédures devrait venir après l'appel de celle-ci.
  D'où, toutes les procédures sont déclarées par une clause "FORWARD" en début de programme.
- Les déclarations de constantes, de types, de variables ne font pas partie de la structure hiérarchique en niveaux des raffinements.
  Elles font l'objet d'un traitement particulier : elles sont regroupées en tête du programme (cf section 3.4.3).

(3.2) Pour l'exemple précédent, on obtient le code suivant :

```
PROGRAM nom du programme;
Déclaration
déclaration des constantes
             des types
             des variables

Déclaration des procédures

R2; FORWARD;
R3; FORWARD;
R4; FORWARD;
R5; FORWARD;
```

Niveau 1

R2; (nom_de la_procédure_=_nom_du Raff_ 2)

A3

A4

Appel à R4; (niveau 2
nom du Raff 4)

A5

A6

R3; (nom_de la_procédure_=_nom_du Raff_ 3)

A7

A8

Appel à R5; (niveau 2
nom du Raff 5)

A9

Niveau 2

R4; (nom_de la_procédure_=_nom_du Raff_ 4)

A10

A11

R5; (nom de la_procédure_=_nom_du Raff_ 5)

A12

A13

Programme principal

A1

A2

Appel à R2; (niveau 1
nom du Raff 2)

Appel à R3; (niveau 1
nom du Raff 3)

FIN.

Où

- les commentaires soulignés en traits pleins sont générés par le codeur;
- les Ai représentent des instructions PASCAL générées par le codeur, correspondant aux instructions et aux structures de l'algorithme développé en pseudocode schématique;
- les commentaires "niveau i" qui accompagnent un appel de procédure sont générés par le codeur et précisent à quel niveau est détaillée la procédure correspondant à l'appel.

Outre les avantages et inconvénients inhérents aux codes structurés en niveaux, on peut ajouter que ce type de codage

- pourrait éventuellement permettre le développement de programme sur plusieurs unités de compilation,
- faciliterait la maintenance.

Remarque.

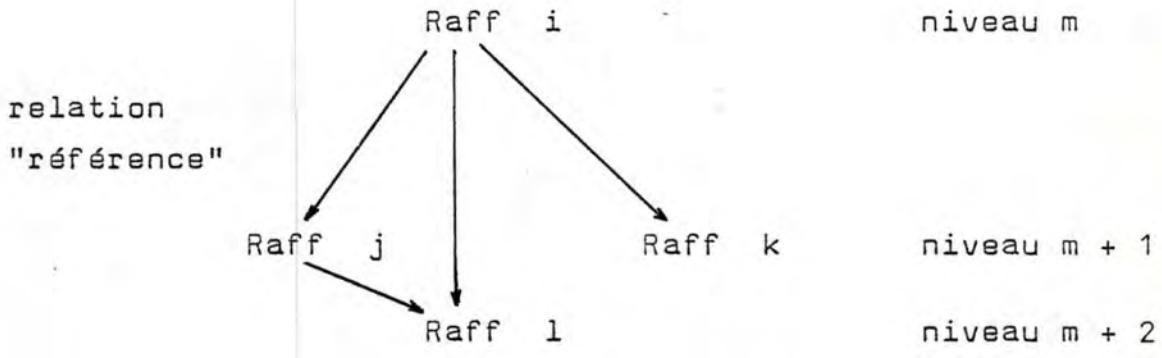
Actuellement, Schémacode ne permet pas de référencer un même raffinement de plusieurs endroits du programme.

Cette limitation provient de la représentation interne des raffinements, nous avons vu à la section 3.2.4.2 que la représentation interne d'un raffinement n'est en fait qu'une suite de records physiques.

Le dernier record d'un raffinement contient un pointeur vers le record de référence à ce raffinement, et un seul pointeur est prévu à cet effet. D'où l'impossibilité de référencer un raffinement de plusieurs endroits différents.

Cependant, si cette restriction était levée, on aurait une nouvelle définition de l'arbre des raffinements : un raffinement pourrait être référencé à plusieurs endroits, par des raffinements appartenant à des niveaux différents.

Supposons cette situation :



Le Raff l serait développé dans le niveau immédiatement inférieur au niveau du dernier raffinement qui le référence.

D'où une nouvelle définition de la structure en niveaux :

- le niveau 0 ne reprend que le Raff 0;
- le niveau 1 reprend certains raffinements référencés par le raff 0;
- le niveau i reprend des raffinements référencés par les différents raffinements des niveaux 0 à (i - 1).

C'est dans cette optique d'évolution de Schémacode que nous avons choisi la traduction des raffinements en procédures.

3.3.3 SOLUTION ADOPTEE.

Il est normal de concevoir qu'un raffinement soit référencé de plusieurs endroits du programme en pseudocode schématique. La technique du remplacement de texte serait assez lourde, puisque le même texte serait recopié plusieurs fois dans le code généré.

Nous avons donc éliminé la technique de remplacement de texte.

La notion de procédure est fort restreinte dans Schémacode : la seule instruction relative aux procédures est l'appel de procédure externe, rédigé en langage cible.

Schémacode ne permet pas la déclaration, le développement et l'appel à des procédures internes à un programme. En effet, toutes les déclarations sont introduites en langage cible grâce à la commande D. Comment pourrait-on dès lors introduire un traitement (le développement de la procédure) qui fasse partie des déclarations : il devrait être introduit en même temps grâce à la commande D, en tant que déclaration, et grâce aux commandes A, P, T, G, C ou R, en tant que traitement ? Cet aspect de la restriction de Schémacode ne touche que le langage de commande.

D'un autre côté, la notion de raffinement est une notion "d'unité logique de construction" d'un programme et la notion de procédure est une notion "d'unité physique lors de la compilation et de l'exécution" du programme.

Il faudrait donc introduire un nouvel attribut à chaque raffinement : de quelle unité de compilation et d'exécution fait-il partie ? Du programme, de la procédure X ou de la procédure Y internes au programme ?

Au vu des différents avantages et inconvénients discutés ci-dessus, la technique retenue a dès lors consisté à introduire la notion de procédure; dans la génération du code PASCAL :

- une procédure correspond à un raffinement,
- les procédures sont regroupées en niveaux.

3.4 INTRODUCTION AU CODEUR PASCAL.

3.4.1 Redéfinition de la structure séquentielle SPC.

Nous allons nous concentrer dans cette section et les sections suivantes sur l'outil réalisé : le codeur PASCAL. Afin de donner une sémantique à la structure séquentielle SPC et de faciliter les traitements nécessités par le codeur PASCAL, nous allons modifier la syntaxe et la sémantique de cette structure SPC.

Rappelons qu'en Schémacode la structure séquentielle n'est constituée que d'une instruction SPC ou d'une partie d'instruction SPC, c'est-à-dire d'un seul élément.

On a par exemple :

```
⌈S11
|...S12        où S1i est le ième élément de
⌊               l'instruction S1.
```

Au sens strict de la définition du pseudo-langage SPC, chacun des éléments S11 et S12 représente une structure séquentielle SPC.

Nous redéfinissons la syntaxe et la sémantique de la structure séquentielle SPC de la façon suivante :

Syntaxe.
La structure séquentielle SPC se compose toujours d'une seule instruction du pseudo-langage, c'est-à-dire éventuellement de plusieurs éléments.

Sémantique.
La sémantique de la structure séquentielle SPC est la sémantique de la seule instruction qui la compose.

On a par exemple :

```
S11
...S12          où S1i est le ième élément de
                l'instruction S1.
```

D'après la nouvelle définition de la structure séquentielle SPC les éléments S11 et S12 représetent une seule structure séquentielle SPC.

C'est de cette redéfinition de la structure séquentielle SPC, dont nous tiendrons compte pour la réalisation du codeur PASCAL.

3.4.2 Le cas particulier du raffinement numéro un.

3.4.2.1 Le contenu du raffinement numéro un et les contraintes associées.

En PASCAL, toutes les déclarations du programme doivent être faites avant toute instruction exécutable de ce programme. A cet effet, les déclarations du programme SPC doivent être regroupées dans le programme SPC, afin de les placer en tête du programme PASCAL généré.

Une solution aurait été de tester chaque instruction du programme SPC afin de savoir s'il s'agissait d'une déclaration; de regrouper ensuite toutes ces déclarations dans un fichier auxiliaire, puis de replacer le contenu de ce fichier, en bon ordre, dans le programme PASCAL généré. Un tel processus présente cependant certaines difficultés : les déclarations d'un programme PASCAL doivent se faire dans un ordre déterminé; il aurait dès lors fallu réaliser une analyse sémantique de chaque déclaration afin de la placer correctement par rapport aux déclarations déjà analysées.

Nous avons estimé que cette solution serait trop complexe pour un simple réarrangement des déclarations; c'est pourquoi l'utilisateur doit le prendre lui-même en charge. L'utilisateur doit donc regrouper toutes ses déclarations dans le raff 1 et les écrire dans l'ordre exigé par PASCAL.

Le raff 1 étant dédicacé aux déclarations, il ne peut donc contenir comme type de structures que des structures séquentielles et comme type d'instructions que des déclarations et des commentaires narratifs.

Lors de la génération du code PASCAL, le codeur n'a qu'à recopier le raff 1 pour obtenir les déclarations du programme PASCAL.

Une extension est à apporter quant au contenu du raff 1. En effet, l'utilisateur a la possibilité d'utiliser le système pour introduire des déclarations par raffinements successifs. Outre des déclarations et des commentaires narratifs, le raff 1 peut donc contenir des commentaires opérationnels.

L'utilisateur a pu, par exemple, introduire la situation suivante : le raff 1 référence successivement :

- le raff 2 qui contient la déclaration des constantes,
- le raff 3 qui contient la déclaration des types,
- le raff 4 qui contient la déclaration des variables,

le raff 4 référence successivement :

- le raff 5 qui contient les variables relatives aux entrées-sorties,
- le raff 6 qui contient les variables relatives au traitement.

Il est clair que toutes ces déclarations doivent être faites dans l'ordre PASCAL et que le codeur doit faire figurer l'ensemble de ces déclarations en un seul bloc dans le programme PASCAL. Le code PASCAL généré ne reflètera donc pas la structure descendante des raffinements de déclarations.

3.4.2.2 Technique de génération des déclarations et contraintes associées.

La technique de génération à adopter pour les raffinements de déclarations, ne peut être celle utilisée pour les raffinements d'instructions exécutables, à savoir produire du code structuré en niveaux avec procédures (cfr sous-section 3.3.2). En effet, il serait absurde de générer une procédure PASCAL qui ne contiendrait que des déclarations du programme SPC.

D'autre part, puisque l'ensemble des déclarations doit figurer en un seul bloc dans le programme PASCAL, le codeur produira les déclarations par remplacement de texte (cfr sous-section 3.3.1). Toutefois, le codeur doit être capable de différencier les raffinements de déclarations, des raffinements d'instructions exécutables puisque la technique de génération qu'il leur applique est différente. Une solution aurait été que le codeur différencie ces deux types de raffinements en examinant leur contenu. Il s'agissait alors de faire une analyse sémantique des instructions contenues dans le raffinement. Cette solution a été rejetée.

Une solution plus simple et plus immédiate est d'imposer à l'utilisateur de libérer le raff 1. Les raffinements de déclarations (le raff 1 et les raffinements référencés à partir du raff 1) constituent donc un groupe de raffinements isolés qui sont traduits séparément des autres par le codeur.

Remarque :
La libération du raff 1 pourrait être automatique. Avant de débuter l'étape du codage, une routine du codeur PASCAL devrait libérer le raff 1. Ceci reste un point à améliorer.

3.4.3 Relations "traduire" et "declar";règles pour le codage automatique.

Le codeur PASCAL traduit la représentation interne d'un programme SPC, en un programme PASCAL. Comme la technique de génération des déclarations est différente du reste du programme SPC (cfr point 3.4.2.2), nous définissons deux relations : la relation "traduire" et la relation "declar".

La relation "declar" notée "D" s'applique aux raffinements de déclarations, aux structures et instructions du pseudo-langage SPC qui peuvent les composer (cfr point 3.4.2.1). Soit D, qui à un des éléments du pseudo-langage SPC fait correspondre zéro, une ou plusieurs unités textuelles PASCAL.

Si Si est un élément du pseudo-langage SPC, nous noterons D(Si) la traduction de Si. Cette traduction appartient au langage PASCAL.
La relation traduire notée "T" s'applique au programme, aux raffinements d'instructions exécutables, aux structures et instructions du pseudo-langage SPC (sauf les déclarations). De la même façon que D, nous noterons T(Si) la traduction de Si; elle appartient également au langage PASCAL.

Ces deux relations vont nous permettre de déterminer un ensemble de règles pour le codage automatique.

3.4.3.1 Relation traduire : traduction d'un programme.

La relation traduire fait correspondre plusieurs unités textuelles PASCAL à un programme SPC. Son schéma de traduction est le suivant :

Soit le programme SPC P, constitué d'un ensemble de raffinement R0, D1... Dm, ...Rn,
soit D1 le raff 1,
soit R0 le raff 0,
soit $R^1 j$ l'ensemble des raffinements du niveau 1,
soit $R^2 j$ l'ensemble des raffinements du niveau 2,
⋮
soit $R^k j$ l'ensemble des raffinements du niveau k avec $k \leq 18$,
soit $\alpha^1 j$ la première instruction de chaque raffinement de Rj^1, un commentaire narratif,
soit $\alpha^2 j$ la première instruction de chaque raffinement de Rj,
⋮
soit $\alpha^k j$ la première instruction de chaque raffinement de $R^k j$,
soit γ une chaîne de caractères représentant le nom du programme SPC,
soit β une chaîne de caractères représentant le nom du fichier contenant le programme SPC.

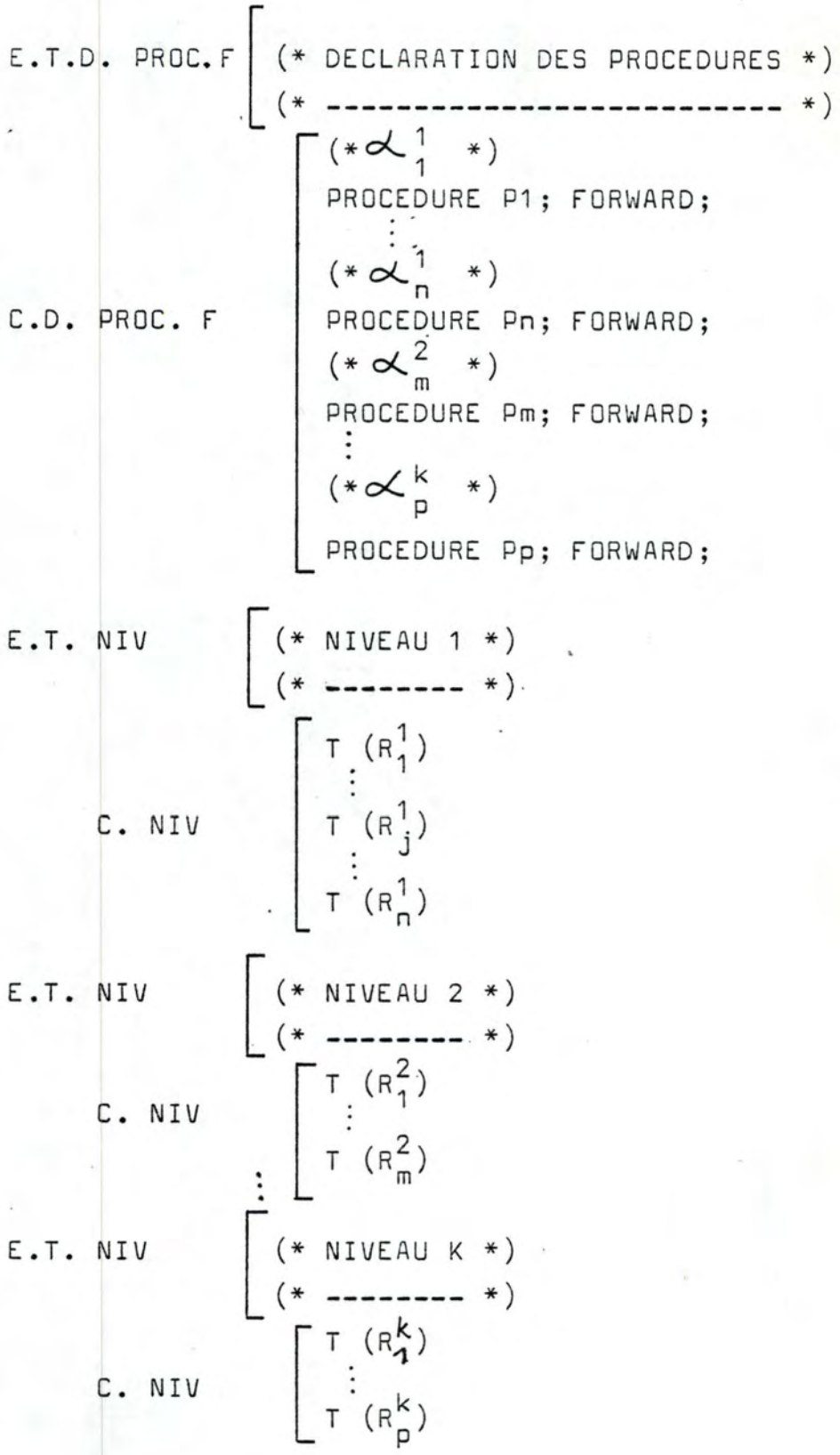
P —TRADUIRE→

E.T. PGRM
```
PROGRAM γ (OUTPUT);
(************************************)
(* β - Ecole Polytechnique de Montréal *)
```

E.T. DECLA
```
(* DECLARATION *)
(* ----------- *)
```

C. DECLA
```
D (D1)
```

F. DECLA
```
(* LITNU : VARIABLE GENEREE PAR LE CODEUR
   LITNU : ARRAY (/0... i/) OF BOOLEAN;
```


E.T.D. PROC.F
(* DECLARATION DES PROCEDURES *)
(* ------------------------- *)
(* α_1^1 *)
PROCEDURE P1; FORWARD;
(* α_n^1 *)
C.D. PROC. F
PROCEDURE Pn; FORWARD;
(* α_m^2 *)
PROCEDURE Pm; FORWARD;
(* α_p^k *)
PROCEDURE Pp; FORWARD;
E.T. NIV
(* NIVEAU 1 *)
(* -------- *)
C. NIV
T (R_1^1)
T (R_j^1)
T (R_n^1)
E.T. NIV
(* NIVEAU 2 *)
(* -------- *)
C. NIV
T (R_1^2)
T (R_m^2)
E.T. NIV
(* NIVEAU K *)
(* -------- *)
C. NIV
T (R_1^k)
T (R_p^k)

```
E.T.P.PRINC ⌈ (* PROGRAMME PRINCIPAL*)
            | (* ------------------*)
            ⌊ BEGIN

C.P. PRINC  ⌈ T (R0)

F.P. PRINC  ⌊ END.
```

La traduction d'un programme SPC contient les unités textuelles PASCAL suivantes :

- l'unité textuelle PASCAL "EN-TETE PROGRAMME".
  Cette unité se compose :
  - de la seconde instruction du raff 0, qui comprend le mot réservé PASCAL PROGRAM et le nom du programme SPC,
  - d'un commentaire PASCAL, qui se compose du nom du fichier contenant le programme SPC.

  Cette unité se note de façon abrégée E.T. PGRM.

- l'unité textuelle PASCAL "EN-TETE DECLARATIONS".
  Cette unité se compose d'un commentaire PASCAL contenant le mot "déclarations". Elle se note de façon abrégée E.T. DECLA.

- l'unité textuelle PASCAL "FIN DECLARATIONS".
  Cette unité se compose :
  - d'un commentaire PASCAL contenant : "litnu : variable générée par le codeur",
  - de la déclaration dans le langage cible, d'un tableau de boolean, de nom LITNU et de dimension i.
    i représente le plus grand numéro du raffinement référencé dans le programme SPC.

  Cette unité s'écrit de façon abrégée F. DECLA.

- l'unité textuelle PASCAL "EN-TETE DECLARATION PROCEDURES FORWARD".
  Cette unité se compose d'un commentaire PASCAL contenant les mots "déclaration des procédures".
  Cette unité s'écrit de façon abrégée E.T.D. PROC. F.

- l'unité textuelle PASCAL "CORPS DECLARATION PROCEDURES FORWARD".
  Cette unité se compose pour chaque raffinement référencé à partir du raff 0 :
  - d'un commentaire PASCAL contenant le nom du raffinement référencé. Ce nom est en fait la première instruction du raffinement référencé,
  - du mot réservé PASCAL PROCEDURE suivi du caractère P, du numéro du raffinement référencé et d'un point virgule.

  Cette unité se note de façon abrégée C.D. PROC. F.

- l'unité textuelle PASCAL "EN-TETE NIVEAU".
  Cette unité se compose d'un commentaire PASCAL contenant le mot "niveau", suivi de son numéro.
  Cette unité se note de façon abrégée E.T. NIV.

- l'unité textuelle PASCAL "EN-TETE PROGRAMME PRINCIPAL".
  Cette unité se compose :
  - d'un commentaire PASCAL contenant les mots "programme principal",
  - du mot réservé PASCAL BEGIN .

  Cette unité se note de façon abrégée E.T.P. PRINC.

- l'unité textuelle PASCAL "FIN PROGRAMME PRINCIPAL".
  Cette unité se compose du mot réservé PASCAL END suivi d'un point.
  Elle se note de façon abrégée F.P. PRINC.

Le corps des déclarations noté de façon abrégée C. DECLA.représente la traduction du raff 1.

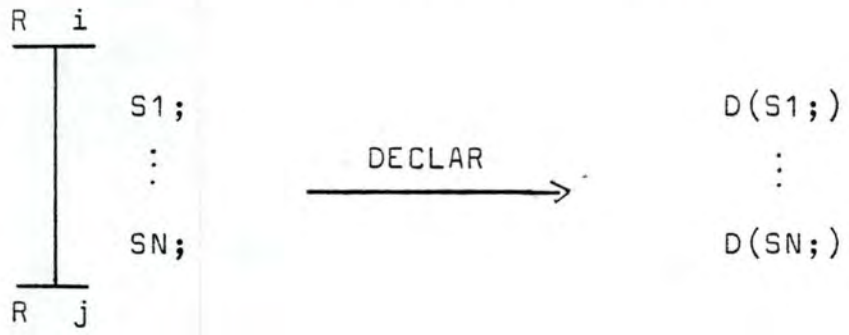
Le corps du niveau noté C. NIV représente la traduction de tous les raffinements qui figurent dans ce niveau.

Le corps du programme principal noté de façon abrégée C.P. PRINC représente la traduction du raff 0.

3.4.3.2 Relation "declar" : traduction d'un raffinement de déclarations.

Le schéma de traduction d'un raffinement de déclarations est le suivant :

soit Si, une structure du pseudo-langage SPC,

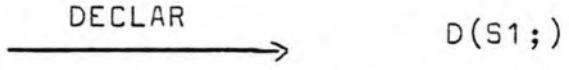


La traduction de ce raffinement est la traduction des structures séquentielles SPC qui le composent.

3.4.3.3 Relation "declar" : traduction d'une structure SPC.

Le schéma de traduction d'une structure séquentielle est le suivant :

soit S1 une structure séquentielle SPC,



La traduction de cette structure est la traduction de l'instruction qui la compose.

3.4.3.4 Relation "declar" : traduction des instructions SPC.

La relation "declar" fait correspondre zéro ou une unité textuelle PASCAL à une instruction SPC.

(1) la déclaration.

Puisque cette instruction SPC est formulée dans le langage cible, on a le schéma de traduction suivant :

soit S une déclaration SPC,

$$S; \xrightarrow{DECLAR} S;$$

La traduction d'une déclaration SPC comprend l'unité textuelle PASCAL composée d'une déclaration PASCAL.

(2) le commentaire narratif.

Son schéma de traduction est le suivant :

soit S un commentaire narratif SPC,

$$S \xrightarrow{DECLAR} (* \quad S \quad *)$$

La traduction du commentaire narratif SPC est l' unité textuelle PASCAL "COMMENTAIRE" se composant du libellé du commentaire narratif SPC délimité par les caractères PASCAL (/)/ *.

(3) le commentaire opérationnel.

Son schéma de traduction est le suivant :

soit S un commentaire opérationnel SPC auquel est associé un numéro de raffinement (i),

soit Di un raffinement de déclarations de numéro i,

$$S(i) \xrightarrow{DECLAR} D(Di)$$

La traduction de cette instruction SPC est la traduction du raffinement de déclarations du numéro i.

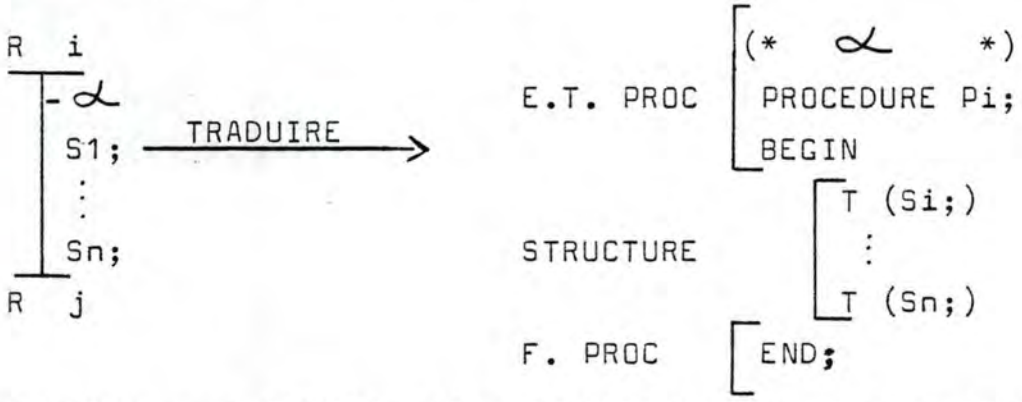
3.4.3.5 Relation "traduire" : traduction d'un raffinement.

La relation "traduire" fait correspondre une unité textuelle PASCAL à un raffinement du pseudo-langage SPC.

Le schéma de traduction d'un raffinement est le suivant :

soit , la chaîne de caractères représentant un commentaire narratif,

soit Si, une structure SPC,



La traduction, en PASCAL, d'un raffinement du pseudo-langage SPC s'appelle une procédure. Celle-ci comprend les deux unités textuelles PASCAL suivantes :

- l'unité textuelle PASCAL "EN-TETE PROCEDURE".

Cette unité se compose :

- d'un commentaire PASCAL qui reprend le libellé du premier commentaire narratif du raffinement. Ce libellé correspond au nom du raffinement,
- du mot réservé PASCAL PROCEDURE , suivi du titre de la procédure et d'un point virgule. Le titre de la procédure est formé du caractère P suivi du numéro du raffinement développé,
- du délimiteur PASCAL BEGIN .

Cette unité se note de façon abrégée E.T. PROC.

- l'unité textuelle PASCAL "FIN PROCEDURE" qui se compose du délimiteur PASCAL END, suivi d'un point virgule.

Cette unité se note de façon abrégée F. PROC.

Entre ces deux unités textuelles, on trouve le corps de la procédure noté STRUCTURE sur le schéma.

3.4.3.6 Relation "traduire" : traduction des structures.

La relation traduire fait correspondre une ou plusieurs unités textuelles PASCAL à une structure du pseudo-langage SPC.

(1) la structure séquentielle.

Son schéma de traduction est le suivant :

S1; $\xrightarrow{\text{TRADUIRE}}$ T(S1;)

La traduction d'une structure séquentielle SPC est la traduction de l'unique instruction SPC qui la compose.

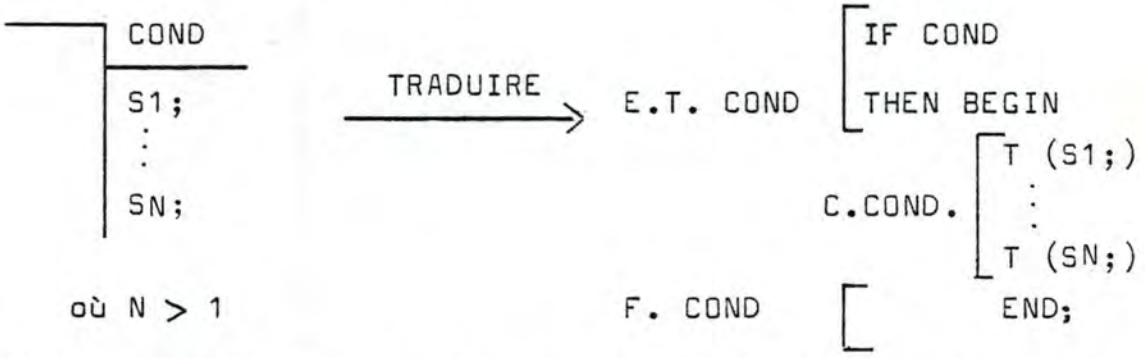
(2) la structure conditionnelle.

On appelle instruction conditionnelle, la traduction en PASCAL de la structure conditionnelle SPC. D'après les règles syntaxiques d'une structure conditionnelle, définies à la sous-section 3.3.2, nous remarquons qu'une structure conditionnelle peut prendre quatre formes distinctes.

(2.1) Le schéma de traduction de la première forme considérée est le suivant :

soit COND, une instruction conditionnelle SPC,

soit Si, une instruction SPC,



La traduction de cette forme de structure conditionnelle comprend les deux unités textuelles PASCAL suivantes :

- l'unité textuelle "EN-TETE DE LA CONDITIONNELLE".

Cette unité se compose :

- des mots réservé PASCAL IF/THEN ,
- du délimiteur PASCAL BEGIN ,
- de l'instruction conditionnelle SPC COND.

Cette unité se note de façon abrégée E.T. COND.

- l'unité textuelle PASCAL "FIN DE LA CONDITIONNELLE".

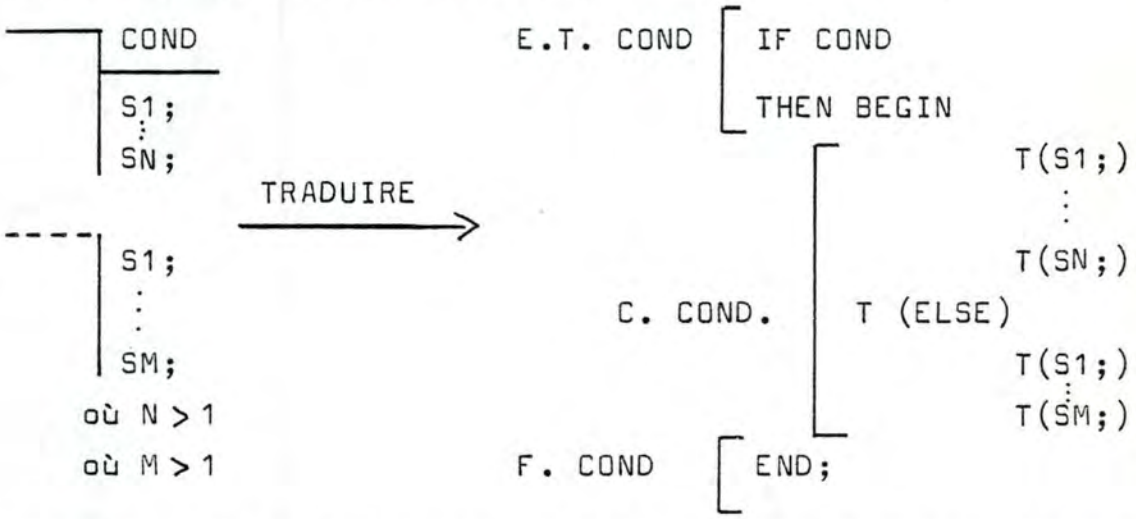
Cette unité se compose du délimiteur PASCAL END, suivi d'un point virgule et se note de façon abrégée F. COND.

Entre ces deux unités textuelles on trouve le corps de la conditionnelle noté de façon abrégée C. COND.

(2.2) Le schéma de traduction de la deuxième forme considérée est le suivant :

soit COND, une instruction conditionnelle SPC,

soit Si, une instruction SPC,



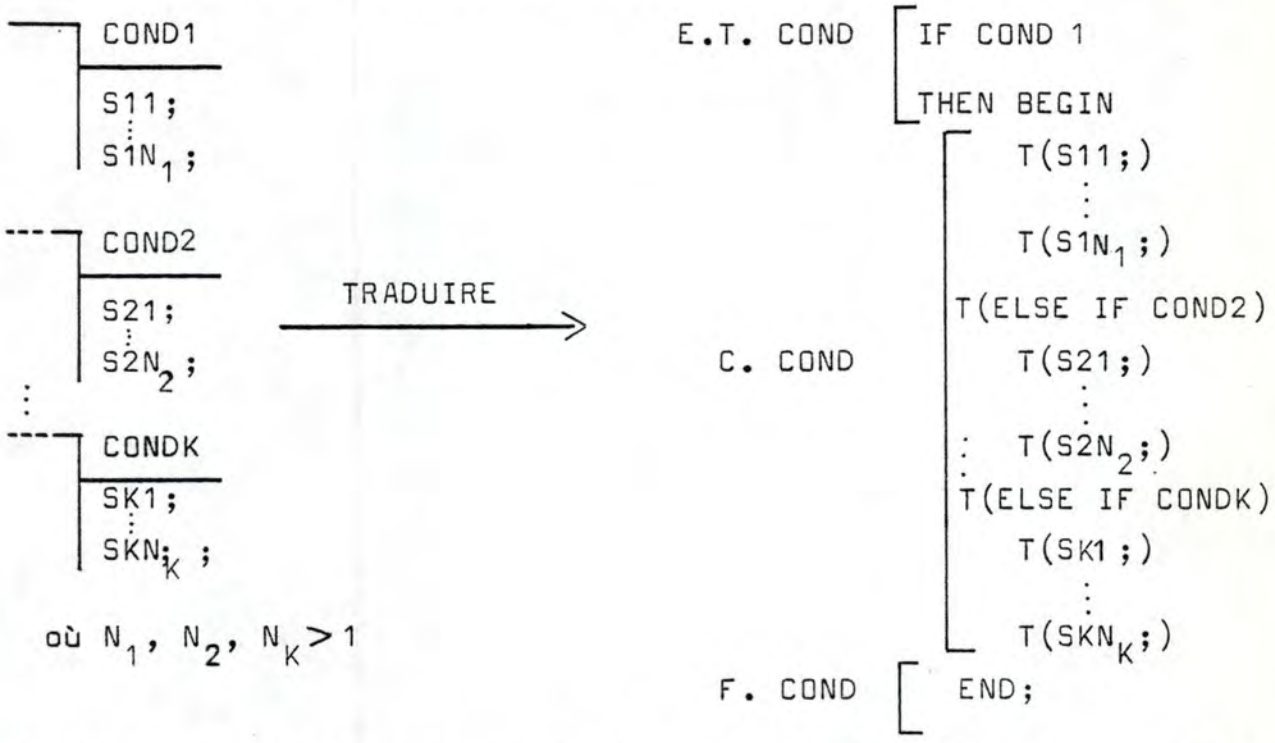
La traduction de cette deuxième forme d'une structure conditionnelle comprend les mêmes unités textuelles que celles qui figurent dans la traduction de la première forme.

Entre ces deux unités textuelles, on retrouve le corps de la conditionnelle noté C. COND.

(2.3) Le schéma de traduction de la troisième forme considérée est le suivant :

soit CONDj, une instruction conditionnelle SPC,

soit Si, une instruction SPC,

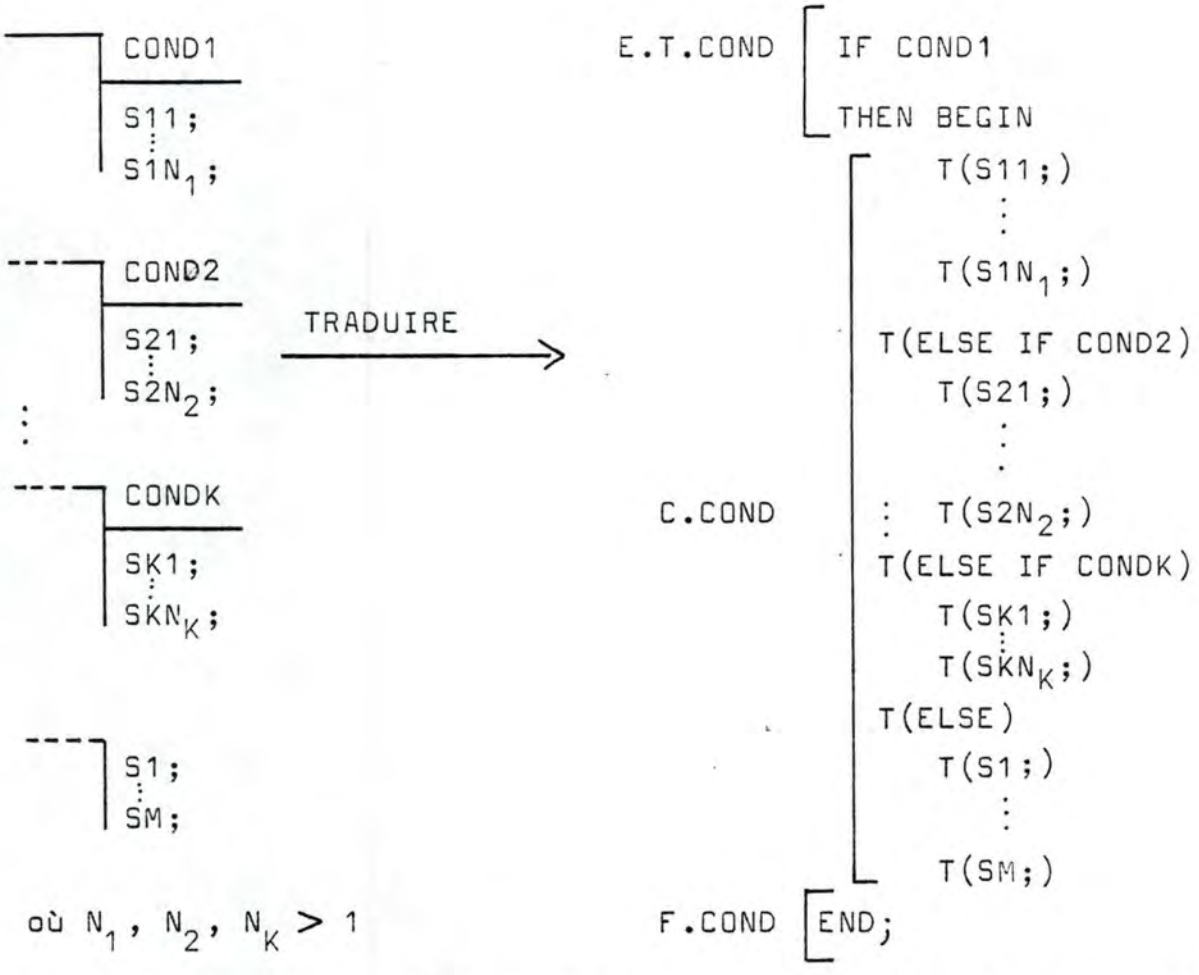


on retrouve, de nouveau, dans cette traduction les deux unités textuelles "EN-TETE" et "FIN CONDITIONNELLE", et entre les deux le corps de la conditionnelle.

(2.4) Le schéma de traduction de la dernière forme considérée est le suivant :

soit CONDj, une instruction conditionnelle SPC,

soit Si, une instruction SPC,



où N_1, N_2, $N_K > 1$

On retrouve dans cette traduction les mêmes unités textuelles PASCAL que précédemment.

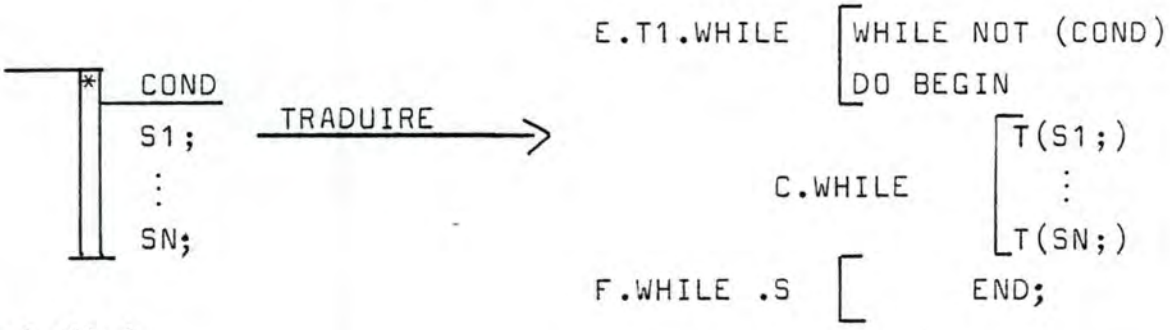
(3) la structure répétitive.

On appelle instruction répétitive, la traduction en PASCAL de la structure répétitive SPC. La syntaxe d'une structure répétitive nous permet de distinguer pour la traduction de cette structure, trois formes de structures répétitives.

(3.1) Nous considérons comme première forme la structure répétitive avec une seule condition de sortie, avec ou sans référence à un raffinement, en première position dans la structure.

(3.1.1) Si cette structure comprend plusieurs instructions SPC et une condition de sortie sans référence à un raffinement;

on a le schéma de traduction suivant : soit COND, une condition de sortie, soit S1 une instruction SPC,



où N > 1

La traduction de cette structure comprend les deux unités textuelles PASCAL suivantes :

- l'unité textuelle "EN-TETE1 WHILE"
  Cette unité se compose :
  - des mots réservés PASCAL WHILE/NOT/DO,
  - des délimiteurs PASCAL BEGIN/(/),
  - de la condition de sortie COND.

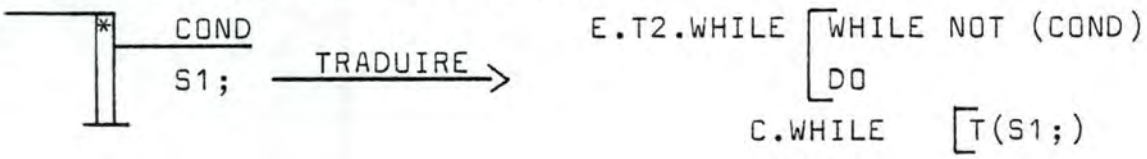
On note cette unité de façon abrégée E.T1.WHILE

- l'unité textuelle "FIN WHILE SIMPLE".
  Cette unité composée du délimiteur PASCAL END suivi d'un point virgule, se note de façon abrégée F.WHILE.S.

Entre ces deux unités textuelles, on trouve le corps du while, noté de façon abrégée C.WHILE.

(3.1.2) Si cette structure comprend une instruction SPC et une condition de sortie sans référence à un raffinement; on a le schéma de traduction suivant :

- soit COND, une condition de sortie,
- soit S1, une instruction SPC,

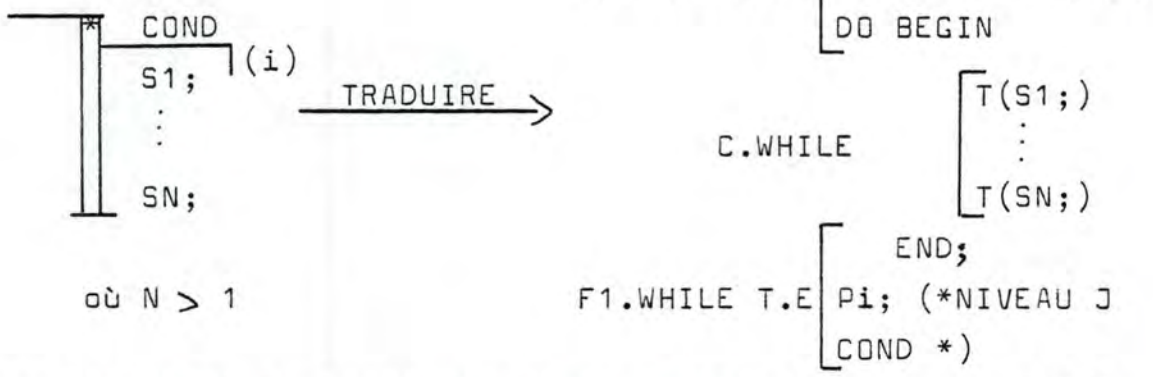


La traduction de cette forme de structure comprend l'unité textuelle PASCAL "EN-TETE 2 WHILE" noté de façon abrégée E.T2.WHILE.
Cette unité se compose :
- des mots réservés PASCAL : WHILE/DO/NOT,
- de la condition de sortie COND.

On trouve ensuite le corps du while noté de façon abrégée C.WHILE.

(3.1.3) Si la condition de sortie de cette forme de structure répétitive est une condition de sortie avec référence à un raffinement, on a, lorsque la structure comprend plusieurs instructions SPC, le schéma de traduction suivant :

- soit COND, une condition de sortie avec référence à un raffinement,
- soit (i), le numéro du raffinement associé à la condition de sortie,
- soit Sk, une instruction SPC,



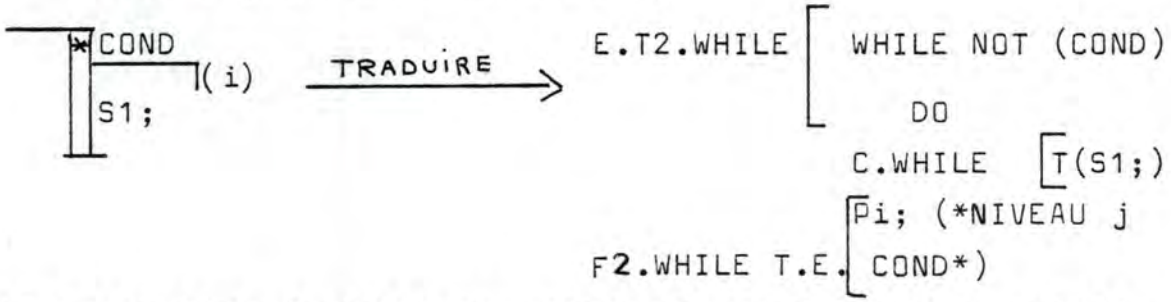
La traduction de cette structure comprend les deux unités textuelles PASCAL suivantes :
- l'unité textuelle "EN-TETE 1 WHILE".
  Cette unité a été décrite dans le point (3.1.1.),
- l'unité textuelle PASCAL "FIN 1 WHILE TRAITEMENT D'EXCEPTION".

Cette unité se compose :

- du délimiteur PASCAL END suivi d'un point virgule,
- d'un appel de procédure correspondant au raffinement référencé. Cet appel se compose du caractère P suivi du numéro du raffinement référencé, puis d'un point virgule,
- d'un commentaire PASCAL qui comprend :
  - le numéro du niveau dans lequel le raffinement est développé,
  - le nom du raffinement qui correspond au libellé de la condition de sortie.
    En effet, l'éditeur de Schémacode impose à l'utilisateur que le nom du raffinement, dans une condition de sortie avec référence à un raffinement, soit le libellé de cette condition de sortie.

(3.1.4) Si cette première forme de structure répétitive comprend une condition de sortie avec référence à un raffinement et une instruction SPC, on a le schéma de traduction suivant :

- soit COND, une condition de sortie avec référence à un raffinement,
- soit (i), le numéro de raffinement associé à la condition de sortie,
- soit S1, une instruction SPC,



La traduction de cette structure comprend les deux unités textuelles PASCAL suivantes :

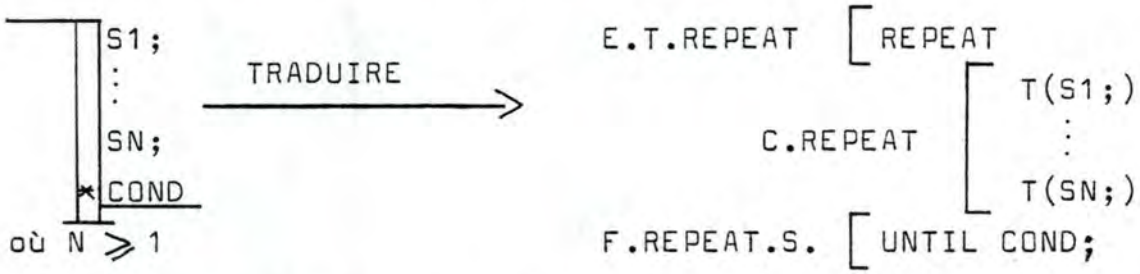
- l'unité textuelle PASCAL "EN-TETE 2. WHILE" qui a été décrite dans le point (3.1.2),

- l'unité textuelle PASCAL "FIN 2 WHILE TRAITEMENT D'EXCEPTION". Cette unité se compose d'un appel de procédure et d'un commentaire PASCAL qui ont été décrits dans le point (3.1.3).
- Entre ces deux unités textuelles PASCAL, on trouve le corps du while noté de façon abrégée C. WHILE.

(3.2) La deuxième forme de structure répétitive envisagée, est la structure répétitive avec une seule condition de sortie, avec ou sans référence à un raffinement, en dernière position dans la structure.

(3.2.1) Si la condition de sortie de cette forme de répétitive est une condition de sortie sans référence à un raffinement, quel que soit le nombre d'instructionsSPC de la structure (une ou plusieurs), on a le schéma de traduction suivant :

- soit COND, une condition de sortie,
- soit Si, une instruction SPC,



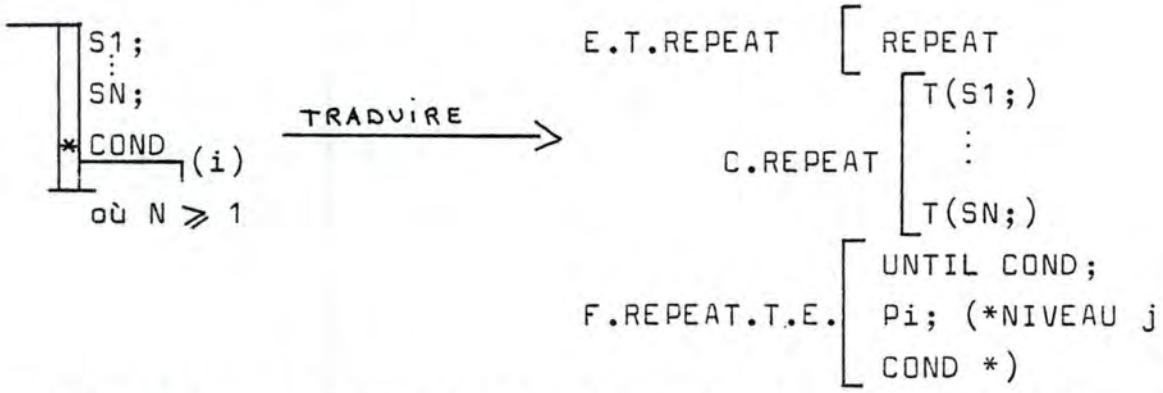
La traduction de cette forme de répétitive comprend les deux unités textuelles suivantes :

- l'unité textuelle "EN-TETE REPEAT". Cette unité se compose du mot réservé PASCAL REPEAT et se note de façon abrégée E.T. REPEAT,
- l'unité textuelle PASCAL "FIN REPEAT SIMPLE". Cette unité se compose :
  - du mot réservé PASCAL UNTIL ,
  - de la condition de sortie COND, suivie d'un point virgule.

Entre ces deux unités textuelles se trouve le corps de la structure répétitive noté de façon abrégée C.REPEAT.

(3.2.2) Si la condition de sortie de cette forme de structure répétitive est une condition de sortie avec raffinement, quel que soit le nombre d'instructions SPC de la structure (une ou plusieurs), on a le schéma de traduction suivant :

- soit Sk, une instruction SPC,
- soit COND, une condition de sortie avec référence à un raffinement,
- soit (i), le numéro du raffinement associé à cette condition de sortie,



La traduction de cette forme de structure produit les deux unités textuelles suivantes :

- l'unité textuelle "EN-TETE REPEAT".
  Elle se compose du mot réservé PASCAL REPEAT et se note de façon abrégée E.T.REPEAT.
- l'unité textuelle PASCAL "FIN REPEAT TRAITEMENT D'EXCEPTION".
  Cette unité se compose :
  - du mot réservé PASCAL UNTIL ,
  - de la condition de sortie COND, suivie d'un point virgule,
  - d'un appel de procédure et d'un commentaire PASCAL qui ont été décrits dans l'unité textuelle FIN 1 WHILE TRAITEMENT D'EXCEPTION.

  Entre ces deux unités textuelles se trouve le corps du repeat noté de façon abrégée C. REPEAT.

(3.3.) La troisième forme de structure répétitive à envisager est celle composée :

- soit d'une seule condition de sortie avec ou sans référence à un raffinement qui n'est ni en première, ni en dernière position de la structure,
- soit de plusieurs conditions de sortie avec ou sans référence à un raffinement.

Afin de faciliter la compréhension du lecteur, nous allons tout d'abord expliquer la sémantique de la traduction de cette forme de structure.
Prenons par exemple une structure répétitive qui comprend :

- une condition de sortie sans référence à un raffinement COND1,
- une condition de sortie avec référence à un raffinement COND2,
- plusieurs autres instructions SPC Sk.

Sa traduction PASCAL est la suivante :

```
                LITNU (/i/) : = FALSE;
                REPEAT
                  T(S11;)
                  ⋮
                  T(S1N1;)
    T(COND1)    IF COND1 THEN LITNU (/i/) : = TRUE
                         ELSE BEGIN
                  T(S21;)
                  ⋮
                  T(S2N2;)
                  IF COND2
                         THEN BEGIN
                               Pi; (*NIVEAU j*)
    T(COND2)                   LITNU (/i/) : = TRUE;
                              END
                         ELSE BEGIN
                  T(S31;)
                  ⋮
                  T(S3N3;)
                  END; END
                UNTIL LITNU (/i/);
```

où N_1, N_2, $N_3 > 1$.

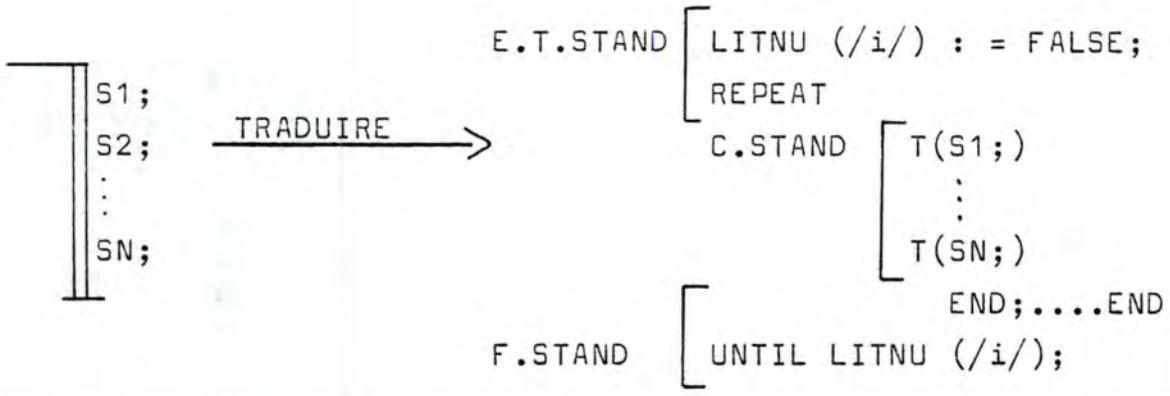
Sémantique de la traduction.

L'exécution de la traduction ci-dessus s'effectue de la façon suivante :

- on exécute T(S11) jusque $T(S1N_1)$
- on évalue COND1, soit v1 sa valeur
  - ° si v1 = vrai, - LITNU (/i/) prend la valeur vraie et l'exécution du THEN est terminée,
    - on exécute ensuite le test de fin de la répétitive qui est vrai; l'exécution de la répétitive est ainsi terminée,
  - ° si v1 = faux, - on exécute T(S21) jusque $T(S2N_2)$,
    - on évalue COND2, soit v2 sa valeur
      - ° si v2 = vrai
        - LITNU (/i/) prend la valeur vraie,
        - on exécute l'appel de procédure Pi, cette procédure correspond au raff i, l'exécution du THEN est terminé,
        - on exécute ensuite le test de fin de la répétitive qui est vrai; l'exécution de la répétitive est terminée,
      - ° si v2 = false
        - on exécute T(S31;) jusque $T(S3N_3;)$; une exécution de la structure répétitive est terminée et on l'exécute à nouveau.

Le schéma de traduction de cette troisième forme de structure répétitive est le suivant :

- soit S1...SN un ensemble d'instructions SPC qui contient au moins une condition de sortie avec ou sans référence à un raffinement.



La traduction de cette forme de structure comprend les deux unités textuelles suivantes :

- l'unité textuelle PASCAL "EN-TETE STANDARD".
  Cette unité se compose :
  - du mot réservé PASCAL REPEAT ,
  - de la variable indicée LITNU (/i/), anagramme de UNTIL, créée pour les besoins de la traduction, où i est le numéro du raffinement où figure la structure répétitive.
  Cette unité se note de façon abrégée E.T.STAND.
- l'unité textuelle PASCAL "FIN STANDARD".
  Cette unité se compose :
  - du mot réservé PASCAL UNTIL ,
  - de la variable indicée LITNU (/i/),
  - d'autant de délimiteurs PASCAL END qu'il n'y a de conditions de sortie dans la structure.

Entre ces deux unités textuelles PASCAL se trouve le corps de la répétitive traduite selon une forme standard noté C.STAND.

3.4.3.7 Relation "traduire" : traduction des instructions.

La relation traduire fait correspondre zéro ou une unité textuelle PASCAL, à une instruction du pseudo-langage SPC. Précisons ce qui en résulte, suivant le type de cette instruction SPC.

(1) l'affectation.

Puisque l'affectation SPC est formulée dans le langage cible, on a le schéma de traduction suivant :

soit S une affectation SPC

S; ——TRADUIRE——> S;

La traduction de cette instruction SPC se compose d'une unité textuelle PASCAL comprenant l'affectation SPC.

(2) l'appel de procédure externe.

Puisque l'appel de procédure externe SPC est formulé dans le langage cible, on a le schéma de traduction suivant :

soit S un appel SPC de procédure externe,

S; ——TRADUIRE——> S;

La traduction de cette instruction SPC se compose d'une unité textuelle PASCAL comprenant l'appel de procédure externe SPC.

(3) l'instruction à coder en première colonne.

La relation "traduire" ne fait correspondre aucune unité textuelle PASCAL à cette instruction.

(4) le commentaire narratif.

Son schéma de traduction est le suivant :

soit S un commentaire narratif,

S ——TRADUIRE——> (* S *)

La traduction du commentaire narratif SPC produit l'unité textuelle PASCAL "COMMENTAIRE". Elle se compose du libellé du commentaire narratif SPC délimité par les caractères PASCAL (/)/*.

(5) le commentaire opérationnel.

Son schéma de traduction est le suivant :

Soit S un commentaire opérationnel
soit (i) un numéro du raffinement associé au commentaire opérationnel

$$S(i) \xrightarrow{\text{TRADUIRE}} \text{Pi; (* NIVEAU j S *)}$$

La traduction du commentaire opérationnel et du numéro de raffinement qui lui est toujours associé comprend l'unité textuelle PASCAL "APPEL PROCEDURE INTERNE PLUS COMMENTAIRE".

Cette unité se compose :

- d'un appel de procédure de nom Pi, suivi d'un point virgule. L'appel de procédure se compose du caractère "P" suivi du numéro de raffinement i associé au commentaire opérationnel,
- d'un commentaire PASCAL qui comprend :
  - l'indication du niveau dans lequel le raffinement est développé,
  - le nom du raffinement correspondant au libellé du commentaire opérationnel.

(6) la condition.

Son schéma de traduction est le suivant :

soit COND une condition SPC,

$$\text{COND} \xrightarrow{\text{TRADUIRE}} \text{T(ELSE IF COND)} \left[\begin{array}{l} \text{END} \\ \text{ELSE IF COND} \\ \text{THEN BEGIN} \end{array}\right.$$

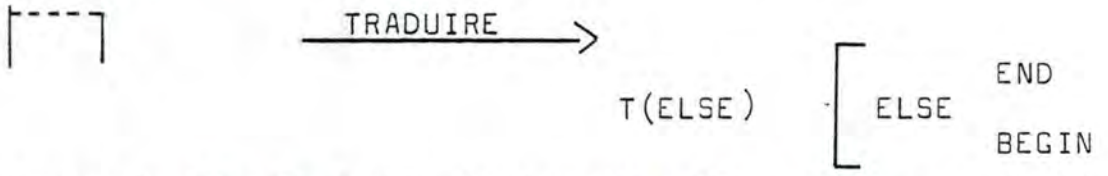
La traduction d'une condition SPC se compose de l'unité textuelle PASCAL "JONCTION ALTERNATIVE SUIVANTE".

Cette unité comprend :

- les délimiteurs PASCAL END/BEGIN ,
- la condition SPC COND,
- les mots réservés PASCAL ELSE/IF .

Cette traduction correspond à T (ELSE IF COND).

(8) l'instruction SINON.
Son schéma de traduction est le suivant :



La traduction de cette instruction SPC se compose de l'unité textuelle PASCAL "JONCTION DERNIERE ALTERNATIVE". Cette unité comprend la suite des mots réservés PASCAL END/ELSE/BEGIN . Cette traduction correspond à T(ELSE COND).

RAPPEL :
La condition et l'instruction SINON ne figurent que dans des structures conditionnelles.

(9) la condition de sortie.
On considère ici uniquement les conditions de sortie figurant dans une structure répétitive à plusieurs conditions de sortie, sans référence à un raffinement, ou dans une structure répétitive avec une seule condition de sortie, sans référence à un raffinement, qui n'est ni en première, ni en dernière position dans la structure répétitive.

Son schéma de traduction est le suivant :
soit COND une condition de sortie,

COND —TRADUIRE→

```
IF COND
THEN LITNU (/i/) : = TRUE
ELSE BEGIN
```

La traduction d'une condition de sortie SPC se compose de l'unité textuelle PASCAL "CONDITION DE SORTIE".

Cette unité comprend :
- des mots réservés PASCAL IF/THEN/ELSE ,
- de la condition de sortie COND,
- de la variable LITNU (/i/)(anagramme de UNTIL) créée pour la traduction
  i représente le numéro du raffinement où figure une condition de sortie
- du délimiteur PASCAL BEGIN .

(10) la condition de sortie avec référence à un raffinement.
On considère ici uniquement les conditions de sortie figurant dans le même type de répétitive que celle définie au point (9) pour la condition de sortie.
Son schéma de traduction est le suivant :

soit COND, une condition de sortie avec référence à un raffinement,

soit (i), le numéro de raffinement associé à cette condition de sortie,

---

$\overline{\text{COND}}_{(i)}$ TRADUIRE ⟶

```
IF COND
THEN BEGIN
  Pi; (* NIVEAU j*)
  LITNU (/k/) : = TRUE;
     END

ELSE BEGIN
```

La traduction de cette instruction se compose de l'unité textuelle PASCAL "CONDITION DE SORTIE AVEC TRAITEMENT D'EXCEPTION".
Cette unité comprend :
- les mots réservés PASCAL IF/THEN/ELSE,
- des délimiteurs PASCAL BEGIN (2 fois)/END/;/,
- de la variable indicée LITNU (/k/) créée pour la traduction où i représente le numéro de raffinement où figure cette instruction SPC,

- d'un appel de procédure de nom Pi suivi d'un point virgule.
  Cet appel est composé :
  - du caractère P suivi du numéro du raffinement référencé,
  - d'un commentaire PASCAL qui comprend l'indication du niveau où le raffinement est développé.

Remarque :

"ELSE BEGIN" ne peut pas figurer dans la traduction :
- soit d'une condition de sortie avec référence à un raffinement point (10),
- soit d'une condition de sortie sans référence à un raffinement point (9),

puisque l'on détectera que ce type d'instruction est la dernière de la structure répétitive.

3.4.3.8 Remarques générales concernant la traduction des structures conditionnelles et répétitives.

Lors de la génération du code PASCAL relatif aux structures conditionnelles et répétitives, certains mots réservés PASCAL seront produits au niveau de la structure, d'autres au niveau des instructions qui la composent.
On aura par exemple des mots réservés PASCAL générés au niveau de la structure dans les unités textuelles PASCAL "EN-TETE CONDITIONNELLE, FIN CONDITIONNELLE, EN-TETE WHILE, FIN WHILE SIMPLE, FIN WHILE TRAITEMENT D'EXCEPTION, EN-TETE REPEAT, FIN REPEAT SIMPLE, FIN REPEAT TRAITEMENT D'EXCEPTION, EN-TETE STANDARD, FIN STANDARD TRAITEMENT D'EXCEPTION.
D'autres seront générés au niveau de la traduction des instructions dans les unités textuelles PASCAL "JONCTION ALTERNATIVE SUIVANTE, JONCTION DERNIERE ALTERNATIVE, CONDITION DE SORTIE, CONDITION DE SORTIE AVEC TRAITEMENT D'EXCEPTION.

Ceci est dû au fait que la notion de structure n'est pas présente comme telle dans la représentation interne du programme SPC.
Les informations relatives à la structure conditionnelle ou répétitive font partie des instructions qui la composent. De plus, par souci de rapidité et d'efficacité du codeur, il nous a été demandé de faire le moins de lecture possible de la représentation interne du programme SPC (une fois chaque structure) et de ne pas créer une représentation interne du programme SPC mieux adaptée à la traduction. Il s'avérait dès lors impossible de produire tous les mots réservés PASCAL au niveau de la structure.
Pour la même raison, les BEGIN et END sont toujours produits dans les structures conditionnelles.
Concernant les structures répétitives, nous avons fait exception au principe précédent. On gagne énormément en clarté du code PASCAL généré, si l'on traduit la structure répétitive par un WHILE ou un REPEAT plutôt que de la traduire toujours sous une forme standard. Nous avons donc décidé de lire préalablement les structures répétitives avant de les traduire. Cette première lecture nous permet de déterminer sous quelle forme la répétitive sera traduite. Elle permet également dans le cas où la structure doit être traduite par un while, d'éliminer les BEGIN et END superflus.

3.4.4 L'analyse d'erreurs en phase de traduction.

Le code PASCAL n'est fourni à l'utilisateur que si certaines conditions sur le programme SPC sont satisfaites. Si de telles conditions ne sont pas satisfaites, il y a production de messages d'erreur à l'écran ou dans un fichier.

On distingue deux types d'erreurs :

3.4.4.1 les erreurs fatales qui entraînent l'arrêt du processus de traduction et la production d'un message d'erreur à l'écran.

(1) C'est un type d'erreur qui rend toute traduction impossible.
C'est : - un cycle de références entre raffinements,
- plus de vingt références imbriquées à partir du niveau qui contient le raff 1,
- plus de vingt références imbriquées à partir du niveau qui contient le raff 0,
- pour des raisons d'implémentation, certains raffinements d'instructions exécutables sont référencés un trop grand nombre de fois.

(2) C'est un type d'erreur qui fournirait une traduction du programme dépourvue de sens (trop d'erreurs de codage dans le code PASCAL généré).
C'est : - le raff 1 qui n'est pas libéré et qui serait alors traduit comme un raffinement d'instructions exécutables,
- le raff 1 et les raffinements, référencés à partir de ce raffinement qui ne contiennent aucune déclaration.
Cette situation est un symptôme de la dispersion des déclarations dans le programme SPC.
Un message d'erreur spécifique est produit pour les situations suivantes :
- le raff 1 n'existe pas,
- le raff 1 ne contient aucune déclaration et aucune référence à un raffinement.

3.4.4.2 les erreurs de codage qui ne provoquent pas l'arrêt du processus de traduction mais entraînent la production d'un message d'erreur dans un fichier auxiliaire. C'est un type d'erreur qui rend ponctuellement le code généré dépourvu de sens.

C'est :
- une déclaration qui figure hors du raff 1 ou d'un raffinement référencé à partir du raff 1,
- une instruction à coder en première colonne, qui n'a pas de sens en PASCAL,
- une structure conditionnelle, répétitive, une instruction d'affectation, un appel de procédure externe dans le raffinement 1 ou un raffinement référencé à partir du raff 1.

La traduction n'est pas arrêtée lors de la détection d'une erreur de codage afin d'en découvrir un maximum sur le programme SPC. L'utilisateur peut ainsi corriger toutes ses erreurs de codage en une fois. Cela limite les retours en arrière fastidieux (édition - traduction du programme SPC), coûteux en temps et en productivité de l'analyste/programmeur.

3.5 ARCHITECTURE DU CODEUR

Le codeur PASCAL créé pour Schémacode constitue essentiellement un outil de traitement de texte.
En entrée, le codeur a le fichier SPC produit par l'éditeur de Schémacode, contenant la représentation interne d'un programme écrit grâce à Schémacode.
Le codeur produit un programme PASCAL dont on connait la structure.

Nous nous sommes inspirées de la démarche de M.A. JACKSON (JAC, 75) pour établir l'architecture globale du codeur :

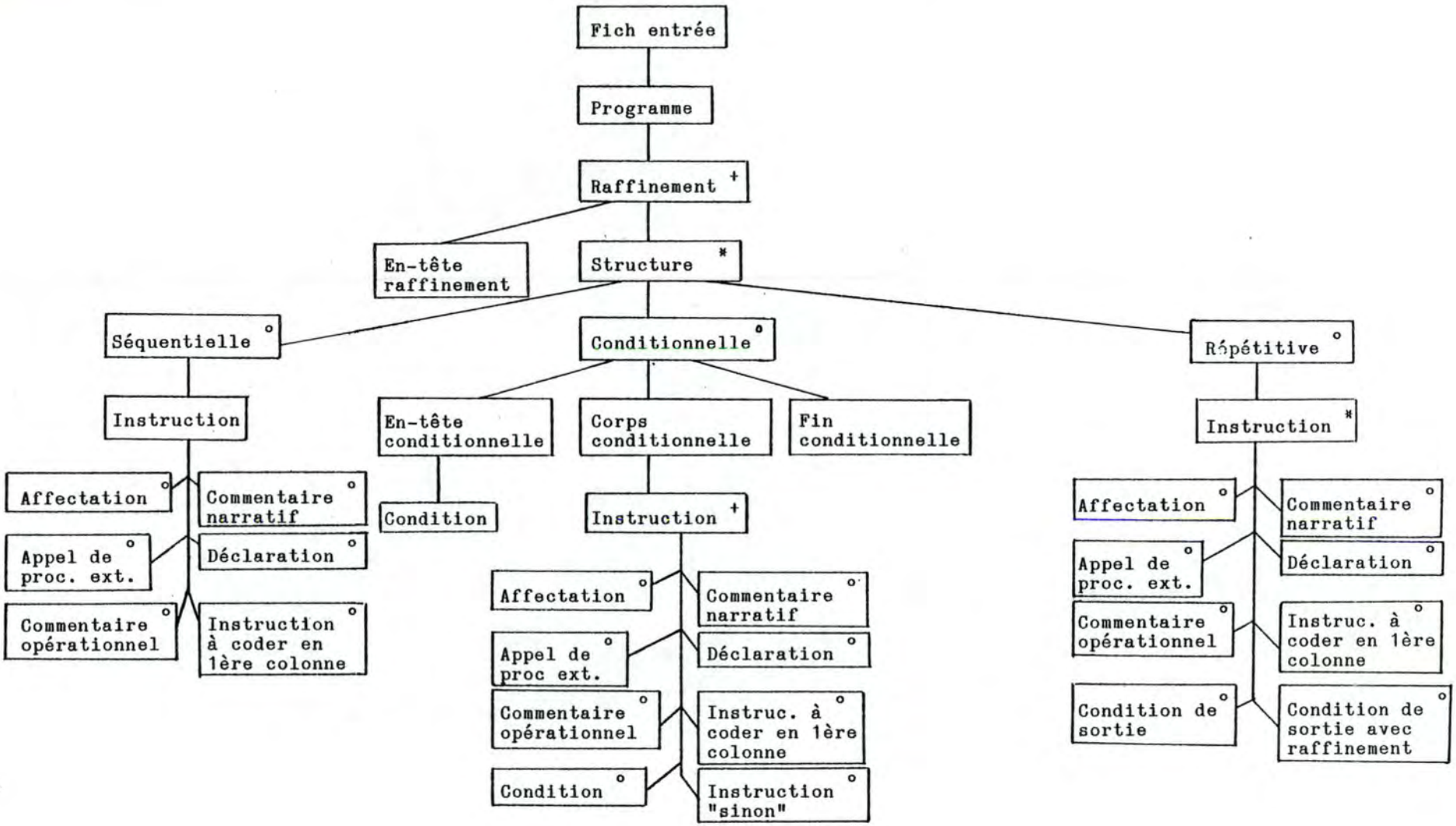
- La structure du programme reflètera les correspondances entre structures de données en entrée et en sortie.
- Chaque décision de design sera localisée dans un composant.

Nous utiliserons également la notation de M.A. JACKSON, à laquelle nous avons ajouté deux symboles :

⊛ signifie zéro ou une occurence,
\+ signifie une ou plusieurs occurences.

3.5.1 Analyse de la structure des données en entrée.

Bien que le fichier SPC soit une suite de records de 72 caractères, après analyse, nous avons déduit la structure logique qui suit.


Fich entrée
Programme
Raffinement +
En-tête raffinement
Structure *
Séquentielle °
Conditionnelle °
Répétitive °
Instruction
En-tête conditionnelle
Corps conditionnelle
Fin conditionnelle
Instruction *
Affectation °
Commentaire narratif °
Appel de proc. ext. °
Déclaration °
Commentaire opérationnel °
Instruction à coder en 1ère colonne °
Condition
Instruction +
Affectation °
Commentaire narratif °
Appel de proc ext. °
Déclaration °
Commentaire opérationnel °
Instruc. à coder en 1ère colonne °
Condition °
Instruction "sinon" °
Affectation °
Commentaire narratif °
Appel de proc. ext. °
Déclaration °
Commentaire opérationnel °
Instruc. à coder en 1ère colonne °
Condition de sortie °
Condition de sortie avec raffinement °

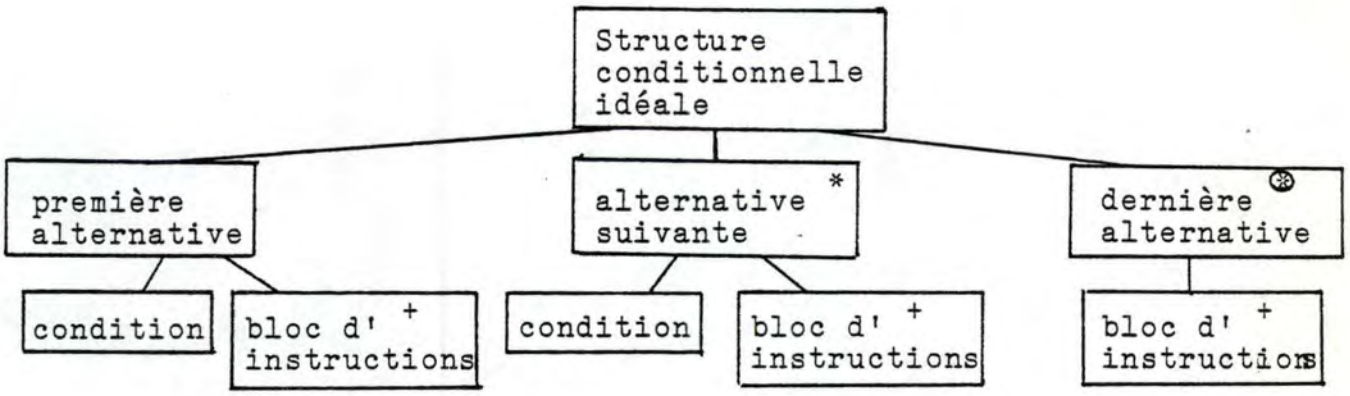
Selon cette structure des données, le corps d'une conditionnelle semble pouvoir être formé de toutes les combinaisons possibles des instructions mentionnées. Mais il n'y a qu'un sous-ensemble de ces combinaisons qui soit syntaxiquement correct. L'éditeur de Schémacode se charge de certaines vérifications et exclut ainsi les combinaisons d' instructions syntaxiquement fausses :

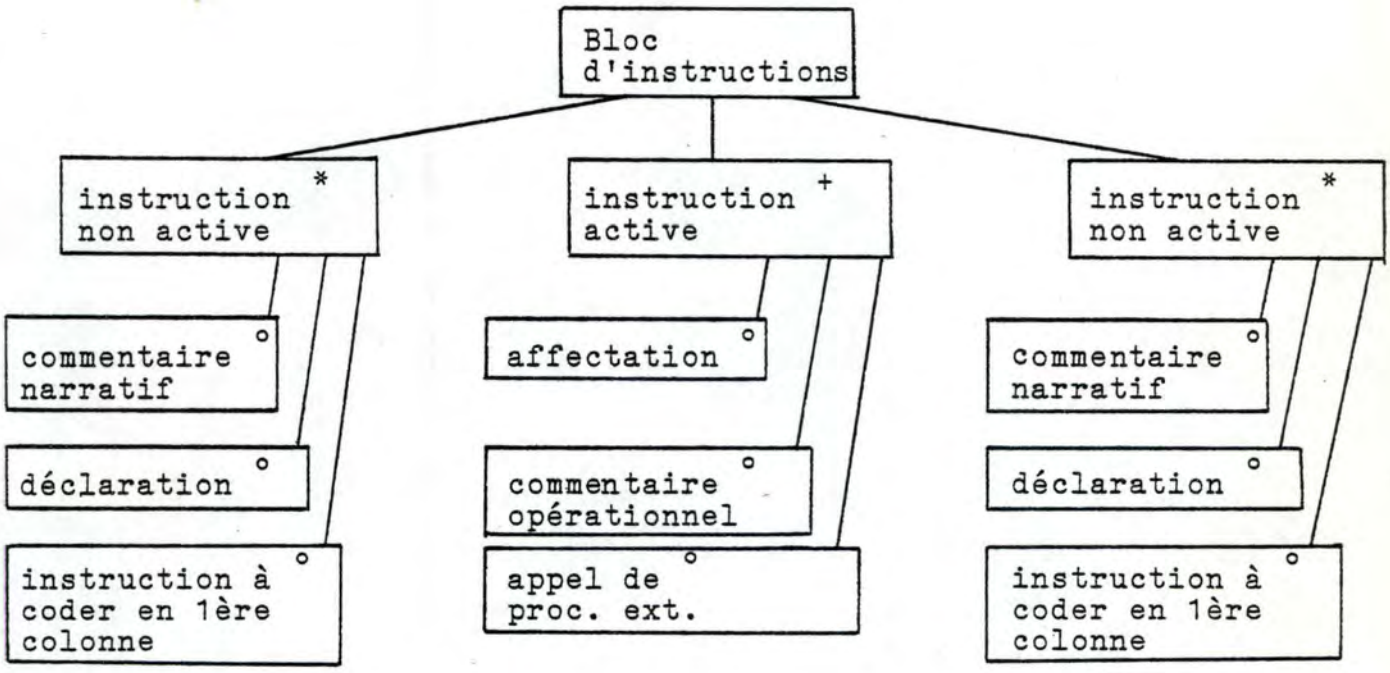
- les alternatives sans action,
- deux instructions "SINON" dans la même structure,
- une alternative "ELSE IF" après l'alternative "SINON".

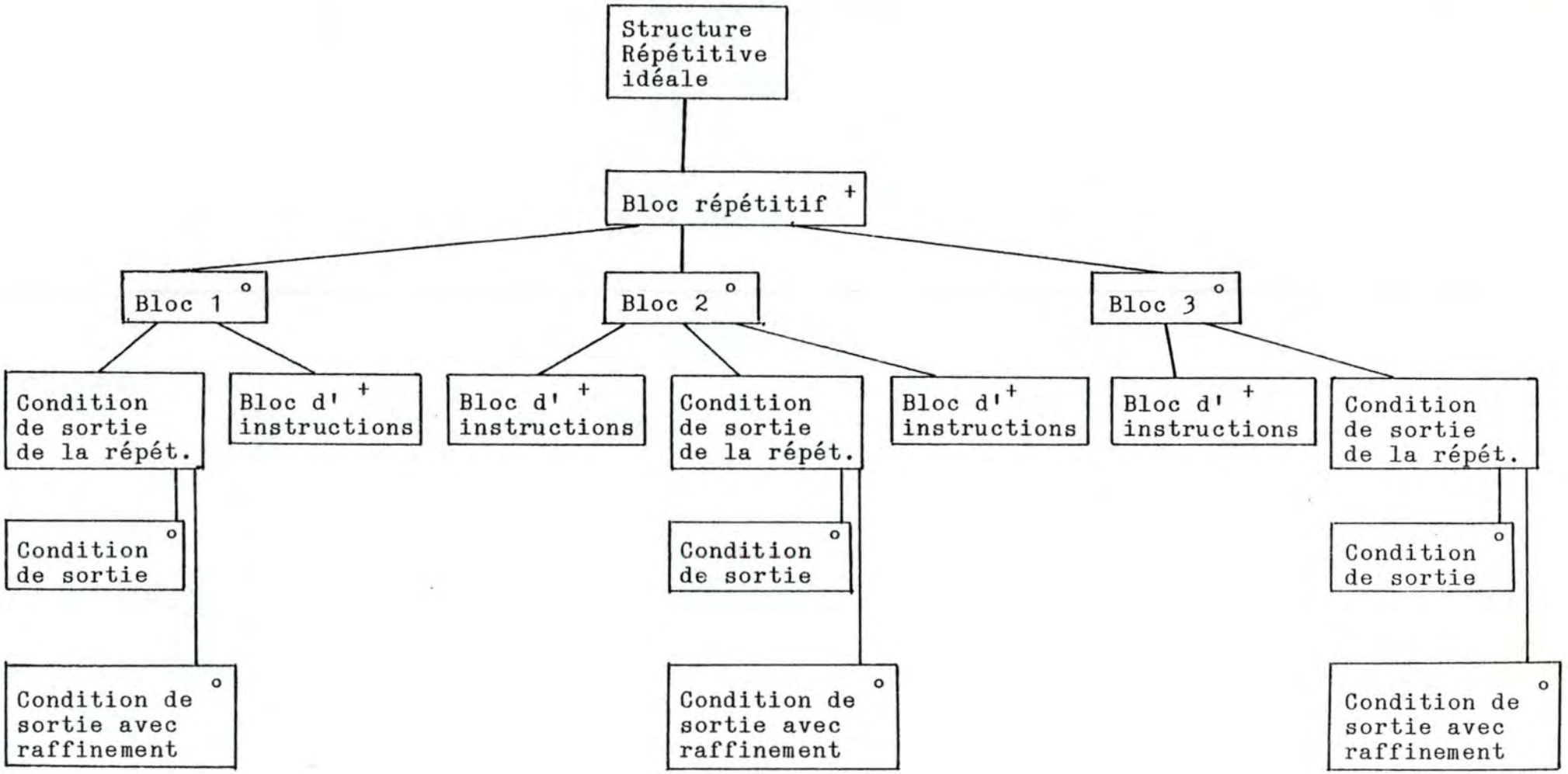
De même, selon cette structure des données, une structure répétitive semble pouvoir être composée de n'importe quelle suite d'instructions parmi celles mentionnées. Comme pour les structures conditionnelles, l'éditeur de Schémacode exclut les combinaisons d'instructions syntaxiquement fausses :

- structure répétitive sans action,
- structure répétitive sans condition de sortie.

Obtenir une structure de données tout à fait exacte reviendrait à définir complètement la syntaxe du pseudo-langage. Cette syntaxe n'est pas exprimée dans la représentation des données (fichier SPC), ce qui, idéalement, devrait être le cas. Nous avons donc analysé de plus près les structures conditionnelles et répétitives de SPC. Nous en proposons les structures syntaxiques et nous les qualifions d'"idéales". Elles sont très lourdes et n'apparaissent pas dans le fichier SPC; elles ne nous serviront donc pas de structures de données en entrée du codeur, mais ellesserviront de base pour constituer l'architecture des traitements.


Structure conditionnelle idéale
première alternative
alternative suivante
dernière alternative
condition
bloc d' instructions
condition
bloc d' instructions
bloc d' instructions


Bloc d'instructions
instruction non active
instruction active
instruction non active
commentaire narratif
déclaration
instruction à coder en 1ère colonne
affectation
commentaire opérationnel
appel de proc. ext.
commentaire narratif
déclaration
instruction à coder en 1ère colonne


Structure Répétitive idéale
Bloc répétitif +
Bloc 1 °
Bloc 2 °
Bloc 3 °
Condition de sortie de la répét.
Bloc d' + instructions
Bloc d' + instructions
Condition de sortie de la répét.
Bloc d' + instructions
Bloc d' + instructions
Condition de sortie de la répét.
Condition de sortie °
Condition de sortie °
Condition de sortie °
Condition de sortie avec raffinement °
Condition de sortie avec raffinement °
Condition de sortie avec raffinement °

Rappelons les notations employées et leur signification :

- ⊛ zéro ou une occurence,
- * zéro, une ou plusieurs occurences,
- + une ou plusieurs occurences,
- o ou exclusif.

La représentation idéale de la structure conditionnelle reflète bien sa syntaxe :

- Elle comporte au moins une alternative "IF...THEN", zéro, un ou plusieurs alternatives "ELSE IF", et si elle contient une alternative "ELSE", elle en contient une seule.

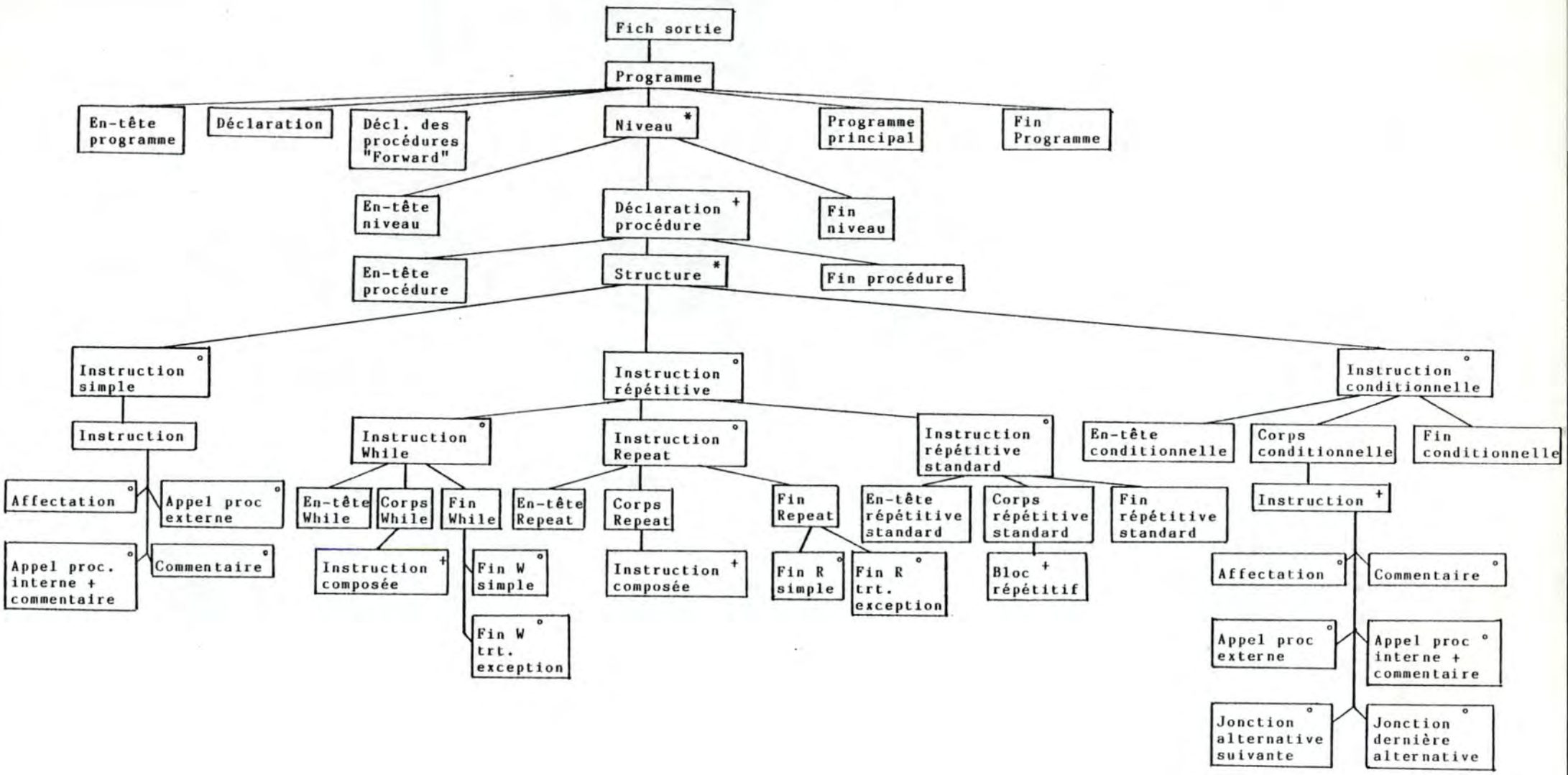
- Chaque alternative contient au moins un bloc d'instructions, qui lui-même est formé d'au moins une action.

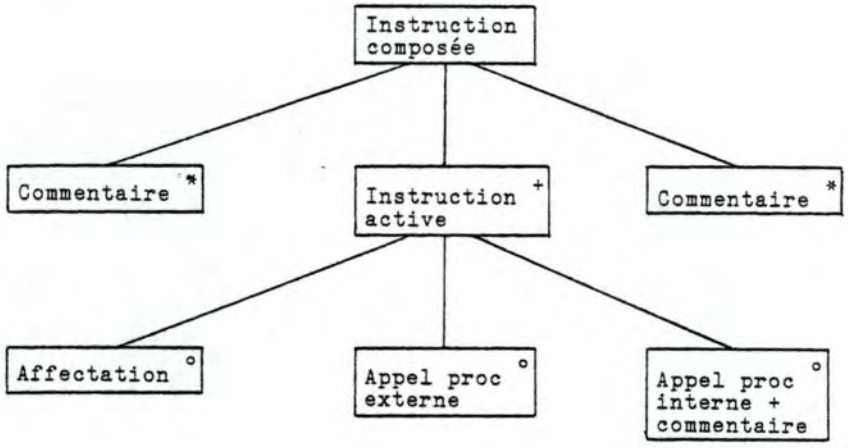
La représentation idéale de la structure répétitive reflète également sa syntaxe : la structure répétitive est formée d'au moins un bloc répétitif, c'est-à-dire au moins une condition de sortie et un bloc d'instructions, donc au moins une action.

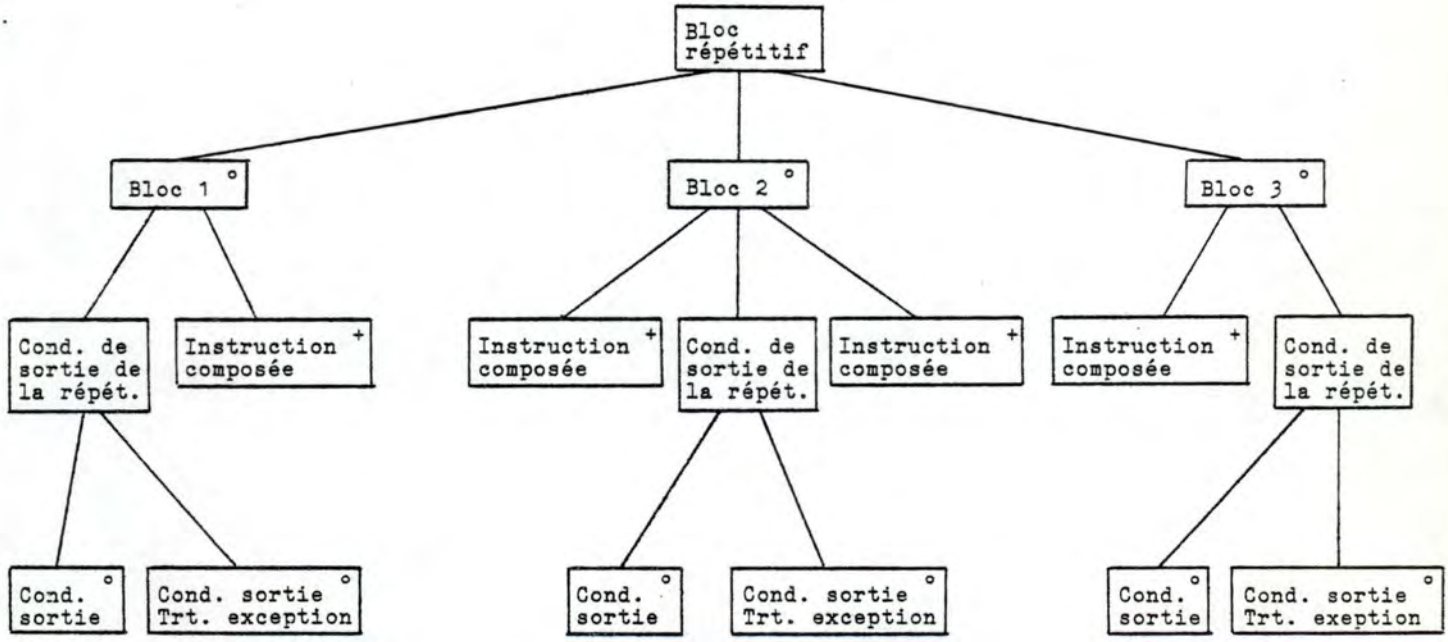
3.5.2 Analyse de la structure des données en sortie.

La structure du code produit est décrite ci-après.

Remarquons que parmi les instructions de la structure des données en sortie, les déclaration et instruction à coder en première colonne ont disparu des niveaux inférieurs de la structure :


Fich sortie
Programme
En-tête programme
Déclaration
Décl. des procédures "Forward"
Niveau *
Programme principal
Fin Programme
En-tête niveau
Déclaration procédure +
Fin niveau
En-tête procédure
Structure *
Fin procédure
Instruction simple °
Instruction répétitive °
Instruction conditionnelle °
Instruction
Instruction While °
Instruction Repeat °
Instruction répétitive standard °
En-tête conditionnelle
Corps conditionnelle
Fin conditionnelle
Affectation °
Appel proc externe °
En-tête While
Corps While
Fin While
En-tête Repeat
Corps Repeat
Fin Repeat
En-tête répétitive standard
Corps répétitive standard
Fin répétitive standard
Instruction +
Appel proc. interne + commentaire °
Commentaire °
Instruction composée +
Fin W simple °
Instruction composée +
Fin R simple °
Fin R trt. exception °
Bloc + répétitif
Affectation °
Commentaire °
Fin W trt. exception °
Appel proc externe °
Appel proc interne + commentaire °
Jonction alternative suivante °
Jonction dernière alternative °


Instruction composée
Commentaire *
Instruction active +
Commentaire *
Affectation °
Appel proc externe °
Appel proc interne + commentaire °


Bloc répétitif
Bloc 1 °
Bloc 2 °
Bloc 3 °
Cond. de sortie de la répét.
Instruction composée +
Instruction composée +
Cond. de sortie de la répét.
Instruction composée +
Instruction composée +
Cond. de sortie de la répét.
Cond. sortie °
Cond. sortie Trt. exception °
Cond. sortie °
Cond. sortie Trt. exception °
Cond. sortie °
Cond. sortie Trt. exeption °

- Il ne peut y avoir de déclaration locale à une procédure du programme; les déclarations sont globales, en tête du programme.
- Les instructions à coder en première colonne n'ont pas de sens en PASCAL; elles ne sont donc pas traduites par le codeur.

Il est évident que les restrictions et obligations imposées par l'éditeur de Schémacode pour les données en entrée se retrouvent dans la structure des données en sortie, même si elles ne sont pas exprimées :

- toute alternative d'une instruction conditionnelle doit contenir au moins une action,

- une seule "jonction dernière alternative" dans l'instruction conditionnelle.

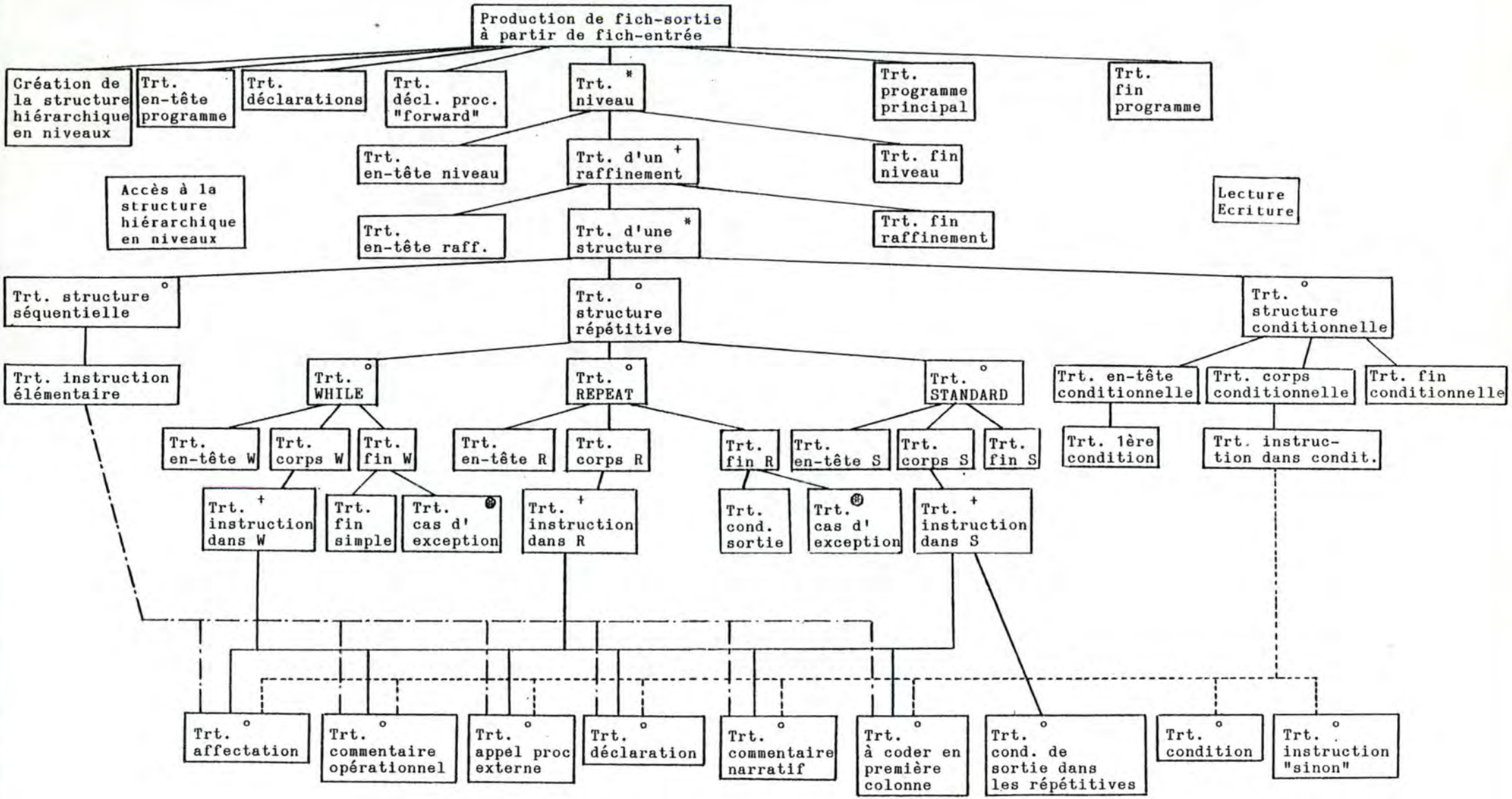
3.5.3 Dérivation de la structure globale des traitements.

Pour obtenir la structure du code PASCAL décrite précédemment, nous avons dû appliquer les traitements suivants aux données d'entrée du codeur.

L'architecture représente la structure de MODULES de traitement; chaque noeud dans l'arbre représente un MODULE. Cela implique que

- un module pourra être physiquement éclaté en plusieurs routines,

- plusieurs modules pourront être rassemblés en une seule routine.


Production de fich-sortie à partir de fich-entrée
Création de la structure hiérarchique en niveaux
Trt. en-tête programme
Trt. déclarations
Trt. décl. proc. "forward"
Trt. * niveau
Trt. programme principal
Trt. fin programme
Trt. en-tête niveau
Trt. d'un + raffinement
Trt. fin niveau
Accès à la structure hiérarchique en niveaux
Lecture Ecriture
Trt. en-tête raff.
Trt. d'une * structure
Trt. fin raffinement
Trt. structure ° séquentielle
Trt. ° structure répétitive
Trt. ° structure conditionnelle
Trt. instruction élémentaire
Trt. ° WHILE
Trt. ° REPEAT
Trt. ° STANDARD
Trt. en-tête conditionnelle
Trt. corps conditionnelle
Trt. fin conditionnelle
Trt. en-tête W
Trt. corps W
Trt. fin W
Trt. en-tête R
Trt. corps R
Trt. fin R
Trt. en-tête S
Trt. corps S
Trt. fin S
Trt. 1ère condition
Trt. instruction dans condit.
Trt. + instruction dans W
Trt. fin simple
Trt. cas d' exception
Trt. + instruction dans R
Trt. cond. sortie
Trt. cas d' exception
Trt. + instruction dans S
Trt. ° affectation
Trt. ° commentaire opérationnel
Trt. ° appel proc externe
Trt. ° déclaration
Trt. ° commentaire narratif
Trt. ° à coder en première colonne
Trt. ° cond. de sortie dans les répétitives
Trt. ° condition
Trt. ° instruction "sinon"

3.5.4 Justification de l'architecture des traitements.

Montrons comment l'architecture reflète les correspondances entre structures de données "idéales" en entrée et en sortie du codeur.

(1) Au fichier SPC contenant un programme développé en pseudo-code schématique correspond un fichier contenant un programme PASCAL.

(2) L'introduction de la notion de "niveau" dans le code généré se reflète au second niveau de l'architecture du codeur. Le module "Création de la structure hiérarchique en niveaux" crée la structure en niveaux du code généré, à partir des raffinements du fichier SPC; les raffinements sont donc groupés en niveaux, et leur traitement se fait par niveau : module "Traitement niveau".

(3) A partir d'un raffinement du fichier SPC, le codeur produit la déclaration d'une procédure PASCAL : module "Traitement raffinement".

(4) Les niveaux 4 et 5 de l'architecture du codeur montre qu'à partir d'une structure SPC, le codeur produit une instruction PASCAL.

Voyons comment chaque structure SPC est traitée :

* La structure séquentielle : elle devient une instruction élémentaire.

* La structure conditionnelle :
  L'en-tête de la structure conditionnelle est obligatoirement une condition. Une condition dans le corps de la

structure est traitée différemment de la première condition : elle dénote une nouvelle alternative dans la structure conditionnelle et est traitée comme telle.
L'instruction "SINON" de SPC dénote la dernière alternative d'une structure conditionnelle et est donc traduite en "jonction de la dernière alternative".

* La structure répétitive : Selon le type de la structure et le nombre d'instructions qui la composent, la structure répétitive est traduite en une instruction REPEAT, en une instruction WHILE ou en une instruction répétitive dite "standard" (cf section 3.4.1). Dans les trois cas, on produit en-tête, corps et fin de l'instruction.

(5) Au niveau le plus bas des structures de données en entrée et en sortie, on retrouve les instructions du pseudo-langage et les instructions élémentaires du langage cible. Bien qu'elles appartiennent à des contextes différents, un seul module traîte chaque type d'instruction.

3.5.5 Décomposition physique.

Une fois la modularisation effectuée (cfr sous-section 3.5.2), il convient de décider si l'on va

(1) faire correspondre à un module une routine,

(2) regrouper certains modules en une routine;
- quand le traitement associé à un de ces modules ne nécessite pas d'en faire une routine séparée,
- quand les traitements associés à certains modules sont similaires,

(3) scinder un module en plusieurs routines;
- quand on peut identifier à l'intérieur de ce module plusieurs traitements distincts.

Suivant ces critères, nous avons adopté la découpe suivante (chaque routine sera spécifiée de façon plus détaillée par la suite) :

- Le module "production de fich-sortie à partir de fich-entrée" sera réalisé par la routine CODPAS.

  entrées : ° fichier SPC contenant la représentation interne d'un programme SPC,
  ° l'identification du support souhaité pour le code généré,

sorties : soit ° le code PASCAL sur le support souhaité;
si le programme SPC ne contient ni erreurs fatales, ni erreurs de codage,
soit ° un message, à l'écran, parmi différents types de messages d'erreur;
s' il existe des erreurs fatales relatives à la création de la structure hiérarchique en niveaux et au contenu du raffinement de déclarations. Le message produit correspond à la première erreur fatale rencontrée,
soit ° différents types de messages d'erreur à l'écran;
s' il existe des erreurs de codage relatives au contenu des raffinements de déclarations et d'instructions exécutables,
° impression du nombre d'erreurs de codage. Ce nombre correspond au nombre de messages d'erreur produits à l'écran.

appelle : les routines CRENIV, TRTRAF, IMPECR.

appelée par : la routine de Schémacode UTILTR.

- Le module "création de la structure hiérarchique en niveaux" sera réalisé par la routine CRENIV.

entrées : ° fichier SPC contenant la représentation interne du programme SPC.

sorties : soit ° représentation de la structure hiérarchique en niveaux des raffinements du programme SPC;
s' il n'y a pas d'erreur fatale.

La structure hiérarchique en niveaux est l'arbre des raffinements tel qu'il a été défini au chapitre 3, sous-section 3.2, point (3) et auquel on a associé la structure en niveaux suivante :

- le niveau 0 ne reprend que le raff 0,
- le niveau -1 ne reprend que le raff 1,
- le niveau i reprend les raffinements référencés par les différents raffinements des niveaux 0 à (i - 1)

soit ° l'identification d'une éventuelle erreur fatale relative à la construction de la représentation de la structure hiérarchique en niveaux et qui aurait empêché la création de celle-ci.

appelle : les routines LECT, LIT, VARTIC (routine de Schémacode).

appelé par : la routine CODPAS.

- Le module "accès à la structure hiérarchique en niveaux" se scinde en quatre routines de fonctions distinctes :
(1) la routine PREM

entrées : ° numéro de niveau,
° représentation de la structure hiérarchique en niveaux,

sorties : ° adresse dans la représentation interne du premier raffinement de ce niveau s' il existe,
° l'identification d'une erreur si celui-ci n'existe pas.

appelé par : ° la routine TRTRAF.

(2) la routine SUIV

entrées : ° numéro de niveau,
° numéro d'un raffinement appartenant à ce niveau,
° représentation de la structure hiérarchique en niveaux.

sorties : ° adresse dans la représentation interne du raffinement qui suit ce raffinement dans le niveau, s' il existe,
° erreur soit
si le raffinement suivant n'existe pas, soit
si le raffinement suivant dans un niveau est demandé, alors que le premier raffinement dans ce niveau ne l'a pas été.

appelé par : la routine TRTRAF.

(3) la routine NIVRAF

entrées : ° numéro de raffinement,
° représentation de la structure hiérarchique en niveaux.

sorties : ° numéro du niveau auquel ce raffinement apprtient.

appelé par : les routines COMOP, PTSOR.

(4) la routine TABSI

entrées : ° la représentation de la structure hiérarchique en niveaux.

sortie : ° le plus grand numéro de raffinement parmi les raffinements de la structure hiérachique en niveaux.

appelé par : la routine TRTRAF.

Chacune de ces quatre routines ne fait appel à aucune autre.

- Nous regroupons les modules "trt en-tête" programme", "trt déclarations", "trt déclarations procédure forward", "trt niveau", "trt programme principal", "trt fin programme", "trt en-tête niveau", "trt d'un raffinement", "trt en-tête raff", "trt fin raff"; ils sont réalisés par la routine TRTRAF.

  entrées : ° fichier SPC contenant la représentation interne d'un programme SPC.

  sorties : soit ° le texte d'un programme PASCAL structuré en niveaux, avec procédures, (cfr exemple annexe 1, point (2.2)), correspondant à la traduction de ce programme SPC,
  ° et d'éventuels messages d'erreur, dans un fichier; s' il existe des erreurs de codage relatives aux raffinements de déclarations et d'instructions exécutables de ce programme SPC,
  ° le nombre d'erreurs de codage détectées lors de la traduction du programme SPC,
  soit ° un message, à l'écran, parmi différents types de messages d'erreurs; s' il existe des erreurs fatales relatives au contenu des raffinements de déclarations.

appelle : les routines IDSTR, IMPLI, IMPECR, LECT, IMPERR, PREM, SUIV, TABSI, LIT et ALPHA (deux routines de Schémacode).

appelé par : la routine CODPAS.

- Nous regroupons les modules "trt d'une structure", , "trt structure séquentielle", "trt instruction", "trt structure conditionnelle", "trt en-tête conditionnelle", "trt première condition", "trt fin conditionnelle", trt corps conditionnelle", "trt instruction", "trt structure répétitive"; ils seront réalisés par la routine IDSTR.

entrées : ° fichier SPC contenant la représentation interne d'un programme SPC, dont l'élément courant (cfr chapitre 3, sous-section 2.4.2 "structure logique du fichier SPC", est le dernier élément de la première structure du raffinement à traduire.

sorties : ° corps d'une procédure PASCAL (cfr chapitre 3, sous-section 4.3) correspondant à la traduction des structures (sauf la première) de ce raffinement,

° et d'éventuels messages d'erreur dans un fichier; s' il existe des erreurs de codage relatives aux raffinements d'instructions exécutables.

appelle : IMPLI, IMPECR, LECT, REPETR, REPETS, REPETW, AFPRO, COMOP, COMNA, EXPBO, SINON FAUCO, SURVOL.

appelé par : TRTRAF.

- Les modules "trt while", "trt en-tête while", "trt corps while", "trt fin while", "trt fin simple", "trt cas d'exception", "trt instruction" sont réalisés par la routine REPETW.

entrées : ° fichier SPC dont l'élément courant est le premier élément de la structure répétitive à traduire,
° nombre d'instructions dans cette structure (y compris les conditions de sortie avec ou sans référence à un raffinement).

sorties : ° instruction répétitive PASCAL While, (cfr chapitre 3, sous-section 4.3) correspondant à la traduction de cette structure répétitive,
° l'élément courant correspondant au premier élément de la structure suivante,si elle existe,
° d'éventuels messages d'erreur dans un fichier; s' il existe des erreurs de codage relatives aux raffinements d'instructions exécutables.

appelle : les routines IMPLI, AFPRO, COMOP, COMNA, FAUTCO, LECT, LMOVE (routine de Schémacode).

appelé par : la routine IDSTR.

- Les modules "trt repeat", "trt en-tête repeat", "trt corps repeat", "trt fin repeat", "trt fin simple", 'trt cas d'exception", "trt instruction" sont réalisés par la routine REPETR.

entrées : ° fichier SPC dont l'élément courant est le premier élément de la structure répétitive à traduire.

sorties : ° instruction répétitive PASCAL Repeat Until, (cfr chapitre 3, sous-section 4.3) correspondant à la traduction de cette structure répétitive,
° l'élément courant correspondant au premier élément de la structure suivante, si elle existe,

° mêmes types de messages d'erreur que REPETW.

appelle : les routines IMPLI, LECT, AFPRO, COMOP, COMNA, FAUTCO.

appelé par : la routine IDSTR.

- Les modules "trt standard","trt en-tête standard", "trt corps standard", "trt fin standard", "trt instruction" seront réalisés par la routine REPETS.

entrées : cfr celles de la routine REPETR ci-dessus.

sorties : ° instruction répétitive PASCAL traduite selon une forme standard, (cfr chapitre 3 - section 4.3) correspondant à la traduction de cette structure répétitive,
° mêmes types de messages d'erreur que REPETW,
° l'élément courant correspondant au premier élément de la structure suivante, si elle existe.

appelle : les routines IMPLI, AFPRO, COMOP, COMNA, PTSOR, FAUTCO, ALPHA (routine de Schémacode).

appelé par : la routine IDSTR.

- Le module "trt cond. de sortie dans les répétitives" sera réalisé par la routine PTSOR.

entrées : ° fichier SPC dont l'élément courant est le premier élément de la condition de sortie avec ou sans référence à un raffinement.

sorties : ° l'unité textuelle PASCAL "condition de sortie" (cfr chapitre 3, sous-section 4.3), correspondant à la traduction d'une condition de sortie sans

référence à un raffinement,

° ou l'unité textuelle PASCAL "condition de sortie avec traitement d'exception" (cfr chapitre 3, sous-section 4.3), correspondant à la traduction d'une condition de sortie avec référence à un raffinement,

° l'élément courant est le premier élément de l'instruction SPC suivante, si elle existe.

appelle : les routines IMPLI, LECT, NIVRAF, ALPHA (routine de Schémacode).

appelé par : la routine REPETS.

- Les modules "trt affectation" et "trt appel procédure externe", dont les traitements sont similaires, sont réalisés par la routine AFPRO.

entrées : ° fichier SPC dont l'élément courant est le premier élément de l'instruction SPC d'affectation, ou appel de procédure externe,

° nombre de blancs à insérer avant le début du texte PASCAL.

sorties : ° l'unité textuelle PASCAL correspondant à la traduction de l'instruction d'affectation ou de l'appel de procédure externe (cfr chapitre 3, section 4.3),

° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

ts

appelle : les routines IMPLI et LECT.

appelé par : les routines IDSTR, REPETR, REPETS, REPETW.

- Le module "trt commentaire narratif" est réalisé par la routine COMNA.

  entrées : ° identiques à celles de AFPRO, mais pour le commentaire narratif SPC.

  sorties : ° l'unité textuelle PASCAL "commentaire" correspondant à la traduction de cette instruction SPC,
  ° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

  appelle : les routines IMPLI et LECT.

  appelé par : par les routines IDSTR, REPETR, REPETS, REPETW.

- Les modules "trt déclaration" et "trt instruction à coder en première colonne", dont les traitements sont similaires, sont réalisés par la routine FAUTCO.

  entrées : ° fichier SPC dont l'élément courant est le premier élément d'une de ces instructions SPC,
  ° numéro du message d'erreur à imprimer,
  ° nombre d'erreurs de codage détectées avant l'élément courant.

  sorties : ° message d'erreur associé à ce numéro, dans un fichier,
  ° deux types de message d'erreur peuvent être produits en sorties. Ils correspondent aux deux types d'erreurs de codage, relatives aux raffinements d'instructions exécutables,
  ° élément courant est le premier élément de l'instruction qui suit cette instruction SPC, si elle existe,

° nombre d'erreurs de codage détectées avant l'élément courant.

appelle : les routines IMPERR et LECT.

appelé par : les routines IDSTR, REPETR, REPETS, REPETW.

- Le module "trt condition" est réalisé par la routine EXPBO.

entrées : ° identiques à celles de AFPRO, mais pour la condition SPC.

sorties : ° l'unité textuelle PASCAL "jonction alternative suivante" correspondant à la traduction de cette instruction SPC (cfr chapitre 3, sous-section 3.4),
° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

appelle : ° les routines IMPLI et LECT.

appelé par : la routine IDSTR.

- Le module "trt sinon" est réalisé par la routine SINON.

entrées : ° le fichier SPC dont l'élément courant est le premier élément de cette instruction.

sorties : ° l'unité textuelle PASCAL "jonction dernière alternative" (cfr chapitre 3, sous-section 3.4) correspondant à la traduction de cette instruction SPC,
° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

appelle : les routines IMPLI et LECT.

appelé par : la routine IDSTR.

- Le module "trt commentaire opérationnel est réalisé par la routine COMOP.

entrées : ° identiques à celle de AFPRO, mais pour le commentaire opérationnel SPC.

sorties : ° l'unité textuelle Pascal "appel procédure interne plus commentaire" correspondant à la traduction de cette instruction (cfr chap. 3 sous-section 4.3),
° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

appelle : les routines IMPLI, LECT, NIVRAF et ALPHA (routine de Schémacode).

appelé par : : les routines IDSTR, REPETS, REPETR, REPETW.

- Le module "lecture/écriture" est réalisé par les routines suivantes :

(1) la routine de lecture LECT

entrées : ° fichier PSC contenant la représentation interne d'un programme SPC et un élément courant dans un raffinement,
° nombre d'éléments jusqu'à et y compris l'élément courant dans ce raffinement.
Remarque :
ce nombre d'éléments ne reprend que les éléments dont la partie TEXTE est non vide (cfr chapitre 3, sous-section 2.4.2).

sorties : ° élément courant devient l'élément qui suit celui-ci dans le fichier SPC, s' il existe.
° éventuellement détection de fin de fichier, fin d'un raffinement, fin d'une structure, fin d'une instruction,
° nombre d'éléments jusqu'à et y compris l'élément courant dans ce raffinement.

appelle : la routine de Schémacode LIT.

appelé par : les routines CRENIV, TRTRAF, IDSTR, REPETR, REPETW, AFPRO, COMOP, COMNA, EXPBO, SINON, PTSOR, FAUTCO, SURVOL.

(2) la routine d'impression IMPLI.

entrées : ° texte : chaîne de caractères,
° nombre de caractères composant le texte,
° nombre de fois qu'il faut l'imprimer,
° nombre de blancs à insérer avant de l'imprimer,
° paramètre déterminant s'il faut éliminer les blancs à la fin du texte,
° paramètre déterminant s' il faut encore ajouter des caractères à la fin de ce texte.

sorties : ° l'impression sur un fichier auxiliaire de ce texte en fonction des paramètres d'entrées.

appelle : les routines de Schémacode EQUAL et LMOVE.

appelé par : les routines TRTRAF, IDSTR, REPETR, REPETS, REPETW, AFPRO, COMOP, COMNA, EXPBO, SINON, PTSOR.

(3) la routine d'impression IMPERR.

entrées : ° numéro d'un message d'erreur,
° fichier SPC dont l'élément courant est celui où figure l'erreur de codage,
° nombre d'éléments jusqu'à et y compris l'élément courant dans le raffinement.

sorties : ° impression sur fichier auxiliaire d'un message d'erreur associé à ce numéro,
° impression du numéro de raffinement et du numéro de l'élément dans ce raffinement, où figure l'erreur de codage,
° impression du texte de cet élément.

appelé par : les routines TRTRAF, IDSTR, FAUTCO.

(4) la routine d'impression IMPECR.

entrées : ° numéro d'un message d'erreur associé à une erreur fatale.

sorties : ° affichage à l'écran du message d'erreur associé à ce numéro,
° affichage de "arrêt de la traduction".

appelle : les routines de Schémacode FPUTC et ADRE.

appelé par : les routines CODPAS, TRTRAF.

- Comme mentionné au chapitre 3 à la sous-section 3.4, nous avons décidé de traduire une structure répétitive soit par un while, soit par un repeat, soit selon une forme standard. Il convient, à cete effet, d'analyser chaque structure répétitive avant de la traduire, ceci est réalisé par la routine SURVOL.

entrées : ° fichier SPC dont l'élément courant est le premier élément de la structure répétitive.

sorties : ° la façon dont la structure répétitive doit être traduite soit par un while,
soit par un repeat,
soit selon un forme standard,
° dans le cas où la structure répétitive doit être traduite par un while, le nombre d'instructions de la structure (y compris les conditions de sortie),
° l'élément courant du fichier SPC reste inchangé.

appelle : la routine LECT.

appelé par : la routine IDSTR.

3.6. DEVELOPPEMENT DES ROUTINES

Nous allons développer chacune des routines du codeur PASCAL en deux étapes. Dans un premier temps, nous spécifierons la routine. En nous basant sur la description des entrées-sorties de cette routine (cfr décomposition physique, section 3.5.5.), nous décrirons ses entrées et ses sorties en termes de paramètres, de variables locales à la routine, de variables globales (figurant dans des commons).

Dans un deuxième temps, nous donnerons le procédé de construction associé à cette routine. A partir des spécifications, nous ferons apparaître :

- des sous-objectifs parmi lesquels l'on distinguera, les sous-objectifs résolus dans cette routine des sous-objectifs résolus par une autre routine;
- des choix de représentation de données.

La terminologie usitée dans le développement des routines de traduction reprend les concepts définis dans la sous-section 3.4.3., relations " TRADUIRE " et " DECLAR" : règles pour le codage automatique. L'unité textuelle PASCAL sera notée de façon abrégée u.t.P.

Remarques préliminaires

- Toutes les routines qui vont être développées, (sauf les toutines d'impression IMPLI et IMPECR et les routines d'accès à la structure hiérarchique PREM, SUIV, TABSI, NIVRAF) utilisent en entrée, le fichier SPC. Ce fichier contient la représentation interne d'un programme SPC. Nous ne noterons donc pas cette entrée, tout en sachant qu'elle figure dans chacune des entrées spécifiées.
- L'élément courant dans le fichier SPC est représenté par les variables raff, struc, type, texte, pntr. Ces variables font partie d'un common. Désormais, nous parlerons de " instr " au lieu de type. Ce nom nous apparaît plus significatif pour parler du type d'une instruction.

- Les routines de Schémacode LIT, VARTIC, LOCAL,ADRE, FPUTC et ALPHA ne seront pas développées ici. Elles ont été spécifiées à la sous-section 3.2.4.3.
- Les algorithmes du codeur PASCAL sont repris en annexe 2.

3.6.1. Développement de la routine CODPAS

3.6.1.1. Spécification

Appel à la routine : CODPAS (modcod, numfic)

Entrées : ° modcod, paramètre de type INTEGER, identifie le support sur lequel le code PASCAL sera produit.

Valeurs possibles :

- 1, si l'utilisateur souhaite l'affichage du code PASCAL;
- 2, si l'utilisateur souhaite l'impression du code PASCAL;
- 0, si l'utilisateur ne souhaite ni l'affichage, ni l'impression du code PASCAL.

Sorties : soit ° un programme PASCAL, structuré en niveaux, correspondant à la traduction d'un programme SPC, si ce programme SPC ne contient ni erreurs fatales, ni erreurs de codage. Le code PASCAL est affiché ou imprimé selon la valeur de modcod;

soit ° une liste de messages d'erreur liés à des erreurs de codage. Ces messages sont affichés. Ils seront explicités par la suite;

° errcod, variable de type INTEGER; elle représente le nombre d'erreurs de codage détectées lors de la traduction du programme SPC. Cette variable fait partie d'un common.
Elle sera affichée avec la liste de messages d'erreur;

soit ° un message, à l'écran, associé à une erreur fatale relative à la création de la structure hiérarchique en niveaux. Ce message est un des quatres messages suivants :

(1) Le raff 1 n'est pas libéré.
Arrêt de la traduction.

(2) Détection d'un cycle de références entre raffinements.
Arrêt de la traduction.

(3) Plus de vingt références imbriquées à partir du niveau qui contient le raff 0.
Arrêt de la traduction.

(4) Certains raffinements d'instructions exécutables sont référencés un trop grand nombre de fois; pour des raisons d'implémentation du codeur, cela constitue une erreur fatale.
Arrêt de la traduction.

Les autres messages seront explicités par la suite;

° numfic, paramètre de type INTEGER, représente le numéro du fichier contenant le résultat de la traduction.

Valeurs possibles :

- 14, si le résultat de la traduction est le code PASCAL;
- 7, si le résultat de la traduction est une liste de messages associés aux erreurs de codage;
- 0, si le résultat de la traduction est un message, à l'écran, correspondant à la première erreur fatale rencontrée.

Ce numéro de fichier est imposé par l'environnement Schémacode et est utilisé par celui-ci.

3.6.1.2. Procédé de construction

Afin d'obtenir le résultat correspondant à la traduction d'un programme SPC, les traitements à envisager sont les suivants :

- Initialiser les supports utilisés pour produire le résultat de la traduction (sous-objectif 1).
- Invoquer la rotine CRENIV, afin de créer la structure hiérarchique en niveaux.
- Invoquer la routine IMPECR, afin d'afficher un message correspondant à l'éventuelle erreur rencontrée lors de la création de la structure hiérarchique en niveau.
- Invoquer la routine TRTRAF, afin de traduire le programme SPC en un programme PASCAL.
- Si aucune erreur fatale n'a été détectée ni par TRTRAF, ni par CRENIV, il faut :
  - ° déterminer la valeur de numfic, (sous-objectif 2).
  - ° Déterminer ce qui sera fourni à l'utilisateur (sous-objectif 3).

  Sinon il faut :
  - ° déterminer la valeur de numfic et effectuer une mise à jour des supports (sous-objectif 4).

Sous-objectif 1 : initialiser les supports utilisés.

Il faut envisager les traitements suivants :
- Se positionner au début des deux fichiers de sortie (fichier n° 14 et fichier n° 7).
- Invoquer une routine " système ", afin d'initialiser l'écran en vue d'un éventuel affichage sur celui-ci.

Sous-objectif 2 : déterminer la valeur de numfic

Si errcod est strictement positive, il faut :

- Donner à numfic la valeur 7.
- Détruire le contenu du fichier numéro 14.
- Ecrire dans le fichier numéro 7, la valeur de errcod, suivie des mots :
  " erreurs détectées ".

Sinon
- Donner à numfic la valeur 14.

Sous-objectif 3 : déterminer ce qui sera fourni à l'utilisateur.

Si modcod = 1 et si errcod est strictement positive, il faut :
- Afficher le contenu du fichier de numéro numfic.

Sinon si modcod = 2, il faut :
- Imprimer le fichier numéro 14.

Sous-objectif 4 : déterminer la valeur de numfic et mise à jour des supports.

A cet effet, il faut :
- Donner à numfic la valeur 0.
- Détruire le contenu du fichier numéro 14.
- Détruire le contenu du fichier numéro 7.
- Invoquer une routine " système ", afin de remettre l'écran dans son état initial.

3.6.2. Développement de la routine CRENIV

3.6.2.1. Spécification

Appel à la routine : CRENIV (errcre)

Entrées : -

Sorties : . errcre, paramètre de type INTEGER.

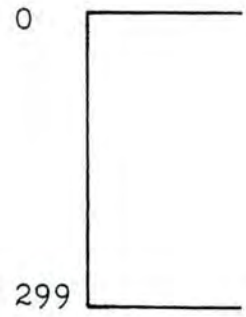
Valeurs possibles :

- ° 0, si aucune erreur fatale n'a été détectée. Dans ce cas, nous obtenons la représentation de la structure hiérarchique en niveaux du programme SPC.
  La structure en niveaux est l'arbre des raffinments, auquel on a associé une structure en niveaux (cfr sous-section 3.3.3.).
  Précisons que dans cette structure, le niveau - 1 ne reprend que le raff 1.
- ° 1, si le raff 1 n'est pas libéré.
- ° 2, s'il existe un cycle de références entre raffinements.
- ° 3, s'il existe plus de vingt références imbriquées à partir du niveau qui contient le raff 0.
- ° 4, si certains raffinements du programme sont référencés un trop grand nombre de fois, pour des raisons d'implémentation cela constitue une erreur fatale.

Choix de représentation

Nous allons expliciter les choix effectués pour représenter la structure hiérrarchique en niveaux.
Celle-ci est représentée par trois tables que nous allons décrire.

(1) la table des raffinements : TABRAF.



A chaque numéro de raffinement (ceux-ci pouvant varier de 0 à 299),correspond un élément de TABRAF.
A chaque élément de TABRAF, correspond un raffinemnt d'instructions exécutables du programme SPC ou le raff 1.
Un élément contient les informations suivantes :

ADD : Adresse du raffinement dans le ficheir SPC.

NUMNIV : Numéro du niveau de ce raffinement.

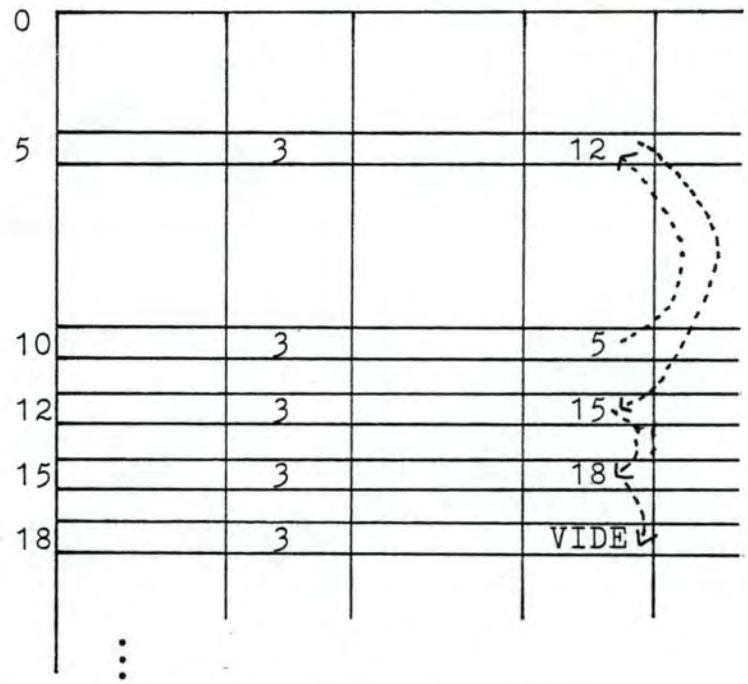
- 1, s'il s'agit du raff 1 des déclarations,
  0, s'il s'agit du raff 0,
  i, si le raffinemnt appartient au niveau i.

RAFSUI : Numéro du raffinement suivant, s'il existe dans le même niveau,
- vide sinon.
Les raffinments d'un même niveau sont chaînés dans un ordre quelconque.

Exemple

Soit le niveau numéro 3 constitué des raffinements numéro 5,10,12,15,18.

TABRAF :



PRPERE : - numéro d'un père du raffinement, s'il existe
- vide, sinon.

SECPER : - adresse dans TABPER (cfr supra) d'un autre père du raffinement, s'il existe
- vide, sinon.

Les éléments de TABRAF, ne correspondant pas à un raffinement du programme SPC ou correspondant à un raffinment de déclarations (sauf le raff 1) contiennent les informations suivantes :

ADD = 0;
NUMNIV = -2;
RAFSUI, PRPERE, SECPER sont vides.
(par convention, vide = 999)

(2) La table des pères : TABPER

Les pères d'un même raffinement sont chaînés dans cette seconde table TABPER, sauf le premier qui est rangé dans TABRAF (PRPERE).
Chaque élément de TABPER contient deux informations :

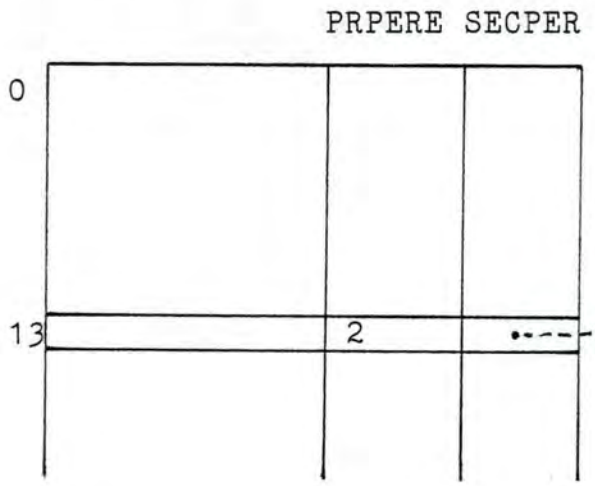
PERE : le numéro d'un des pères du raffinement;
PERSUIV : l'adresse dans TABPER d'un autre père.

Les pères d'un raffinement sont chaînés dans un ordre quelconque.

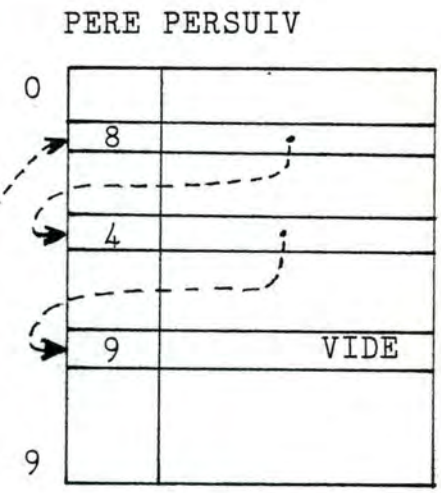
Exemple

Le raffinemnt 13 a comme père les raffinements 2,4,8,9.

TABRAF :



TABPER :

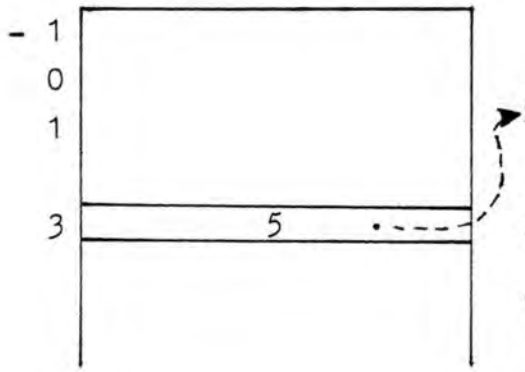


(3) La table des niveaux : TABNIV

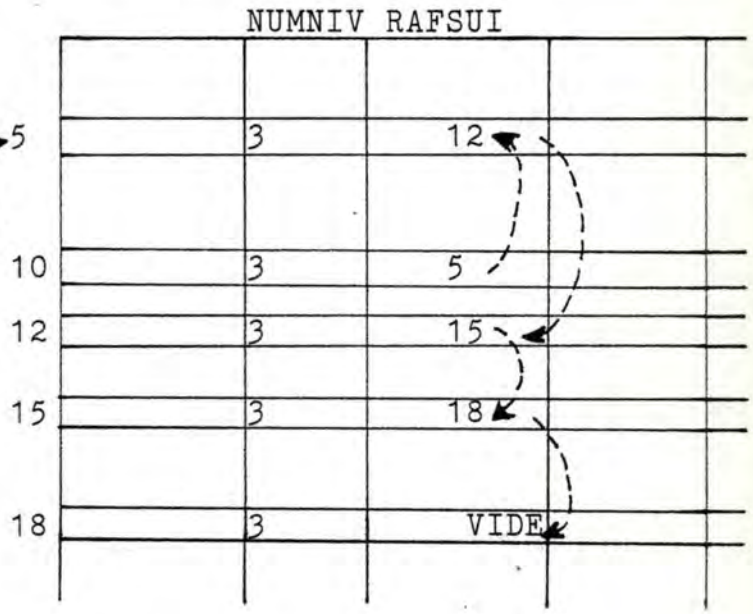
Nous avons vu que les raffinements d'un même niveau sont chaînés dans TABRAF.
A chaque numéro de niveau, variant de - 1 à 18, correspond un élément de TABNIV.
Chaque élément de TABNIV contient le numéro du premier raffinement de ce niveau.

Ainsi pour le premier exemple, nous obtenons :

TABNIV :



TABRAF :



Ces trois tables font partie d'un common. Leurs éléments sont tous de type INTEGER * 2.

Raisons de ce choix de représentation

Ce mode de représentation a été choisi pour

- faciliter les accès à la structure hiérarchique en niveaux.
  Il est facile et rapide de trouver tous les raffinements d'un niveau donné (cfr les routines PREM et SUIV).
  Il est facile et rapide de trouver le niveau d'un raffinement donné (cfr la routine NIVRAF).
  Notons que l'adressage se fait de façon directe grâce au numéro de raffinement pour TABRAF et au numéro de niveau pour TABNIV.
- constuire aisément la structure hiérarchique en niveaux à partir d'un programme SPC (cfr procédé de construction de CRENIV).

3.6.2.2. Procédé de construction

Pour construire une structure hiérarchique en niveaux, il faut :

- Construire la structure hiérarchique.
  (sous-objectif 1)
- Déterminer les niveaux .
  (sous-objectif 2)

Sous-objectif 1 : construire la structure hiérarchique

Cette étape consiste à représenter la structure hiérarchique induite par le programme SPC. Cette structure hiérarchique sera représentée par les éléments PRPERE, SECPER, de la table TABRAF et par les éléments PERE et PERSUIV de la table TABPER.

Nous associerons donc à chaque raffinement, la liste de ses pères.

A cet effet, il faut parcourir l'ensemble des raffinements du programme SPC.

D'une manière générale, le parcours d'un raffinement i s'effectue de la façon suivante :
Pour chaque instruction du raffinement i,

- Si cette instruction est une référence à un raffinemnt j,
  il faut :
  - ° Ajouter le raff i à la liste des pères du raff j.
  - ° Parcourir le raff j.

La construction de la structure hiérarchique s'effectue de la façon suivante :

- Pour chacun des 300 raffinements qui peuvent figurer dans un programme SPC, il faut effectuer des initialisations
  (sous-objectif 3).
- Parcourir le raff 0.

Ainsi, tous les raffinements accessibles à partir du raff 0 seront parcourus et la structure hiérarchique sera représentée.

Sous-objectif 3 : effectuer des initialisations

Pour chaucun des 300 raffinements qui peuvent figurer dans le programme SPC, il faut assigner la valeur vide aux différentes informations liées au raffinement.

- son adresse := VIDE,
- son niveau := VIDE,
- le raffinement suivant dans le niveau := VIDE,
- son premier père := VIDE,
- son second père := VIDE.

Ceci correspond à assigner la valeur 999 aux différentes informations contenues dans chaque élément de TABRAF.

- TABRAF (ADD) := 999,
- TABRAF (NUMNIV) := 999,
- TABRAF (RAFSUI) := 999,
- TABRAF (PRPERE) := 999,
- TABRAF (SCPERE) := 999.

- LIBRE, variable de type INTEGER, représente l'adresse de la première place libre dans TABPER. Elle est initialisée à 1.

Ces initialisations sont effectuées dans le Block Data du programme.

Sous-objectif 4 : Parcours du raff 0

Vu la structure interne des instructions d'un raffinement, la récursivité dans le parcours d'un raff i peut être aisément retirée.

Rappel sur la représentation interne des raffinements

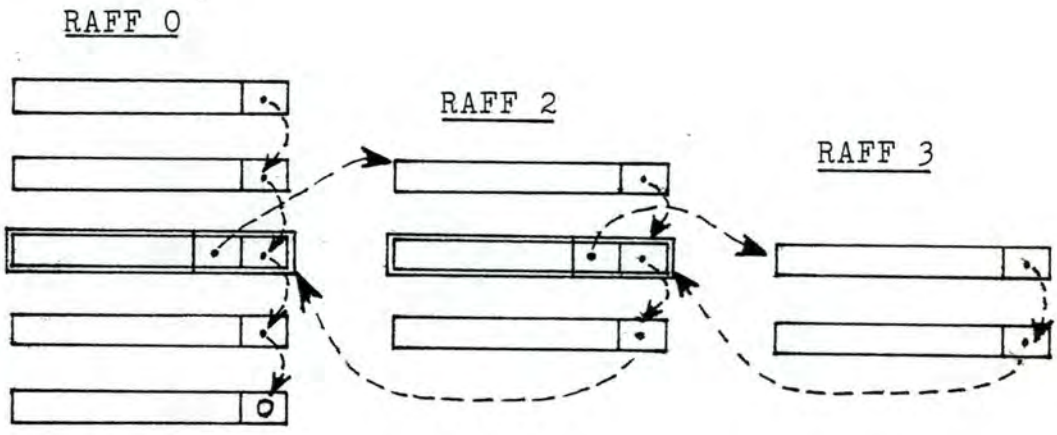
- Chaque élément de chaque instruction contient l'adresse de l'élément suivant de cette instruction ou l'adresse du premier élément de l'instruction suivante.
  Si celle-ci n'existe pas (fin de raffinement), on dispose de l'adresse de l'instruction de référence à ce raffinement. L'adresse obtenue vaut 0, si ce raffinement n'est pas référencé.

- Chaque instruction de " référence à un raffinement " possède l'adresse du premier élément du raffinement référencé.
- Chaque élément possède le numéro de raffinement auquel il appartient.

Exemple

Voici trois groupes d'éléments associés respectivement aux raffinements 0,2,3 d'un programme SPC.
On condière que le raff 0 référence le raff 2, et que ce dernier référence le raff 3.

La représentation internede cette situation est la suivante :



Parcourir le raff 0, revient alors à effectuer les traitements suivants :

- Lire le premier élément du raff 0, grâce à la routine LIT.
- Tant que non (fin fichier ou erreur fatale) faire :
  - ° Si instruction de référence à un raff j
    Alors, - Ajouter le numéro de raffinement auquel appartient cette instruction (RAFF) à la liste des pères du raffinement référencé j.
    (sous-objectif 5).
    - Lire le premier élément du raffinement référencé j, grâce à la routine LIT.

Sinon - Lire l'élément suivant, grâce à la routine LECT.

N.B. Une erreur fatale se présente, lorsque la table TABPER est pleine.

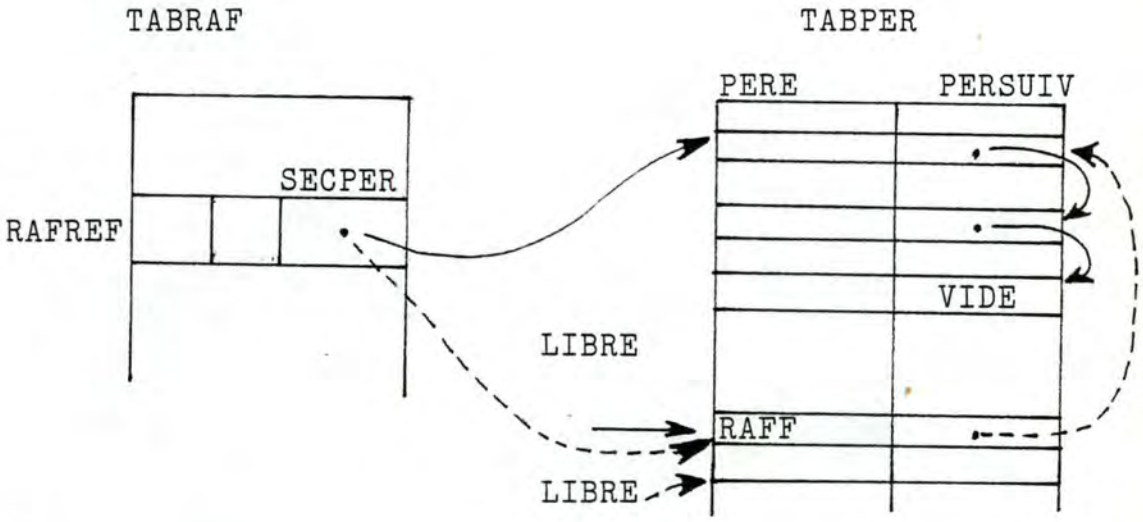
Sous-objectif 5 : Ajouter le numéro de raffinement (RAFF) à la liste des pères du raffinement référencé j.

Soit RAFREF, le numéro du raffinement référencé.
Réaliser ce sous-objectif revient à envisager les traitements suivants :

- SI le raffinement RAFREF a déjà un père,
  ALORS - lui ajouter un nouveau père (RAFF) dans TABPER (sous-objectif 6).
  SINON - le premier père de RAFREF est le raffinement RAFF.
  TABRAF (RAFREF, PRPERE) : = RAFF .

Sous-objectif 6 : Ajouter un nouveau père (RAFF) DANS TABPER

- SI la table TABPER est pleine,
  ALORS - erreur fatale := . true.,
  - errcre := 4.
  SINON le raffinement RAFREF devient le deuxième père du raffinement RAFF.
  - TABPER (LIBRE, PERE) := RAFF
  - TABPER (LIBRE, PERSUIV) := TABRAF (RAFREF, SECPER)
  - TABRAF (RAFREF, SECPER) := LIBRE
  - LIBRE := LIBRE + 1



N.B.

° Flèche en trait plein = ancienne situation
° Flèche en trait pointillé = chaînage de RAFF avec les autres pères de RAFREF

Sous-objectif 2 : Déterminer les niveaux

Réaliser ce sous-objectif, revient à réaliser les sous-objectifs suivants :

- Traiter les raffinements inaccessibles.
  (sous-objectif 7)
- Déterminer le niveau - 1.
  (sous-objectif 8)
- Déterminer le niveau 0,
  (sous-objectif 9)
- Déterminer les niveaux i (i> 0)
  (sous-objectif 10)

Sous-objectif 7 : Traiter les raffinements inaccessibles

- Pour chacun des raffinements de TABRAF,
  ° Si PRPERE = 999 (ce raffinement n'a pas de père, il est inaccessible)

ALORS - PRPERE := 999
- NUMNIV := - 2,
- ADD := 0.

Remarque : Le raff 0 et le raff 1 sont donc considérés temporairement comme inaccessibles.

Sous-objectif 8 : Déterminer le niveau - 1

- SI le raff 1 est libéré,
  ALORS ° Pour la position correspondant au raff 1 dans TABRAF
  - NUMNIV := - 1,
  - ADD := adresse du raff 1 dans le fichier SPC, via la routine VARTIC.

  ° Pour la position correspondant au niveau - 1 dans TABNIV
  - TABNIV (1) := positon correspondant au raff 1 dans TABRAF.

  SINON, - erreur fatale := . true .
  - errcre := 1.

Sous-objectif 9 : Déterminer le niveau 0

Le raff 0 fera toujours partie du niveau 0 et sera le seul raffinement de ce niveau.
Réaliser ce sous-objectif, revient à effectuer les traitements suivants :
- Pour la position correspondant au raff 0 dans TABRAF :
  ° NUMNIV := 0,
  ° ADD := adresse du raff 0 dans le fichier SPC.

- Pour la position correspondant au niveau 0 dans TABNIV :
  - ° TABNIV (0) := position correspondant au raff 0 dans TABRAF.

Sous-objectif 10 : Déterminer le niveau i

Un raffinement accessible de numéro k appartient au niveau i (i > o) si et seulement si tous ses pères sont de niveau < i.

De toute façon, trouver les raffinements du niveau 1 revient à trouver tous les raffinements, ne possédant pas encore de numéro de niveau, (les raffinements inaccessibles possèdent déjà un numéro de niveau) dont les pères sont d'un niveau 1.
Nous dirons que les raffinements auxquels on n'a pas encore attribué de numéro de niveau sont " non traités ".

De façon générale, trouver les raffinements du niveau i revient à trouver tous les raffinements non traités, dont les pères sont d'un niveau < i.
Ce traitement est effectué tant qu'il existe des raffinements non traités (ceci constitue une condition de sortie dans la répétitive correspondant aux traitements des niveaux).

Une erreur fatale peut survenir pour deux raisons :

- Lorsque l'on détecte un cycle dans les références aux raffinements.
  Ce cycle sera détecté lorsque, pour construire le niveau i, l'on est incapable de trouver des raffinements non traités dont tous les pères sont de niveau < i, alors qu'il existe encore des raffinements non traités (FIC, 81).
- Lorsque le nombre maximum de niveaux est atteint (20 niveaux). Ce nombre correspond à la dimension de TABNIV.

Nous déduisons de ces deux points, deux nouvelles conditions de sortie dans la répétitive correspondant aux traitements des niveaux.

Nous obtenons ainsi, pour le traitement des niveaux, une répétitive qui se compose des traitements suivants :
(initialement NIV := 1)

- Examiner si tous les raffinements sont traités.
  Si c'est le cas, il faut sortir de la répétitive.
  (sous-objectif 11)
- Examiner si le nombre maximum de niveaux est atteint (NIV = 18). Si c'est le cas,il faut :
  - ° Effectuer un traitement d'erreur (sous-objectif 12)
  - ° Sortir de la répétitive.
- Mettre PASSER à . false.
- Trouver et chaîner les raffinements du niveau NIV.
  (sous-objectif 13)
- Si PASSER = . false . (on n'a pas trouvé de raffinements non traités dont les pères sont traités) il faut
  - ° Effectuer un traitement d'erreur (détection d'un cycle) (sous-objectif 14).
  - ° Sortir de la répétitive.
- NIV := NIV + 1
- Itérer.

La variable logique PASSER permet de déterminer s'il existe un raffinement non traité dont tous les pères sont de niveau < NIV . Dans ce cas PASSER vaudra . true .

Sous-objectif 11 : Examiner si tous les raffinements sont traités

Afin de voir si tous les raffinements sont traités, il suffit de compter les raffinements traités à l'aide de la variable RAFTRT (de type INTEGER).
Ainsi dans les sous-objectifs 7,10, il faut totaliser le nombre de raffinements dans RAFTRT.

Examiner si tous les raffinements sont traités se résume à faire le test :

- RAFTRT = 300 (nombre total de raff)

Sous-objectif 12 : Effectuer un traitement d'erreur

Dans le cas où NIV > 18,

- erreur fatale := . true .,
- errcre := 3.

Sous-objectif 13 : Trouver et chaîner les raffinements du niveau NIV.

Pour chacun des raffinements de TABRAF

- Si celui-ci est non traité (sous-objectif 15) et si tous ses pères sont traités (sous-objectif 16) ALORS, ce raffinement appartient au niveau NIV.
  Il faut :
  - ° Chaîner ce raffinement (RAFF) avec les autres raffinements du niveau.
    (sous-objectif 17)
  - ° Remplir les champs NIV et ADD de la position correspondant à ce raffinement dans TABRAF.
    - ADD := adresse dans SPC de ce raffinement, via la routine VARTIC,
    - NUMNIV := NIV
  - ° PASSER := . true . (il existe un raffinenement appartenant à ce niveau).

Sous-objectif 15 : voir si un raffinement est non traité

Un raffinement est traité si le champ NUM$_{NIV}$ correspondant à la position dans TABRAF de ce raffinement est vide (999).

Sous-objectif 16 : Examiner si tous les pères du raffinement sont traités

Convention : La variable PRTRTE vaut . true . tant que tous les pères rencontrés sont traités (leur niveau NUMNIV < NIV).

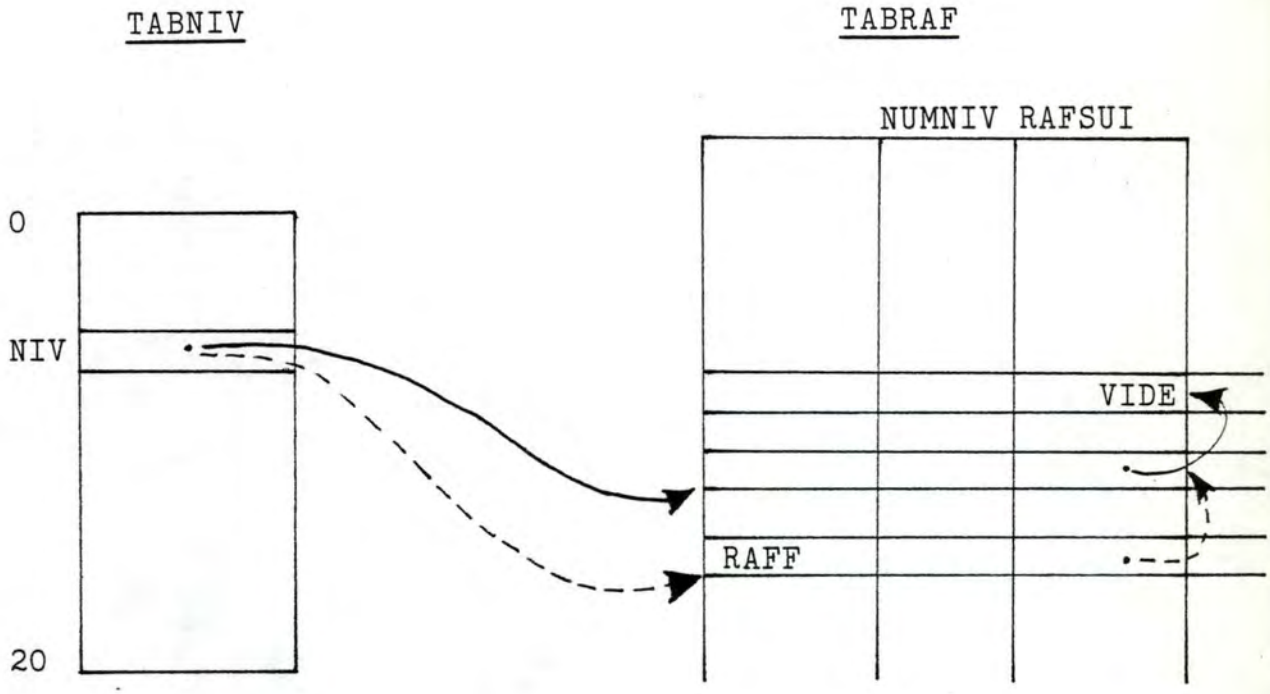
Réaliser ce sous-objectif revient à effectuer les traitements suivants :

- Pour la position de TABRAF correspondant au premier père du raff i, il faut déterminer si le champ NUMNIV < NIV.
  Si c'est le cas, le premier père est traité.
  N.B. le premier père existe toujours vu que le raff i est accessible.
- Parcourir la liste des pères suivants et examiner s'ils sont traités.
  Tant que la liste n'est pas vide et PRTRTE
  faire - Si le père courant est traité
  ALORS le père courant = le père suivant dans la liste des pères
  Sinon PRTRTE := . false.

Sous-objectif 17 : Chaîner le raffinement(RAFF) avec les autres raffinements du niveau

Le raffinement RAFF devient le premier raffinement du niveau NIV.
On a donc :

- Pour la position correspondant à RAFF dans TABRAF : faire
  - TABRAF (NUMNIV) := NIV
  - TABRAF (RAFSUI) := TABNIV (NIV)
  - TABNIV (NIV) := RAFF
- Incrémenter RAFTRT de 1



N.B.

° Flèche en trait plein = ancienne situation
° Flèche en trait pointillé = chaînage de RAFF
avec les autres raffinements
du niveau NIV.

Sous-objectif 14 : Effectuer un traitement d'erreur

Il s'agit de la détection d'un cycle de références entre raffinements :

- Erreur fatale := . true .
- errcre := 2

3.6.3. Développement de la routine PREM

3.6.3.1. Spécification

Appel à la routine : PREM (niveau, adrsra erprem

Entrées :
- ° niveau, paramètre de type INTEGER *2. Représente un numéro de niveau;
- ° Représentation de la structure hiérarchique en niveaux.

Sorties :
- ° adrsra, paramètre de type INTEGER *2 représente l'adresse dans le fichier SPC du premier raffinement du niveau, s'il existe;
- ° erprem, paramètre de type INTEGER. Valeurs possibles
  - ° 1, si le premier raffinement du niveau n'existe pas.
  - ° 0, sinon.

3.6.3.2. Procédé de construction

Déterminer l'adresse du premier raffinement d'un niveau donné revient à effectuer les traitements suivants :
(après avoir initialiser erprem à0)

- Examiner la position correspondant à ce numéro de niveau dans TABNIV
  - ° Si celle-ci ≠ vide(999) (elle contient alors le numéro du premier raffinement)
    Alors, pour la position correspondant à ce numéro de raffinement dans TABRAF,
    - ADRSRA := TABRAF (ADD)
    - Retenir le numéro de niveau et le numéro et le numéro de son premier raffinement; ils nous serviront dans SUIV.

- NIPREC := NIVEAU
- RAPREC := numéro du premier raffinement du niveau

Sinon (le premier raffinement d'un niveau donné n'existe pas)

- ERPREM := 1

3.6.4. Développement de la routine SUIV

3.6.4.1. Spécification

Appel à la routine : SUIV (niveau, adrsra, srsuiv

Entrées :

- ° niveau, paramètre de type INTEGER *2, représente un numéro de niveau.
- ° Représentation de la structure hiérarchique en niveaux.
- ° nivprec, paramètre de type INTEGER représente un numéro de niveau.
- ° rafprec, paramètre de type INTEGER représente un numéro de raffinement.

Ces deux dernières variables font parties d'un common. Leur existence provient de l'appel à la routine PREM qui précède toujours celui de SUIV.

Sorties :

- ° adrsra, paramètre de type INTEGER *2 représente l'adresse dans le fichier SPC du raffinemnt qui suit rafprec dans le le même niveau.
- ° ersuiv, paramètre de type INTEGER

  Valeurs possibles
  - ° 2, si le raffinement qui suit rafprec dans le niveau n'existe pas.
  - ° 3, si le raffinement suivant dans le niveau est demandé alors que le premier raffinement du même niveau ne l'a pas été.
  - ° 0, sinon.

3.6.4.2. Procédé de construction

Déterminer l'adresse du raffinement qui suit rafprec à partir d'un niveau donné, revient à effectuer les traitements suivants :

- ersuiv := 0
- Si NIVPREC = NIVEAU
  ALORS, déterminer l'adresse dans SPC du raffinement suivant dans ce niveau
  (sous-objectif 1)
  SINON, ersuiv = 3.

Sous-objectif 1 : Déterminer l'adresse dans SPC du raffinement suivant dans ce niveau.

- Pour la position correspondant à rafprec dans TABRAF
  - ° Si le champ RAFSUI associé à cette position ≠ vide (999) (il contient alors le numéro du raffinemnt suivant)
    Alors - pour la position correspondant à ce numéro de raffinement dans TABRAF, ADRSRA := TABRAF (ADD).
    - rafprec := TABRAF (RAFSUI)
    Sinon (il n'y a pas de raffinement suivant)
    - ersuiv := 2

3.6.5. Développement de la routine NIVRAF

3.6.5.1. Spécification

Appel à la routine : NIVRAF (numéro, niveau)

Entrées :
- ° représentation de la structure hiérarchique en niveau;
- ° numéro, paramètre de type INTEGER * 2 représente un numéro de raffinement.

Sorties :
- ° niveau, paramètre de type INTEGER * 2 représente le numéro du niveau associé associé à ce raffinement.

3.6.5.2. Procédé de construction

Pour la position correspondant à ce numéro de raffinement dans TABRAF,

- Mettre le contenu du champ NUMNIV associé à cette position dans la variable NIVEAU
  NIVEAU := TABRAF (NUMNIV)

3.6.6. Développement de la routine TABSI

3.6.6.1. Spécification

Appel à la routine : TABSI (maxira)

Entrées : ° représentation de la structure hiérarchique en niveau

Sortie : ° maxira, paramètre de type INTEGER * 2 représente le plus grand numéro de raffinement figurant dans le programme SPC.

3.6.6.2. Procédé de construction

Il s'agit de parcourir TABRAF par ordre décroissant de numéro de raffinement :

- Pour chaque élément de TABRAF, examiner le champ NUMNIV associé à l'élément
  - ° Si celui-ci = - 2 (le raffinement n'appartient pas au programme SPC)
    Alors, numéro de raffinement = numéro de raffinement - 1
    Sinon, on sort de la répétitive et MAXIRA := numéro de l'élément courant de TABRAF .

3.6.7. Développement de la routine TRTRAF

3.6.7.1. Spécification

Appel à la routine : TRTRAF

Entrées : -

Sorties : soit ° le texte d'un programme PASCAL, structuré en niveaux, avec des procédures, correspondant à la traduction d'un programme SPC. Ce texte PASCAL figure dans le fichier qui lui est réservé;

° d'éventuels messages d'erreur, dans un fichier, s'il existe des erreurs de codage relatives aux raffinements de déclarations.

(1) L'instruction a coder en première colonne est interdite.
*** libellé de cette instruction ***
raff i, ligne j

(2) Le raff 1 et ceux qu'il référence ne peuvent contenir que des déclarations, des commentaires et des références à d'autres raffinements.
*** libellé de cette instruction ***
raff i, ligne j

i représente le numéro du raffinement où figure l'erreur
j représente le numéro de la ligne dans le raff i, où figure l'erreur.

D'autres messages d'erreur de ce type seront explicités par la suite;

° errcod, variable de type INTEGER représente le nombre d'erreurs de codage détectées lors de la traduction du programme SPC;

soit ° un message à l'écran parmi les messages suivants, s'il existe une erreur fatale relative aux raffinements de déclarations.

(1) Le raff 1 n'existe pas
arrêt de la traduction.

(2) Le raff 1 et les raffinements référencés à partir de celui-ci ne contiennent aucunes déclarations.
arrêt de la traduction.

(3) Il y a plus de vingt références imbriquées à partir du niveau qui contient le raff 1
arrêt de la traduction.

(4) Le raff 1 ne contient aucune déclaration et aucune référence à un raffinement.

3.6.7.2. Procédé de construction

Au vu de la structure d'un programme PASCAL (cfr exemple annexe 1 au point 2.2.) et d'après les règles de traduction (cfr sous-section 3.4.3), il s'agit de produire :

(1) l'en-tête du programme PASCAL
(sous-objectif 1);
(2) les déclarations de constantes, types, variables de ce programme
(sous-objectif 2);
(3) les déclarations forward des procédures du programme
(sous-objectif 3);
(4) les déclarations du texte des procédures structurées en niveaux
(sous-objectif 4);
(5) le programme principal de ce programme PASCAL
(sous-objectif 5);

S'il n'y a pas d'erreur fatale.

La mise en page d'un programme PASCAL consiste à :
- Indenter les instructions PASCAL produites
- Souligner les titres suivants :
  - ° déclarations
  - ° déclarations des procédures
  - ° niveau i $1 \leq i \leq 18$
  - ° programme principal
  - ° le mot réservé " PROGRAM " suivi du nom de ce programme
- Passer des lignes blanches.

Sous-objectif 1 : production de l'en-tête du programme PASCAL

Réaliser le sous-objectif 1, revient à :

- Invoquer la routine PREM, afin d'accéder au premier raffinement du niveau 0 (le raff 0).
- Invoquer les routines LIT et LECT, afin d'accéder aux deux premières instrucitons du raff 0.
- Invoquer la routine IMPLI, afin de produire l'u.t.P. " en-tête programme ".

Sous-objectif 2 : production des déclarations de constantes, types, variables de ce programme.

Réaliser le sous-objectif 2, revient à :

- Invoquer la routine PREM, afin d'accéder au premier raffinement du niveau - 1 (le raff 1).
- Si ce premier raffinement n'existe pas, on invoque la routine IMPECR afin de produire un message associé à cette erreur fatale. La réalisation du sous-objectif 2 est impossible.
- S' il n'y a pas d'erreur fatale, on peut :
  - ° Invoquer la routine IMPLI, afin de produire l'u.t.P. " en-tête déclarations ".
  - ° Invoquer la routine LIT, afin de lire la première instruction du raff 1.
  - ° Produire le corps des déclarations.
    Cela correspond à la traduction de toutes les instructions du raff 1;
    à moins que l'on ne détecte une erreur fatale.
    (sous-objectif 6).
  - ° Si aucune erreur fatale n'a été détectée lors de la réalisation du sous-objectif 6, on peut invoquer la routine IMPLI, afin de produire l'u.t.P. " fin déclaration ".
  - ° Sinon, il faut
    Invoquer la routine IMPECR, afin d'afficher un message correspondant à l'erreur fatale rencontrée.

Sous-objectif 3 : Production des déclarations forward des procédures du programme

La réalisation de ce sous-objectif n'est possible que si aucune erreur fatale n'a été rencontrée jusqu'à présent.

Réaliser le sous-objectif 3, revient à :

- Invoquer la routine PREM afin d'accéder au premier raffinement du niveau 1.
- Si ce premier raffinement existe, invoquer la routine IMPLI, afin de produire l'U.t.P. " en-tête déclarations procédures forward ".
- Produire l'u.t.P. " corps déclarations procédures forward "
  (sous-objectif 7).

Sous-objectif 4 : les déclarations du texte des procédures structurées en niveaux

Réaliser ce sous-objectif, revient à :

- Invoquer la routine PREM, afin d'accéder au premier raffinement du niveau 1.
- Pour chacun des niveaux de la structure hiérarchique en niveaux,
  - ° Invoquer la routine IMPLI et ALPHA, afin de produire l'u.t.P. " en-tête niveau ".
  - ° Produire le texte PASCAL du corps d'un niveau. Cela correspond à la traduction des raffinements de ce niveau
    (sous-objectif 8).
  - ° Invoquer la routine PREM, afin d'obtenir le premier raffinement du niveau suivant, s'il existe.

Sous-objectif 5 : Production du programme principal d'un programme PASCAL

Réaliser ce sous-objectif revient à :

- Invoquer la routine IMPLI, afin de produire l'u.t.P. " en-tête programme principal ".
- Invoquer la routine PREM afin d'accéder au raff 0.

- Invoquer les routines LIT et LECT, afin d'accéder à la troisième structure du raff 0.
- Mettre à jour le compteur de ligne pour le raff 0.
- Invoquer la routine IDSTR, afin de traduire toutes les structures (sauf les deux premières qui ont déjà été traduites) du raff 0.
- Invoquer la routine IMPLI, afin de produire l'u.t.P. " fin programme principal ".

Sous-objectif 6 : Produire le corps des déclarations

Réaliser ce sous-objectif, revient à effectuer les traitements suivants :

- S'il s'agit d'une structure non séquentielle il faut :
  - ° invoquer la routine IMPERR, afin d'envoyer un message relatif à cette erreur de codage.
  - ° incrémenter de 1 errcod.
  - ° invoquer la routine LECT jusqu'à ce que l'on ait lu toute cette structure.

  Sinon,
  - ° identifier l'instruction et produire le texte PASCAL correspondant à l'instruction. (sous-objectif 9)

Sous-objectif 9 : identifier l'instruction et produire le texte PASCAL correspondant

- S'il s'agit d'un commentaire narratif

  Il faut :
  - ° Invoquer les routines IMPLI et LECT, afin de produire l'u.t.P. " commentaire ".
  - ° Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.
- S'il s'agit d'une déclaration

  Il faut :
  - ° Invoquer les routines IMPLI et LECT afin de produire une déclarations PASCAL.
  - ° Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

- S'il s'agit d'un commentaire opérationnel
  il faut : ° produire le texte PASCAL correspondant
  au commentaire opérationnel
  (sous-objectif 10).
- Sinon
  il faut : ° Incrémenter de 1 errcod.
  ° Invoquer la routine IMPERR, afin de produire le message d'erreur correspondant à l'instructio fautive.
  ° Invoquer la routine LECT, jusqu'à ce que l'on ait lu toute l'instruction fautive.
  ° Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

Sous-objectif 10 : Production du texte PASCAL correspondant au commentaire opérationnel

- si le commentaire opérationnel ne coïncide pas avec une fin de raffinement, (cas d'une référence vers un raffineemnt
  il faut :
  ° Incrémenter de 1, le nombre de références imbriquées à partir du raff 1.
  ° Si ce nombre de références imbriquées est strictement supérieur à vingt, invoquer la routine IMPECR, afin de produire à l'écran un message relatif à cette erreur fatale.
  ° Sinon
  il faut :
  ° Appliquer la technique du remplacement de texte, c'est-à-dire mettre le raffinement à la place de sa référence.
  A cet effet il suffit
  ° De retenir le numéro de la ligne où figure le commentaire opérationnel dans le raffinement courant.

° D'initialiser le compteur de lignes pour le raffinement référencé.
° D'invoquer la routine LIT, afin de lire la première instruction du raffinement référencé.

- Sinon (cas d'une fin de raffinement)

Cela signifie, que l'on a terminé de remplacer la référence à un raffinement par le raffinement lui-même.
On est revenu à l'endroit où figurait la référence à un raffinement dans le raffinement courant.

Il faut alors :

° Restaurer le numéro de ligne correspondant à cet endroit dans le raffinement courant.
° Invoquer la routine LECT, afin de continuer à lire dans le raffinement courant.

Sous-objectif 7 : Produire l'u.t.P. " corps déclarations procédures forward "

- Pour chacun des niveaux de la structure hiérarchique en niveaux,
  ° Pour chacun des raffinements de ce niveau,

  il faut :
  ° Invoquer la routine LIT, afin d'accéder à la première instruction du raffinement. Cette première instruction reprend le nom du raffinement.
  ° Produire un commentaire PASCAL, reprenant le nom de ce raffinement.
  ° Produire les mots réservés " Procédure Pi; FORWARD "; où i est le numéro du raffinement que l'on trouve dans la variable " raff " de l'élément courant.
  ° Invoquer la routine SUIV, afin d'obtenir le raffinement suivant dans ce niveau, s'il existe.

  ° Invoquer la routine PREM, afin d'obtenir le premier raffinement du niveau suivant, s'il existe.

Sous-objectif 8 : Produire le texte PASCAL du corps d'un niveau

Pour chacun des raffinements du niveau,

Il faut :

- Initialiser le compteur de ligne du raffinement.
- Invoquer la routine LIT afin de lire la première instruction du raffinement.
- Invoquer les routines IMPLI et ALPHA afin de produire l'u.t.P. " en-tête procédure ".
- Invoquer la routine ISDTR, afin de produire le corps de la procédure associéeau raffinement.
- Invoquer la routine IMPLI, afin de produire l'u.t.P. " fin de la procédure ".
- Invoquer la routine SUIV, afin d'obtenir le raffinement suivant dans ce niveau, si il existe.

3.6.8. Développement de la routine IDSTR

3.6.8.1. Spécification

Appel à la routine : IDSTR

Entrées
- l'élément courant représente le dernier élément de la première structure du raffinement à traduire.

Sorties
- le corps d'une procédure PASCAL, dans un fichier, correspondant à la traduction des structures (sauf la première) de ce raffinement;
- des messages d'erreur, dans un fichier, associés aux éventuelles erreurs de codage relatives aux raffinements d'instructions exécutables.

(1) l'instruction à coder en première colonne est interdite
*** texte de cette instruction ***
raff i ligne j

(2) une déclaration ne peut figurer en dehors des raffinements de déclarations:
le raff 1 et ceux qu'il référence
*** texte de cette instruction ***
raff i ligne j

i : représente le numéro du raffinement où figure l'erreur.

j : représente le numéro de la ligen dans le raffinement, où figure l'erreur.

3.6.8.2. Procédé de construction

D'après les règles de traduction, produire le corps d'une procédure PASCAL, revient à effectuer les traitements suivants :

- Invoquer la routine LECT, afin de lire le premier élément de la deuxième structure du raffinement à traduire.
- Pour chacune des structures du raffinement à traduire, il faut :
    - ° Identifier son type.
    - ° Produire l'instruction PASCAL correspondante.
        - S'il s'agit d'une structure séquentielle, il faut produire l'instruction simple PASCAL correspondante. (sous-objectif 1)
        - S'il s'agit d'une structure conditionnelle, il faut produire l'instruction conditionnelle PASCAL correspondante. (sous-objectif 2)
        - S'il s'agit d'une structure répétitive, il faut produire l'instruction répétitive PASCAL correspondante. (sous-objectif 3)

Sous-objectif 1 : Produire l'instruction simple PASCAL correspondante

Cela revient à envisager les traitements suivants :

- Identifier l'instruction SPC composant la structure séquentielle.
- Invoquer la routine de traduction correspondant au résultat de l'identification.
    - ° On invoque la routine AFPRO, s'il s'agit d'une affectation, ou d'un appel de procédure externe.
    - ° On invoque la routine COMNA, s'il s'agit d'un commentaire narratif.
    - ° On invoque la routine COMOP, s'il s'agit d'un commentaire opérationnel.
- Invoquer la routine FAUTCO, qui imprime, dans un fichier, le message d'erreur correspondant à l'identification d'une déclaration ou d'une instruction à coder en première colonne.

Sous-objectif 2 : Produire l'instruction conditionnelle PASCAL correspondante

Cela revient à envisager les traitements suivants :

- Invoquer les routines IMPLI et LECT, afin de produire l'u.t.P. " en-tête de la conditionnelle ".
- Invoquer la routine LECT, afin de lire le premier élément de la seconde instruction de la structure.
- Produire le corps de l'instruction conditionnelle PASCAL. Cela correspond à la traduction des instructions (sauf la première) de la structure conditionnelle SPC. (sous-objectif 4)
- Invoquer la routine IMPLI afin de produire l'u.t.P. " fin de la conditionnelle ".

Sous-objectif 3 : Produire l'instruction répétitive PASCAL correspondante

Cela revient à envisager les traitements suivants :

- Invoquer la routine SURVOL, afin de déterminer la forme selon laquelle la structure répétitive SPC doit être traduite.
- Invoquer la routine de traduction correspondant au résultat de l'identification.
    - ° On invoque la routine REPETR, s'il s'agit de produire une instruction répétitive PASCAL REPEAT .... UNTIL.
    - ° On invoque la rotuine REPETW, s'il s'agit de produire une instruction répétitive PASCAL WHILE.
    - ° On invoque la routine REPETS, s'il s'agit de produire une instruction répétitive PASCAL, traduite selon une forme standard.

Sous-objectif 4 : Produire le corps de l'instruction conditionnelle PASCAL

Pour chacune des instructions SPC composant la structure il faut :

- Identifier son type.
- Invoquer la routine de traduction correspondant au résultat de l'identification.
    - ° On invoque la routine EXPBO, s'il s'agit d'une instruction conditionnelle SPC.
    - ° On invoque la routine SINON, s'il s'agit de l'instruction SPC sinon.

    cfr le sous-objectif 1 pour la suite du traitement.

3.6.9. Développement de la routine REPETW

3.6.9.1. Spécification

Appel à la routine : REPETW (nbinst)

Entrées :
- ° l'élément courant est le premier élément de la structure répétitive;
- ° nbinst, paramètre de type INTEGER représente le nombre d'instructions dans cette structure (y compris les conditions de sortie avec ou sans référence à un raffinement).

Sorties :
- ° l'instruction répétitive PASCAL While, correspondant à la traduction de cette structure répétitive;
- ° l'élément courant est le premier élément de la structure suivante, si elle existe;
- ° d'éventuels messages d'erreurs dans un fichier, s'il existe des erreurs de codage relatives aux raffinements d'instructions exécutables.
  (cfr spécification des sorties de IDSTR)

3.6.9.2. Procédé de construction

D'après les règles de traduction, produire l'instruction répétitive PASCAL While, revient à effectuer les traitements suivants :

- Invoquer les routines IMPLI et LECT, afin de produire l'u.t.P. " en-tête While ".
  La forme de cette u.t.P. dépend du type de la condition de sortie (avec ou sans référence à un raffinement) et de nbinst.
- Produire le corps du While,
  c.à.d.
  - ° Si nbinst = 2, il faut produire le texte PASCAL correspondant à la traduction de l'unique instruction de cette structure.
    (sous-objectif 1)

° si nbinst est strictement supérieure à 2, il faut produire le texte PASCAL correspondant à la traduction de chacune des instructions de la structure répétitive.
(sous-objectif 2)

- Si la condition de sortie, est une condition sans référence à un raffinement, il faut :
  ° Invoquer la routine IMPLI, afin de produire l'u.t.P. " fin While simple ".

- Sinon il faut :
  ° Invoquer les routines IMPLI NIVRAF et ALPHA, afin de produire l'u.t.P.
  " fin While traitement d'exception ".

Sous-objectif 1 : Production du texte PASCAL correspondant à la traduction de l'unique instruction de cette structure

Réaliser ce sous-objectif, revient à envisager les traitements suivants :

° Identifier le type de l'instruction.
° Invoquer la routine AFPRO, s'il s'agit d'une affectation ou d'un appel de procédure externe.
° Invoquer la routine COMOP s'il s'agit d'un commentaire opérationnel.

Sous-objectif 2 : Production du texte PASCAL correspondant à la traduction de chacune des instructions de la structure répétitive

Réaliser ce sous-objectif, revient à envisager les traitements suivants :

Pour chacune des instructions de la structure

Il faut : ° Identifier son type.
° Invoquer la routine correspondant au résultat de l'identification.

° On invoque la routine AFPRO
s'il s'agit d'une affectation ou d'un appel de procédure externe.
° On invoque la routine COMOP, s'il s'agit d'un commentaire opérationnel.
° On invoque la routine COMNA, s'il s'agit d'un commentaire narratif.

- Invoquer la routine FAUTCO, qui imprime un message relatif à une erreur de codage.
Il y a une erreur de codage lorsque l'instruction identifiée est une déclaration ou une instruction à coder en première colonne.

3.6.10. Développement de la routine REPETR

3.6.10.1. Spécification

Appel à la routine : REPETR

Entrées
- ° l'élément courant est le premier élément de la structure répétitive.

Sorties
- ° l'instruction répétitive PASCAL REPEAT ... UNTIL, correspondant à la traduction de cette structure répétitive;
- ° pour la suite, cfr les spécifications des sorites de a routine REPETW.

3.6.10.2. Procédé de construction

D'après les règles de traduction,
produire l'instruction répétitive Repeat ... Until revient à effectuer les actions suivantes :

- Invoquer la routine IMPLI, afin de produire l'u.t.P. " en-tête Repeat ".
- Produire le corps du Repeat ... Until, c.à.d. produire le texte PASCAL correspondant à la traduction de chacune des instructions (sauf la condition de sortie) de la structure répétitive. (sous-objectif 1)
- Si la condition de sortie, est une condition sans référence à un raffinement, il faut :
  - ° Invoquer les routines IMPLI et LECT, afin de produire l'u.t.P. " fin repeat simple ".
- Sinon, il faut :
  - ° Invoquer les routines IMPLI, NIVRAF et ALPHA, afin de produire l'u.t.P. " fin repeat traitement d'exception ".

Sous-objectif 1 : Produire le texte PASCAL, correspondant à la traduction de chacune des instructions de la structure répétitive

Les traitements sont identiques à ceux décrits dans le sous-objectif 2 de la routine REPETW.

3.6.11. Développement de la routine REPETS

3.6.11.1. Spécification

Appel à la routine REPETS

Entrées
- cfr les spcécifications des entrées de la routine PEPETR.

Sorties
- une instruction répétitive PASCAL correspondant à la traduction, selon une forme standard, d'une structure répétitive SPC;
- pour la suite, cfr les spécifications des sorties de la routine REPETW.

3.6.11.2. Procédé de construction

Pour produire une instruction répétitive PASCAL selon une forme standar, revient à réaliser les traitements suivants :

- Invoquer les routines IMPLI et ALPHA, afin de produire l'u.t.P. " en-tête standard ".
- Produire le corps de cette instruction répétitive c.à.d. produire le texte PASCAL correspondant à la traduction de chacune des instructions de la structure répétitive.
  (sous-objectif 1)
- Invoquer les routines IMPLI et ALPHA afin de produire l'u.t.P. " fin standard ".

Sous-objectif 1 : Produire le texte PASCAL correspondant à la traduction de chacune des instructions de la structure répétitive

Les traitements sont identiques à ceux décrits dans le sous-objectif 2 de la routine REPETW.

3.6.12. Développement de la routine PTSOR

3.6.12.1. Spécification

Appel à la routine : PTSOR

Entrées : ° l'élément courant est le premier élément d'une condition de sortie avec ou sans référence à un raffinement.

Sorties : soit ° l'u.t.P. " condition de sortie " correspondant à la traduction d'une condition de sortie, sans référence à un raffinement, dans une structure répétitive qui doit être traduite selon la forme standard;

soit ° l'u.t.P. " condition de sortie avec traitement d'exception " correspondant à la traduction d'une condition de sortie, avec référence à un raffinement, dans une structure répétitive à traduire selon la forme standard;

° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

3.6.12.2. Procédé de construction

D'après les règles de traduction, les traitements à envisager sont les suivants :

- S'il s'agit d'une condition de sortie sans référence à un raffinement, il faut :
  - ° Déterminer le nombre de chiffres significatifs dans le numéro du raffinement.
    Ce numéro de raffinement se trouve dans la variable " raff " de l'élément courant.
  - ° Invoquer la routine ALPHA, afin de transformer ce nombre en une chaîne de caractères.
  - ° Invoquer la routine IMPLI, jusqu'à ce que l'on ait produit le texte PASCAL de l'u.t.P. : " condition de sortie ".

° Invoquer la routine IMPLI, afin de lire le premier élément de l'instruction suivante.
° Si cette instruction SPC coïncide avec une fin de structure, le " else begin " dans l'u.t.P. ne sera pas produit.
° Sinon, incrémenter de 1 le nombre de " begin " produits (pour la génération du nombre de " end " correspondant).

- Sinon, il faut :

° Déterminer le nombre de chiffres significatifs dans le numéro du raffinement référencé.
° Invoquer la routine NIVRAF, afin de connaître le niveau auquel appartient ce raffinement.
° Déterminer le nombre de chiffres significatifs dans le numéro de niveau.
° Invoquer la routine ALPHA, afin de transformer le numéro du raffinement en une chaîne de caractères.
° Invoquer la routine ALPHA, afin de transformer le numéro de niveau en une chaîne de caractères.
° Invoquer la routine IMPLI, jusqu'à ce que l'on ait produit le texte PASCAL de l'u.t.P.
" condition de sortie avec traitement d'exception ".
° Invoquer la routine IMPLI, afin de lire le premier élément de l'instruction suivante.
° Si cette instruction coïncide avec la fin de la structure répétitive, le " else begin " ne sera pas produit.
° Sinon, incrémenter de 1 le nombre de " begin " produits.

3.6.13. Développement de la routine AFPRO

3.6.13.1. Spécification

Appel à la routine : AFPRO (indent)

Entrées :
- ° l'élément courant est le premier élément d'une instruction d'affectation ou d'un appel de procédure externe;
- ° indent, paramètre de type INTEGER, représente le nombre de blancs à insérer avant le début du texte PASCAL produit.

Sorties :

soit
- ° une affectation PASCAL dans un fichier correspondant à la traduction de l'affectation SPC;

soit
- ° un appel de procédure externe PASCAL dans un fichier, correspondant à la traduction de l'appel SPC de procédure externe;
- ° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

3.6.13.2. Procédé de construction

D'après les règles de traduction, il s'agit de réaliser les traitements suivants :

- pour chacun des éléments de l'instruction SPC à traduire, il faut :
  - ° Invoquer les routines IMPLI et LECT jusqu'à obtenir leur équivalent en PASCAL.
- Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

3.6.14. Développement de la routine COMNA

3.6.14.1. Spécification

Appel à la routine COMNA (indent)

Entrées :
- ° l'élément courant est le premier élément du commentaire narratif SPC;
- ° indent, paramètre de type INTEGER, représente le nombre de blancs à insérer avant le début du texte PASCAL.

Sorties :
- ° un commentaire PASCAL, dans un fichier, correspondant à la traduction du commentaire narratif SPC;
- ° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

3.6.14.2. Procédé de construction

D'après les règles de traduction, pour obtenir un commentaire PASCAL, il faut :

- Pour chacun des éléments de l'instruction SPC à traduire; il faut invoquer les routines IMPLI et LECT jusqu'à ce que l'on ait produit ce commentaire PASCAL.
- Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

3.6.15. Développement de la routine FAUTCO

3.6.15.1. Spécification

Appel à la routine FAUTCO (nummes)

Entrées :

- l'élément courant est le premier élément d'une déclaration SPC ou d'une instruction à coder en première colonne;
- nummes, paramètre de type INTEGER, représente le numéro du message d'erreur à imprimer.
  valeurs possibles :
  - 2, s'il s'agit d'une instruction à coder en première colonne
  - 3, s'il s'agit d'une déclaration;
- errcod, variable de type INTEGER, représente le nombre d'erreurs de codage détectées avant l'élément courant. Cette variable fait partie d'un common.

Sorties :

- le message d'erreur de numéro nummes, dans un fichier.
  Le libelle des deux messages se trouve dans les spécifications des sorties de la routine IDSTR;
- l'élément courant est le premier élément de l'instruction qui suit l'instruction mentionnée en entrée, si elle existe;
- errcod conserve la même valeur.

3.6.15.2. Procédé de construction

Afin de produire le message d'erreur adéquat, les traitements suivants doivent être réalisés :

- Incrémenter de 1, errcod.
- Invoquer la routine IMPERR, afin d'imprimer le message associé à cette erreur de codage.
- Invoquer la routine LECT afin de lire tous les éléments de cette instruction et le premier élément de l'instructinn suivante.

3.6.16. Développement de la routine EXPBO

3.6.16.1. Spécification

Appel à la routine EXPBO

Entrées : ° l'élément courant est le premier élément de la condition SPC.

Sorties : ° l'u.t.P. " jonction alternative suivante " correspondant à la traduction de cette instruction SPC;
° l'élément courant est le premier élément de l'instruction suivante, si elle existe.

3.6.16.2. Procédé de construction

D'après les règles de traduction, les traitements à envisager pour produire cette u.t.P. sont les suivantes :

- Invoquer la routine IMPLI, afin de produire les mots réservés PASCAL qui composent l'u.t.P. " jonction alternative suivante ".
- Invoquer les routines IMPLI et LECT jusqu'à ce que l'on ait produit l'équivalent PASCAL de la condition SPC.
- Invoquer la routine LECT afin de lire le premier élément de l'instruction suivante.

3.6.17. Développement de la routine SINON

3.6.17.1. Spécification

Appel à la routine SINON

Entrées :
- ° l'élément courant est l'unique élément de l'instruction SPC " Sinon ".

Sorties :
- ° l'u.t.P. " jonction dernière alternative " correspondant à la traduction de cette instruction SPC;
- ° l'élément courant est le premier élément de l'instruction suivante.

3.6.17.2. Procédé de construction

D'après les règles de traduction, les traitements à effectuer pour produire cette u.t.P. sont les suivantes :

- Invoquer la routine IMPLI, afin de produire chacun des mots réservés qui composent l'u.t.P.
- Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

3.6.18. Développement de la routine COMOP

3.6.18.1. Spécification

Appel à la routine COMOP (indent)

Entrées
- l'élément courant est l'unique élément du commentaire opérationnel SPC;
- indent, paramètre de type INTEGER, représente le nombre de blancs à insérer avant le début du texte PASCAL.

Sorties
- l'u.t.P. " appel procédure interne plus commentaire " correspondant à la traduction de cette instruction SPC;
- l'élément courant est le premier élément de l'instruction qui suit ce commentaire opérationnel.

3.6.18.2. Procédé de construction

D'après les règles de traduction, les traitements à effectuer pour produire cette u.t.P. sont les suivants :

- Déterminer le numéro du raffinement référencé.
  Ce numéro est représenté par la variable " pntr " de l'élément courant.
- Invoquer la routine NIVRAF, afin de déterminer le numéro de niveau de ce raffinement.
- Compter le nombre de chiffres significatifs dans le numéro du raffinement référencé.
- Compter le nombre de chiffres significatifs dans le numéro de niveau.
- Invoquer la routine ALPHA, afin de convertir le numéro du raffinement référencé en une chaîne de caractères.
- Invoquer la routine IMPLI jusqu'à ce que l'on ait produit l'appel de procédure interne et le commentaire qui lui est associé.
  Le libellé du commentaire opérationnel qui doit figurer dans le commentaire de l'u.t.P. se trouve dans la variable " texte " de l'élément courant.
- Invoquer la routine LECT, afin de lire le premier élément de l'instruction suivante.

3.6.19. Développement de la routine LECT

3.6.19.1. Spécification

Appel à la routine LECT

Entrées :

- un élément courant.
  Rappelons que l'élément courant dans le fichier SPC est représenté par les variables raff, struc, instr, texte, pntr.
  Les différentes valeurs que peuvent prendre ces cinq variables ont été décrites à la sous-section 3.2.4.2. point (2);
- nolign, variable de type INTEGER, fait partie d'un common. Elle représente le nombre d'éléments pour lesquels la variable texte est non vide, jusqu'à et y compris l'élément courant dans un raffinement.

Sorties :

- l'élément courant devient le suivant de celui-ci;
- nolign garde la même signification;
- finins, variable de type LOGICAL.
  Valeurs possibles :
  - . true . lorsque l'on détecte les fins d'instructions, de structures de raffinements et la dernière instruction du raff 0,
  - . false . sinon;
- finstruc, variable de type LOGICAL.
  Valeurs possibles :
  - . true . lorsque l'on détecte les fins de structures, de raffinements et la dernière instruction du raff 0,
  - . false . sinon;

° finraf, variable de type LOGICAL.

Valeurs possibles :

- . true . lorsque l'on détecte la fin des raffinements et la dernière instruction du raff 0,
- . false . sinon;

° finfic, variable de type LOGICAL.

Valeurs possibles :

- . true . lorsque l'on détecte la dernière instruction du raff 0,
- . false . sinon;

Ces quatre variables font partie d'un common.

3.6.19.2. Procédé de construction

Il faut :

- Détecter la fin du raff 0
  (sous-objectif 1)

Sous-objectif 1 : détecter la fin du raff 0

- Si la variable " pntr (5) " = 0 (variable associée à l'élément courant) : il n'y a pas d'éléments suivant. Il faut, alors,
  - ° Donner à finfic, la valeur . true .
  - ° Donner à finraf, la valeur . true .
  - ° Donner à finstruc, la valeur . true .
  - ° Donner à finins, la valeur . true .
- Sinon, il faut :
  - ° Retenir le numéro du raffinement contenu dans la variable " raff " de l'élément courant.
  - ° Invoquer la routine LECT, afin de lire l'élément suivant, qui devient le nouvel élément courant.
  - ° Retenir la valeur de " pntr (5) " associé au nouvel élément courant.
  - ° Détecter la fin d'un raffinement.
    (sous-objectif 2)

Sous-objectif 2 : détectr la fin d'un raffinement

- Si " raff " associée à l'élément courant
  ≠ " raff " associée au nouvel élément courant, il faut :
  - ° Donner a finraf, la valeur . true .
  - ° Donner à finstrue, la valeur . true .
  - ° Donner à finins, la valeur . true .

- Sinon, il faut :
  - ° Compter le nombre de lignes dans le raffinement courant.
    (sous-objectif 3)
  - ° Détecter la fin d'une structure et d'une instruction.
    (sous-objectif 4)

sous-objectif 3 : compter le nombre de lignes dans le raffinement courant

- Si " instr " ≠ " sinon ", il faut :
  - ° incrémenter de 1 nolign, afin de reproduire sa valeur dans les messages associés aux erreurs de codage.
    La valeur de nolign doit représenter le nombre de lignes de code PASCAL dans un raffinement. Comme l'instruction SPC " sinon ", ne correspond à aucune ligne de code dans un raffinement, elle n'est pas comptée.

Sous-objectif 4 : détecter la fin d'une structure et d'une instruction

- Si (" struc " ≠ " ")
  et (" struc " ≠ " S " ou " instr " ≠ " - "), il faut :
  - ° Donner à finstruc, la valeur . true .
  - ° Donner à finins, la valeur . true .

- Sinon si
  " instr " ≠ " - ", il faut :
  - ° Donner à finins, la valeur . true .

3.6.20. Développement de la routine IMPLI

3.6.20.1. Spécification

Appel à la routine : IMPLI (tab, impres, long,
repet, finli, reduct

Entrées :

- tab, paramètre de type INTEGER, indique le nombre de blancs à insérer avant d'écrire le texte PASCAL;
- impres, tableau de LOGICAL * 1 de dimension égale à long, représente le texte à écrire dans le fichier;
- long, paramètre de type INTEGER, représente le nombre de caractères du texte contenu dans impres;
- repet, paramètre de type INTEGER, représente le nombre de fois qu'il faut écrire le texte contenu dans impres;
- finli, paramètre de type LOGICAL;
  Valeurs possibles :
  - . true . si ce que l'on écrit dans le fichier termine une ligne à l'écran,
  - . false . si l'on compte encore écrire sur la même ligne que celle où on vient d'écrire;
- réduct, paramètre de type LOGICAL
  Valeurs possibles :
  - . true . si l'on veut supprimer les blancs à la fin du texte contenu dans impres,
  - . false . sinon.

Sortie :

- l'impression, dans un fichier, du texte PASCAL en fonction des paramètres d'entrées.

3.6.20.2. Procédé de construction

Poure produire le texte contenu dans la variable impres, en fonction des paramètres d'entrées, les traitements à envisager sont les suivants :

- Mettre un blanc dans le tampon, jusqu'à ce que le nombre de blancs insérés soit égal à TAB
  - ° Si le tampon est plein pendant cette opération, il faut :
    - Vider le tampon dans un fichier.
    - Le blanchir.
    - Se repositionner au début du tampon.
- Si reduct vaut . true ., les deux traitements suivants sont à envisager jusqu'à ce qu'il n'y ait plus de blancs à supprimer dans la variable " texte " ou que long soit strictement inférieure à cinq.
  - ° Comparer les caractères du texte PASCAL cinq par cinq, en commençant par la fin.
  - ° Si ces cinq caractères sont des blancs, il faut diminuer la longueur du texte de cinq blancs.
- Si repet ≠ 0, il faut écrire le contenu de la variable impres autant de fois que repet le demande.
  - ° Si le tampon est plein pendant cette opération, les mêmes traitements que ceux décrits ci-dessus sont à envisager.
- Si finli = . true . et que l'on n'est pas passé à la ligne, il faut :
  - ° Ecrire le tampon.
  - ° Blanchir le tampon.
  - ° Se repositionner au début du tampon.

3.6.21. Développement de la routine IMPERR

3.6.21.1. Spcéfication

Appel à la routine IMPERR (LERR, NOMESS)

Entrées :

- l'élément courant est celui où figure l'erreur de codage;
- lerr, tableau de 72 éléments déclarés LOGICAL * 1, représente la variable " texte " associée à l'élément courant;
- nomess, paramètre de type INTEGER représente le numéro d'un message d'erreur
- nolign, variable de type INTEGER, appartient à un common.
  Cette variable représente le nombre d'éléments jusqu'à y compris l'élément courant dans le raffinement.

Sorties :

- Impression sur un fichier du message d'erreur assoicé à ce numéro;
- Impression de lerr;
- Impression du numéro du raffinement et du numéro de la ligne dans ce raffinement, où figure l'erreur de codage (nolign).

3.6.21.2. Procédé de construction

Afin de produire ce type de message d'erreur, les traitements à envisager sont les suivants :

- Ecrire le message associé à nomess.
- Ecrire le contenu de la variable lerr.
- Ecrie " raffinement numéro : ".
- Ecrire le numéro du raffinement.
  Ce numéro est contenu dans la variable " raff " associée à l'élément courant.
- Ecrire " ligne numéro ".
- Ecrire nolign.

3.6.22. Développement de la routine IMPECR

3.6.22.1. Spécification

Appel à la routine IMPECR (nomerr)

Entrées :
- ° nomerr, paramètre de type INTEGER représente le numéro d'un message d'erreur associé à une erreur fatale.

Sorties :
- ° affichage d'un message d'erreur associé à nomerr;
- ° affichage de " arrêt de la traduction ".

3.6.22.2. Procédé de construction

Afficher un message d'erreur revient à effectuer les traitements suivants :

- Initialiser l'écran.
- Ecrire le message associé à nomerr.
- Ecrire " arrêt de la traduction ".

3.6.23. Développement de la routine SURVOL

3.6.23.1. Spécification

Appel à la routine SURVOL (typrep, nbinst)

Entrées :
- l'élément courant est le premier élément d'une structure répétitive.

Sorties :
- typrep, paramètre de type INTEGER * 2.
  Valeurs possibles :
  - repeat, si la structure répétitive doit être traduite par un REPEAT ... UNTIL;
  - While, si la structure répétitive doit être traduite par un WHILE;
  - stand, si la structure répétitive doit être traduite selon une FORME STANDARD
    Les constantes repeat, while, stand, appartiennent à un common;
- nbinst, paramètre de type INTEGER, représente le nombre d'instructions (y compris les conditions de sortie) dans une structure répétitive devant être traduite par un while;
- l'élément courant reste inchangé.

3.6.23.2. Procédé de construction

Afin de déterminer la façon dont la structure répétitive doit être traduite, les traitements à effectuer sont les suivants :

- Sauver un numéro de ligne du raffinement courant.
  Cette ligne correspond au premier élément de la structure répétitive.
- Sauver la référence vers l'élément courant.
- Pour chaque instruction de la structure répétitive,

il faut :

- ° Incrémenter de 1 nbinst.
- ° Déterminer s'il s'agit d'une condition de sortie.
  (sous-objectif 1)
- ° Lire l'instruction suivante.
  (sous-objectif 2)
  jusqu'à ce que l'on ait détecté la fin d'une structure répétitive
  (sous-objectif 3)
  ou jusqu'à ce que l'on ait identifié plus d'une condition de sortie avec ou sans référence à un raffinement.
  (sous-objectif 4)

Sous-objectif 1 : déterminer s'il s'agit d'une condition de sortie

- Si la variable " instr " associée à l'élément courant = " S " (condition de sortie),

il faut :

- ° Incrémenter de 1 le nombre de condition de sortie.
- ° Mémoriser, dans la variable positc, la position dans la structure répétitive de cette condition de sortie.

Sous-objectif 2 : lire l'instruction suivante

il faut :

- Invoquer la routine LECT, jusqu'à ce que l'on obtienne le premier élément de l'instruction suivante.

Sous-objectif 3 : détection de la fin d'une structure répétitive

Quand on a détecté une fin de structure, il faut examiner positc.

- Si positc = 1
  il faut assigner à typrep la valeur while.
- Si positc = nbinst
  il faut assigner à typrep la valeur repeat.
- Sinon il faut assigner à tyrep la valeur stand.

Sous-objectif 4 : identification de plus d'une condition de sortie avec ou sans référence à un raffinement

Il faut assigner à typrep la valeur stand.

3.7. TESTS DES ROUTINES

Notre objectif n'est pas de présenter ici, une série de jeux de tests, mais plutôt de présenter la technique qui a été utilisée pour tester les différentes routines du codeur PASCAL. Les jeux de tests ont été établis à trois moments distincts de la réalisation du codeur : la spécification, la conception et la construction des différentes routines.
A chacun de ces moments correspond un type de jeux de tests : les tests " Black-box ", les tests " White-box " et les tests d'intégration.

(1) Tests " Black-box "

Les jeux de tests " Black-box " sont déterminés à partir des spécifications. Le choix des données tests est indépendant de l'algorithme associé à la routine.
Vu la structure des données, l'application stricte des principes des jeux de tests " Black-box " conduit à une redondance excessive des jeux de tests pour les routines de traduction.
Ainsi, par exemple, les routines TRTRAF et IDSTR ont des données de jeux de tests communes : les structures et instructions d'un programme SPC.
Un autre exemple est que le codeur PASCAL devrait reprendre entre autre les données tests associées à toutes les routines de traduction.
Cette redondance est intrinsèque au principe des jeux de tests " Black-box ". En effet, au vu d'une spécification, l'on ne peut déterminer ce qui sera effectivement réalisé par la routine de ce qui sera effectivement réalisé par d'autres routines.

(2) Tests " White-box "

Les jeux de tests " White-box " sont déterminés à partir de l'algorithme associé à une routine. Ces jeux de tests offrent moins de redondance que les précédants.

(3) Tests d'intégration

Ces jeux de tests sont déterminés lors de l'intégration des différentes routines.

(4) Application des tests

Les jeux de tests sont appliqués aux différentes routines. Les résultats obtenus sont alors comparés aux résultats attendus.
Pour les tests d'intégration, nous avons appliqué la démarche " bottom-up ". Nous avons donc testé, en premier lieu, les routines de " bas-niveau " : celles qui n'en appellent aucune. On teste ensuite les routines dans lesquelles les routines appelées sont déjà testées.

C H A P I T R E 4 :

EVALUATION CRITIQUE

4. EVALUATION CRITIQUE

La réalisation du codeur PASCAL nous a permis d'utiliser le logiciel Schémacode pendant plus de trois mois.
Nous avons pu, grâce à la connaissance que nous en avons acquise, dégager un certain nombre d'avantages mais aussi de faiblesses liées à ce logiciel.
Les critiques que nous allons apporter au logiciel Schémacode seront constructives. Elles nous aideront à élaborer, dans la seconde partie du mémoire, un système qui remédierait (en partie) aux faiblesses du système Schémacode.
Nous évaluerons également le produit réalisé et intégré à cet environnement.

4.1. EVALUATION DU SYSTEME UTILISE : LE LOGICIEL SCHEMACODE.

Celui-ci sera évalué en termes de
- L'objectif de Schémacode.
- Le pseudo-langage de Schémacode.
- La représentation interne des programmes SPC.
- Performance de l'outil.

4.1.1. L'objectif de Schémacode

(1) Cet outil encourage l'utilisateur à appliquer une démarche descendante de programmation.

(2) Il permet d'écrire des programmes SPC, plus lisibles par rapport à un programme écrit en FORTRAN.

(3) En temps qu'outil d'aide à la documentation, Schémacode permet d'insérer automatiquement au début de chaque raffinement du commentaire (le nom du raffinement). Il donne également à l'utilisateur la possibilité d'introduire du commentaire au sein du programme. Des commentaires pertinants permettent de refléter la démarche descendante suivie.

(4) L'objectif réel de Schémacode n'apparaît pas très clairement.

(1) Schémacode comme générateur de code.
L'intérêt habituel d'un générateur de code est qu'il permet de concevoir, valider et maintenir un programme dans un pseudo-langage de haut niveau. Dans cette optique, la structure de la version générée importe peu (elle devrait à la limite être invisible au programmeur), puisque c'est la version en pseudo-langage que l'on conçoit , valide et maintient.
Mais, cette optique n'est-elle pas un peu faussée, puisque

- SPC n'est pas réellement un pseudo-langage de haut niveau,
- Toutes les erreurs de compilation et d'exécution sont données sur le code généré (FORTRAN-PASCAL) et non sur le programme SPC.
  Le programmeur doit alors effectuer la démarche inverse du codeur : à partir d'une instruction dans le programme généré, découvrir sa position dans le programme Schémacode. L'utilisateur est donc obligé de maîtriser et de consulter le code généré.

(2) Schémacode comme producteur de programmes bien documentés.
Dans l'optique de Schémacode, la documentation d'un programme correspond aux commentaires plus ou moins pertinants que l'utilisateur y a insérés. Mais l'utilisateur n'est pas réellement forcé de documenter son programme SPC.
De plus, cette documentation ne concerne que la phase de programmation et non les différentes étapes précédant celles-ci (conception, spécification architecture).
L'outil Schémacode ne permet donc de fournir comme documentation d'un programme SPC, que le programme SPC lui-même avec éventuellement des commentaires !

Il semble donc que Schémacode ne réponde ni à l'objectif habituel d'un générateur de code, ni à l'objectif d'un producteur de programmes bien documentés.

(5) La notion de raffinement dans Schémacode est une notion floue, non définie, donc assez arbitraire. La compréhension d'un programme SPC peut parfois être rendue délicate, lorsque le nombre de raffinements est très élevé.

4.1.2. Le pseudo-langage de Schémacode

4.1.2.1. Avantages

(1) Le pseudo-langage SPC offre des structures qui n'existent pas en FORTRAN : la structure conditionnelle (if...then; if ... then ... else; if ... then ... else if... else if ... else) et la structure répétitive avec ou sans traitement d'exception.
Le programmeur FORTRAN ne doit plus gérer les GOTO et les étiquettes de ses programmes.
Ceci permet une mise au point plus aisée d'un programme FORTRAN.

(2) Le caractère graphique des structures SPC, permet d'avoir une vue très claire de l'ossature des raffinements. Ceci permet une compréhension plus aisée d'un programme SPC que d'un programme FORTRAN.

4.1.2.2. Inconvénients

(1) Le pseudo-langage SPC, qui permet de décrire un programme SPC, est en fait un langage hybride fait de la réunion de deux langages : le pseudo-code schématique (SPC) et le langage cible (FORTRAN ou PASCAL). SPC n'étant pas un vrai pseudo-langage, l'on peut dire que Schémacode ne possède pas un préprocesseur mais plutôt un éditeur de structures, couplé de codeurs.

(2) La syntaxe et la sémantique des instructions SPC, ne sont définies que par le langage de commande.

(3) La notion de délimiteur (fin de structure, fin de raffinement) fait partie du langage de commande et non du langage SPC.

(4) Le logiciel Schémacode ne permet pas l'imbrication des structures conditionnelles et répétitives.
Ainsi, par exemple, l'introduction d'une structure conditionnelle au sein d'une structure répétitive nécessite l'utilisation du mécanisme de référence à un raffinement. Ce raffinement contiendra la structure conditionnelle.
Cette contrainte entraîne le morcellement excessif d'un programme SPC.

(5) Pour des raisons d'implémentation de la représentation interne d'un programme SPC, Schémacode ne permet pas de référencer un même raffinement de plusieurs endroits différents dans le programme SPC.

(6) Le concept de procédure interne n'existe pas dans Schémacode. Il aurait pu être remplacé par la notion de raffinement avec paramètres.

(7) Le numéro d'un raffinement est imposé par le système Schémacode. Il est attribué par compostage et ne permet pas de refléter une structure arborescente. Il aurait été plus intéressant d'avoir une numérotation Rij où i désigne un numéro de niveau et j un numéro de raffinement dans ce niveau.

(8) Le nom du raffinement dans l'instruction SPC " condition de sortie avec référence à un raffinement ", est le libellé de la condition de sortie. Il aurait été plus pertinant, que le nom du raffinement référencé puisse être choisi par l'utilisateur.

4.1.3. La représentation interne des programmes SPC

(1) Dans cette représentation interne, la seule relation physiquement existante est la notion d'élément suivant indépendamment de toute notion de raffinement.
Ainsi, la notion de raffinement référencé par un raffinement donné est diluée dans l'ensemble du fichier contenant la représentation interne d'un programme SPC.
Pour pouvoir constituer la structure hiérarchique en niveaux, il faut alors parcourir tout ce fichier.
On pourrait envisager une représentation interne plus riche, telle qu'une structure d'arbre pour représenter les notions de raffinements, structures, instructions SPC.

(2) Dans cette représentation interne, aucune distinction n'est faite entre les instructions SPC et les instructions du langage cible.
La conséquence est qu'il faut effectuer un grand nombre de tests pour déterminer ce qui doit être traduit de ce qui doit être transcrit.

(3) La gestion des espaces libres du fichier SPC est inefficace. En effet, la taille du fichier SPC ne diminue jamais.
Tous les espaces libres correspondant à des instructions supprimées, restent chaînés dans SPC.
Ils sont utilisés lors de l'introduction de nouvelles instructions.

4.1.4. Performance de l'outil

L'éditeur Schémacode est très lent. Cette lenteur provient principalement de la technique d'affichage.
En effet, l'ensemble de l'écran est trop souvent affiché, alors qu'il n'y a aucune modification de celui-ci.

4.2. EVALUTATION DU PRODUIT REALISE : LE CODEUR PASCAL.

(1).Le caractère arbitraire du concept de raffinement et le fait que les imbrications de structures conditionnelles et répétitives ne sont pas permises, peuvent donner lieu à des procédures très fines.

.Le caractère morcellé du programme SPC est conservé dans le code PASCAL produit.

.En vue de faciliter à l'utilisateur la consultation du code produit, nous avons traduit un raffinement par une procédure.
Cette traduction est réalisée avec un souci de mise en page pour l'ensemble du programme PASCAL.

.Pour atténuer le caractère morcellé, les différentes procédures du programme PASCAL sont présentées par niveaux.

(2) Le concept d'instruction répétitive à plusieurs points de sortie, n'existe pas en PASCAL. Nous en avons proposé une traduction, qui n'est pas toujours élégante.

C H A P I T R E 5 :

EBAUCHE D'UN SYSTEME ALTERNATIF

5. EBAUCHE D'UN SYSTEME ALTERNATIF

5.1. INTRODUCTION

Au vu des avantages et inconvénients du système Schémacode, nous proposons, dans ce chapitre, un nouveau système d'aide à la programmation descendante.
Nos motivations sont de déterminer l'impact

- d'un pseudo-langage moins restrictif,
- d'une représentation interne sous forme d'arbre qui reflète la syntaxe de ce pseudo-langage,
- de l'intégration d'un éditeur syntaxique à ce système,

sur le travail de conception d'un nouveau codeur PASCAL.
Nous allons définir un pseudo-langage et sa représentation interne. Nous discuterons la possibilité d'intégrer à ce système un éditeur syntaxique.
Nous expliciterons ensuite, l'architecture d'un nouveau codeur PASCAL. Nous donnerons la technique de génération et les règles de traduction qu'il applique. Enfin, nous spécifierons chacun des modules de l'architecture, puis nous donnerons les procédés de construction et les algorithmes qui y sont associés.

5.2 DEFINITION D'UN PSEUDO-LANGAGE.

5.2.1 Introduction.

Au vu des divers avantages et inconvénients du pseudo-langage de Schémacode, nous proposons un pseudo-langage alternatif.

De Schémacode, nous conservons :

- la structure de l'instruction conditionnelle,
- l'idée des instructions répétitives à plusieurs conditions de sortie, avec traitement d'exception attaché à certaines de ces conditions de sortie.

Par contre, nous supprimons des restrictions telles que :

- l'interdiction d'imbriquer les instructions répétitives et conditionnelles,
- l'interdiction de référencer un raffinement en plusieurs points de programme,
- la limitation à neuf éléments dans chaque branche ou clause d'une conditionnelle et dans une répétitive,
- le numéro de raffinement imposé.

Nous ajoutons la notion d'instruction composée, les délimiteurs d'instructions et de raffinements au niveau des instructions et des raffinements.

Nous définissons donc un pseudo-langage "RAFCODE". Il porte sur les traitements uniquement. Il comprend les instructions suivantes :

- l'affectation,
- l'instruction composée,
- l'instruction conditionnelle,

- l'instruction répétitive,
- la référence à un raffinement,
- le commentaire.

Nous définissons ces instructions sous leur forme canonique. En annexe, le lecteur intéressé pourra trouver l'ensemble des règles formelles de type BNF qui définissent complètement le pseudo-langage.

5.2.2 Les symboles de base.

Les symboles de base sont ceux d'ALGOL 60; nous y avons ajouté quelques symboles spéciaux :

DO / ==> / // / OD / SI / ALORS / OUSI / SINON / FINSI /

RAFF / FAR / REF / %

5.2.3 Les constructions du pseudo-langage.

Classiquement, les constructions possibles d'un langage sont les déclarations, les expressions, les instructions et les programmes.

Nous ne traiterons pas les déclarations; par contre nous ajoutons un type de construction : le raffinement.

Les constructions du pseudo-langage définies sont donc

les expressions,
les instructions,
les raffinements,
les programmes.

(1) Les expressions

Les expressions de "RAFCODE" sont basées sur celles d'ALGOL 60.

(2) Les instructions

Une instruction du pseudo-langage est une des instructions suivantes :

- une affectation,
- une instruction composée,

- une instruction conditionnelle,
- une instruction répétitive,
- une référence à un raffinement,
- un commentaire.

(2.1) Affectation

Syntaxe : x := E

Sémantique : soit X, une variable,
E, une expression,

L'exécution de l'affectation se déroule comme suit : - on évalue E, soit v sa valeur,
- on affecte à la variable x la valeur v.

(2.2) Instruction composée

Syntaxe : ins1; ins2; ...; insn

Sémantique : soit ins1, ins2, ..., insn des instructions du pseudo-langage; l'instruction composée est une liste d'instructions qui doivent être exécutées dans l'ordre où elles apparaissent.

(2.3) Instruction conditionnelle

```
Syntaxe     : SI    cond1 ALORS ins1
              OUSI  cond2 ALORS ins2
              .
              .
              .
              OUSI  condi ALORS insi
```

```
.
.
.
OUSI condn ALORS insn
SINON ins(n + 1)
FINSI
```

Sémantique : soit cond1, cond2, ..., condn, des expressions logiques,

ins1, ins2, ..., ins(n + 1) des instructions,

L'exécution de l'instruction conditionnelle ci-dessus s'effectue comme suit :

Plaçons-nous dans une situation générale : on évalue condi, soit v, sa valeur;

si v = vrai, on exécute insi et l'exécution de l'instruction conditionnelle est terminée;

si v = faux, on évalue cond(i + 1) et on se retrouve dans la même situation qu'avec condi;

si, pour tout i, (i variant de 1 à n), condi a la valeur faux alors on exécute ins(n + 1) et l'exécution de l'instruction conditionnelle est terminée.

(2.4) Instruction répétitive

Syntaxe :

```
DO  ins0;
    cond1 ==> ins1 // exception 1;
    cond2 ==> ins2;
    .
    .
    .
    condi ==> insi // exception i;
```

```
        .
        .
        .
        condn ==> insn;
    OD
```

Sémantique : soit cond1, condi, ..., condn, des expressions logiques appelées conditions d'exécution, ins0, ins1, ..., insn, des instructions, exception 1, ..., exception n, des instructions.

L'exécution de l'instruction répétitive ci-dessus s'effectue comme suit :

on exécute ins0 si elle existe,
plaçons-nous dans une situation générale,
on évalue condi, soit v, sa valeur;
si v = faux, on exécute exception i si elle existe et l'exécution de l'instruction répétitive est terminée;

si v = vrai, on exécute insi si elle existe, on évalue cond(i + 1) et on se retrouve dans la même situation qu'avec condi.

Si, pour tout i (i variant de 1 à n), condi a la valeur vrai, alors on exécute à nouveau l'instruction répétitive.

Remarque : Pour tout i, insi et/ou exception i peuvent être l'instruction vide mais il existe au moins un j tel que insj soit différent de l'instruction vide.

(2.5) Référence à un raffinement

Syntaxe : REF ijk ; % nom du raffinement %

Sémantique : soit ijk le numéro du raffinement référencé.

L'exécution d'une référence à un raffinement consiste en un branchement vers le raffinement référencé; elle provoque l'exécution de ce raffinement.

(2.6) Commentaire

Syntaxe : % ceci est un commentaire %

Sémantique : un commentaire est un texte libre qui n'a aucun effet lors de l'exécution du programme dans lequel il apparaît.

Restriction : le commentaire ne peut contenir le caractère "%".

(3) Les raffinements

Syntaxe : RAFF ijk % nom du raffinement %;
inscomp FAR

Sémantique : soit ijk, le numéro du raffinement, inscomp, une instruction composée.

L'exécution d'un raffinement consiste à exécuter l'instruction composée qui forme le raffinement.

Pour tout raffinement d'un programme, sauf le Raff 0,

- l'exécution d'un raffinement est lancée par l'exécution de la référence à ce raffinement,
- la fin de l'exécution d'un raffinement provoque le branchement vers l'instruction qui suit la référence au raffinement exécuté.

Le Raff 0 est exécuté en premier lieu dès le lancement de l'exécution du programme et la fin de l'exécution du Raff 0 coïncide avec la fin de l'exécution du programme.

(4) Les programmes

Syntaxe : PROGRAM nom du programme; (déclaration)
r0; r1; r2; ...; rn.

Sémantique : soit r0, r1, ..., rn des raffinements.

L'exécution d'un programme consiste à exécuter le Raff 0 de ce programme.

Remarque : nous avons mis déclaration entre parenthèses car elles sont à cette place là mais nous ne les traitons pas dans le cadre de ce pseudo-langage.

5.3. LA REPRESENTATION INTERNE DE " RAFCODE ".

5.3.1. Notion de représentation interne

L'utilisateur introduit son programme à l'aide d'un éditeur. Au cours de l'opération d'édition, l'éditeur crée une représentation interne (RI) du programme. Cette représentation interne est la forme unique de manipulation de programmes par différents outils (codeurs, compilateurs, interpréteurs, formateurs ...).
D'après (KRA, 82) et (MOS, 82) les objectifs d'une représentation interne sont les suivants :

(1) Elle doit être équivalente au texte source; on doit pourvoir traduire un programme écrit dans un langage source dans sa représentation interne et réciproquement (aux modifications de mise en pages près).
Elle doit donc contenir toutes les informations du texte source soit explicitement, soit moyennant un calcul simple.

(2) Elle doit faciliter la manipulation et la modification interactives des programmes.

(3) Dans le cas qui nous occupe, elle doit faciliter le processus de traduction du codeur PASCAL.

5.3.2. Représentation interne arborescente

D'après (MOS, 82) le squelette d'une RI arborescente est un arbre dans lequel chacun des sommets correspond à une unité syntaxique et où les arcs dénotent les fils syntaxiques du sommet considéré. Ce type d'arbre est appelé arbre syntaxique.
La syntaxe du pseudo-langage " RAFCODE " a été définie à l'aide d'une grammaire BNF. Chacune des règles de cette grammaire peut être représentée sous la forme d'un arbre syntaxique.

A chaque noeud de l'arbre est associé un symbole non terminal de la grammaire. A chaque feuille est associé un symbole terminal de la grammaire. La RI d'un programme " RAFCODE " sera donc la représentation de la strucutre de cet arbre.
L'avantage d'un tel type de représentation est que la manipulatin d'un programme se fait en termes d'unités syntaxiques et non en termes de chaînes de caractères. Cela facilite l'analyse, la vérification, le codage, la conception, le formatage, le parcours d'un programme, ...

5.3.3. Interfaçage avec un analyseur syntaxique.

Avant de définir la RI des programmes " RAFCODE ", il convient d'envisager la possibilité d'introduire un analyseur syntaxique.
- Soit au niveau de l'édition d'un programme;
- Soit au niveau du codage de ce programme.

L'analyse syntaxique consiste à vérifier que le programme écrit en " RAFCODE " est syntaxiquement correct; c'est-à-dire que le programme respecte la syntaxe du pseudo-langage. A cet effet, l'analyseur essaie de déterminer l'arbre syntaxique associé au programme.

(1) Au niveau de l'édition d'un programme.

Au cours de l'introduction d'un programme; l'éditeur vérifie de façon interactive que ce programme est syntaxiquement correct et crée la RI sous forme arborescente de ce programme. Ce genre d'éditeur est appelé " éditeur syntaxique ". Un tel éditeur est donc guidé par la syntaxe du langage; l'objet élémentaire de manipulation est l'unité syntaxique et non la chaîne de caractères. Dès lors, la structure d'un programme est produite et analysée en termes d'unités syntaxiques. Mais un éditeur syntaxique fait généralement plus que de l'analyse syntaxique. A chaque unité syntaxique du langage est associé un modèle, appelé aussi " pattern ", reconnu et produit par l'éditeur.

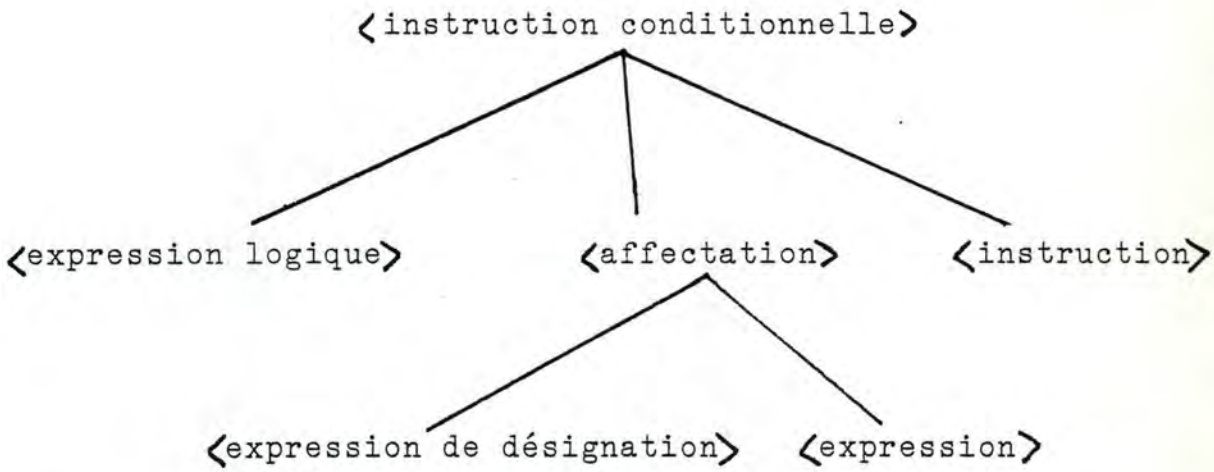
Un modèle est constitué de mots réservés et d'emplacements destinés à être remplis. L'utilisateur, après avoir sélectionné un modèle particulier, peut alors remplir chacun des emplacements de ce modèle par un nouveau modèle. L'éditeur vérifie la possibilité syntaxique de l'opération.
Chaque modèle sélectionné correspond à un noeud de l'arbre syntaxique. Chaque emplacement dans ce modèle correspond à un sous-arbre de ce noeud.
Prenons, par exemple, le cas d'un utilisateur qui veut introduire une instruction conditionnelle.
A la simple frappe de la touche adéquate, l'éditeur fait apparaître le modèle de l'instruction conditionnelle sous une forme syntaxiquement correcte :

SI <expression logique>
ALORS <instruction>
SINON <instruction>
FINSI

Si l'utilisateur veut remplir l'emplacement <instruction> associé au " ALORS " par une affectation; l'éditeur remplace l'emplacement <instruction> par le modèle de l'affectation et il renseigne l'utilisateur sur la nature des emplacements à remplir. On obtient alors :

SI <expression logique>
ALORS <expression de désignation> := <expression>
SINON <instruction>
FINSI

Ainsi, l'arbre syntaxique associé au dernier exemple se représente comme suit :



(2) Au niveau du codage d'un programme

Dans ce cas, l'analyseur syntaxique est inclus au préprocesseur. L'analyseur vérifie que la chaîne de caractères représentant un programme écrit dans le pseudo-langage, est syntaxiquement correcte. Si c'est le cas, le codage est effectué. Si ce n'est pas le cas, l'utilisateurdoit corriger les erreurs de son programme avant de relancer le codage.
Une différence fondamentale par rapport à la possibilité d'introduire un analyseur au niveau de l'édition est qu'ici, l'éditeur n'a pas besoin de créer une RI arborescente, il fournit au préprocesseur le programme sous forme d'une chaîne de caractères.

(3) Choix effectué

La possibilité retenue est celle d'introduire l'analyseur syntaxique au niveau de l'édition. Le codeur travaille alors à partir d'une RI syntaxiquement correcte; elle sera sous forme arborescente et facile à parcourir.
Le codeur n'aura donc pas de vérification à effectuer.
Ce choix présente l'avantage de fournir directement la RI d'un programme syntaxiquement correct, à des outils tel qu'un codeur, un compilateur, un interpréteur... et de permettre la mise au point d'un programme plus rapidement.

Grâce au fait que l'introduction d'un programme s'effectue par insertion de modèles correspondants aux instructions du pseudo-langage et que l'écriture des mots-clés et séparateurs est effectuée par l'éditeur, le programmeur est dégagé de préoccupations syntaxiques contraignantes. L'éditeur peut ainsi également servir de guide pour l'apprentissage progressif du pseudo-langage.
L'éditeur offre également des fonctions de désignation et d'édition puissantes. Les commandes d'édition admettent comme arguments des unités syntaxiques.
Il est possible de les désigner de plusieurs manières.
Une première manière consiste à désigner directement l'unité à éditer grâce à un curseur sur l'écran.

On peut également désigner l'unité à éditer par son nom. C'est le cas, par exemple, lorsque l'utilisateur demande à l'éditeur d'insérer une instruction conditionnelle dans son programme.

Les emplacements des modèles possèdent des noms : l'utilisateur peut ainsi demander à l'éditeur de se positionner sur la première expression logique de l'instruction conditionnelle ou sur la partie instruction de celle-ci.

Une dernière fonction de désignation consiste à se déplacer par rapport à l'unité syntaxique courante.
On passera, par exemple à l'unité précédente, suivante, à l'unité englobante ou encore au premier emplacement de l'unité courante.
Outre ces fonctions de désignation et d'édition, l'éditeur offre l'équivalent d'un éditeur de texte, (insérer, détruire, échanger des objets), à la différence près que les objects manipulés sont des unités syntaxiques (MOS, 82), (DRA, 82) .

5.3.4. Implémentation physique de " RAFCODE "

Des choix ont été effectués quant aux techniques utilisées pour représenter physiquement l'arbre syntaxique associé à un programme " RAFCODE ", à savoir :

- Pour les noeuds auxquels est associé un nombre fixe de fils, nous mettrons dans le noeud père, un pointeur vers chacun des fils;
- Pour les noeuds auxquels est associé un nombre variable de fils, nous mettrons dans le noeud père, un pointeur vers le premier fils. Les autres fils sont chaînés entre eux.

En général, pour chaque noeud de l'arbre, nous appliquerons une double solution :

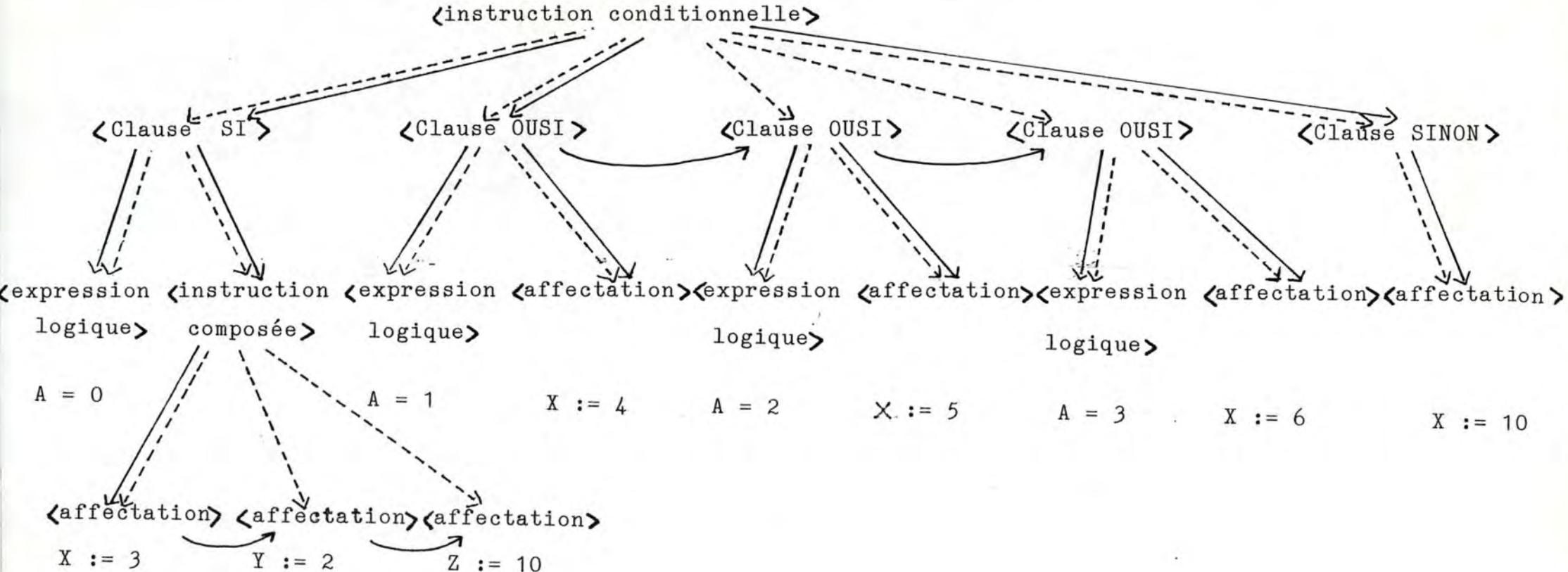
- Pour une partie des fils, ceux dont le nombre est fixe et connu, les pointeurs seront dans le noeud père;
- Les autres fils, ceux dont le nombre est variable, seront chaînés entre eux. Le noeud père possèdera un pointeur vers le premier de ses fils.

Prenons, par exemple, l'instruction conditionnelle suivante :

```
SI          A = 0
ALORS           X := 3;
                Y := 2;
                Z := 10;
OUSI        A = 1
ALORS           X := 4;
OUSI        A = 2
ALORS           X := 5;
OUSI        A = 3
ALORS           X := 6;
SINON           X := 0;
```

La représentation physique de l'arbre associé à cette instruction se présente comme suit :

Représentation physique de l'arbre associé à l'instruction conditionnelle :



° Les flèches en traits pleins représentent un pointeur

° Les flèches en traits pointillés représentent l'arbre syntaxique

5.3.5. Définition de la représentation interne de " RAFCODE "

A chaque noeud de l'arbre syntaxique associé à un programme " RAFCODE ", correspond un enregistrement PASCAL. Cet enregistrement diffère suivant la nature du noeud. De plus, un enregistrement particulier est défini pour la notion de niveau dans le programme PASCAL. Cette notion est équivalente à la notion de niveau définie pour le codeur de Schémacode.
Chaque enregistrement est composé de plusieurs champs. Chaque champ possède un nom et un type.
Les enregistrements ne comprenant que du texte sont des enregistrements de type TEXTE. Le texte associé à chacune des expression " RAFCODE ", au nom du raffinement, au nom du programme, aux instructions " RAFCODE " telles que l'affectation, et le commentaire, est représenté par une suite chaînée de blocs de caractères.
L'enregistrement de type TEXTE se définit comme suit :

INTEGER	PACKED ARRAY [1 .. 50] OF CHAR	↑TEXTE
LG	CONTENU	TSVT

Dans ce dessin,
chaque champ de l'enregistrement est représenté par une case. Le type du champ est noté à l'intérieur de la case, tandis que le nom du champ est indiqué en dessous de celle-ci.

LG : Nombre de caractères effectivement présents dans le champ CONTENU.

CONTENU : Champ contenant les caractères.

TSVT : Référence vers le bloc de caractères suivant, s'il existe.

Les autres enregistrements sont de type CONSTR, que nous allons définir ci-après.
Le premier champ de chacun de ces enregistrements porte le même nom : IDCONS. Il peut prendre les valeurs suivantes : PRG, NIV, RAFF, AFFECT, COMMENT, REF, COMPOS, CONDIT, SIOUSI, REPET, BLOC, EXCEPT. Cet ensemble de valeurs détermine le type IDENCONSTR.
La valeur de ce premier champ détermine le nom et le type des autres champs de l'enregistremet.
Comme pour les enregistrements de type TEXTE, chacun des champs d'un enregistrement de type CONSTR est représenté par une case. Son type est noté à l'intérieur de la case; son nom se trouve en dessous de celle-ci, tandis que la valeur du champ IDCONS est notée au dessus de la case qui lui est associée.
Décrivons à présent ces enregistrements :

(1) <u>enregistrement relatif au programme</u>

PRG

IDENCONSTR	↑TEXTE	↑CONSTR
IDCONS	NOMPRG	PREMNIV

NOMPRG : référence vers le nom du programme

PREMNIV : référence vers le niveau 1.

(2) <u>Enregistrement relatif à un niveau</u>

NIV

IDENCONSTR	INTEGER	↑CONSTR	↑CONSTR
IDCONS	NUMNIV	PREMRAF	SVTNIV

NUMNIV : Champ contenant le numéro du niveau.
PREMRAF : Référence vers le
premier raffinement de ce niveau.
SVTNIV : Référence vers le
niveau suivant dans le programme.

(3) enregistrement relatif à un raffinement

RAFF

IDENCONSTR	INTEGER	↑TEXTE	↑CONSTR	↑CONSTR
IDCONS	NUMRAF	NOMRAF	INSRAF	SVTRAF

NUMRAF : Champ contenant le numéro du raffinement.
NUMRAF : Référence vers le nom du raffinement.
INSRAF : Référence vers la première instruction de
ce raffinement.
SVTRAF : Référence vers le raffinement suivant dans le
même niveau.

Les enregistrements relatifs aux instructions

(1) L'affectation

AFFECT

IDENCONSTR	↑TEXTE	↑CONSTR
IDCONS	TEXAFF	SVTAFF

TEXAFF : Référence vers l'affectation.
SVTAFF : Référence vers
l'instructin suivante dans le raffinement.

(2) Le commentaire

COMMENT

IDENCONSTR	↑TEXTE	↑CONSTR
IDCONS	TEXCOM	SVTCOM

TEXCOM : Référence vers le commentaire.
SVTCOM : Référence vers l'instruction suivante dans le raffinement.

(3) La référence à un raffinement

REF

IDENCONSTR	INTEGER	↑CONSTR
IDCONS	NUMREF	SVTREF

NUMREF : Champ contenant le numéro du raffinement référencé.
SVTREF : Référence vers l'instruction suivante dans le raffinement.

(4) L'instruction composée

COMPOS

IDENCONSTR	↑CONSTR
IDCONS	PREMINS

PREMINS : Référence vers la première instruction de cette instruction composée.

(5) L'instruction conditionnelle

CONDIT

IDENCONSTR	↑CONSTR	↑CONSTR	↑CONSTR	↑CONSTR
IDCONS	INSSI	INSOUSI	INSINON	SVTCONDIT

INSI : référence vers la clause SI.
INSOUSI : référence vers la première clause OUSI.
INSINON : référence vers la clause SINON.
SVTCONDIT : référence vers l'instruction suivante dans le raffinement.

Enregistrement associé à la clause SI, ou à la clause OUSI

SIOUSI

IDENCONSTR	↑TEXTE	↑CONSTR	↑CONSTR
IDCONS	TEXCOND	INSALORS	SVTOUSI

TEXCOND : Référence vers l'expression.
INSALORS : Référence vers l'instruction composant la clause SI ou la clause OUSI.
SVTOUSI : Référence vers la clause OUSI suivante (si l'enregistrement est associé à une clause OUSI).

(6) L'instruction répétitive

REPET

IDENCONSTR	↑CONSTR	↑CONSTR	↑CONSTR
IDCONS	PREMINS	BLOCREP	SVTREP

PREMINS : Référence vers l'instruction d'en-tête de l'instruction répétitive.
BLOCREP : Référence vers le premier bloc répétitif de la structure.
SVTREP : Référence vers l'instruction suivante dans le raffinement.

Enregistrement associé à un bloc répétitif

BLOC

IDENCONSTR	↑TEXTE	↑CONSTR	↑CONSTR	↑CONSTR

IDCONS CONDR INSR EXCEPTION SVTBLOC

CONDR : Référence vers l'expression logique du bloc répétitif.
INSR : Référence vers l'instruction du bloc répétitif.
EXCEPTION : Référence vers l'instructin composant l'exception.
SVTBLOC : Référence vers le bloc répétitif suivant de l'instruction répétitive.

Les instructions telles que l'affectation, le commentaire, la référence à un raffineemnt et les expression de " RAFCODE " sont représentées dans la RI telles qu'elles apparaissent dans le pseudo-langage, sous forme de chaînes de caractères. Les feuilles de l'arbre associé à la RI d'un programme " RAFCODE " représentent donc ces instructins et expressions. Il n'est pas nécessaire d'aller plus en détails dans la décomposition de ces instructions puisque le processus de traduction consiste à transcrire ces instructions et expressions " RAFCODE " pour obtenir le code généré. S'il n'en était pas ainsi, le codeur devrait à partir des unités syntaxiques, reformer ces expressions et instructions avant de les recopier.
Concernant la possibilité d'introduire un éditeur syntaxique à ce système, nous prévoyons qu'une analyse syntaxique soit effectuée mais sans représenter l'arbre syntaxique associé. L'utilisateur introduirait donc les instructions d'affectation, le commentaire, la référence à un raffinement sous forme de texte plutôt que sous la forme de modèle. Une fois l'introduction d'une telle instruction terminée, l'éditeur vérifie la correction syntaxique de celle-ci et crée un enregistrement correspondant à cette instruction.

5.4. TECHNIQUE DE GENERATION ET LANGAGE CIBLE

Nous allons réaliser un codeur PASCAL qui appliquera la même technique de génération que celle qui a été décrite à la section 3.3.2.
Nous respecterons également la forme du code généré (du code PASCAL, structuré en niveaux, avec des procédures).
Réaliser un codeur PASCAL qui respecte les caractéristiques de l'ancien, nous permettra de dégager l'impact d'une RI arborescente sur la conception du générateur de code.

5.5. REGLES DE TRADUCTION

Ayant proposé un pseudo-langage " RAFCODE " alternatif à celui de Schémacode, nous proposons maintenant les règles de traduction à appliquer pour produire un programme PASCAL à partir d'un programme écrit en RAFCODE.
Rappelons que seules les constructions relatives aux traitements sont définies dans RAFCODE; le codeur dont nous faisons l'ébauche ne traduit donc que des structures de traitement : instructions, raffinements d'instructions et programmes sans partie " déclarations ".
Nous établissons une relation " TRADUIRE " notée " T " entre la forme canonique de chaque construction du pseudo-langage et sa forme PASCAL.

(1) Traduire un programme

```
PROGRAM  nom programme;
( déclarations )
R0; R1; R2; .....; Rr.
      |
      ↓ T
PROGRAM  nom programme;                    ] en-tête programme
( déclarations )
(* niveau 1 * )                           ] en-tête niveau
      T (R1)                               }
      T (R2)                               }
      :                                    } corps niveau
      T (Rk)                               }
                                           ] fin niveau
(* niveau 2*)
      T (R1)
      T (Rl+1)
      :
      T (Rm)
      :
(* niveau N *)
      T (Rp)
      :
      T (Rr)
(* PROGRAMME PRINCIPAL *)                  ] en-tête programme principal
BEGIN
      T (intruction du Raff 0)             ] corps programme principal
END.                                       ] fin programme principal
```

(2) Traduire un raffinement

```
RAFF    ijk  % nom du raffinement %;
Instruction composée

FAR
  |
  | T
  v
(* nom du raffinement *)                    ] nom procédure
PROCEDURE  Pijk;                            ] en-tête procédure
BEGIN                                       ]
T ( instruction composée )                  ] corps procédure
END;                                        ] fin procédure
```

(3) Traduire une instruction

(3.1.) L'instruction composée

```
ins 1;                          T ( ins 1);
ins 2;                          T ( ins 2);
:          ------T------>       :
ins n                           T ( ins n)
```

(3.2.) L'affectation

```
x : = E    ------T------>   x : = E
```

(3.3.) Le commentaire

```
% commentaire %  ------T------>  (* commentaire *)
```

(3.4.) La référence à un raffinement

```
REF  ijk; % nom du raffinement référencé %
  |
  | T
  v
(* nom du raffinement référencé *)   ] nom procédure
     Pijk                           ] appel procédure
```

(3.5.) L'instruction conditionnelle

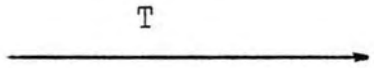
soit ins i une instruction composée;

```
SI          cond 1
            ALORS       ins 1
OUSI        cond 2
            ALORS       ins 2
:
OUSI        cond i
            ALORS       ins i
:
OUSI        cond n
            ALORS       ins n
SINON                   ins (n+1)
FINSI
```



```
IF   T (cond 1)                          ⎫ première alternative
THEN T ( ins 1)                          ⎭
ELSE   IF  T ( cond 2 )                  ⎫
       THEN T ( ins 2 )                  ⎭ seconde alternative
       :
       ELSE  IF   T (cond i )
             THEN   BEGIN
                    T (ins i )
                    END
       :
       ELSE  IF  T ( cond n )
             THEN T ( ins n)
             ELSE T ( ins n+1)           ⎫ dernière alternative
                                         ⎭
```

La condition

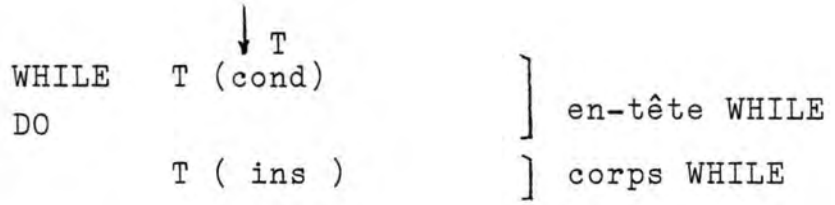


(3.6.) L'instruction répétitive

Selon la forme de la répétitive, elle sera traduite différemment.

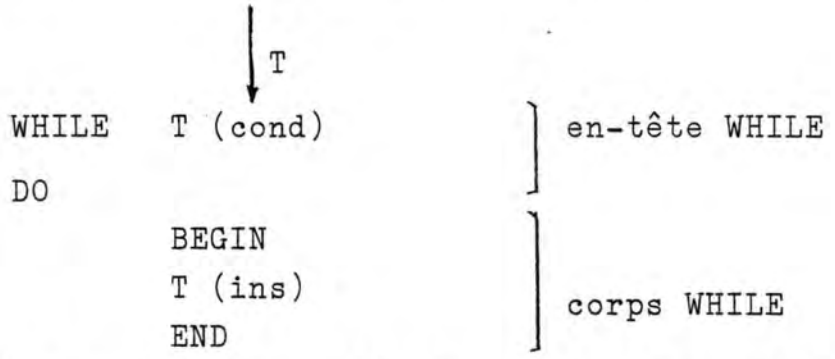
(3.6.1.) Les instructions répétitives n'ayant pas d'instruction en tête de la répétitive et composée d'un seul bloc répétitif sont traduites en un " WHILE " PASCAL.

° DO cond ==> ins; OD



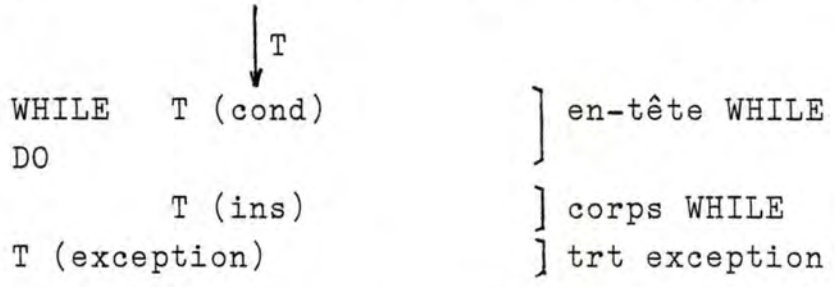
° Si ins est une instruction composée

DO cond ==> ins; OD



° Si un traitement est attaché à la non-réalisation de la condition d'exécution :

DO cond ==> ins // exception; OD



(3.6.2.) Les instructions répétitives ayant une instruction en tête de la répétitive et composée d'un seul bloc répétitif sans instruction sont traduites par un " REPEAT " PASCAL.

° DO ins;
cond; OD

↓ T

REPEAT] en-tête REPEAT
T (ins)] corps REPEAT
UNTIL T (cond)] fin REPEAT

° Si ins est une instruction composée et si un traitement d'exception est attahcé à la non-réalisation de la condition d'exécution :

DO ins;
cond // exception;
OD

↓ T

REPEAT] en-tête REPEAT
T (ins)] corps REPEAT
UNTIL T (cond)] fin REPEAT
T (exception)] trt exception REPEAT

(3.6.3.) Les instructions répétitives composées de plusieurs blocs répétitifs ou d'une instruction en tête de la répétitive et d'un bloc répétitif avec instruction sont traduites en une instruction PASCAL de type " REPEAT ... UNTIL " que nous appellerons répétitive standard.

La condition d'arrêt est constituée par une variable logique "LITNU" indicée par le numéro de raffinement dans lequel on se trouve et le niveau d'imbrication de l'instruction répétitive par rapport à une autre répétitive.

Supposons que :

- On se trouve dans le raffinement k,
- On se trouve au lème niveau d'imbrication d'instruction répétitive.

```
DO      ins 0;
        cond 1 ==> ins  1      // exception 1;
        :
        cond i ==> ins  i      // exception i;
        :
        cond n ==>ins n        //  exception n;
OD
        |T
LITNU   [k,l] : = TRUE;                    ]
REPEAT                                     ] en-tête standard
    T (ins 0);                             ] instruction d'en-tête standard
    IF   T (cond 1)
    THEN    BEGIN
              T (ins 1);
              :
              IF   T (cond i)
              THEN  BEGIN
                        T (ins i);
                        :
                        IF     T (cond n)
                        THEN     BEGIN
                                     T (ins n)
                                 END
                        ELSE     BEGIN
                                     T (exception n);
                                     LITNU [k,l] : = FALSE
                                 END
                    END
```

```
        :
        ELSE        BEGIN                          ⎫ traitement
                        T (exception i );          ⎪ exception
                        LITNU  [k,1]: = FALSE      ⎭ standard
                    END
    END

:
ELSE   BEGIN
         T (exception 1);
         LITNU [k,1] : = FALSE          ⎫ fin traitement
       END                              ⎪ exception
                                        ⎭ standard

UNTIL    LITNU [k,1] = FALSE            ⎫ fin répétition standard
```

(3.6.4.) Les conditions d'exécution des instructions répétitives sont simplement recopiées.

Les " exceptions " sont des instructions du pseudo-langage et sont traduites comme telles.

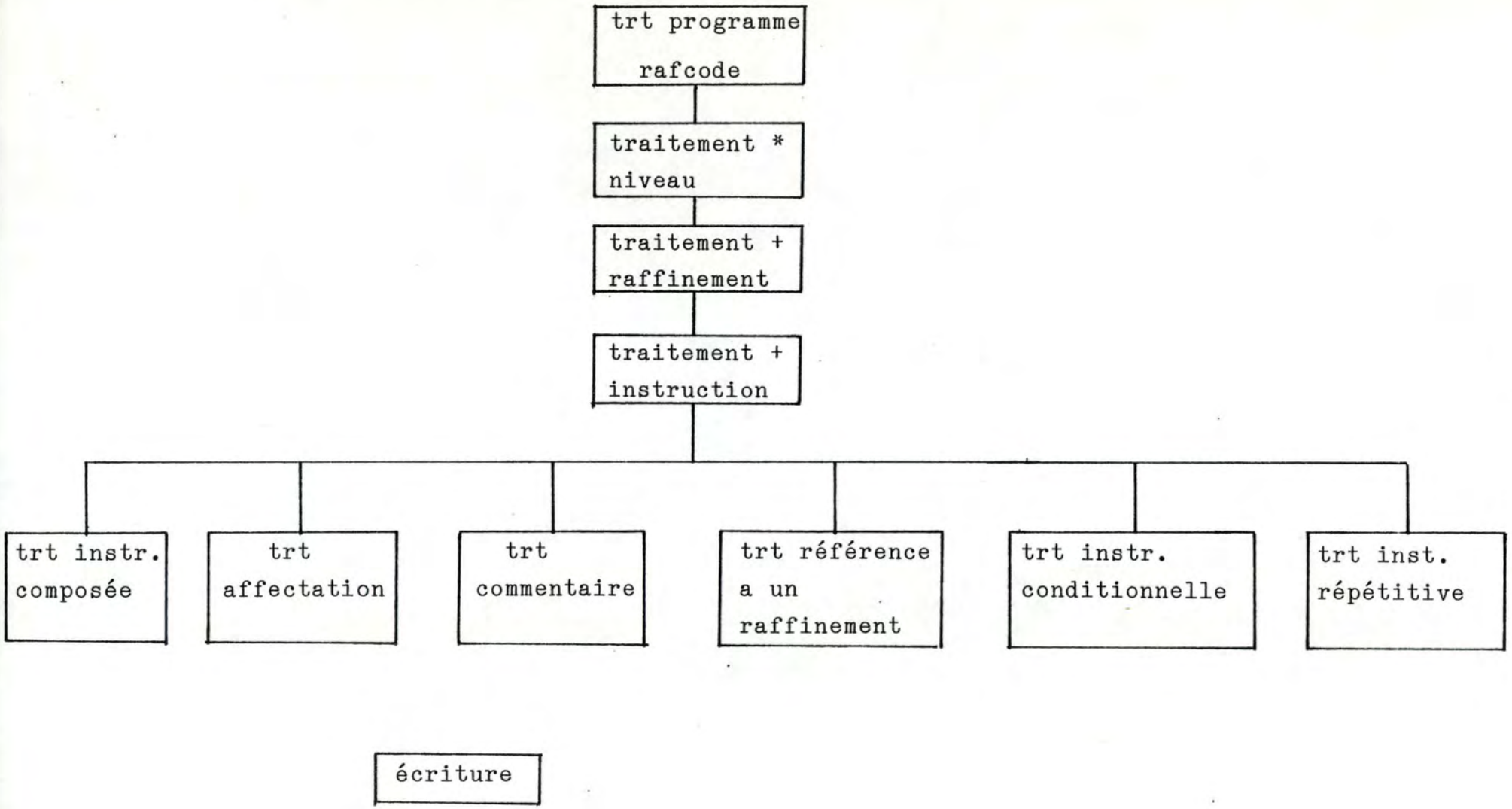
5.6. ARCHITECTURE D'UN CODEUR

A partir de la définition du pseudo-langage " RAFCODE " et des règles de traduction, nous en avons déduit l'architecture d'un codeur PASCAL.
(voir dessin page suivante)

A chaque module est associé une case. Chaque module identifie un traitement (pour l'interprétation de ce schéma, cfr la section 3.5.).
Rappelons que chaque traitement peut être répété

- Une astérisque (*) à l'intérieur de la case, indique que celui-ci peut être répété zéro, une ou plusieurs fois;
- Un plus (+) indique que le traitement peut être répété une ou plusieurs fois;
- La lettre O, symbolise le OU EXCLUSIF.

ARCHITECTURE DES TRAITEMENTS



5.7. SPECIFICATION DES MODULES ET PROCEDES DE CONSTRUCTION ASSOCIES

Pour chaque module de l'architecture de traitements, nous allons spécifier ses entrées et ses sorties, et construire en quelques lignes le traitement effectué par ce module.

Une fois pour toutes, nous précisons qu'en entrée, chaque module a le fichier contenant la représentation interne (RI) du programme à coder; chaque module a également accès aux " variables globales " par exemple le niveau d'indentation du code produit (noté NIVIND).

Les traitements sont construits en fonction des règles de traduction décrites à la section 5.5.
Dans un module, chaque traitement (" faire ceci ") qui nécessite l'exécution d'un autre module (XYZ) sera noté de façon standard : faire ceci (XYZ).
En annexe 5, on pourra trouver les algorithmes dérivés des spécifications et procédés de construction des modules.

(1) TRAITEMENT PROGRAMME RAFCODE

TRAITEMENT PROGRAMME RAFCODE est le module de tête du codeur. L'entrée et la sortie de ce module sont celles du codeur.

En entrée, le module a la référence au premier enregistrement, celui relatif au programme.

En sortie, TRAITEMENT PROGRAMME RAFCODE produit un programme PASCAL structuré en niveaux et mis en page.

TRAITEMENT PROGRAMME RAFCODE invoque les modules TRAITEMENT NIVEAU, TRAITEMENT INSTRUCTION et ECRITURE.

Procédé de construction associé

Pour obtenir un programme PASCAL à partir de la RI d'un programme RAFCODE, les traitements à envisager sont les suivants :

- Initialiser le niveau d'indentation NIVIND à 1.
- Produire l'en-tête du programme à partir de l'enregistrement relatif au programme (ECRITURE).
- Produire un niveau de procédures PASCAL pour chaque niveau (1 à N) de raffinements du programme RAFCODE (TRAITEMENT NIVEAU).
- Créer l'en-tête du programme principal (ECRITURE).
- Produire le corps du programme principal à partir du contenu du Raff 0 (TRAITEMENT INSTRUCTION).
- Créer la fin du programme principal (ECRITURE).

(2) TRAITEMENT NIVEAU

TRAITEMENT NIVEAU est le module de traitement des niveaux 1 à N du programme.

En entrée, TRAITEMENT NIVEAU a la référence à l'enregistrement relatif au niveau à traiter.

En sortie, TRAITEMENT NIVEAU produit un niveau du programme PASCAL mis en page.

TRAITEMENT NIVEAU invoque les modules suivants :
TRAITEMENT RAFFINEMENT et ECRITURE,
et est invoqué par le module TRAITEMENT PROGRAMME RAFCODE.

Procédé de construction associé.

Pour obtenir un niveau PASCAL à partir de la RI d'un niveau RAFCODE, il faut :
- Créer l'en-tête du niveau (ECRITURE).
- Produire une procédure PASCAL mise en page à partir de chaque raffinement du niveau RAFCODE (TRAITEMENT RAFFINEMENT).
- Créer la fin du niveau (ECRITURE).

(3) TRAITEMENT RAFFINEMENT

TRAITEMENT RAFFINEMENT est le module de traitement des raffinements 1 à R du programme.

En entrée, TRAITEMENT RAFFINEMENT a la référence à l'enregistrement relatif au raffinement à traiter.

En sortie, TRAITEMENT RAFFINEMENT produit la déclaration d'une procédure PASCAL mise en page.

TRAITEMENT RAFFINEMENT invoque les modules TRAITEMENT INSTRUCTION et ECRITURE, et est invoqué par le module TRAITEMENT NIVEAU.

Procédé de construction associé

Pour obtenir la déclaration d'une procédure PASCAL à partir de la RI d'un raffinement RAFCODE, les traitements à effectuer sont les suivants :

- Produire le nom de la procédure en commentaire à partir du nom du raffinement (ECRITURE).
- Produire l'en-tête de la procédure à partir du numéro de raffinement (ECRITURE).
- Produire le corps de la procédure à partir de l'instruction composée qui forme le raffinement (TRAITEMENT INSTRUCTION).
- Créer la fin de la procédure (ECRITURE).

(4) TRAITEMENT INSTRUCTION

TRAITEMENT INSTRUCTION est le module d'identification et de traitement des instructions.

En entrée, TRAITEMENT INSTRUCTION a la référence à l'enregistrement relatif à l'instruction à traiter.

En sortie, TRAITEMENT INSTRUCTION produit une instruction PASCAL mise en page et renvoie la valeur de la référenceà l'instruction suivante à traiter.

TRAITEMENT INSTRUCTION invoque les modules suivants : TRT AFFECTATION, TRT COMMENTAIRE, TRT INSTRUCTION COMPOSEE, TRT REFERENCE A UN RAFFINEMENT, TRT INSTR. CONDITIONNELLE, TRT INSTR. REPETITIVE, et est invoqué par les modules

TRAITEMENT PROGRAMME RAFCODE, TRAITEMENT RAFFINEMENT, TRT INSTR. COMPOSEE, TRT INSTR. REPETITIVE, TRT INSTR. CONDITIONNELLE.

Procédé de construction associé.

Pour produire une instruction PASCAL à partir de la RI d'une instruction RAFCODE, il faut :

- Identifier le type de l'instruction à traiter.
- Invoquer le module adéquat pour le codage de l'instruction.

(5) TRT INSTR. COMPOSEE

TRT INSTR. COMPOSEE est le module de traitement des instructions composées.

En entrée, le module a la référence à l'enregistrement relatif à l'instruction composée à traiter.

En sortie, TRT INSTR. COMPOSEE produit une instruction composée PASCAL mise en page.

TRT INSTR. COMPOSEE invoque les modules TRAITEMENT INSTRUCTION et ECRITURE
et est invoqué par le module TRAITEMENT INSTRUCTION.

Procédé de construction associé.

Pour produire une séquence d'instructions PASCAL à partir de la RI d'une instruction composée RAFCODE, on effectue les traitements suivants :

- Produire la première instruction PASCAL de la séquence à partir de la première instruction RAFCODE.
- Produire un point-virgule (ECRITURE)et produire une instruction PASCAL à partir de chacune des instructions suivantes dans l'instruction composée (TRAITEMENT INSTRUCTION).

(6) TRT AFFECTATION

TRT AFFECTATION est le module de traitement des affectations.

En entrée, TRT AFFECTATION a la référence à l'enregistrement relatif à l'affectation à traiter.

En sortie, TRT AFFECTATION produit une affectation PASCAL mise en page.

TRT AFFECTATION invoque le module ECRITURE, et est invoqué par le module TRAITEMENT INSTRUCTION.

Procédé de construction associé.

Pour produire l'affectation PASCAL, il suffit d'accéder au texte de l'affectation RAFCODE et de l'écrire (ECRITURE).

(7) TRT COMMENTAIRE

TRT COMMENTAIRE est le module de traitement des commentaires.

En entrée, TRT COMMENTAIRE a la référence à l'enregistrement relatif au commentaire à traiter.

En sortie TRT COMMENTAIRE produit un commentaire mis en page.

TRT COMMENTAIRE invoque le module ECRITURE, et est invoqué par le module TRAITEMENT INSTRUCTION.

Procédé de construction associé

Pour produire le commentaire, il suffit d'accéder au texte du commentaire RAFCODE et de l'écire en commentaire PASCAL (ECRITURE).

(8) TRT REFERENCE A UN RAFFINEMENT

Ce module traite les références à un raffinement.

En entrée, TRT REFER. a la référence à l'enregistrement relatif à la " référence à un raffinement " à traiter.

En sortie, TRT REFER. produit un appel de procédure conforme à la syntaxe PASCAL et un commentaire.

TRT REFER. invoque le module ECRITURE, et est invoqué par TRAITEMENT INSTRUCTION.

Procédé de construction associé

Pour produire en commentaire le nom de la procédure, TRT REFER. écrit en commentaire le nom du raffinement référencé (ECRITURE).
Pour produire l'appel de procédure, TRT REFER. accède au numéro de raffinement référencé et crée l'appel de procédure (ECRITURE).

(9) TRT INSTR. CONDITIONNELLE

Ce module traite les instructions conditionnelles.

En entrée, TRT CONDIT. a la référence à l'enregistrement relatif à l'instruction conditionnelle à traiter.

En sortie, TRT CONDIT. produit une instruction conditionnelle PASCAL mise en page.

TRT CONDIT. invoque les modules TRAITEMENT INSTRUCTION et ECRITURE,
et est invoqué par TRAITEMENT INSTRUCTION.

Procédé de construction associé

Pour produire une instruction conditionnelle PASCAL à partir de la RI d'une instruction conditionnelle RAFCODE, on procède comme suit :

- Accéder à la référence de la première alternative.
- Produire la première alternative de l'instruction conditionnelle.
- Pour chaque alternative suivante de la forme " OUSI ... ALORS ",
  - ° écrire " ELSE (ECRITURE),
  - ° incrémenter le niveau d'indentation de un,
  - ° produire l'alternative PASCAL correspondant.

- Produire la dernière alternative de l'instruction conditionnelle PASCAL à partir de l'alternative " SINON " de l'instruction RAFCODE, si cette alternative existe (TRAITEMENT INSTRUCTION).
- Décrémenter le niveau d'indentation du nombre de fois qu'il a été incrémenté.

On constate que produire la première alternative et produire une alternative suivante à partir d'un cas " OUSI... ALORS " revient à un même sous-objectif : il s'agit de produire une alternative PASCAL de type " IF... THEN ". Pour atteindre ce sous-objectif, on exécutera les traitements suivants :

- Produire " IF condition ", à partir du texte de la condition de l'alternative traitée (ECRIRE).
- Produire " THEN " et l'instruction consécutive à la condition de l'alternative traitée à partir de l'instruction de l'alternative RAFCODE traitée (TRAITEMENT INSTRUCTION).

(10) TRT INSTR. REPETITIVE

Ce module traite les instructions répétitives.

En entrée, TRT REPET. a la référence à l'enregistrement relatif à l'instruction répétitive à traiter.

En sortie, TRT REPET. produit une instruction répétitive PASCAL mise en page.

TRT REPET. invoque les modules suivants : TRAITEMENT INSTRUCTION et ECRITURE et est invoqué par TRAITEMENT INSTRUCTION.

Procédé de construction associé

TRT REPET. produit trois types d'instructions répétitives PASCAL selon la structure de l'instruction répétitive RAFCODE. Traiter une instruction répétitive consistera donc à :

- Déterminer la structure de la répétitive RAFCODE : sous-objectif 1.
- Selon le résultat de l'exécution du premier sous-objectif :
  - Produire une instruction PASCAL de type WHILE : sous-objectif 2,

ou

° Produire une instruction PASCAL de type REPEAT : sous-objectif 3,

ou

° Produire une instruction répétitive PASCAL que nous avons appelée " STANDARD" : sous-objectif 4.

Sous-objectif 1 :

Déterminer la structure de l'instruction répétitive à traiter. Ce sous-objectif est constitué d'un parcours de la RI arborescente de l'instruction répétitive.

- Une instruction répétitive ne comprenant qu'un seul bloc répétitif
  - ° sera de type " WHILE " s'il y a une instruction dans le bloc répétitif mais qu'il n'y a pas d'instruction en tête de la répétitive.
  - ° sera de type " REPEAT " s'il n'y a pas d'instruction dans le bloc répétitif mais qu'il y a une instruction en tête de la répétitive.
- Dans tous les autres cas, la répétitive sera de type " STANDARD ".

Sous-objectif 2 :

Produire une instruction PASCAL de type " WHILE " à partir d'une instruction répétitive RAFCODE (ayant un seul bloc répétitif avec instruction mais n'ayant pas d'instruction en tête de la répétitive) nécessite les traitements suivants :

- Produire l'en-tête du WHILE à partir du texte de la condition du seul bloc répétitif (ECRIRE).
- Produire le corps du WHILE à partir de l'instruction contenue dans le bloc répétitif (TRAITEMENT INSTRUCTION).

- Si le bloc répétitif contient un traitement d'exception, produire l'instruction PASCAL équivalente (TRAITEMENT INSTRUCTION).

Sous-objectif 3 :

Produire une instruction PASCAL de type " REPEAT " à partir d'une instruction répétitive RAFCODE (ayant un seul bloc répétitif sans instruction mais ayant une instruction en tête de la répétitive) nécessite les traitements suivants :

- Ecrire l'en-tête du REPEAT (ECRITURE).
- Produire le corps du REPEAT à partir de l'instruction en tête de la répétitive RAFCODE (TRAITEMENT INSTRUCTION).
- Produire la fin du REPEAT à partir de la condition du bloc répétitif (ECRIRE).
- Si le bloc répétitif contient un traitement d'exception, produire l'instruction PASCAL équivalente (TRAITEMENT INSTRUCTION).

Sous-objectif 4.

Produire une instruction répétitive PASCAL de type " STANDARD " à partir d'une instruction répétitive RAFCODE (ayant soit un seul bloc répétitif avec une instruction et une instruction en tête de la répétitive, soit au moins deux blocs répétitifs) nécessite les traitements suivants :

- Créer l'en-tête de la répétitive standard (ECRIRE).
- Incrémenter le niveau d'indentation de un.
- Produire un premier groupe d'instruction si la répétitive contient une instruction en tête (TRAITEMENT INSTRUCTION).
- Traiter les blocs répétitifs : sous-objectif 5
- Créer la fin de répétitive standard (ECRIRE).

Sous-objectif 5 :

Produire le corps de la répétitive standard à partir des blocs répétitifs RAFCODE est réalisé comme suit :

- Produire la condition à partir de la condition d'exécution RAFCODE du bloc répétitif traité (ECRIRE).
- Produire le " THEN " et l'instruction consécutive à la condition, à partir de l'instruction du bloc répétitif (TRAITEMENT INSTRUCTION).
- S'il y a un bloc répétitif suivant, le traiter, c'est-à-dire exécuter à nouveau le sous-objectif 5.
- Produire le traitement d'exception lié à la non-exécution de la condition d'exécution du bloc, à partir du cas d'exception du bloc répétitif traité (TRAITEMENT INSTRUCTION, ECRITURE).

(11) ECRITURE

Ce module effectue l'écriture sur fichier du code produit; cette écriture se fait par chaînes de caractères successives sur des lignes de 80 caractères.

En entrée, le " curseur " a une position déterminée,
ECRITURE reçoit en paramètre :
- la référence à la première chaîne de caractères du texte à écrire,
- un facteur de répétition,
- un indicateur de fin de ligne (faut-il aller à la ligne après avoir écrit le texte ?).

ECRITURE se sert de NIVIND.

En sortie, le texte est écrit et la position du " curseur " est soit la position suivant le dernier caractère écrit, soit la première position de la ligne suivante.

Procédé de construction associé

A partir d'où on était arrivé ou à partir de la position imposée par l'indentation, on écrit la première chaîne de caractères et les chaînes suivantes; on répète le texte autant de fois qu'il faut. Si le texte est trop long on va à la ligne.

C H A P I T R E 6 :

CONCLUSION

6. CONCLUSION

Nous tirerons tout d'abord une conclusion à propos de Schémacode, outil d'aide à la programmation descendante et ensuite une conclusion relative au produit réalisé, le codeur PASCAL.

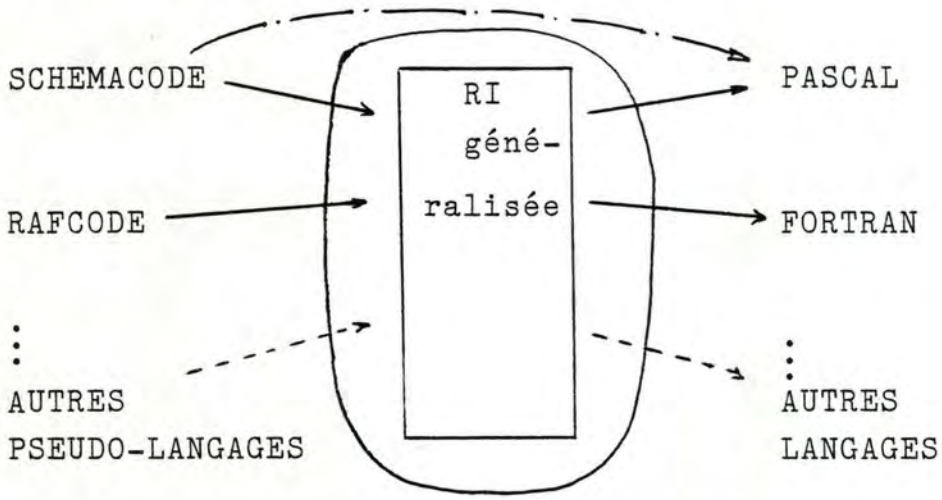
Après l'analyse détaillée de Schémacode, ses concepts, sa RI, son fonctionnement, peut-on dire que Schémacode est un logiciel d'aide à la programmation descendante ?
Le but de Schémacode est d'aider l'uilisateur à obtenir une structure de programme reflétant la structure de son raisonnement, de telle sorte que les programmes obtenus soient plus aisément communicables.
Cependant, l'interdiction d'imbriquer les structures est souvent une entrave pour l'utilisateur et a pour conséquence de fournir des programmes morcellés et peu lisibles.
Il est évident que Schémacode orienté FORTRAN représente un progrès : certaines structures SPC n'existent pas en tant que telles en FORTRAN. Le codage automatique permet de traduire ces structures SPC en instructions FORTRAN équivalentes et chaque type de structure est toujours traduit de la même façon, ce qui rend les programmes plus homogènes.
Quant à Schémacode orienté PASCAL, cela représente-t-il vraiment un bénéfice ? PASCAL contient les mêmes structures que Schémacode (à part la structure répétitive avec points de sortie multiples). L'apport principal est la structure du code en niveaux, qui reflète la structure de raisonnement.
Mais posons la question qui aurait dû être posée avant tout : est-ce un but utile et raisonnable que de vouloir produire un code reflétant la structure de raisonnement ?

Le codeur PASCAL créé pour Schémacode est très particulier; il ne pourra être utilisé dans aucun autre contexte.
Cela provient du fait qu'à partir d'un programme écrit en pseudocode Schématique, on produit directement le programme PASCAL équivalent.

La structure et les programmes du codeur sont tout à fait liés à Schémacode et à PASCAL.
Une autre solution aurait pu être de passer de la RI d'un programme SPC a une RI intermédiaire, compatible avec les structures PASCAL; cette RI aurait pu être l'intermédiaire entre plusieurs pseudo-langages et PASCAL. A la limite, on pourrait même imaginer une RI intermédiaire tout à fait généralisée, de telle sorte qu'à partir de n'importe quel pseudocode, on pourrait obtenir un programme écrit dans n'importe quel langage en deux étapes :

1 - Analyse et transformation du pseudo-langage en une RI généralisée;
2 - Codage : traduction de la RI généralisée dans le le langage choisi.

Dans le dessin qui suit, nous chématisons l'effet du codeur par le trait mixte et le résultat obtenu par une RI généralisée en traits pleins.



PREPROCESSEUR
UNIVERSEL

Le codeur réalisé pour Schémacode est compliqué. En effet la RI de Schémacode est peu adoptée aux traitements à effectuer. Pour cette raison, nous avons proposé une RI arborescente dans le cadre d'un système alternatif.
En gardant le même type de code généré on peut conclure que les traitements sont fort simplifiés grâce à la RI arborescente :

- La structure hiérarchique en niveaux est complètement décrite dans la RI; on ne doit plus ni créer la structure hiérarchique, ni en conserver la représentation. La RI rend disponibles les informations nécessaires : tous les niveaux d'un programme, tous les raffinements d'un niveau, le niveau d'un raffinement donné.
- Tous les problèmes d'accès aux entités syntaxiques sont résolus grâce aux parcours d'arbres; les routines de lecture et accès à la structure hiérarchique en niveau disparaissent (elles étaient au nombre de sept dans le codeur PASCAL).
  En cours de génération, on peut grâce à des primitives de manipulation d'arbres, produire toutes les informations nécessaires.
- Une vue globale des instructions et des raffinements permet de localiser le traitement des instructions et raffinements et la génération des mots réservés qui leur sont propres.
- L'architecture des traitements est largement simplifiée.

Une extension possible du pseudo-langage RAFCODE serait d'introduire la notion de raffinement avec paramètres. Comme pour Schémacode, on ne veut pas introduire la notion de procédure interne parallèlement à la notion du raffinement (cf section 3.3.3.). Or, tout le monde connait le grand intérêt des procédures avec paramètres; pour remplacer cette notion, on envisagerait plutôt d'introduire les raffinements avec paramètres.

BIBLIOGRAPHIE

(BER, 80) BERGLAND G.D.,

A Guided Tour of Program Design Methodologics,
IEEE Computer, October 1981, pp 13 à 37.

(DON, 79) DONZEAU-GOUGE V., HUET G., KAHN G., LANG B.,
Introduction au système MENTOR et à ses applications,

Actes des Journées Francophones sur la Certification du Logiciel, Genève, Janvier 1979.

(DON, 80) DONZEAU-GOUGE V., HUET G., KAHN G., LANG B.,
Programming Environments based on Structured Editors, The MENTOR Experience,

Rapport de Recherche n° 6, INRIA, juillet 1980.

(EST, 82) ESTUBLIERJ., KRAKOWIAK S., MOSSIERE J.,
ROUZAUD Y., Design Principles of the Adèle Programming Environment.

Soumis à la 6ème Conférence de Génie Logiciel, septembre 1982.

(FIC, 81) FICHEFET J.,
Modèles Mathématiques de Recherche opérationnelle,
Notes de cours, Namur 1981.

(JAC, 75) JACKSON M.A.,
Principles of Program Design,
Academic Press, 1975.

(KER, 81) KERNIGHAM B.W., PLAUGER P.J.,
Software Tools in Pascal,
Bell Telephone Laboratories Inc., Whitesmiths Ltd, 1981.

(KRA, 82) KRAKOWIAK S.,
Systèmes intégrés de production de logiciel : concepts et réalisations,
TSI, Volume I, n° 3, 1982, pp 127 à 200

(LEC, 80) LE CHARLIER B.,
Définition du langage LSD 80 et Spécifications des procédures de l'interpréteur du langage LSD 80
non publié, 1980.

(MOS, 82) MOSSIERES J., RAYMOND J., ROUZAUD Y.,
Représentation interne et manipulation des programmes dans l'atelier logiciel Adèle,
Colloque Génie Logiciel AFCET, Paris, Janvier 1982

(NAU, 62) NAUR P., éditeur
Revised Report on the Algorithmic Language ALGOL 60,
Communiqué ACM, 1962.

(PLA, 81) PLAMONDON R., ROBILLARD P.N.,
Harness a Computer to write Better Software, Faster,
Electronic Design, July 23, 1981.

(ROB, 82 a) ROBILLARD P.N., PLAMONDON R.,
Outils pour la création automatique de programmes structurés en FORTRAN,
Ecole Polytechnique de Montréal, juillet 1982.

(ROB, 82 b) ROBILLARD P.N., PLAMONDON R.,
Schémacode User's Guide,
Ecole Polytechnique de Montréal, July 19, 1982.

(TRE, 79) TREMBLAY J.P., SORENSON P.G.,
An Introduction to Data Structures with Applications,
International Student edition, Mc Graw Hill, 1979.

(VLA, 82) VAN LAMSWEERDE A.,
Les outils d'aide au développement de logiciels: un aperçu des tendances actuelles,
15èmes Journées internationales de l'informatique et de l'automatisme, Paris, juin 1982.

(WIR, 76) WIRTH N.,
Algorithms + Data Structures = Programs,
Prentice-Hall, New Jersey, 1976.

Facultés Universitaires N.D. de la Paix - NAMUR

INSTITUT D'INFORMATIQUE

SYSTEME D'AIDE

A LA

PROGRAMMATION DESCENDANTE

ANNEXES

ISABELLE LEPAGE

THERESE PAUWEN

Mémoire présenté
en vue de l'obtention du grade de
LICENCIE ET MAITRE EN INFORMATIQUE

année académique 1982-1983

A N N E X E 1 :

EXEMPLE DE TRADUCTION

TRADUCTIONS DE L'EXEMPLE. (section 3.2.1)

(1) REMPLACEMENT DE TEXTE.

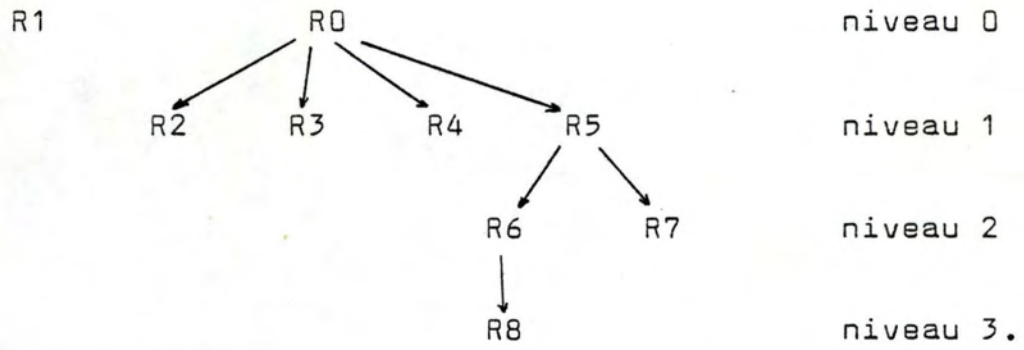
```
PROGRAM  MCSCHEM (DONNEES, OUTPUT);
(* programme de mise à jour de l'inventaire d'un entrepôt *)
(* d'aliments. Cet entrepôt reçoit (R) ou distribue (D) *)
(* des aliments. Chaque transaction est enregistrée sur une *)
(* carte perforée contenant comme information : *)
(*       le nom de l'aliment *)
(*       le type de transaction *)
(*       la quantité affectée *)
(* l'ensemble des carte est trié en ordre alphabétique sur *)
(* le nom des aliments *)     *)
(* le rapport de gestion est imprimé toutes les semaines. *)
(*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * )

(* DECLARATION *)
(* ----------- *)
(* NOMARTICLE   : nom de l'article groupe courant *)
(* CMPTGR       : nombre de groupes rencontrés *)
(* NETCHG       : changement net du groupe courant *)
(* DONNEES      : nom du fichier des données *)
(* ITEM         : transaction courante *)
(* GROUPE       : nom de l'article de la transaction courante *)
(* RD           : type de la transaction *)
(* QTE          : quantité affectée de la transaction courante *)
```

```
TYPE
      TRANSACTION = RECORD
                       GROUPE : PACKED ARRAY  1..4  OF CHAR;
                       RD : CHAR;
                       QTE : INTEGER
                    END;
      FICHIN = FILE OF TRANSACTION;
VAR   DONNEES : FICHIN;
      ITEM : TRANSACTION;
      CMPTGR, NETCHG : INTEGER;
      NOM ARTICLE  : PACKED ARRAY  1..4  OF CHAR;

BEGIN

(* INITIALISATION *)
(* ------------- *)
(* initialisation du nombre de groupes rencontrés *)
CMPTGR : = 0;
(* lecture de la première transaction *)
RESET (DONNEES);
ITEM : = DONNEES↑ ;
(* IMPRESSION EN-TETE *)
(* ----------------- *)
WRITELN; WRITELN ('' : 28, 'RAPPORT DE L''INVENTAIRE');
WRITELN ('' : 27, '*' : 25);
WRITELN;
WRITELN ('' : 10, 'ARTICLES', '' : 10, 'CHANGEMENT NET');

(* IMPRESSION DU CORPS *)
(* ------------------ *)
WHILE NOT (EOF)
DO BEGIN
     (* initialisation du nouveau groupe *)
      NOM ARTICLE  : = ITEM.GROUPE;
      NETCHG : = 0;
```

```
      (* CALCUL CHANGEMENT NET DU GROUPE *)
      (* ------------------------------ *)
       WHILE NOT (ITEM.GROUPE <> NOMARTICLE)
       DO BEGIN

          (* CALCUL QUANTITE AFFECTEE *)
           IF RD = 'D' THEN ITEM.QTE : = - ITEM.QTE;
          (* mettre à jour le changement net *)
           NETCHG : = NETCHG + ITEM.QTE;
          (* lecture d'une nouvelle transaction *)
           GET (DONNEES);
           ITEM : = DONNEES↑ ;
           END;
       (* IMPRESSION LIGNE DU GROUPE *)
        WRITELN ('' : 12, ITEM.GROUPE, '' : 17, NETCHG);
       (* modification du nombre de groupes rencontrés *)
        CMPTGR : = CMPTGR + 1;
   END;

(* IMPRESSION RESUME *)
(* ----------------- *)
WRITELN; WRITELN ('' : 4, '-' : 42);
WRITELN;
WRITELN ('' : 4, 'NOMBRE DE GROUPES MODIFIES = ', CMPTGR);
WRITELN;
STOP
END.
```

(2) STRUCTURE DU CODE EN NIVEAUX.

L'arbre des raffinements de l'exemple se présente comme suit :



(2.1) Structure du code en niveaux avec GOTO.

```
PROGRAM MCSCHEM (DONNEES, OUTPUT);
(* programme de mise à jour de l'inventaire d'un entrepôt *)
(* d'aliments. Cet entrepôt reçoit (R) ou distribue (D) *)
(* des aliments. Chaque transaction est enregistrée sur une *)
(* carte perforée contenant comme information : *)
(*       le nom de l'aliment *)
(*       le type de transaction *)
(*       la quantité affectée *)
(* l'ensemble des cartes est trié en ordre alphabétique sur *)
(* le nom des aliments *)
(* le rapport de gestion est imprimé toutes les semaines. *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* DECLARATION *)
(* ----------- *)
(* NOMARTICLE : nom de l'article groupe courant *)
(* CMPTGR     : nombre de groupes rencontrés *)
(* NETCHG     : changement net du groupe courant *)
(* DONNEES    : nom du fichier des données *)
(* ITEM       : transaction courante *)
(* GROUPE     : nom de l'article de la transaction courante *)
(* RD         : type de la transaction *)
(* QTE        : quantité affectée de la transaction courante *)
```

```
DECLARATION DES ETIQUETTES

TYPE
     TRANSACTION = RECORD
                     GROUPE : PACKED ARRAY  1..4  OF CHAR;
                     RD : CHAR;
                     QTE : INTEGER
                   END;
     FICHIN = FILE OF TRANSACTION;
VAR  DONNEES : FICHIN;
     ITEM : TRANSACTION;
     CMPTGR, NETCHG : INTEGER;
     NOMARTICLE : PACKED ARRAY  1..4  OF CHAR;

     BEGIN
     (* INITIALISATION NIVEAU 1, § 1 *)
     GOTO 1110
0111 (* IMPRESSION EN-TETE  NIVEAU 1, § 2 *)
     GOTO 1210
0121 (* IMPRESSION DU CORPS  NIVEAU1,§ 3 *)
     GOTO 1310
0131 (* IMPRESSION RESUME  NIVEAU 1, § 4 *)
     GOTO 1410
0141 STOP
     END.
```

```
     (* NIVEAU 1 *)
     (***********)
1110 (* §1 INITIALISATION *)
     (* ---------------- *)
     (* initialisation du nombre de groupes rencontrés *)
     CMPTGR : = 0;
     (* lecture de la première transaction *)
     RESET (DONNEES);
     ITEM : = DONNEES↑;
     GOTO 0111;

1210 (* §2 IMPRESSION EN-TETE *)
     (* -------------------- *)
     WRITELN; WRITELN ('' : 28, 'RAPPORT DE L"INVENTAIRE');
     WRITELN ('' : 27, '*' : 25);
     WRITELN;
     WRITELN ('', 'ARTICLES', '' : 10, 'CHANGEMENT NET');
     GOTO 0121

1310 (§3 IMPRESSION DU CORPS *)
     (* ------------------- *)
     WHILE NOT (EOF)
     DO BEGIN
          (* initialisation du nouveau groupe *)
           NOMARTICLE  : = ITEM.GROUPE;
           NETCHG : = 0;

           (* CALCUL DU CHANGEMENT NET DU GROUPE NIVEAU 2, §1 *)
            GOTO 2110;

1321       (* IMPRESSION LIGNE DU GROUPE NIVEAU 2, §2 *)
            GOTO 2210;

1331       (* modification du nombre de groupes rencontrés *)
```

```
            CMPTGR : = CMPTGR + 1;
        END;
    GOTO 0131;
1410 (* §4 IMPRESSION RESUME *)
    (* ------------------- *)
    WRITELN; WRITELN ('' : 4, '-' : 42);
    WRITELN;
    WRITELN ('' : 4, 'NOMBRE DE GROUPES MODIFIES = ', CMPTGR);
    WRITELN;
    GOTO 0141;

    (* NIVEAU 2 *)
    (**********)
2110 (* §1 CALCUL CHANGEMENT NET DU GROUPE *)
    (* ---------------------------------- *)
    WHILE NOT (ITEM GROUPE <> NOMARTICLE)
    DO BEGIN
       (* CALCUL QUANTITE AFFECTEE NIVEAU 3, § 1 *)
        GOTO 3110

2121    (* mettre à jour le changement net *)
        NETCHG : = NETCHG + ITEM.QTE;
       (* lecture d'une nouvelle transaction *)
        GET (DONNEES);
        ITEM : = DONNEES ↑ ;
       END;
    GOTO 1321

2210 (* §2 IMPRESSION LIGNE DU GROUPE *)
    (* ---------------------------- *)
    WRITELN ('' : 12, ITEM.GROUPE, '' : 17, NETCHG);
    GOTO 1331
```

```
      (* NIVEAU 3 *)
      (************)

3110  (* §1 CALCUL QUANTITE AFFECTEE *)
      (* -------------------------- *)
      IF RD = 'D' THEN ITEM.QTE = - ITEM.QTE;
      GOTO 2121;
```

Remarque.
Nous n'avons pas traité la déclaration des étiquettes; ce serait sans doute un des points délicats dans la génération de code structuré en niveaux avec GOTO.

(2.2) Structure du code en niveaux avec Procédures.

```
PROGRAM MCSCHEM (DONNEES, OUTPUT);
     (* DECLARATION *)
     (* ----------- *)
     (* NOMARTICLE : nom de l'article groupe courant *)
     (* CMPTGR     : nombre de groupes rencontrés *)
     (* NETCHG     : changement net du groupe courant *)
     (* DONNEES    : nom du fichier des données *)
     (* ITEM       : transaction courante *)
     (* GROUPE     : nom de l'article de la transaction courante *)
     (* RD         : type de la transaction *)
     (* QTE        : quantité affectée de la transaction courante *

TYPE
     TRANSACTION = RECORD
                     GROUPE : PACKED ARRAY  1..4  OF CHAR;
                     RD : CHAR;
                     QTE : INTEGER
                   END;
     FICHIN = FILE OF TRANSACTION;
VAR  DONNEES : FICHIN;
     ITEM : TRANSACTION ;
     CMPTGR , NETCHG : INTEGER;
      NOMARTICLE : PACKED ARRAY  1..4  OF CHAR;
     (* LITNU : VARIABLE GENEREE PAR LE CODEUR *)

        LITNU : ARRAY  1..8  OF BOOLEAN;
```

```
(* DECLARATION DES PROCEDURES *)
(* ------------------------- *)
(* Initialisation *)
P2; FORWARD;
(* Impression en-tête *)
P3; FORWARD;
(* Impression du corps *)
P4; FORWARD;
(* Impression résumé *)
P5; FORWARD;
(* Calcul changement net du groupe *)
P6; FORWARD;
(* Impression ligne du groupe *)
P7; FORWARD;
(* Calcul quantité affectée *)
P8; FORWARD;
```

```
(* NIVEAU 1 *)
(* -------- *)
(* INITIALISATION *)
PROCEDURE P2;
BEGIN
   (* initialisation du nombre de groupes rencontrés *)
   CMPTGR : = 0;
   (* lecture de la première transaction *)
   RESET (DONNEES);
   ITEM : = DONNEES↑ ;
END;

(* IMPRESSION EN-TETE *)
PROCEDURE P3;
BEGIN
   WRITELN; WRITELN ('' : 28, 'RAPPORT DE L"INVENTAIRE');
   WRITELN ('' : 27, '*' 25);
   WRITELN;
   WRITELN ('' : 10, 'ARTICLES', '' : 10, 'CHANGEMENT NET');
END;

(* IMPRESSION DU CORPS *)
PROCEDURE P4;
BEGIN
   WHILE NOT (EOF)
   DO BEGIN
         (* initialisation du nouveau groupe *)
         NOMARTICLE : = ITEM.GROUPE;
         NETCHG : = 0;
         P6; (* niveau 2
         calcul du changement net du groupe *)
         P7; (* niveau 2
         impression ligne du groupe *)
         (* modification du nombre de groupes rencontrés *)
         CMPTGR : = CMPTGR + 1;
         END ;
END;
```

```
(* IMPRESSION RESUME *)
PROCEDURE P5;
BEGIN
   WRITELN; WRITELN ('' : 4, '-' : 42);
   WRITELN;
   WRITELN ('' : 4, 'NOMBRE DE GROUPES MODIFIES = ', CMPTGR);
   WRITELN;
END;

(* NIVEAU 2 *)
(* -------- *)
(* CALCUL CHANGEMENT NET DU GROUPE *)
PROCEDURE P6;
BEGIN
   WHILE NOT (ITEM.GROUPE <> NOMARTICLE)
   DO BEGIN
      P8; (* niveau 3
      calcul quantité affectée *)
      (* mettre à jour le changement net *)
      NETCHG : = NETCHG + ITEM.QTE;
      (* lecture d'une nouvelle transaction *)
      GET (DONNEES);
      ITEM : = DONNEES ↑ ;
      END;
END;

(* IMPRESSION LIGNE DU GROUPE *)
PROCEDURE P7;
BEGIN
   WRITELN ('' : 12, ITEM.GROUPE, '' : 17, NETCHG );
END;
```

```
(* niveau 3 *)
(* -------- *)
(* calcul quantité affectée *)
PROCEDURE P8;
BEGIN
   IF RD = 'D' THEN ITEM.QTE : =-ITEM.QTE;
END;

(* PROGRAMME PRINCIPAL *)
(* ------------------- *)
BEGIN
(* Programme de mise-à-jour de l'inventaire d'un entrepôt *)
(* d'aliments. Cet entrepôt reçoit (R) ou distribue (D) *)
(* des aliments. Chaque transaction est enregistrée sur *)
(* une carte perforée contenant comme information :
(*       le nom de l'aliment
(*       le type de transaction
(*       la quantité affectée
(* L'ensemble des cartes est trié en ordre alphabétique
(* sur le nom des aliments.
(* Le rapport de gestion est imprimé toutes les semaines.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

P2; (* niveau 1
initialisation *)
P3; (niveau 1
impression en-tête *)
P4; (* niveau 1
impression du corps *)
P5; (* niveau 1
impression résumé *)
STOP
END.
```

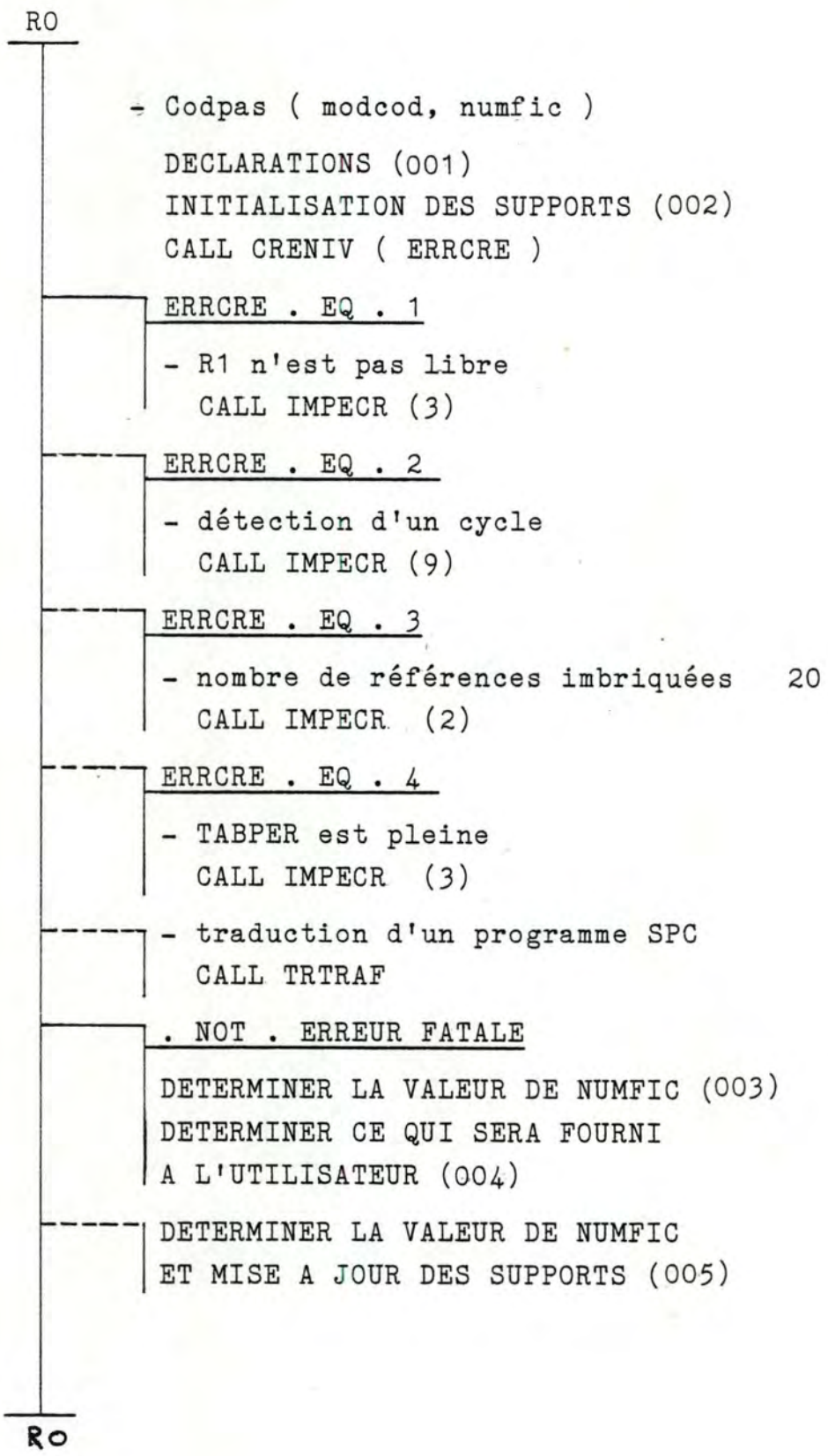
A N N E X E 2 :

CONCEPTION DES ROUTINES DU CODEUR REALISE POUR SCHEMACODE

2. CONCEPTION DES ROUTINES DU CODEUR REALISE POUR SCHEMACODE.

Nous présentons, ici, les algorithmes associés aux routines du codeur PASCAL.
Sur demande de MONTREAL, ils sont écrits dans le pseudo-code (langage SPC et FORTRAN) de Schémacode.

1. Conception de la routine CODPAS



R2

- Initialisation des supports
  REWIND 14
  REWIND 7
- initialisation de l'écran
  CALL CRLF

R0

R3

- Déterminer la valeur de numfic

ERRCOD . NE . 0

- Il y a des erreurs de codage
  WRITE (7,11)
  11 FORMAT (70 X)
  WRITE (7,12)
  12 FORMAT (1x, I3, ERREURS DETECTEES)
  NUMFIC = 7
  REWIND 14
  ENDFILE 14

- il n'y a pas d'erreur de codage
  NUMFIC = 14

R0

R4

- Déterminer ce qui sera fourni à l'utilisateur

MODCOD . EQ . 1 . OR . ERRCOD . NE . 0

- Affichier le contenu du fichier
  de numéro numfic

MODCOD . EQ . 2

- Imprimer le fichier numéro 14

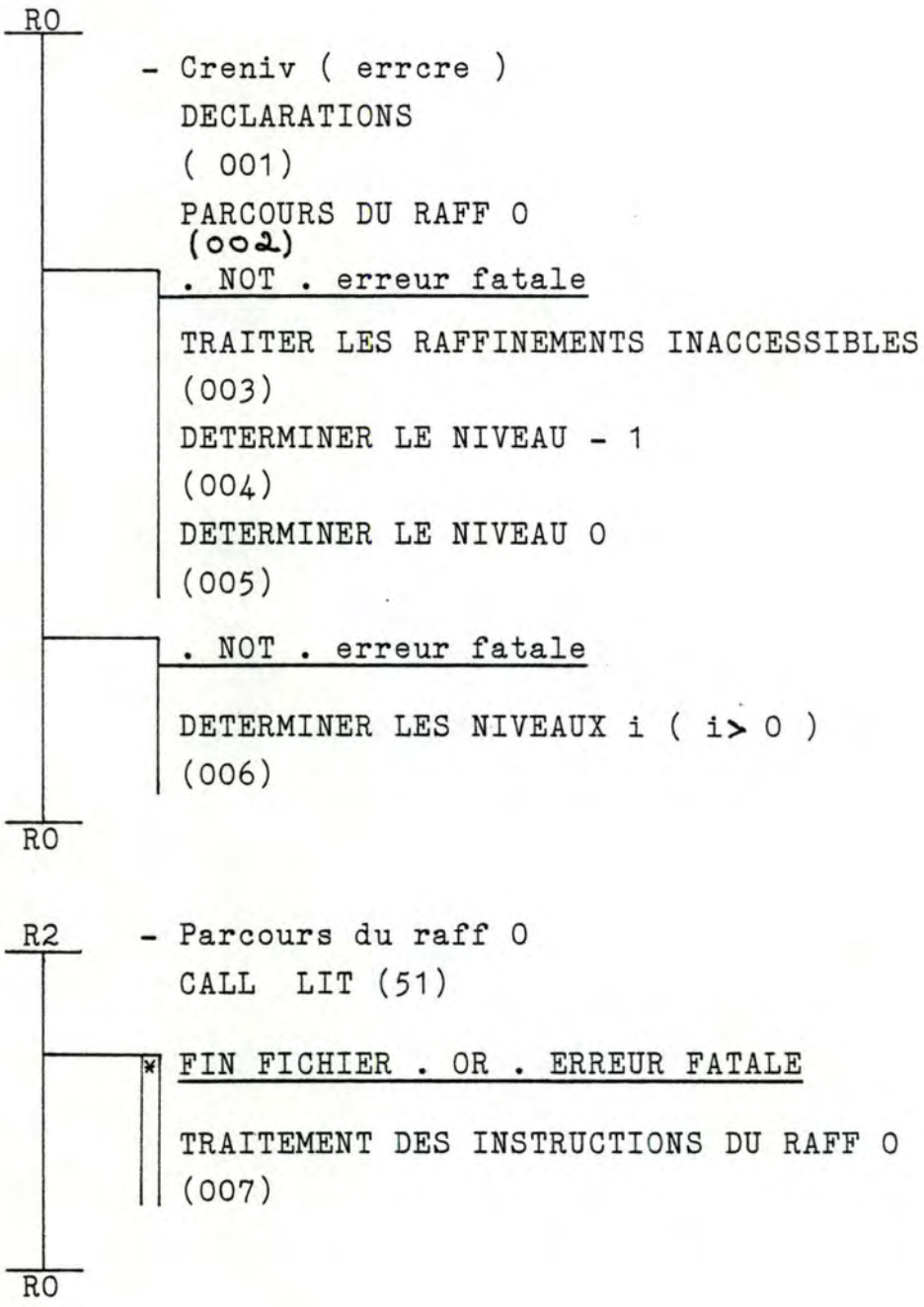
R0

R5

- Déterminer la valeur de numfic et
  mise à jour des supports
  NUMFIC = 0
  REWIND 7
  ENDFILE 7
  REWIND 14
  ENDFILE 14
- On remet l'écran dans l'état initial
  CALL NOCRLF

R0

2. Conception de la routine CRENIV



R7

- Traitement des instruction du raff 0

instruction de référence à un raffinement

```
AJOUTER " RAFF " A LA LISTE DES PERES
DU RAFFINEMENT REFERENCE : " RAFREF ".
(008)
INDEX = PNTR (4)
CALL   LIT (INDEX)
```

```
CALL   LECT
```

R2

R8

- Ajouter " raff " à la liste des pères du raffinement
- référencé " raref ".
- POSRAF = la position correspondant à " rafref " dans TABRAF

TABRAF (POSRAF, PRPERE) . NE . 999

- le raffinement " rafref " a déjà un père dans TABRAF

```
AJOUTER A " RAFREF " UN NOUVEAU PERE " RAFF "
DANS TABRAF
(009)
```

- le premier père de " rafref " est le raffinement " raff "

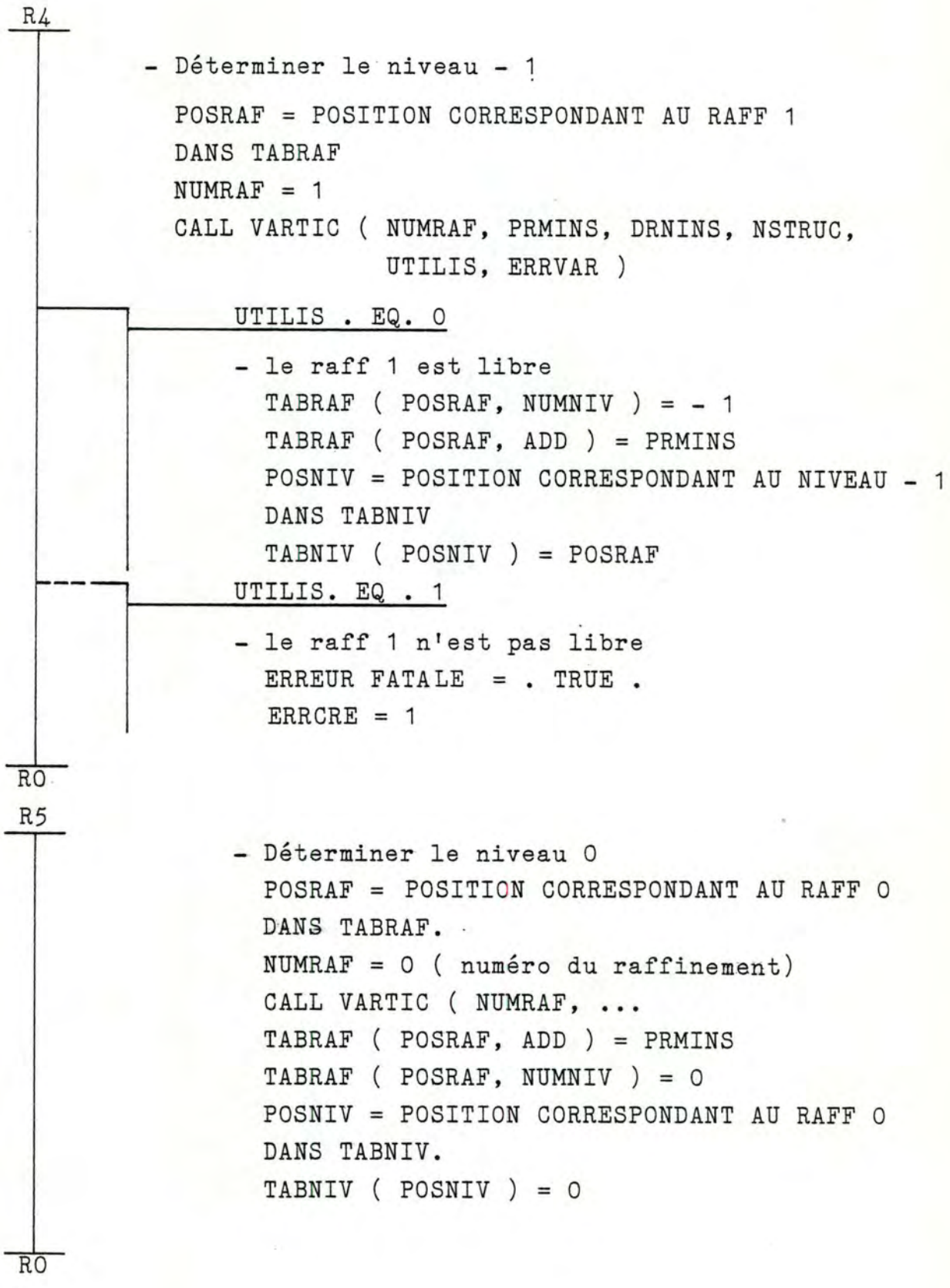
```
TABRAF ( POSRAF, PRPERE ) = RAFF
```

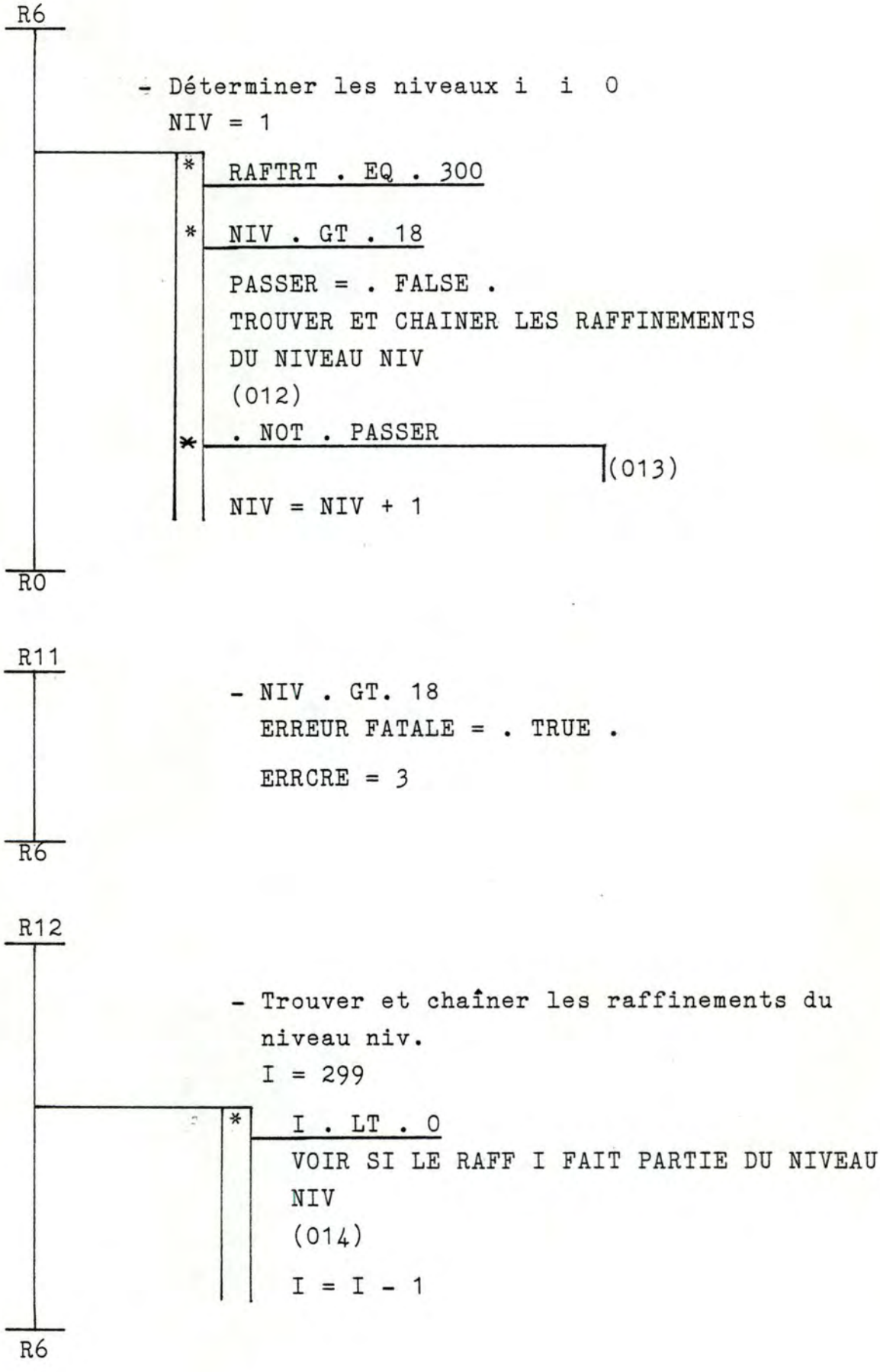
R7

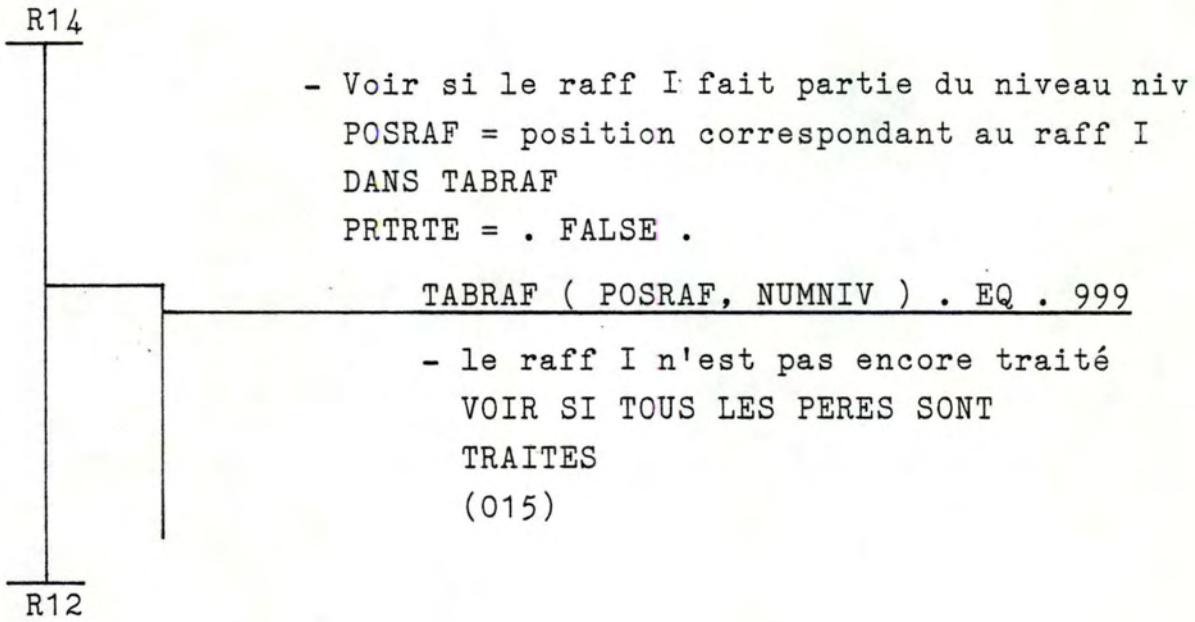
```
R9
   - Ajouter à " rafref " un nouveau père " raff "
     dans TABRAF
     LIBRE . LE . 10
     - la table TABPER est pleine
       ERREUR FATALE = . TRUE .
       ERRCRE = 4
     - Le raffinement " rafref " devient le deuxième
       père du raffinement " raff "
       TABPER ( LIBRE, PERE ) = RAFF
       TABPER ( LIBRE, PERE ) = TABRAF ( RAFREF, SECPER )
       TABRAF ( RAFREF, SECPER ) = LIBRE
       LIBRE   = LIBRE + 1
R8
```

```
R3
   - Traiter les raffinements inaccessibles
     j = 0
       j . GT . 299
       - TRAITER LE RAFF j
         (010)
         j = j + 1
R0
```

```
R10
   - Traiter le raff j
     POSRAF = POSITION CORRESPONDANT A j DANS
     TABRAF
         TABRAF ( POSRAF, PERE ) . EQ . 999
         - le raff j est inaccessible
           TABRAF ( POSRAF, PRPERE ) = 999
           TABRAF ( POSRAF, NUMNIV ) = - 2
           TABRAF ( POSRAF, ADD ) = 0
           RAFTRT = RAFTRT + 1
R3
```


R4
- Déterminer le niveau - 1
POSRAF = POSITION CORRESPONDANT AU RAFF 1
DANS TABRAF
NUMRAF = 1
CALL VARTIC (NUMRAF, PRMINS, DRNINS, NSTRUC,
UTILIS, ERRVAR)
UTILIS . EQ. 0
- le raff 1 est libre
TABRAF (POSRAF, NUMNIV) = - 1
TABRAF (POSRAF, ADD) = PRMINS
POSNIV = POSITION CORRESPONDANT AU NIVEAU - 1
DANS TABNIV
TABNIV (POSNIV) = POSRAF
UTILIS. EQ . 1
- le raff 1 n'est pas libre
ERREUR FATALE = . TRUE .
ERRCRE = 1
R0
R5
- Déterminer le niveau 0
POSRAF = POSITION CORRESPONDANT AU RAFF 0
DANS TABRAF.
NUMRAF = 0 (numéro du raffinement)
CALL VARTIC (NUMRAF, ...
TABRAF (POSRAF, ADD) = PRMINS
TABRAF (POSRAF, NUMNIV) = 0
POSNIV = POSITION CORRESPONDANT AU RAFF 0
DANS TABNIV.
TABNIV (POSNIV) = 0
R0


R6
- Déterminer les niveaux i i 0
NIV = 1
* RAFTRT . EQ . 300
* NIV . GT . 18
PASSER = . FALSE .
TROUVER ET CHAINER LES RAFFINEMENTS
DU NIVEAU NIV
(012)
* . NOT . PASSER
(013)
NIV = NIV + 1
R0
R11
- NIV . GT. 18
ERREUR FATALE = . TRUE .
ERRCRE = 3
R6
R12
- Trouver et chaîner les raffinements du
niveau niv.
I = 299
* I . LT . 0
VOIR SI LE RAFF I FAIT PARTIE DU NIVEAU
NIV
(014)
I = I - 1
R6


R14
- Voir si le raff I fait partie du niveau niv
POSRAF = position correspondant au raff I
DANS TABRAF
PRTRTE = . FALSE .
TABRAF (POSRAF, NUMNIV) . EQ . 999
- le raff I n'est pas encore traité
VOIR SI TOUS LES PERES SONT
TRAITES
(015)
R12

R15

- Voir si tous ses pères sont traités
  POSRAF = POSITION CORRESPONDANT AU PREMIER
  PERE DU RAFF I

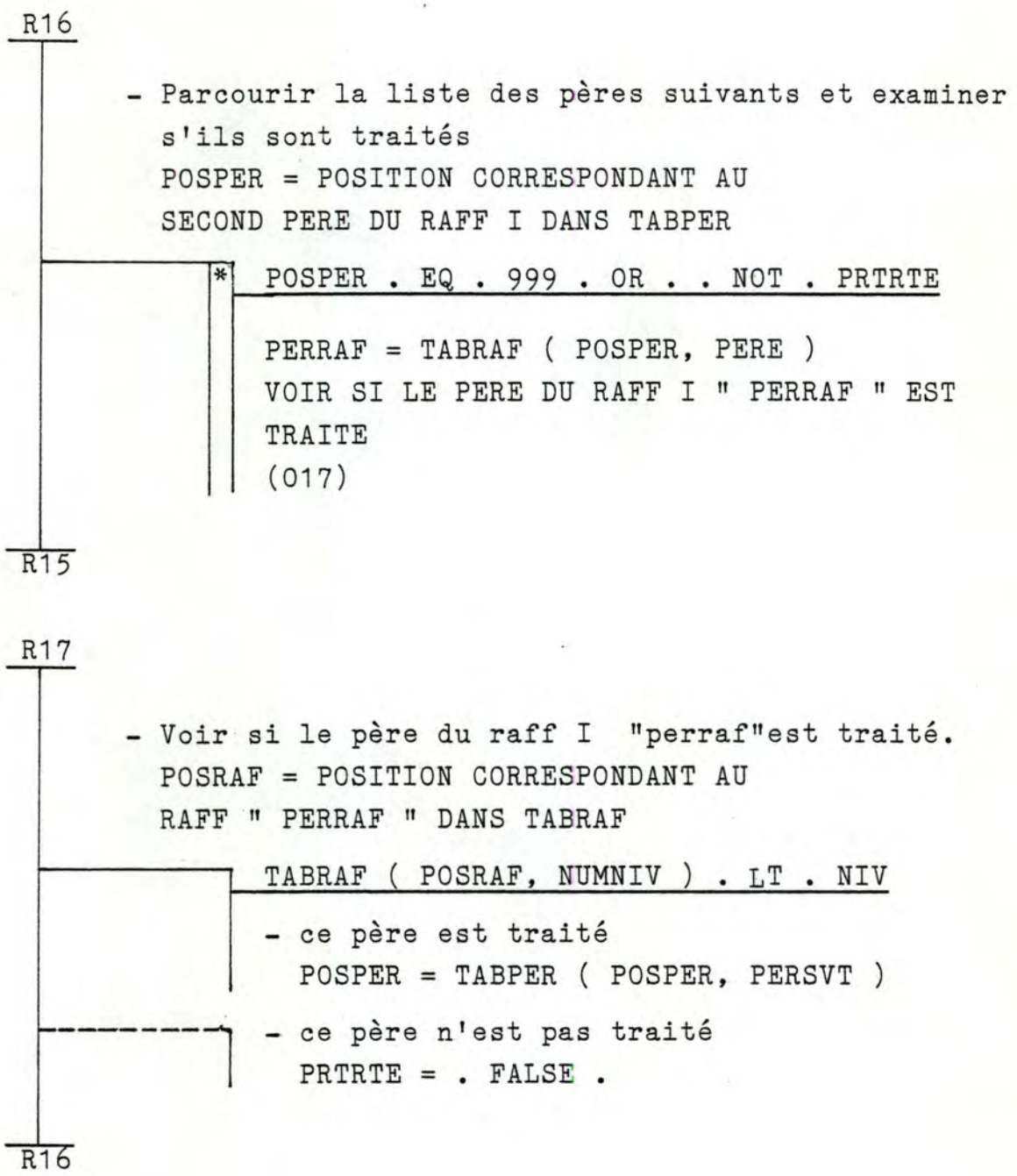
TABRAF (POSRAF, NUMNIV) . LT . NIV

- le premier père du raff I est traité
  PRTRTE = . TRUE .
  PARCOURIR LA LISTE DES PERES SUIVANTS
  ET EXAMINER S'ILS SONT TRAITES
  (016)

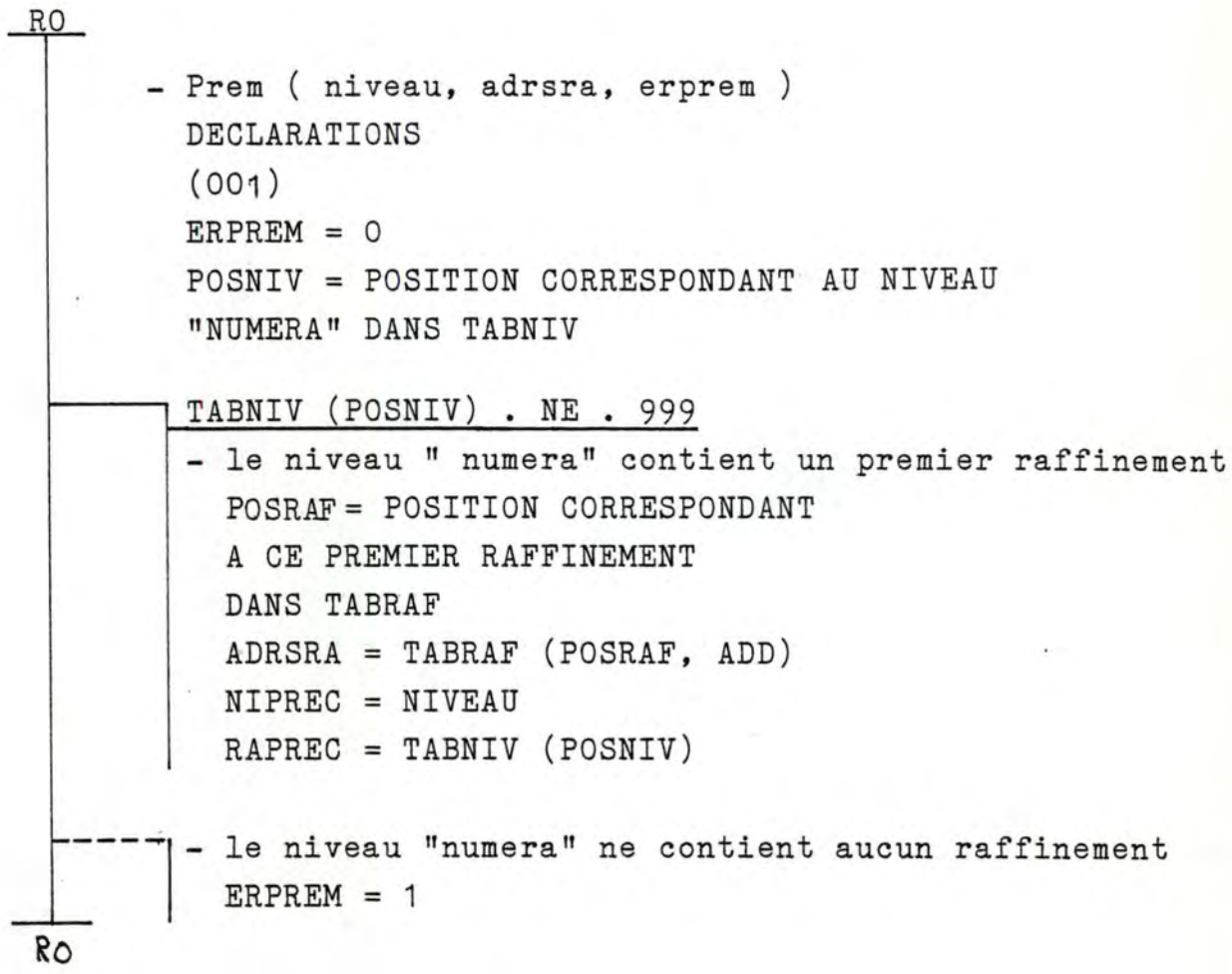
PRTRTE

- Tous les pères du raff I sont traités
  NUMRAF = I (numéro du raffinement)
  CALL VARTIC (NUMRAF, PRMINS ...
  POSRAF = POSITION CORRESPONDANT AU
  RAFF I DANS TABRAF
- On remplit le champ ADD du raff I
  TABRAF (POSRAF, ADD) = PRMINS
- Chaîner le raff I avec les autres
  raffinements du niveau
  POSNIV = POSITION CORRESPONDANT
  AU NIVEAU NIV DANS TABNIV
  TABRAF (POSRAF, NUMNIV) = NIV
  TABRAF (POSRAF, RAFSUI) = TABNIV(POSNIV
  TABNIV (POSNIV) = I

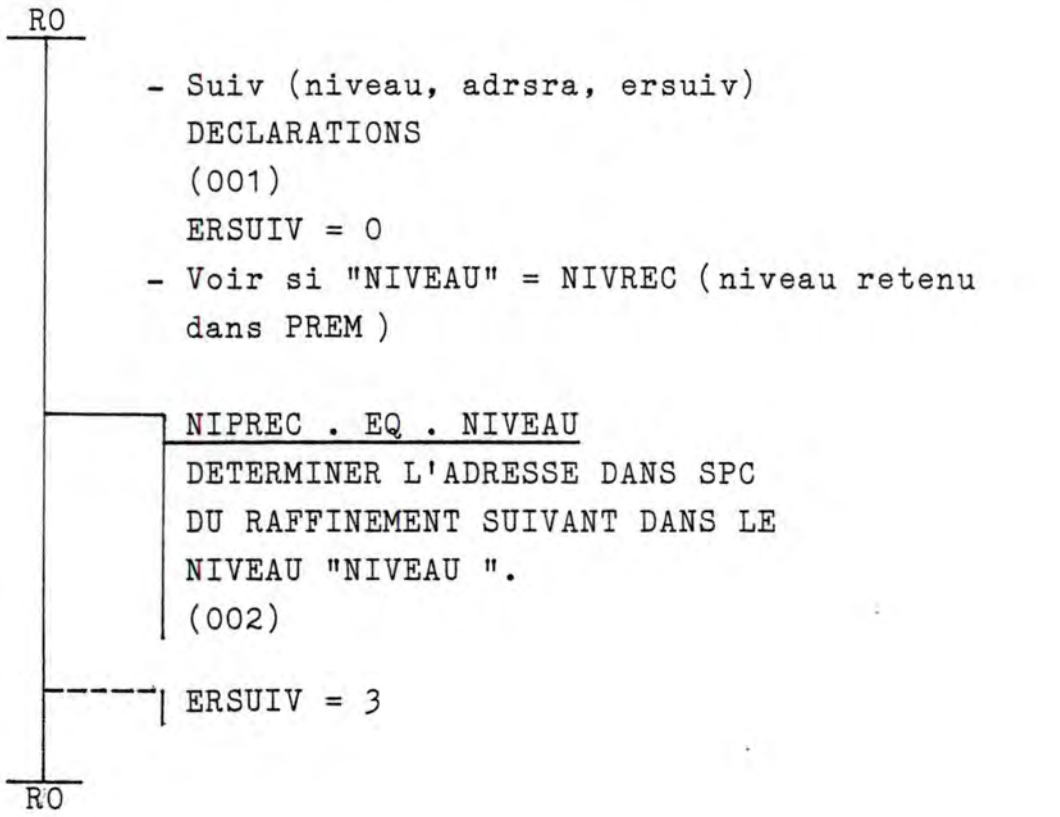
R14

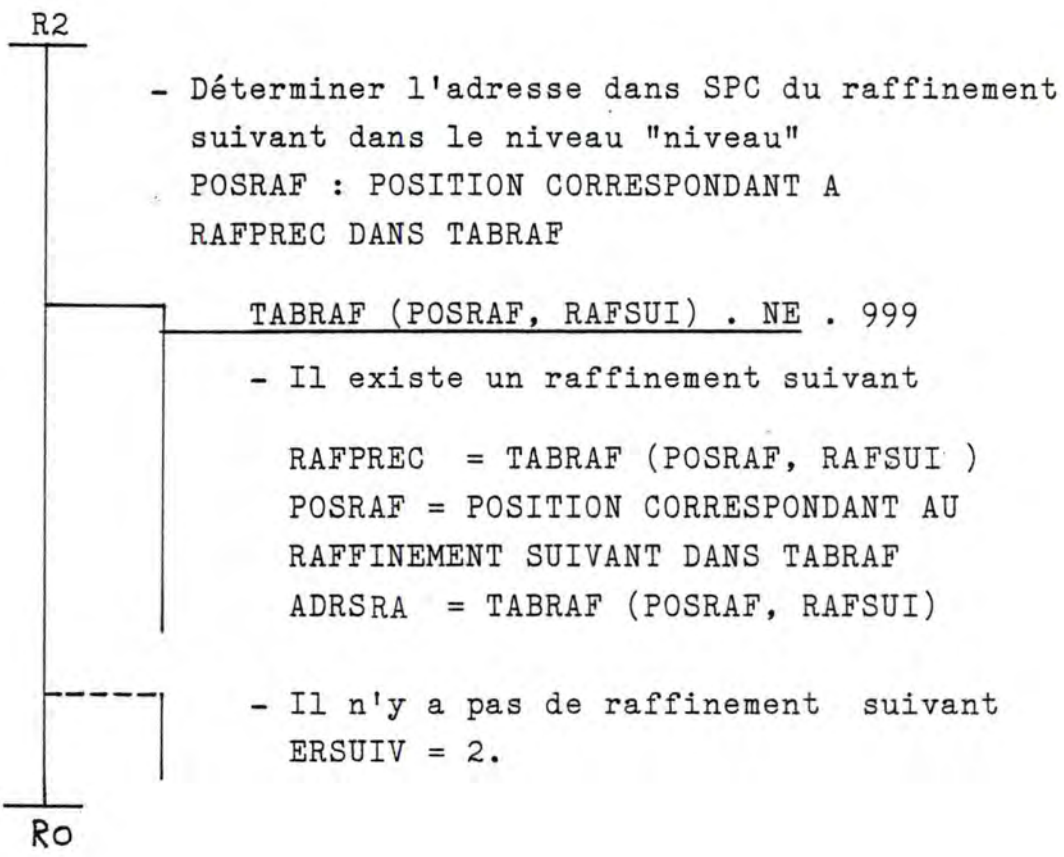

R16
- Parcourir la liste des pères suivants et examiner s'ils sont traités
POSPER = POSITION CORRESPONDANT AU SECOND PERE DU RAFF I DANS TABPER
*
POSPER . EQ . 999 . OR . . NOT . PRTRTE
PERRAF = TABRAF (POSPER, PERE)
VOIR SI LE PERE DU RAFF I " PERRAF " EST TRAITE
(017)
R15
R17
- Voir si le père du raff I "perraf"est traité.
POSRAF = POSITION CORRESPONDANT AU RAFF " PERRAF " DANS TABRAF
TABRAF (POSRAF, NUMNIV) . LT . NIV
- ce père est traité
POSPER = TABPER (POSPER, PERSVT)
- ce père n'est pas traité
PRTRTE = . FALSE .
R16

3. Conception de la routine PREM



4. Conception de la routine SUIV



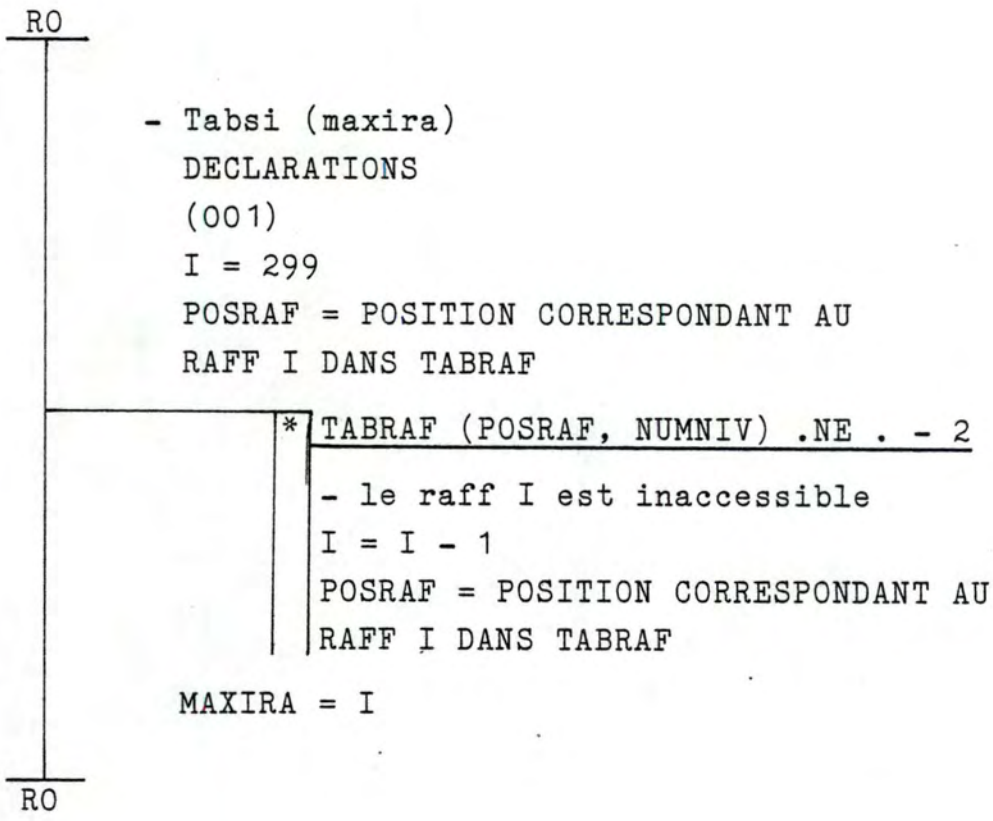


5. Conception de la routine NIVRAF

RO

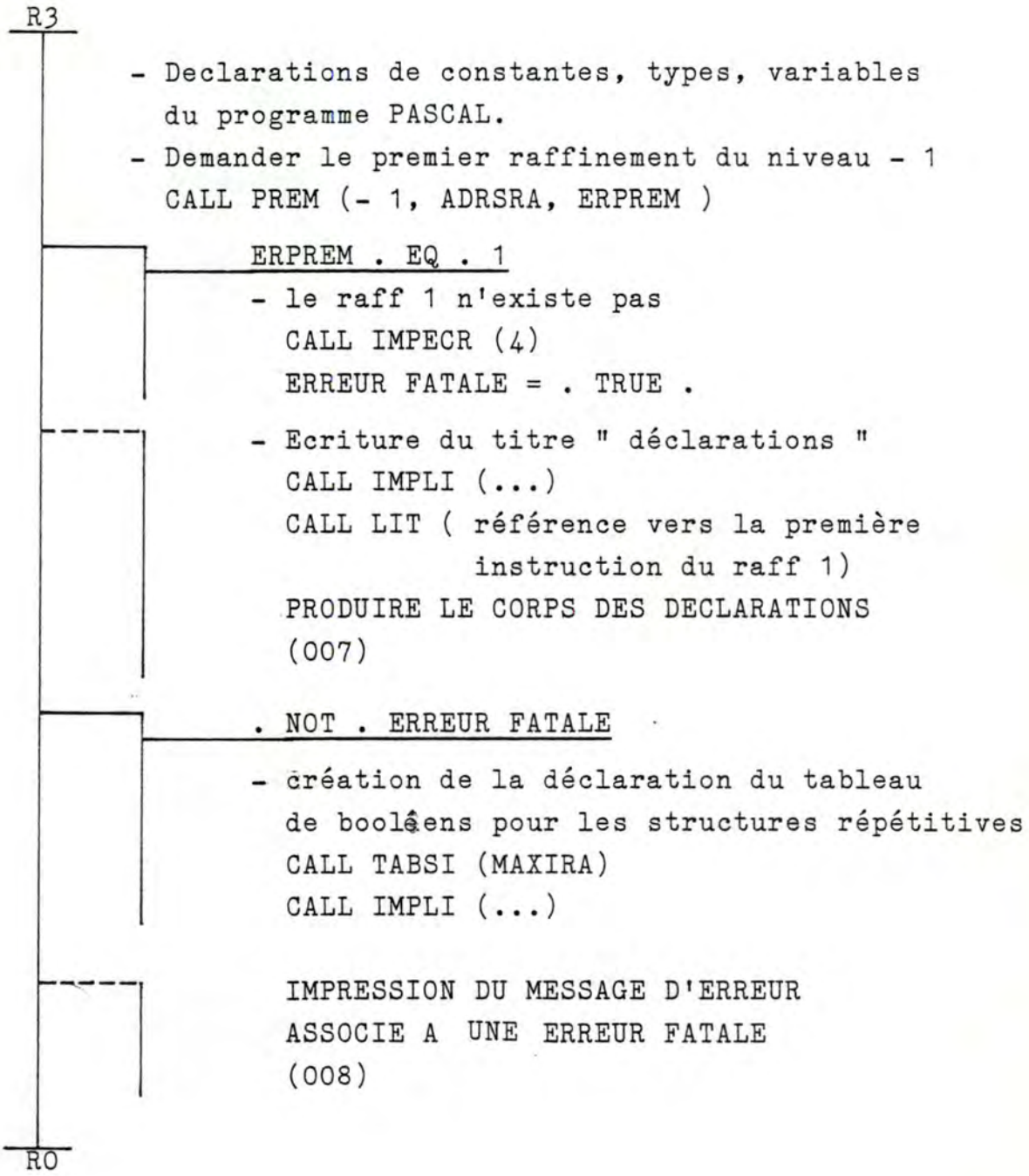
- Nivraf (numéra, niveau)
  DECLARATIONS
  (001)
  POSRAF = POSITION CORRESPONDANT A
  "NUMERA" DANS TABRAF
  NIVEAU = TABRAF (POSRAF, NUMNIV)

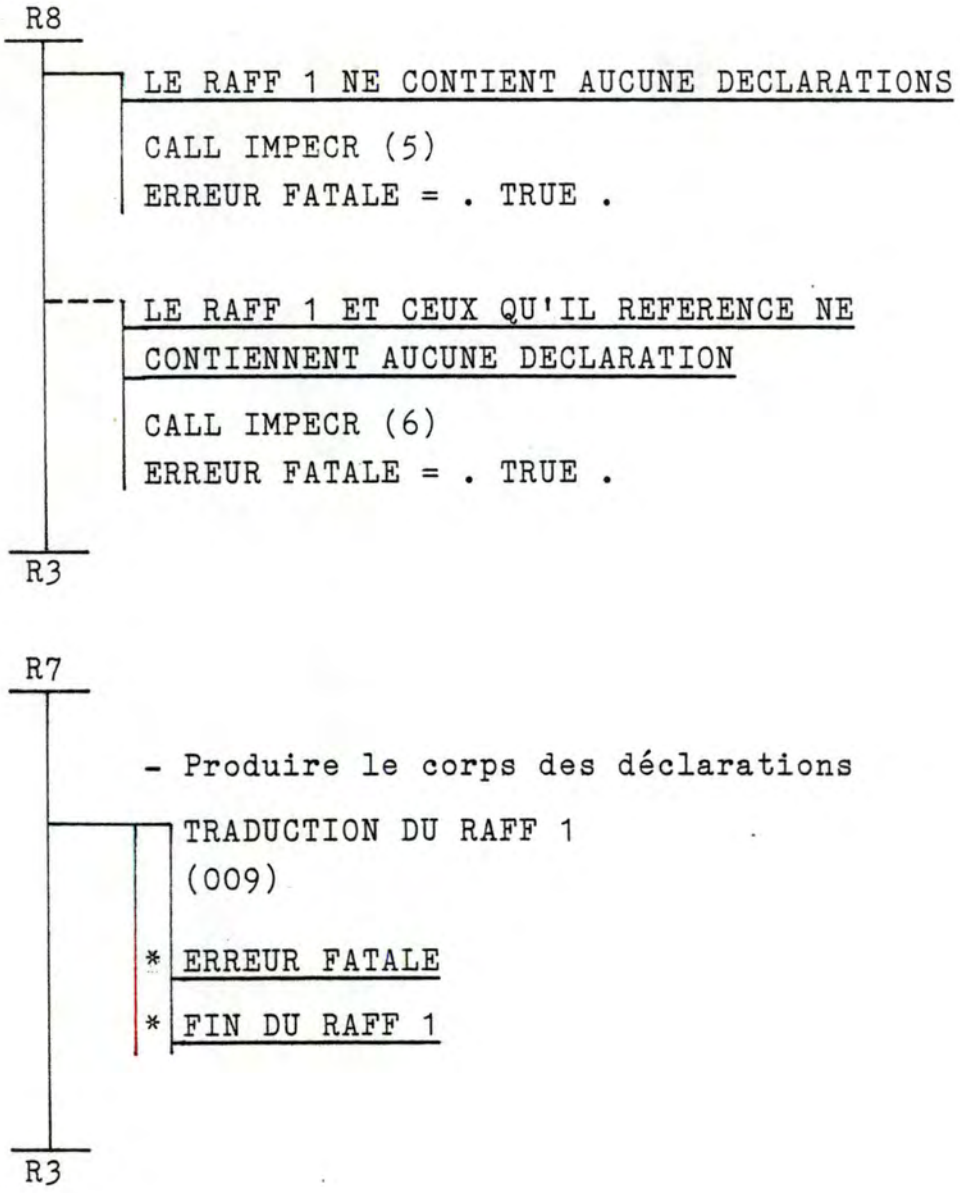
6. Conception de la routine TABSI

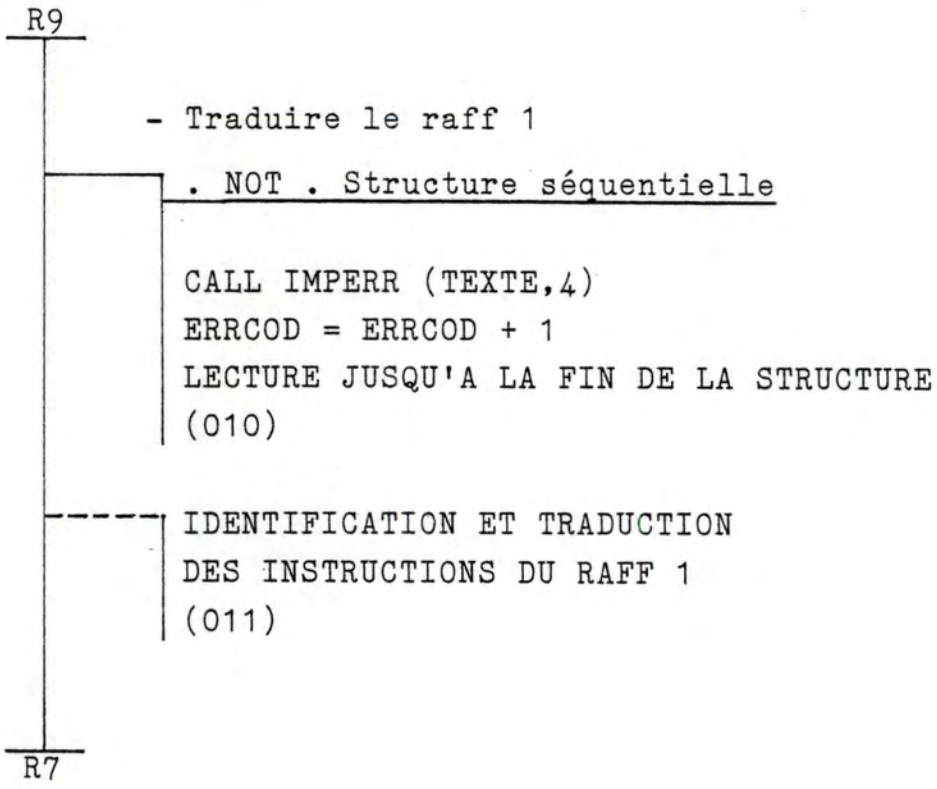


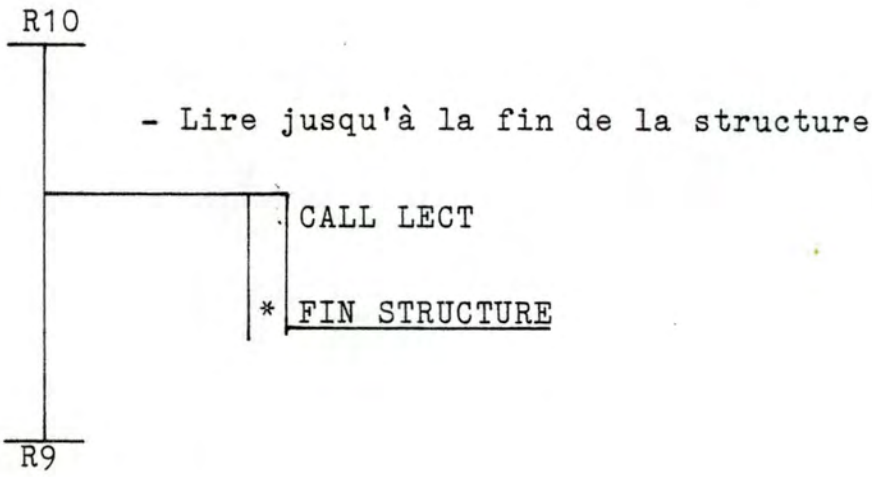
7. Conception de la routine TRTRAF

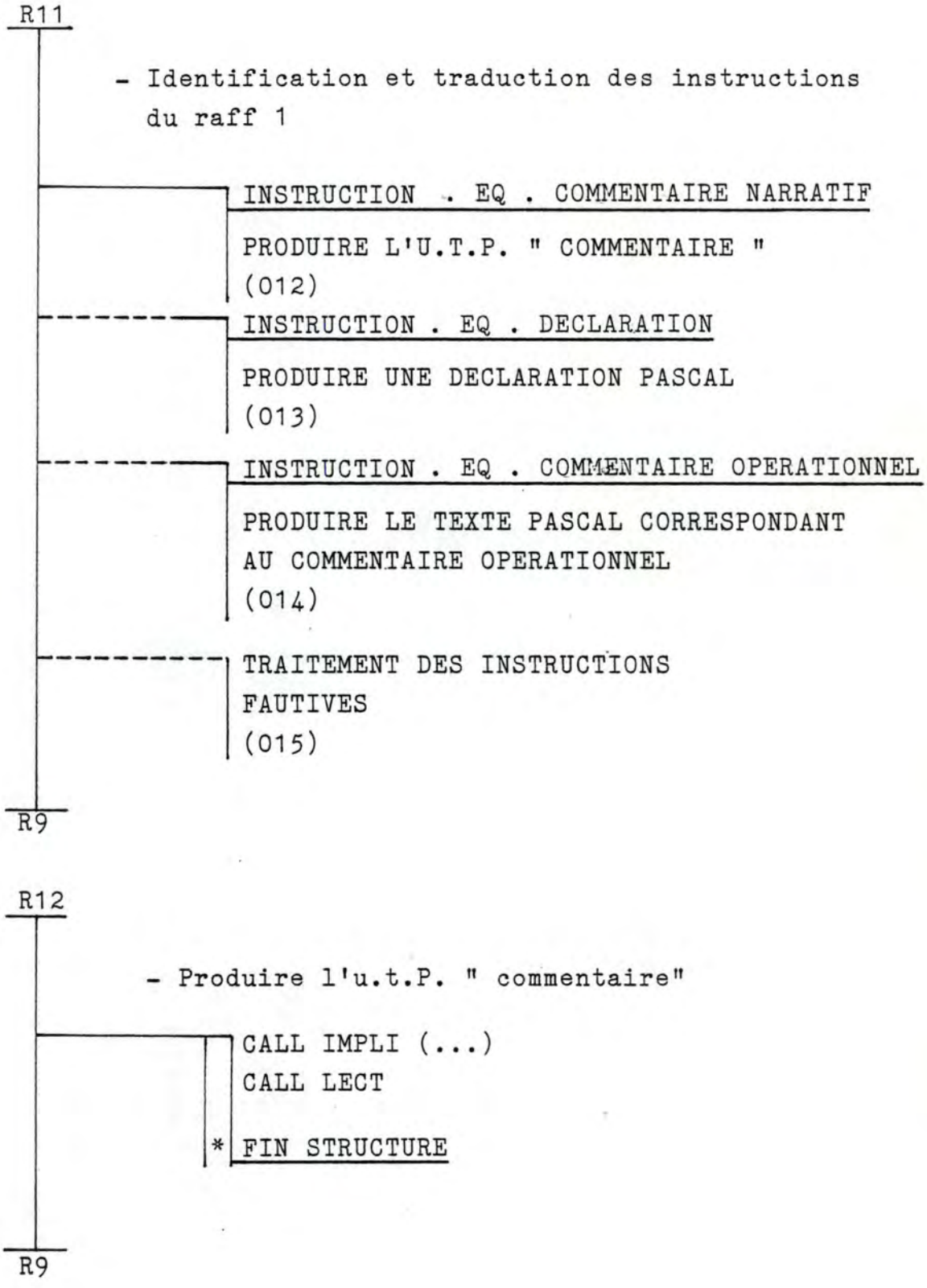
```
RO
|
|   - Trtraf
|     DECLARATIONS
|     (001)
|     EN-TETE DU PROGRAMME PASCAL
|     (002)
|     DECLARATIONS DE CONSTANTES, TYPES,
|     VARIABLES DU PROGRAMME PASCAL
|     (003)
|     . NOT . ERREUR FATALE
|     DECLARATION FORWARD DES PROCEDURES
|     (004)
|     DECLARATION DU TEXTE DES PROCEDURES
|     STRUCTUREES EN NIVEAUX
|     (005)
|     PROGRAMME PRINCIPAL DU PROGRAMME
|     PASCAL
|     (006)
|
RO

R2
|
|   --En-tête du programme PASCAL
|     CALL PREM (0, ADRSRA, ERPREM)
|     CALL LIT (51)
|     CALL LECT
|   --Ecrire le nom du fichier
|     CALL IMPLI (TABO, TEXTE, 72,. TRUE ., . TRUE .)
|     CALL LIT (51)
|   --Ecrire l'instruction "PROGRAM"
|     CALL IMPLI(TABO, TEXTE, 72, . FALSE., . TRUE.)
|
RO
```


R3
- Declarations de constantes, types, variables du programme PASCAL.
- Demander le premier raffinement du niveau - 1
CALL PREM (- 1, ADRSRA, ERPREM)
ERPREM . EQ . 1
- le raff 1 n'existe pas
CALL IMPECR (4)
ERREUR FATALE = . TRUE .
- Ecriture du titre " déclarations "
CALL IMPLI (...)
CALL LIT (référence vers la première instruction du raff 1)
PRODUIRE LE CORPS DES DECLARATIONS
(007)
. NOT . ERREUR FATALE
- création de la déclaration du tableau de booléens pour les structures répétitives
CALL TABSI (MAXIRA)
CALL IMPLI (...)
IMPRESSION DU MESSAGE D'ERREUR
ASSOCIE A UNE ERREUR FATALE
(008)
R0


R8
LE RAFF 1 NE CONTIENT AUCUNE DECLARATIONS
CALL IMPECR (5)
ERREUR FATALE = . TRUE .
LE RAFF 1 ET CEUX QU'IL REFERENCE NE
CONTIENNENT AUCUNE DECLARATION
CALL IMPECR (6)
ERREUR FATALE = . TRUE .
R3
R7
- Produire le corps des déclarations
TRADUCTION DU RAFF 1
(009)
* ERREUR FATALE
* FIN DU RAFF 1
R3


R9
- Traduire le raff 1
. NOT . Structure séquentielle
CALL IMPERR (TEXTE,4)
ERRCOD = ERRCOD + 1
LECTURE JUSQU'A LA FIN DE LA STRUCTURE
(010)
IDENTIFICATION ET TRADUCTION
DES INSTRUCTIONS DU RAFF 1
(011)
R7


R10
- Lire jusqu'à la fin de la structure
. CALL LECT
* FIN STRUCTURE
R9


R11
- Identification et traduction des instructions du raff 1
INSTRUCTION . EQ . COMMENTAIRE NARRATIF
PRODUIRE L'U.T.P. " COMMENTAIRE "
(012)
INSTRUCTION . EQ . DECLARATION
PRODUIRE UNE DECLARATION PASCAL
(013)
INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL
PRODUIRE LE TEXTE PASCAL CORRESPONDANT AU COMMENTAIRE OPERATIONNEL
(014)
TRAITEMENT DES INSTRUCTIONS FAUTIVES
(015)
R9
R12
- Produire l'u.t.P. " commentaire"
CALL IMPLI (...)
CALL LECT
*
FIN STRUCTURE
R9

R13

- Produire une déclaration PASCAL

CALL IMPLI (...)
CALL LECT
* FIN STRUCTURE

R9

R14

- Produire le texte PASCAL correspondant
- au commentaire opérationnel

. NOT . FIN RAFFINEMENT

nombre d'imbrications = nombre d'imbircations + 1

TRAITER LE COMMENTAIRE OPERATIONNEL
(016)

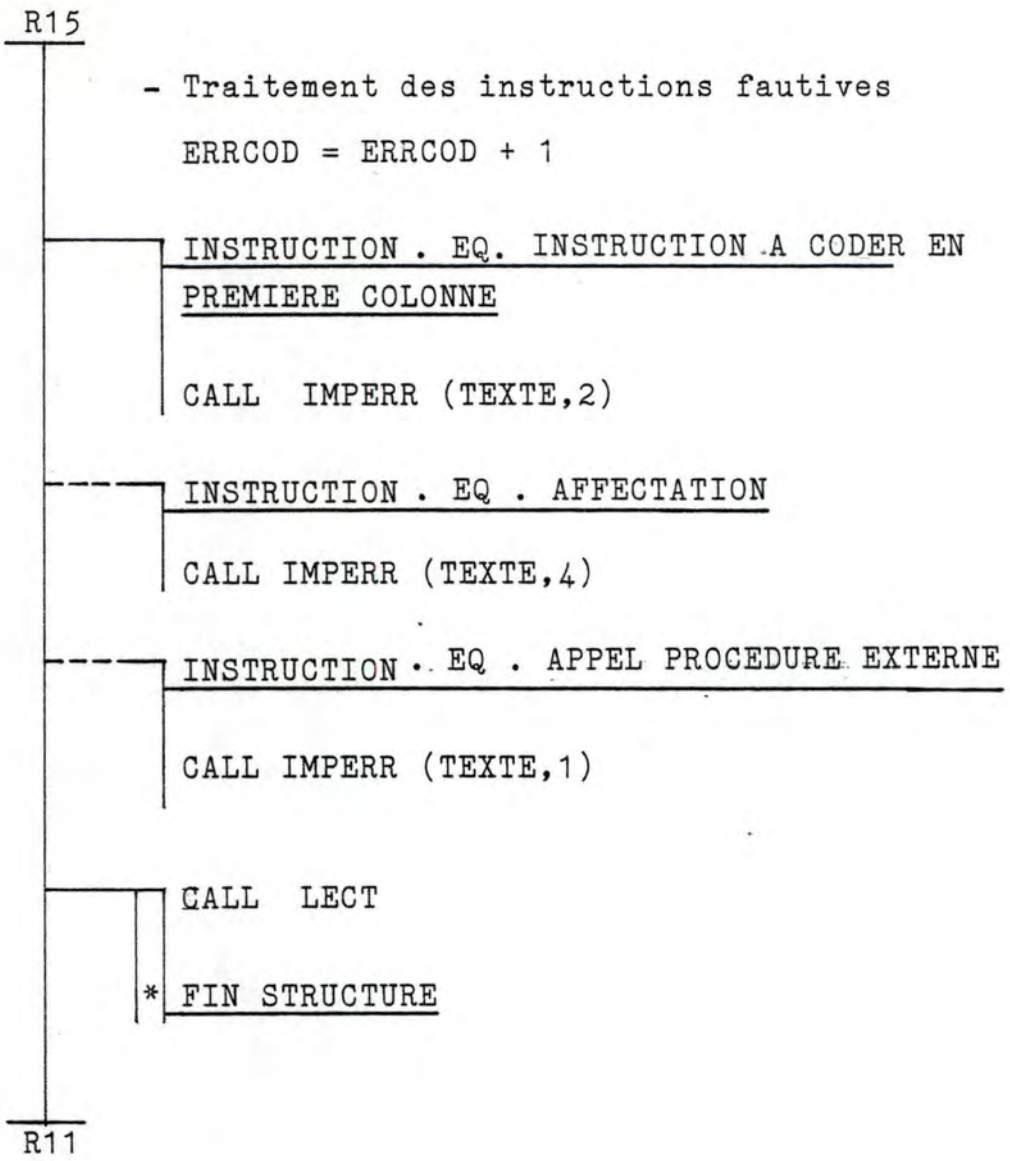
- Restaurer le numéro de la ligne du raffinement
  où se trouve le commentaire opérationnel
  NOLIGN = LGNCOM
  CALL LECT

R9

R16

- Traiter le comentaire opérationnel
- Sauver le numéro de la ligne où se trouve
  le commentaire opérationnel
  LGNCOM = NOLIGN
  NOLIGN = 1
  CALL LIT (numéro du raffinement référencé)

R14


R15
- Traitement des instructions fautives
ERRCOD = ERRCOD + 1
INSTRUCTION . EQ. INSTRUCTION A CODER EN PREMIERE COLONNE
CALL IMPERR (TEXTE,2)
INSTRUCTION . EQ . AFFECTATION
CALL IMPERR (TEXTE,4)
INSTRUCTION . EQ . APPEL PROCEDURE EXTERNE
CALL IMPERR (TEXTE,1)
CALL LECT
*
FIN STRUCTURE
R11

R4

- Déclartations FORWARD des procédures
  NIV = 1
  CALL PREM (NIV, ADRSRA, ERPREM)

ERPREM . EQ . 0

- le premier raffinement du niveau 1 existe
- écrire le titre : déclarations
  CALL IMPLI (...)

* ERPREM . EQ . 1 (le premier raffinement du niveau "niv" n'existe pas)

PRODUIRE L'U.T.P. " CORPS DECLARATIONS PROCEDURES FORWARD ".
(017)

- Demander l'adresse du premier raffinement du niveau suivant
  NIV = NIV + 1
  CALL PREM (NIV, ADRSRA, ERPREM)

R0

R17

- Produire l'u.t.p. " corps déclarations procédures forward ".
  ERSUIV = 0

* ERSUIV . EQ . 2 (il n'y a plus de raffinement dans le niveau " niv ")

CALL LIT (numéro du raffinement courant)

- Ecrire le texte PASCAL correspondant à l'u.t.p.
  CALL IMPLI (...)
- Demander l'adresse du raffinement suivant dans le niveau " niv "
  CALL SUIV (NIV, ADRSRA, ERSUIV)

R4

R5

- Déclaration du texte des procédures structurées en niveaux .
  NIV = 1
  CALL PREM (NIV, ADRSRA, ERPREM)

* ERPREM . EQ . 2 (le premier raffinement du niveau " niv " n'existe pas)

- Ecrire le titre " niveau "
  CALL IMPLI (TEXTE,...)
  PRODUIRE LE CORPS DU NIVEAU "NIV"
  (018)
- Demander le premier raffinement du niveau suivant
  NIV = NIV + 1
  CALL PREM (NIV, ADRSRA, ERPREM)

R0

R18

- Produire le corps du niveau "niv"
  ERSUIV = 0

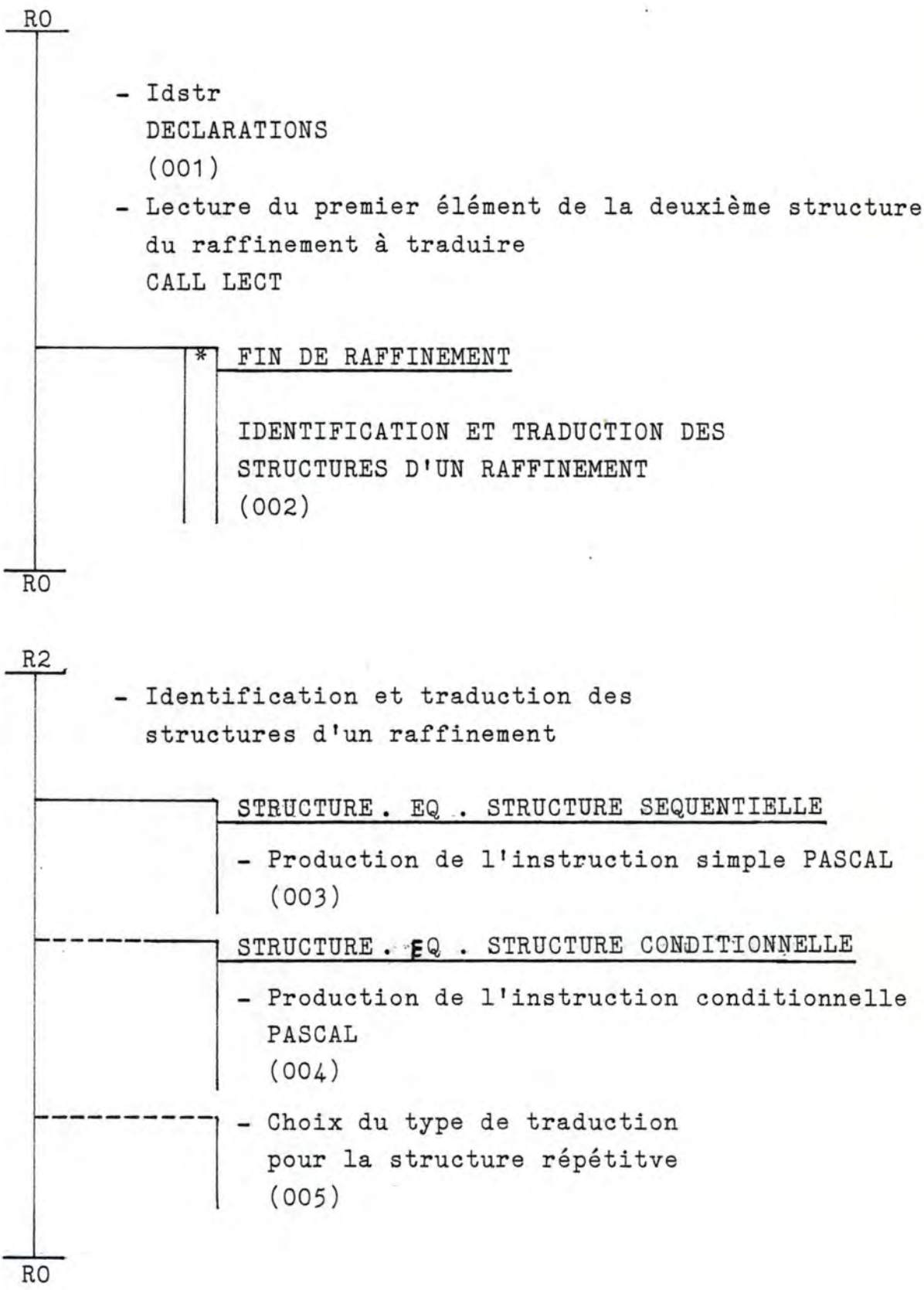
* ERSUIV . EQ . 2 (il n'y a plus de raffinement dans le niveau "niv")

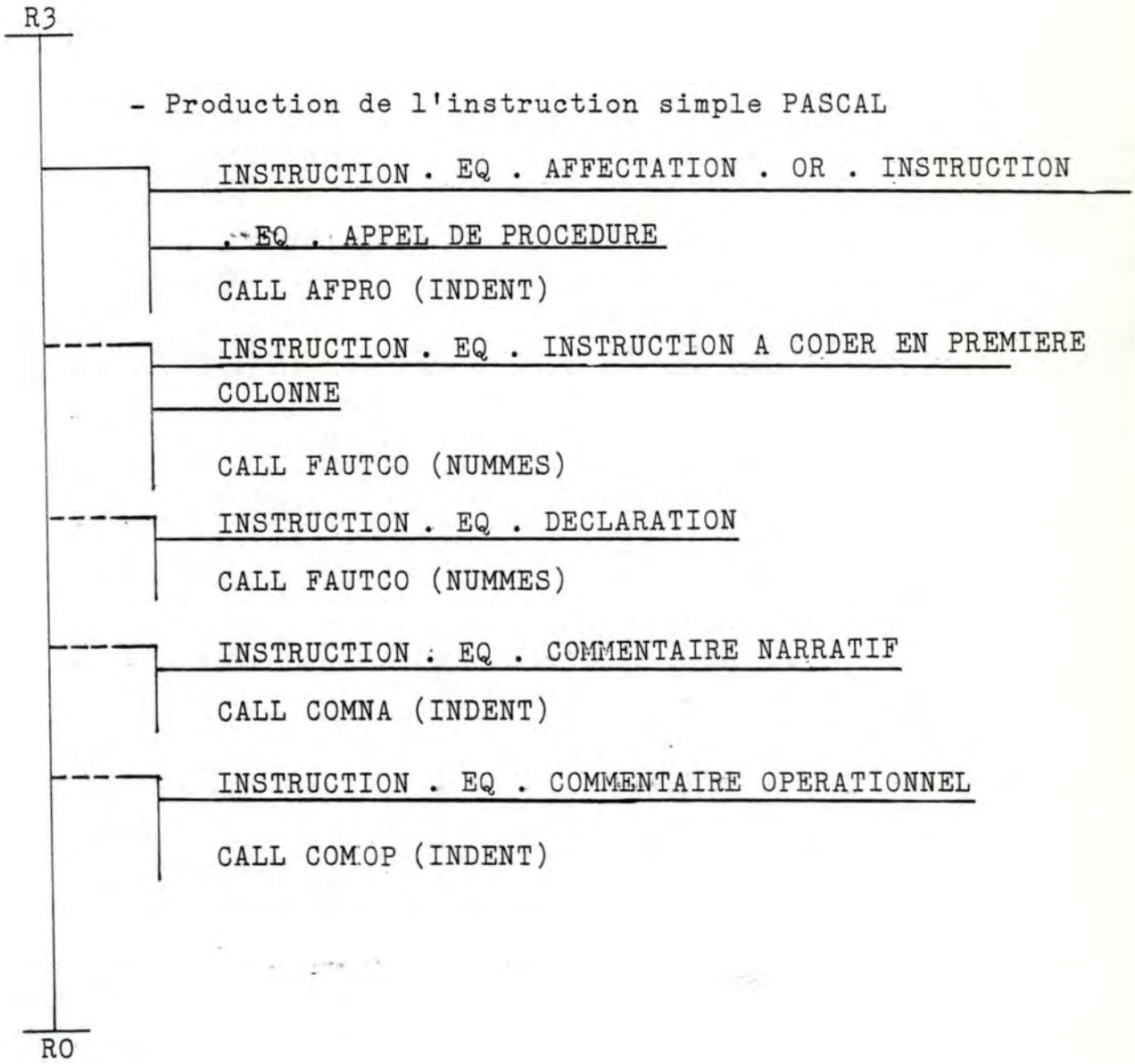
NOLIGN = 1
CALL LIT
- Ecrire l'en-tête de la procédure
  CALL IMPLI (...)
- Produire le corps de la procédure
  CALL IDSTR
- Ecrire la fin de la procédure
  CALL IMPLI (...)
- Demander l'adresse du raffinement suivant dans le niveau "niv"
  CALL SUIV (NIV, ADRSRA, ERSUIV)

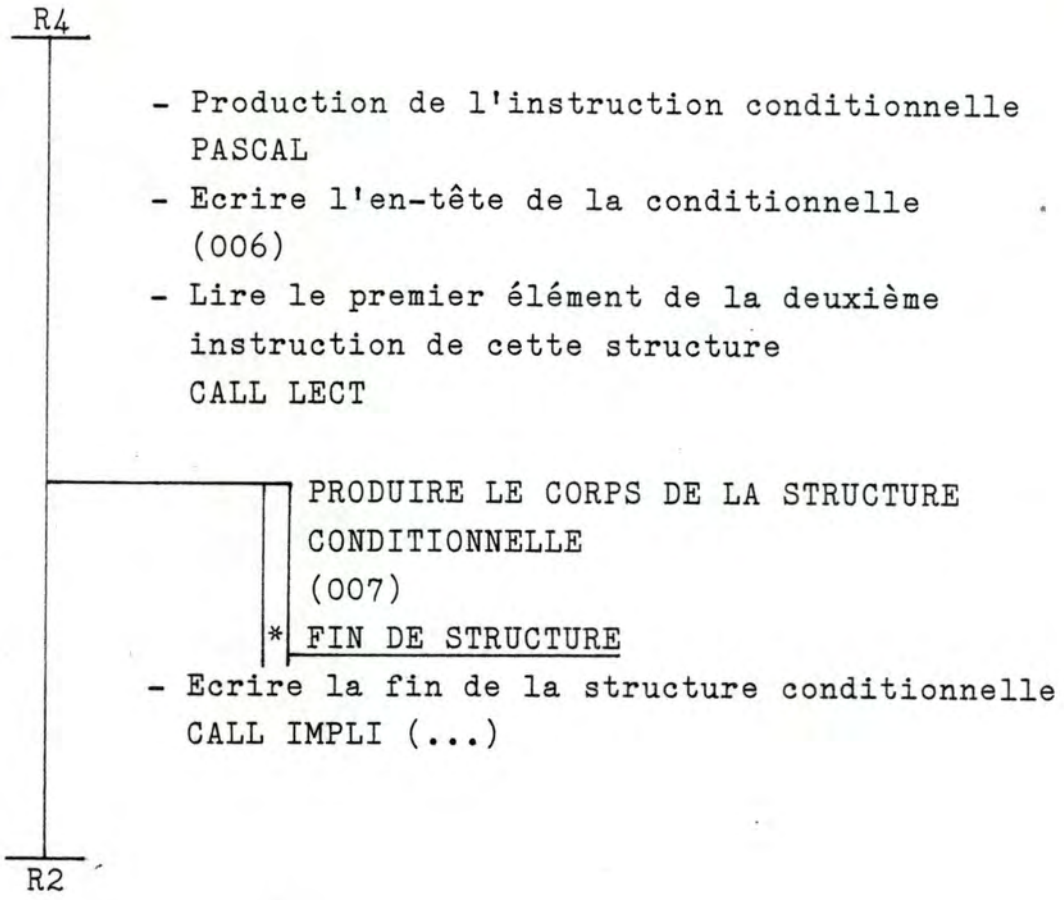
R5

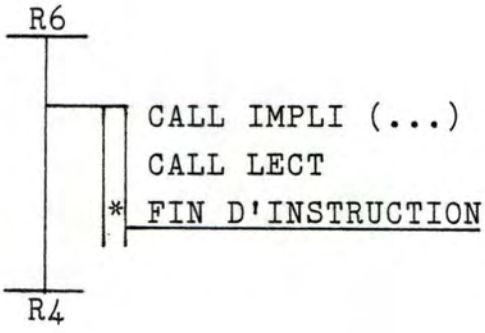
- Programme principal du programme PASCAL
- Ecrire l'en-tête du programme principal
  CALL IMPLI (...)
  CALL PREM (0, ADRSRA, ERPREM)
  NOLIGN = 1
- Traduire les structures du raff 0
  (sauf les deux premières)
  CALL IDSTR
- Ecrire l'u.t.P. " fin programme principal "
  CALL IMPLI (...)

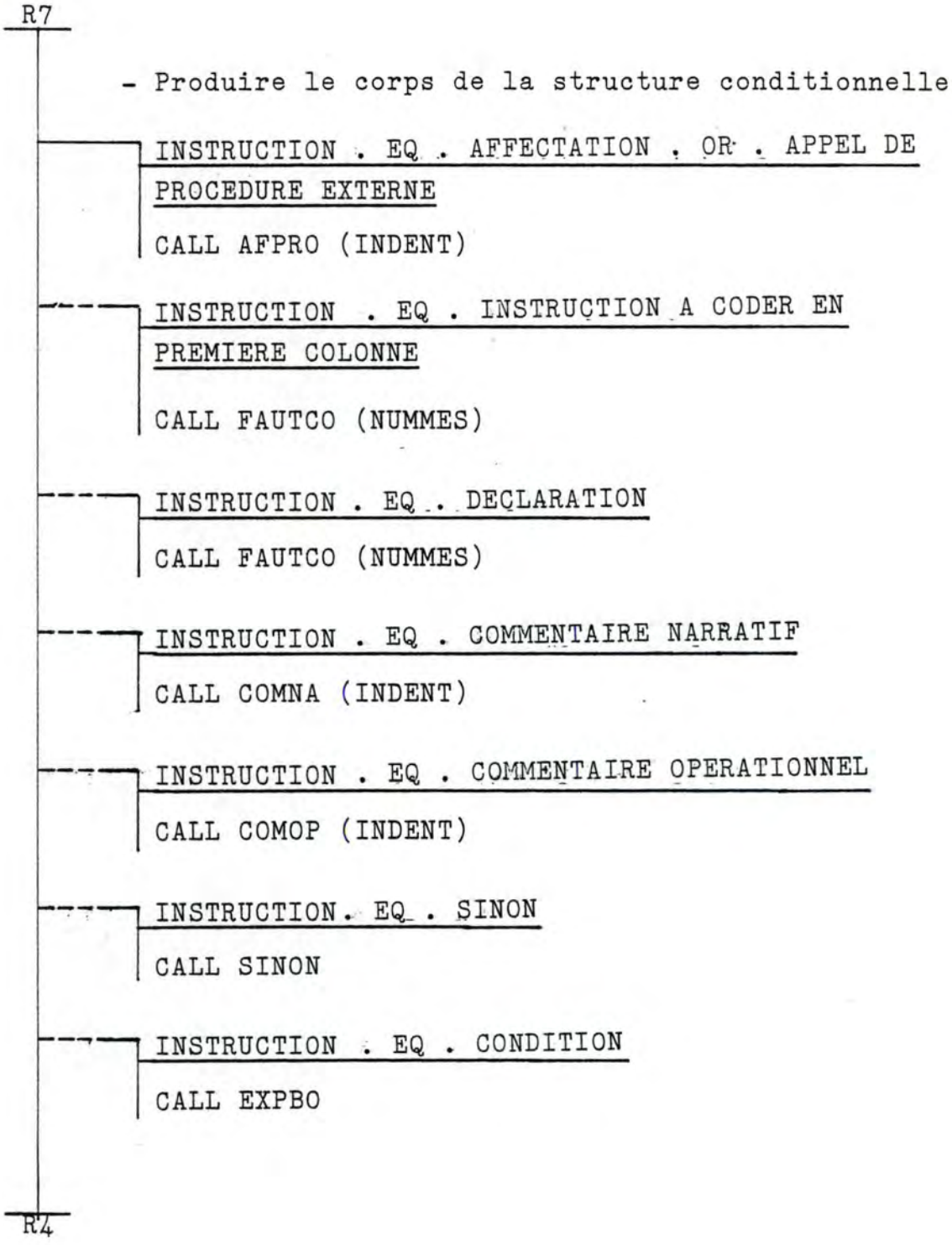
8. Conception de la routine IDSTR

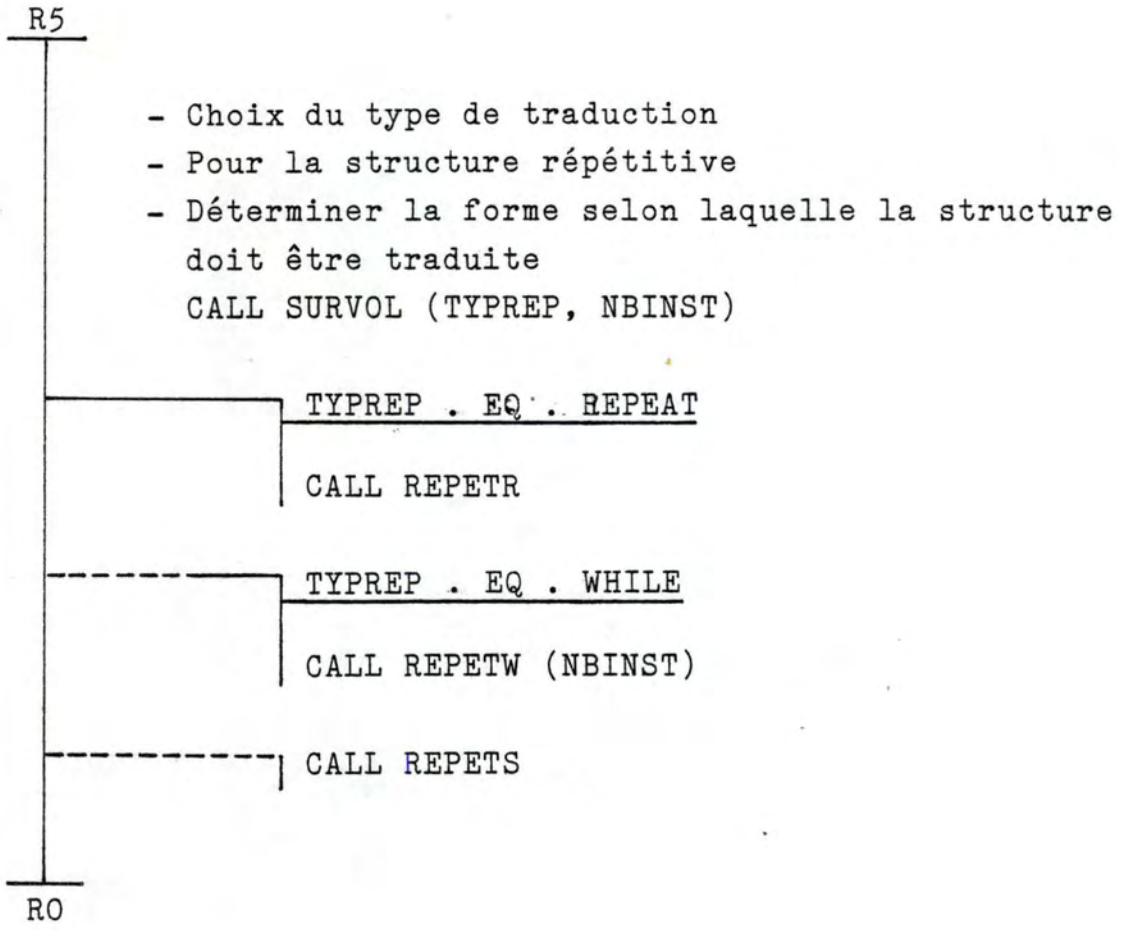



R3
- Production de l'instruction simple PASCAL
INSTRUCTION . EQ . AFFECTATION . OR . INSTRUCTION
. EQ . APPEL DE PROCEDURE
CALL AFPRO (INDENT)
INSTRUCTION . EQ . INSTRUCTION A CODER EN PREMIERE
COLONNE
CALL FAUTCO (NUMMES)
INSTRUCTION . EQ . DECLARATION
CALL FAUTCO (NUMMES)
INSTRUCTION . EQ . COMMENTAIRE NARRATIF
CALL COMNA (INDENT)
INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL
CALL COMOP (INDENT)
RO


R4
- Production de l'instruction conditionnelle PASCAL
- Ecrire l'en-tête de la conditionnelle (006)
- Lire le premier élément de la deuxième instruction de cette structure CALL LECT
PRODUIRE LE CORPS DE LA STRUCTURE CONDITIONNELLE (007)
* FIN DE STRUCTURE
- Ecrire la fin de la structure conditionnelle CALL IMPLI (...)
R2


R6
CALL IMPLI (...)
CALL LECT
* FIN D'INSTRUCTION
R4


R7
- Produire le corps de la structure conditionnelle
INSTRUCTION . EQ . AFFECTATION . OR . APPEL DE PROCEDURE EXTERNE
CALL AFPRO (INDENT)
INSTRUCTION . EQ . INSTRUCTION A CODER EN PREMIERE COLONNE
CALL FAUTCO (NUMMES)
INSTRUCTION . EQ . DECLARATION
CALL FAUTCO (NUMMES)
INSTRUCTION . EQ . COMMENTAIRE NARRATIF
CALL COMNA (INDENT)
INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL
CALL COMOP (INDENT)
INSTRUCTION . EQ . SINON
CALL SINON
INSTRUCTION . EQ . CONDITION
CALL EXPBO
R4


R5
- Choix du type de traduction
- Pour la structure répétitive
- Déterminer la forme selon laquelle la structure doit être traduite
CALL SURVOL (TYPREP, NBINST)
TYPREP . EQ . REPEAT
CALL REPETR
TYPREP . EQ . WHILE
CALL REPETW (NBINST)
CALL REPETS
RO

10. Conception de la routine REPETR

RO

- Repetr
  DECLARATIONS
  (001)
- Ecrire l'en-tête de la structure repeat
  CALL IMPLI (...)

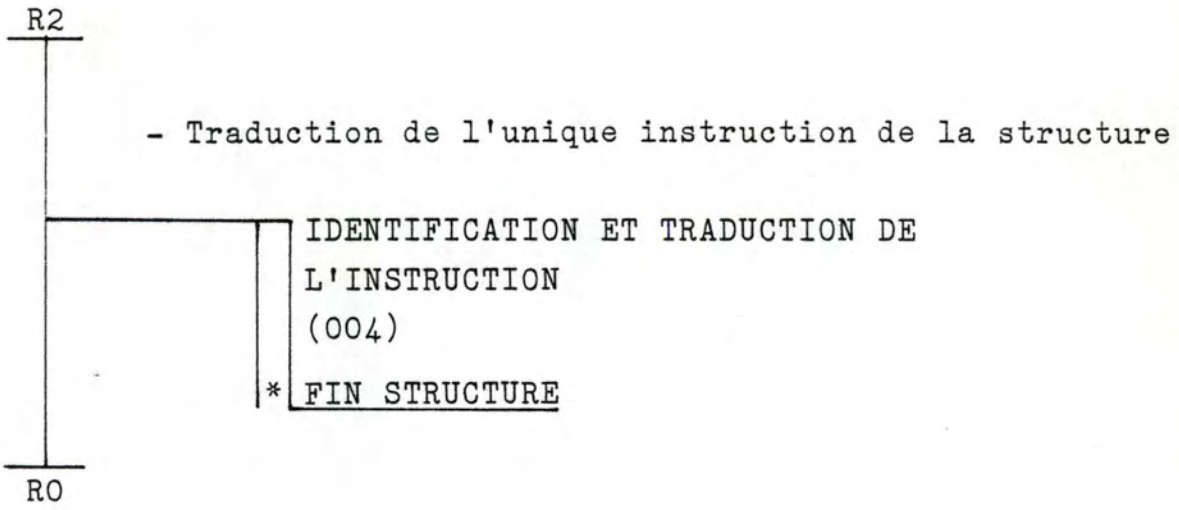
* INSTRUCTION . EQ . CONDITION DE SORTIE

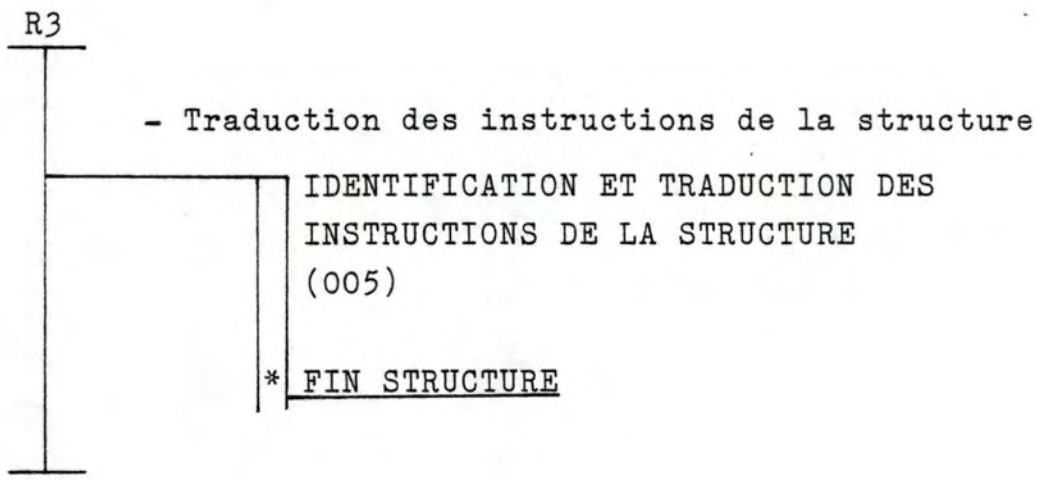
PRODUIRE LE CORPS DU REPEAT
(002)

INSTRUCTION . EQ . CONDITION DE SORTIE
SANS REFERENCE A UN RAFFINEMENT

- Ecrire la fin simple du repeat
  CALL IMPLI (...)
  CALL LECT

- Ecrire la fin du repeat avec un traitement d'exception
  CALL IMPLI (...)
  CALL NIVRAF (numéro de raffinement de cette répétitive, NIVEAU)
  COMPTER LE NOMBRE DE CHIFFRES SIGNIFICATIFS INIVBK DANS LE NUMERO DE NIVEAU
- Transformer le numéro du niveau en une chaîne de caractères (NUMERO)
  CALL ALPHA (NIVEAU, INIVBK, NUMERO)
  CALL IMPLI (...)


R2
- Traduction de l'unique instruction de la structure
IDENTIFICATION ET TRADUCTION DE
L'INSTRUCTION
(004)
* FIN STRUCTURE
RO


R3
- Traduction des instructions de la structure
IDENTIFICATION ET TRADUCTION DES
INSTRUCTIONS DE LA STRUCTURE
(005)
* FIN STRUCTURE

R4

- Identification et traduction de l'instruction

INSTRUCTION . EQ . AFFECTATION . OR . APPEL DE PROCEDURE EXTERNE

CALL AFPRO (INDENT)

INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL

CALL COMOP (INDENT)

R2

R5

- Traduction des instructions de la structure

INSTRUCTION . EQ . AFFECTATION . OR . APPEL DE PROCEDURE EXTERNE

CALL AFPRO (INDENT)

INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL

CALL COMOP (INDENT)

INSTRUCTION . EQ . COMMENTAIRE NARRATIF

CALL COMNA (INDENT)

INSTRUCTION . EQ . INSTRUCTION A CODER EN PREMIER COLONNE

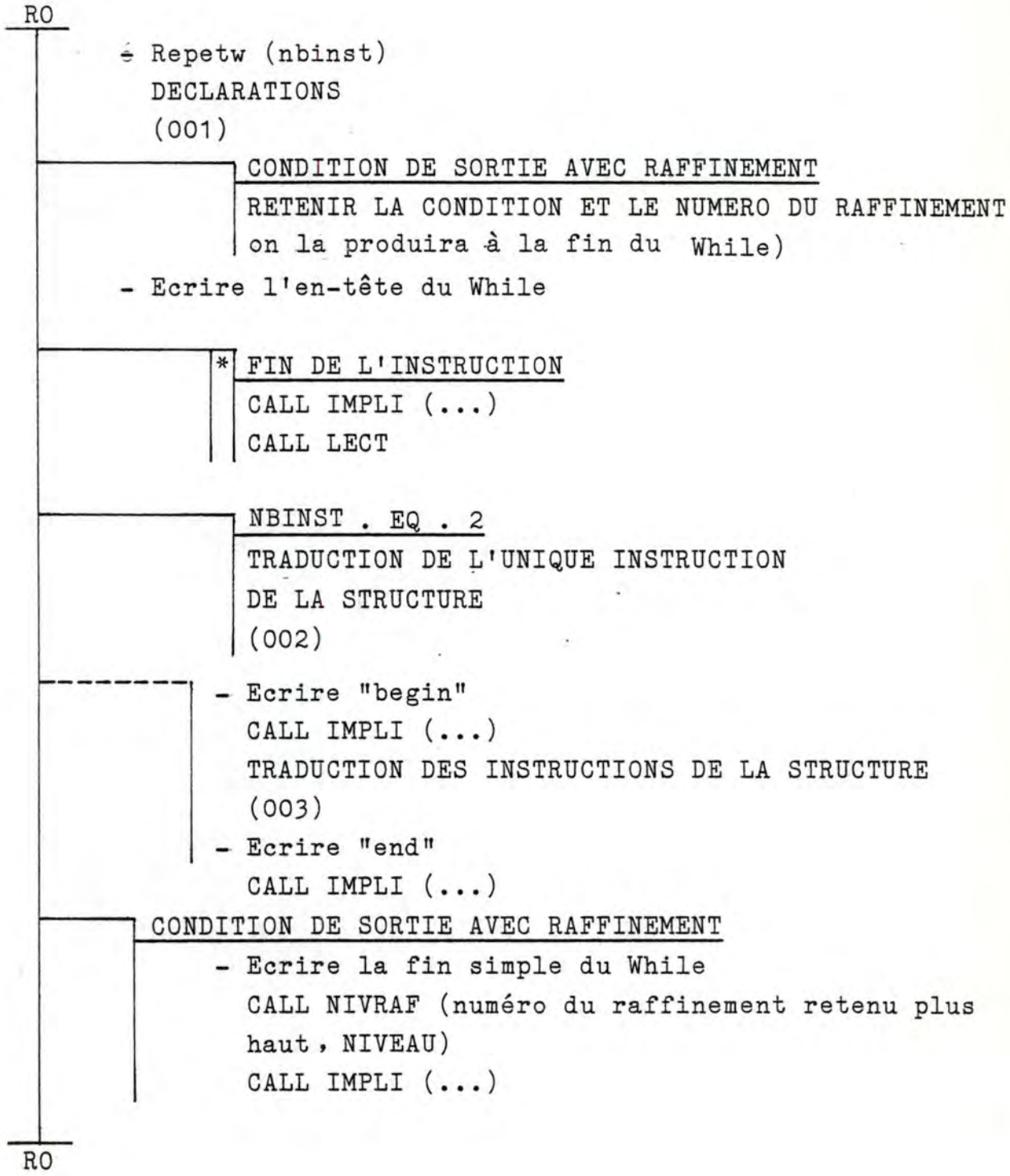
CALL FAUTCO (NUMMES)

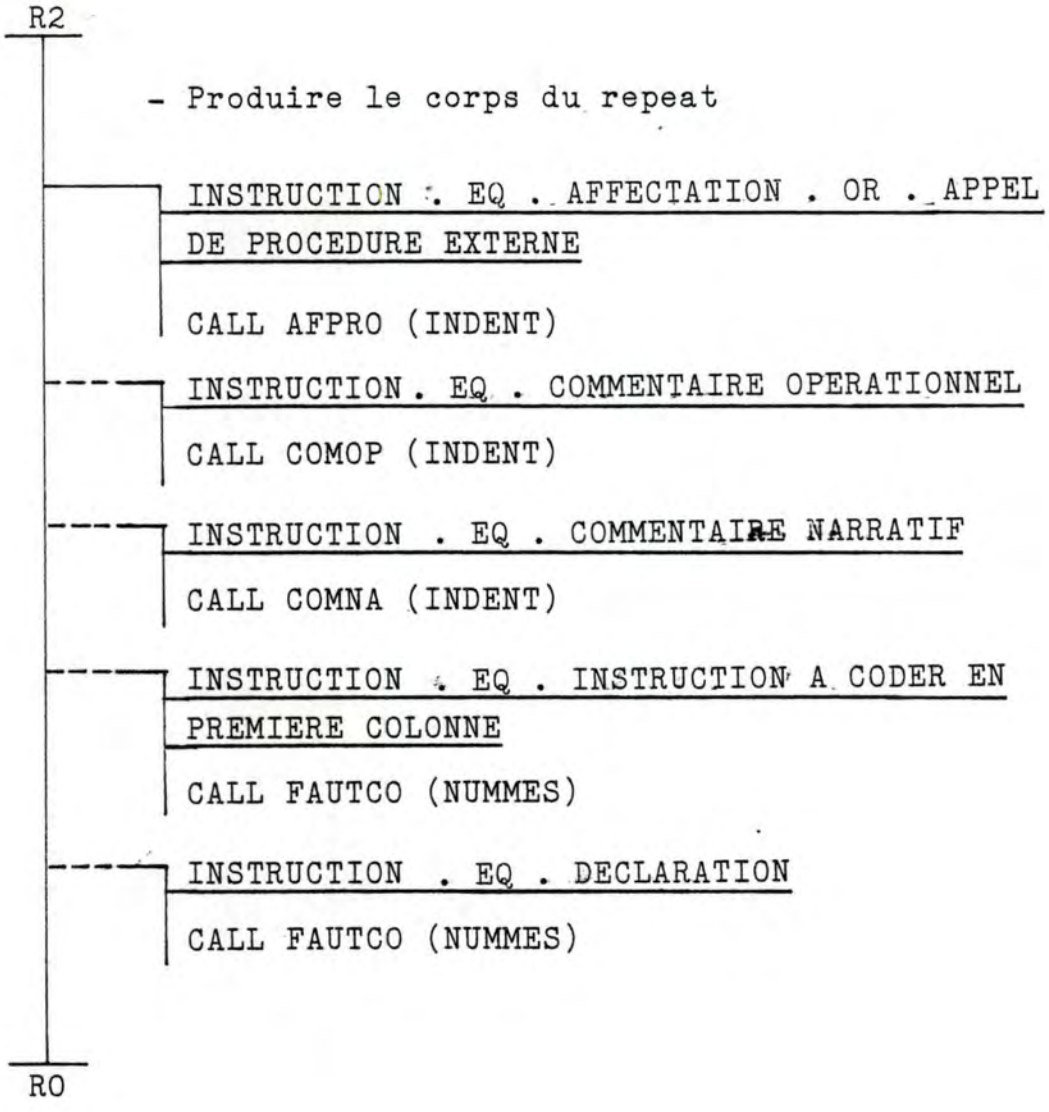
INSTRUCTION . EQ . DECLARATION

CALL FAUTCO (NUMMES)

R3

9. Conception de la routine REPETW




R2
- Produire le corps du repeat
INSTRUCTION . EQ . AFFECTATION . OR . APPEL DE PROCEDURE EXTERNE
CALL AFPRO (INDENT)
INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL
CALL COMOP (INDENT)
INSTRUCTION . EQ . COMMENTAIRE NARRATIF
CALL COMNA (INDENT)
INSTRUCTION . EQ . INSTRUCTION A CODER EN PREMIERE COLONNE
CALL FAUTCO (NUMMES)
INSTRUCTION . EQ . DECLARATION
CALL FAUTCO (NUMMES)
RO

11. Conception de la routine REPETS

R0

- Repets
  DECLARATION
  (001)
- Ecrire l'en-tête de la structure
  CALL ALPHA (...)
  CALL IMPLI (...)

PRODUIRE LE CORPS DE CETTE STRUCTURE
REPETITIVE
(002)

* FIN STRUCTURE

- Ecrire la fin de la structure
  CALL ALPHA (...)
  CALL IMPLI (...)

R0

R2

- Produire le corps de cette structure
  répétitive

INSTRUCTION . EQ . AFFECTATION . OR . APPEL DE PROCEDURE EXTERNE

CALL AFPRO (INDENT)

INSTRUCTION . EQ . COMMENTAIRE OPERATIONNEL

CALL COMOP (INDENT)

INSTRUCTION . EQ . COMMENTAIRE NARRATIF

CALL COMNA (INDENT)

INSTRUCTION . EQ . INSTRUCTION A CODER EN PREMIERE COLONNE

CALL FAUTCO (NUMMES)

INSTRUCTION . EQ . DECLARATION

CALL FAUTCO (NUMMES)

12. Conception de la routine PTSOR

RO

- Ptsor
  DECLARATIONS
  (001)

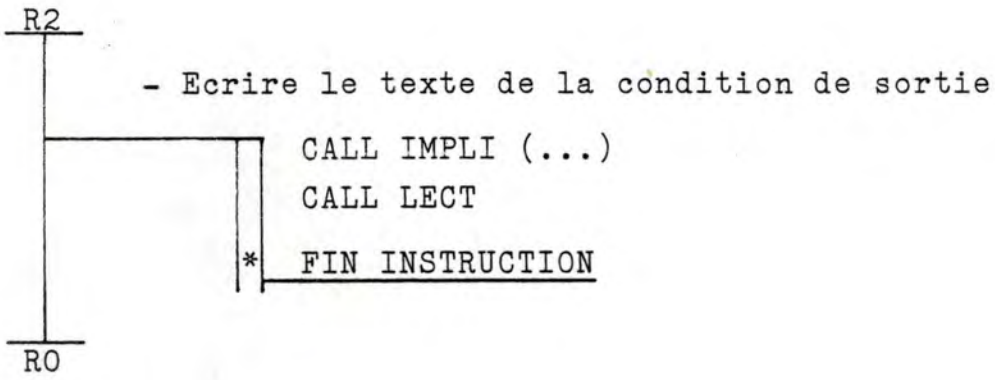
INSTRUCTION . EQ . CONDITION DE SORTIE SANS REFERENCE A UN RAFFINEMENT

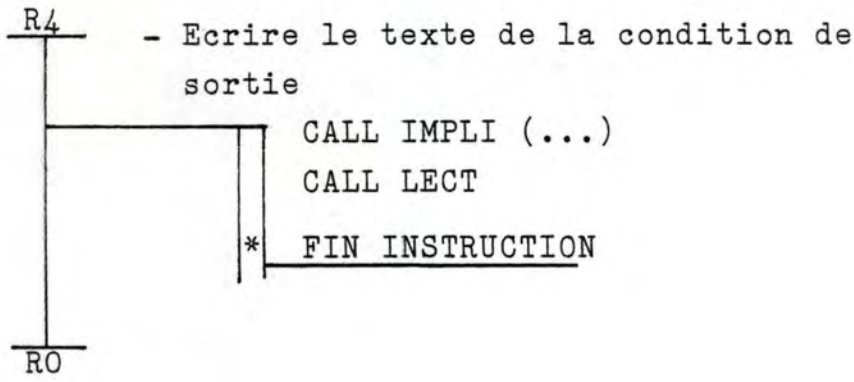
- Produire le texte PASCAL de l'u.t.P.
  "condition de sortie "
  CALL NIVRAF (numéro de raffinement de la condition de sortie)
  CALL ALPHA
- ECRIRE LE TEXTE DE LA CONDITION DE SORTIE
  (002)
  COMPTER LE NOMBRE DE "BEGIN"
  (003)

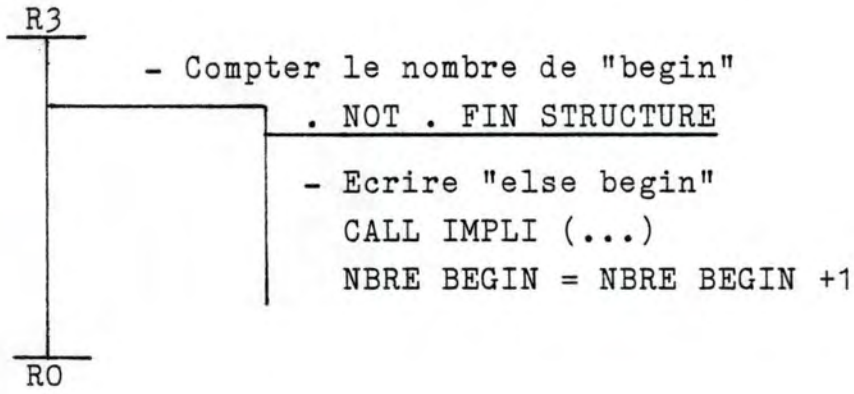
INSTRUCITON . EQ . CONDITION DE SORTIE AVEC REFERENCE A UN RAFFINEMENT

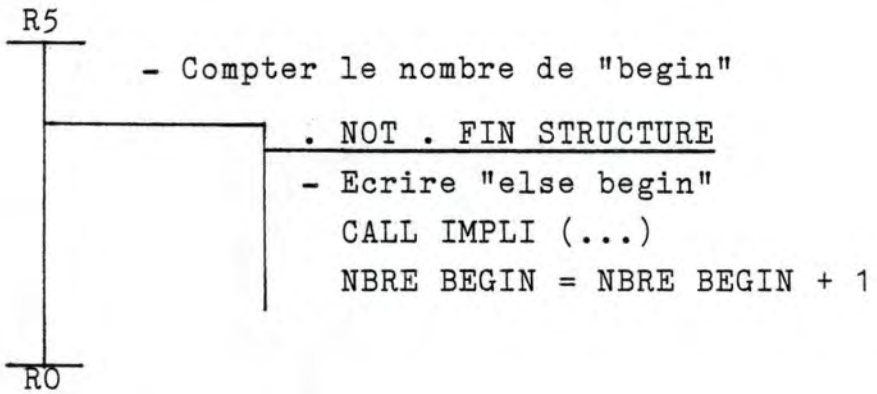
- Produire le texte PASCAL de l'u.t.P.
  "condition de sortie avec traitement d'exception"
  COMPTER LE NOMBRE DE CHIFFRES SIGNIFICATIFS DANS LE NUMERO DU RAFFINEMENT REFERECE (RAFREF).
  CALL NIVRAF (RAFREF , NIVEAU).
  COMPTER LE NOMBRE DE CHIFFRES SIGNIFICATIFS DANS LE NUMERO DE NIVEAU.
  CALL ALPHA (RAFREF, IRAFCO, NUMERO).
  CALL ALPHA (NIVEAU, INIVBK, NUMERR).
  ECRIRE LE TEXTE DE LA CONDITION DE SORTIE
  (004)
  COMPTER LE NOMBRE DE "BEGIN"
  (005)

RO

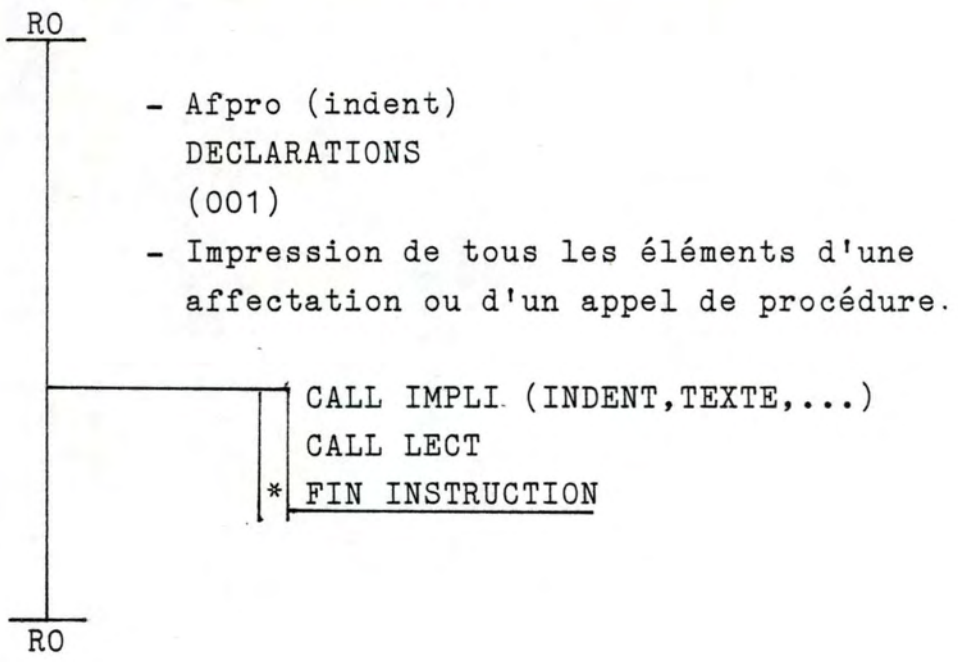

R2
- Ecrire le texte de la condition de sortie
CALL IMPLI (...)
CALL LECT
*
FIN INSTRUCTION
RO


R4
- Ecrire le texte de la condition de sortie
CALL IMPLI (...)
CALL LECT
*
FIN INSTRUCTION
RO

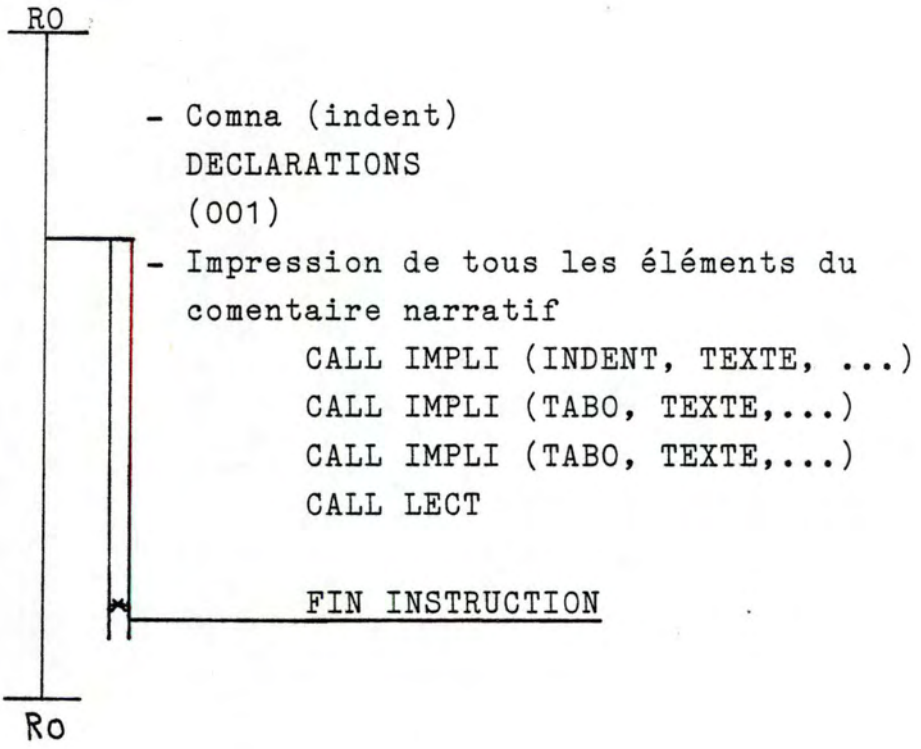

R3
- Compter le nombre de "begin"
. NOT . FIN STRUCTURE
- Ecrire "else begin"
CALL IMPLI (...)
NBRE BEGIN = NBRE BEGIN +1
RO


R5
- Compter le nombre de "begin"
. NOT . FIN STRUCTURE
- Ecrire "else begin"
CALL IMPLI (...)
NBRE BEGIN = NBRE BEGIN + 1
RO

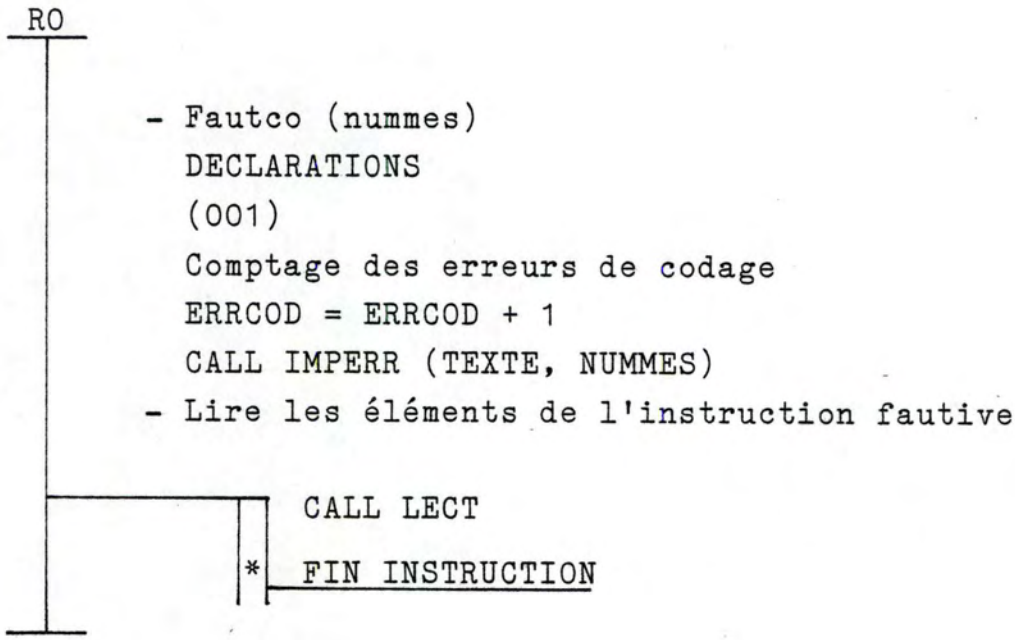
13. Conception de la routine AFPRO



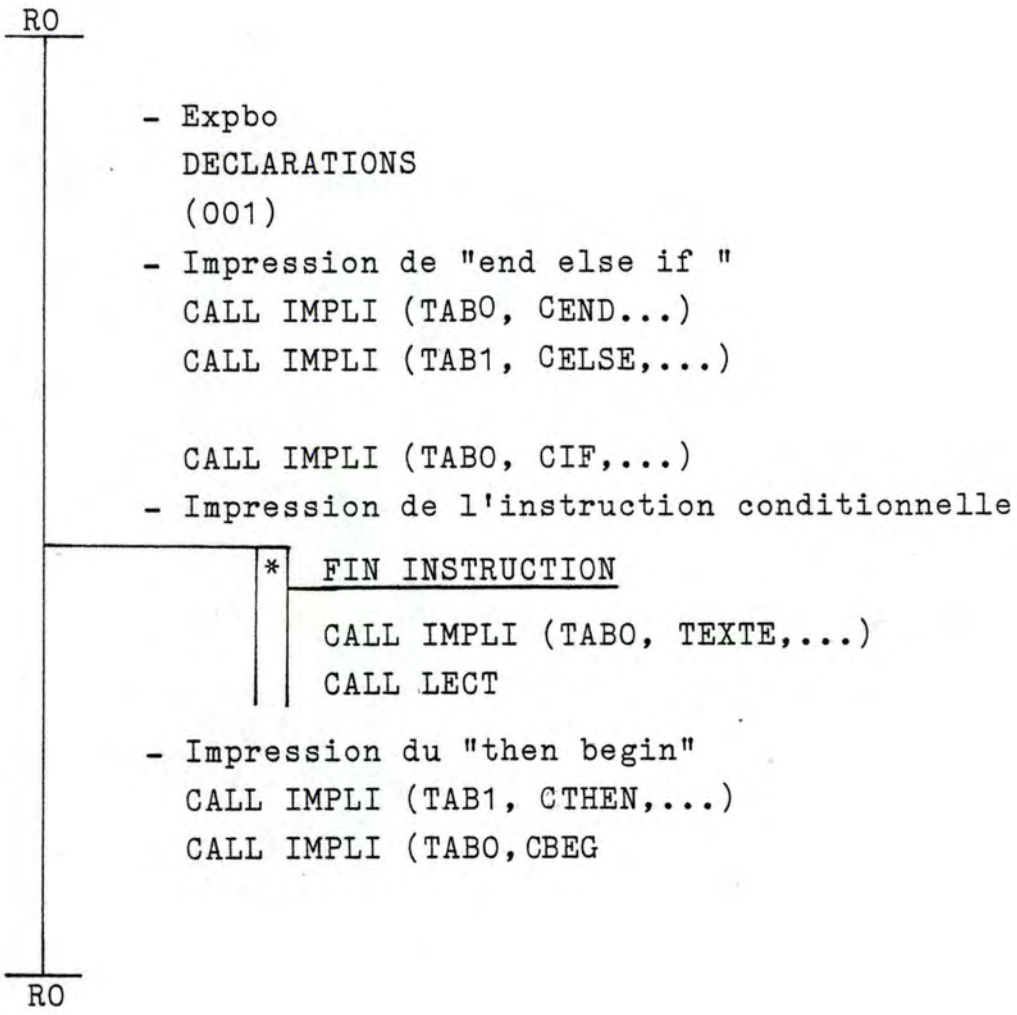
14. Conception de la routine COMNA



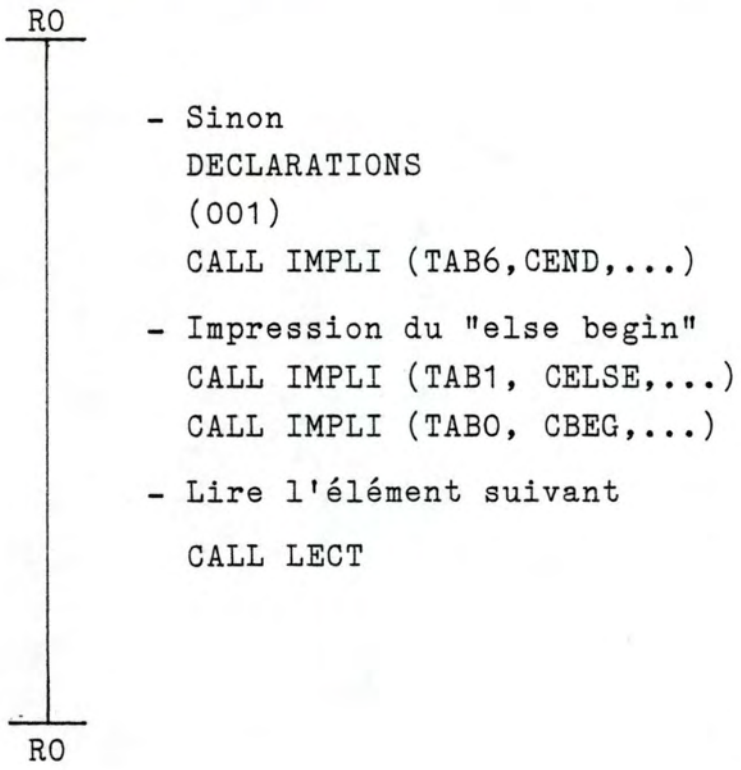
15. Conception de la routine FAUTCO



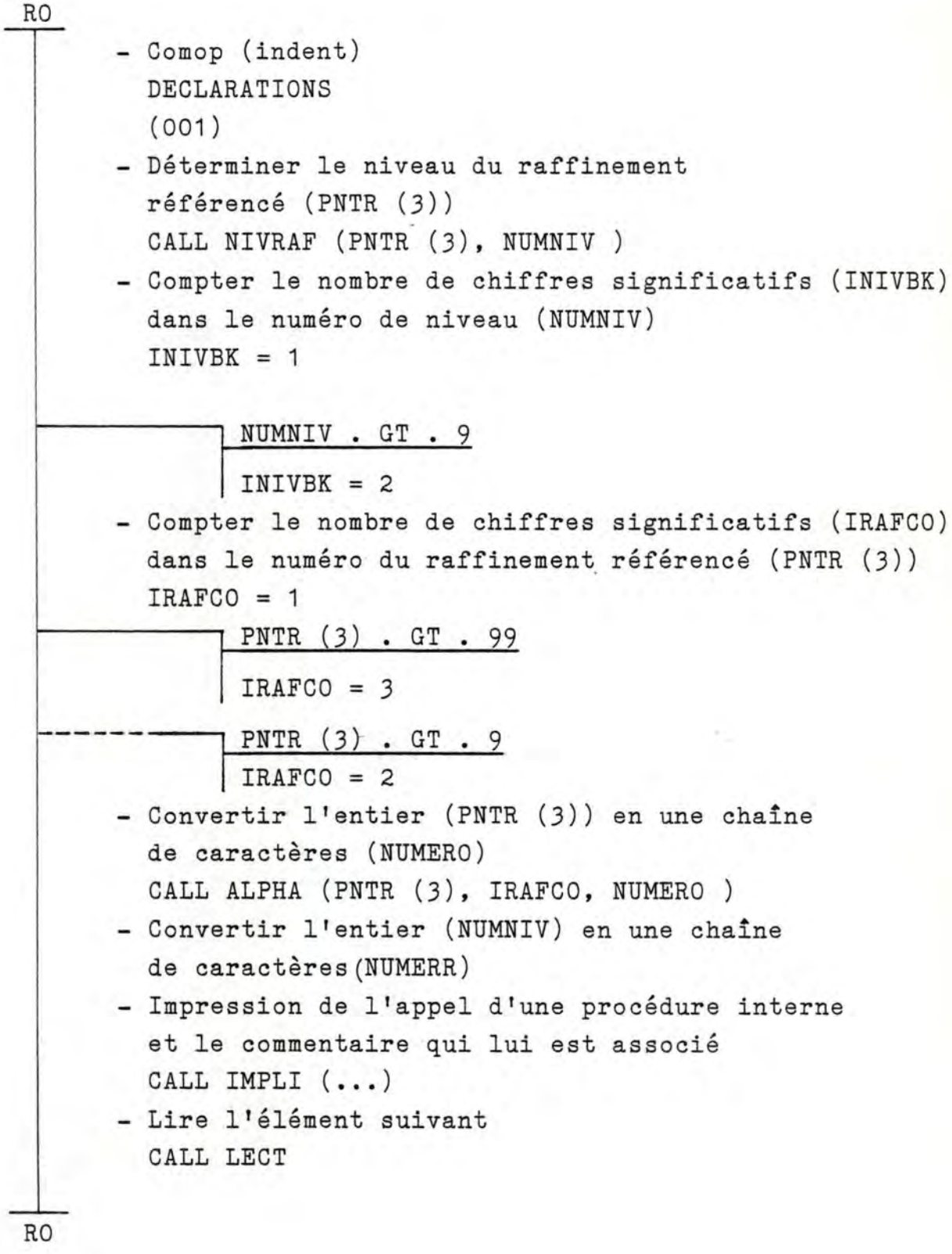
16. Conception de la routine EXPBO



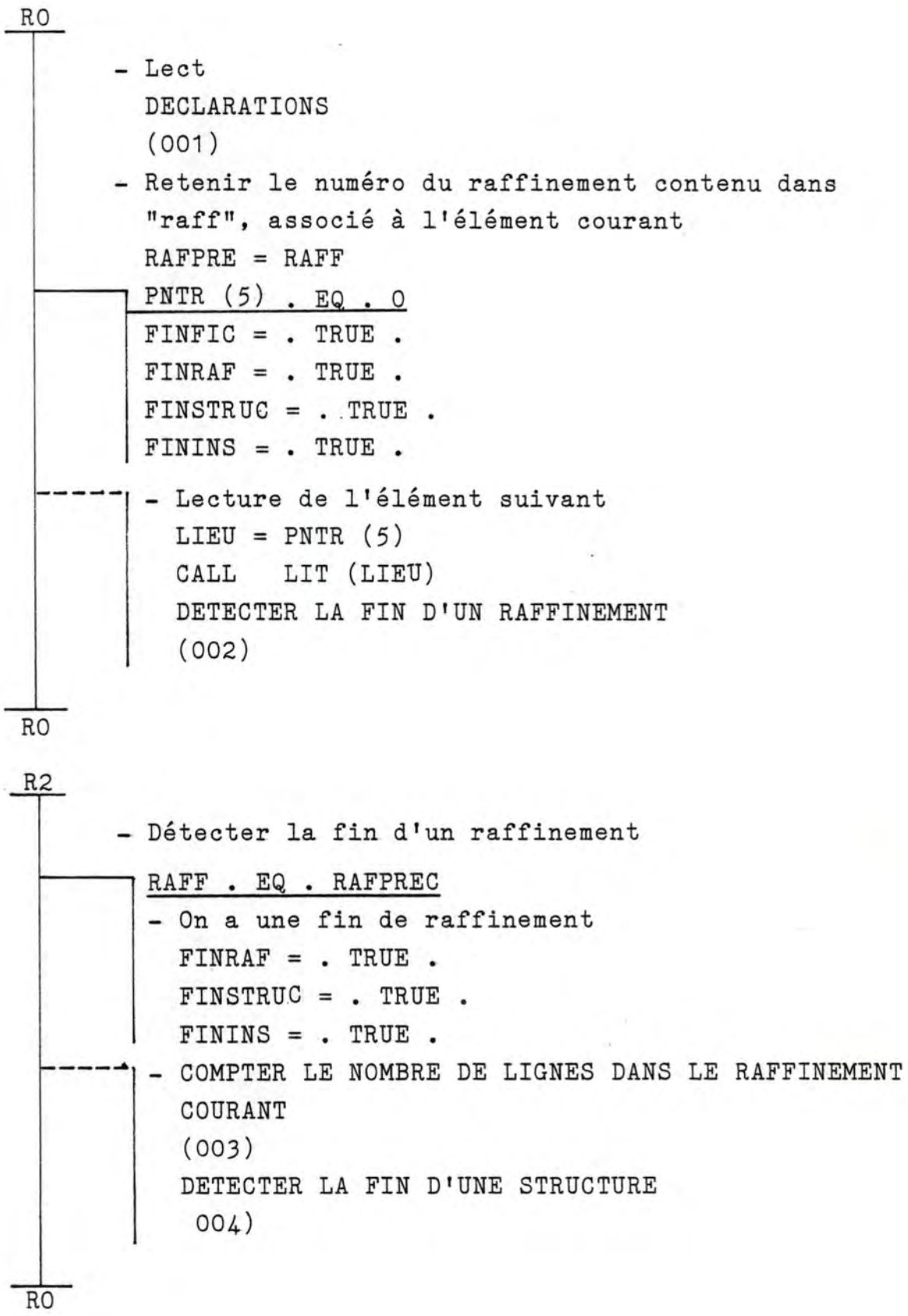
17. Conception de la routine SINON

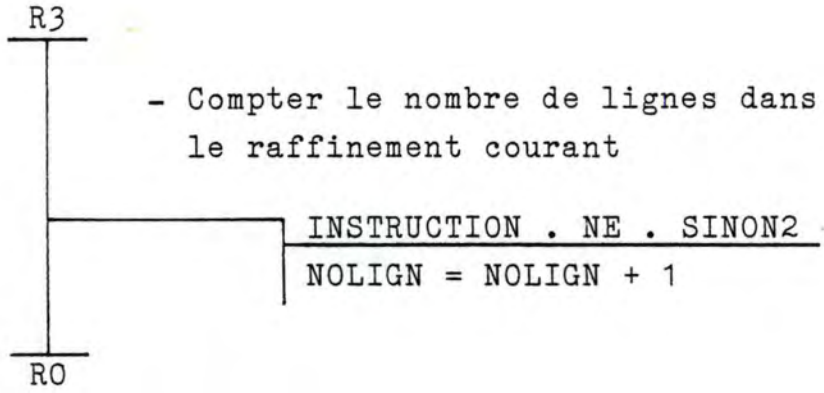


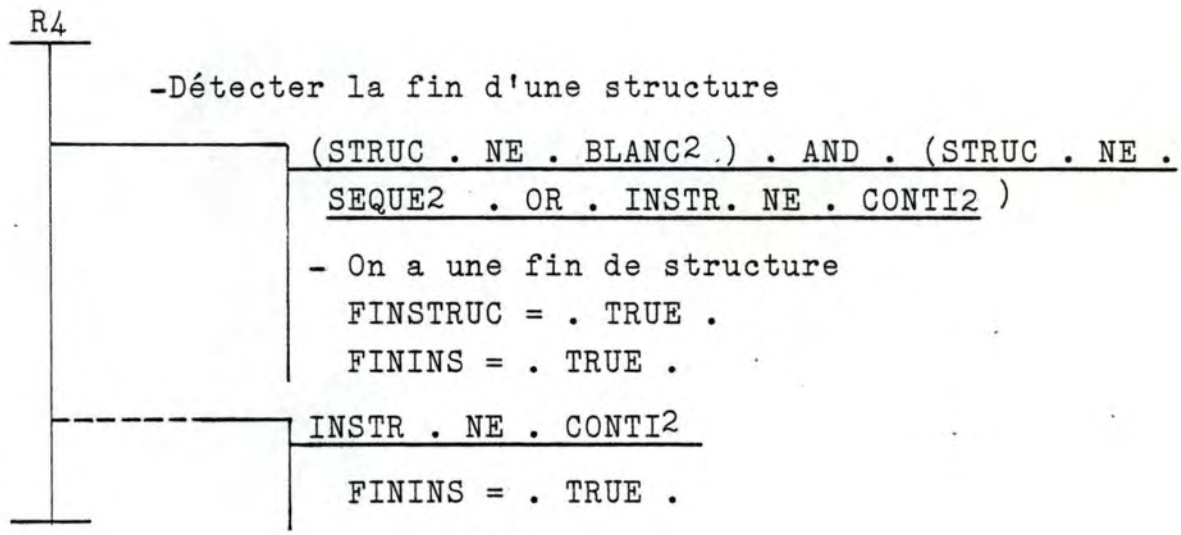
18. Conception de la routine COMOP



19. Conception de la routine LECT



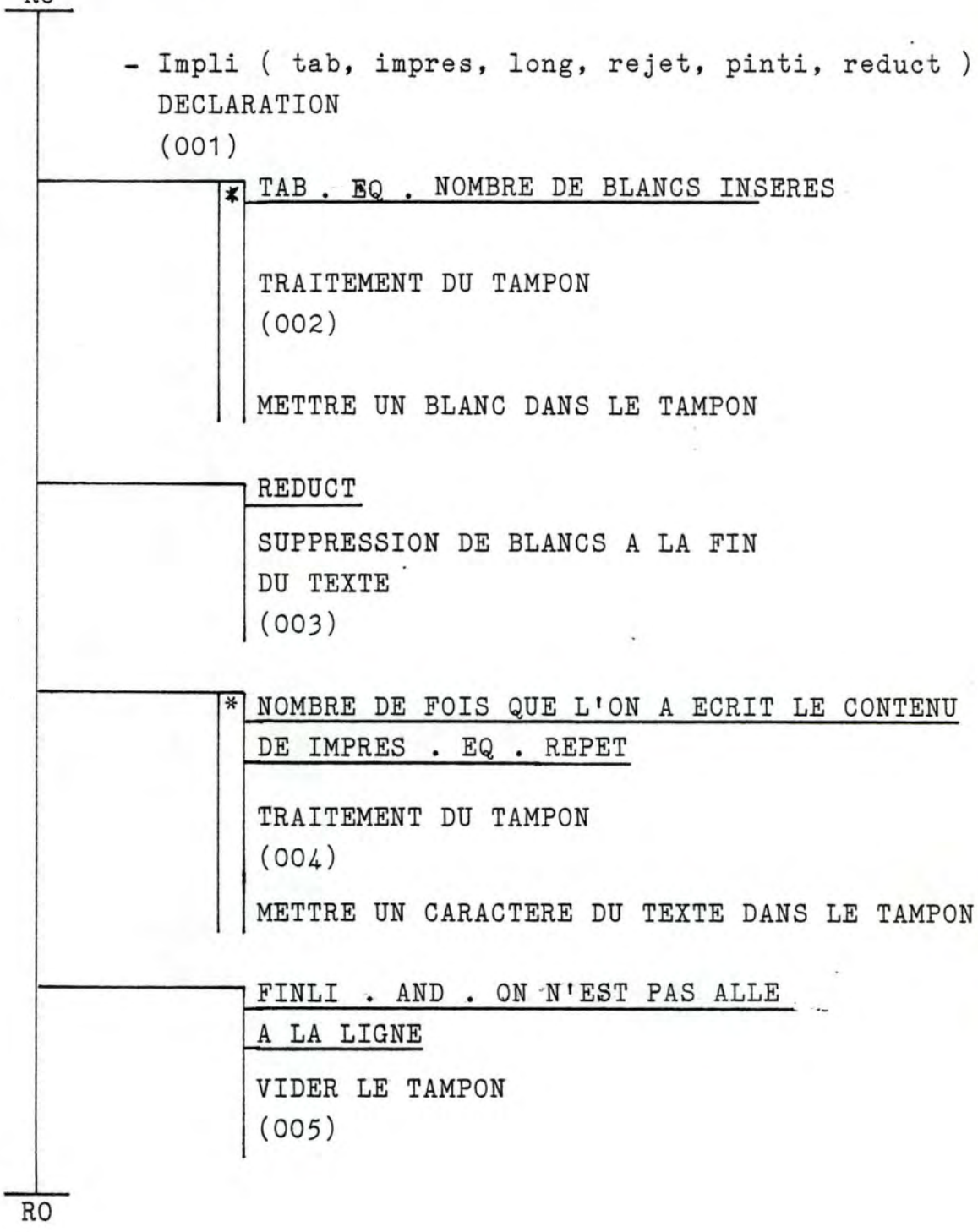




Dans l'algorithme ci-dessus, on a utilié les notations suivantes :

BLANC2 = " "
SEQUE2 = " S "
CONTI2 = " - "
SINON2 = " SINON "

20. Conception de la routine IMPLI



R2

- Traitement du tampon

TAMPON PLEIN PENDANT QUE L'ON Y MET DES BLANCS

ECRIRE LE TAMPON
BLANCHIR LE TAMPON
SE REPOSITIONNER AU DEBUT DU TAMPON

R0

R3

- Suppression de blancs à la fin du texte

* LONG . LT . 5 . OR . PLUS DE BLANCS A SUPPRIMER DANS IMPRES

COMPARER LES CARACTERES DU TEXTE
PASCAL CINQ PAR CINQ, EN COMMENCANT
PAR LA FIN

TRAITEMENT DE CES CINQ CARACTERES
(006)

R0

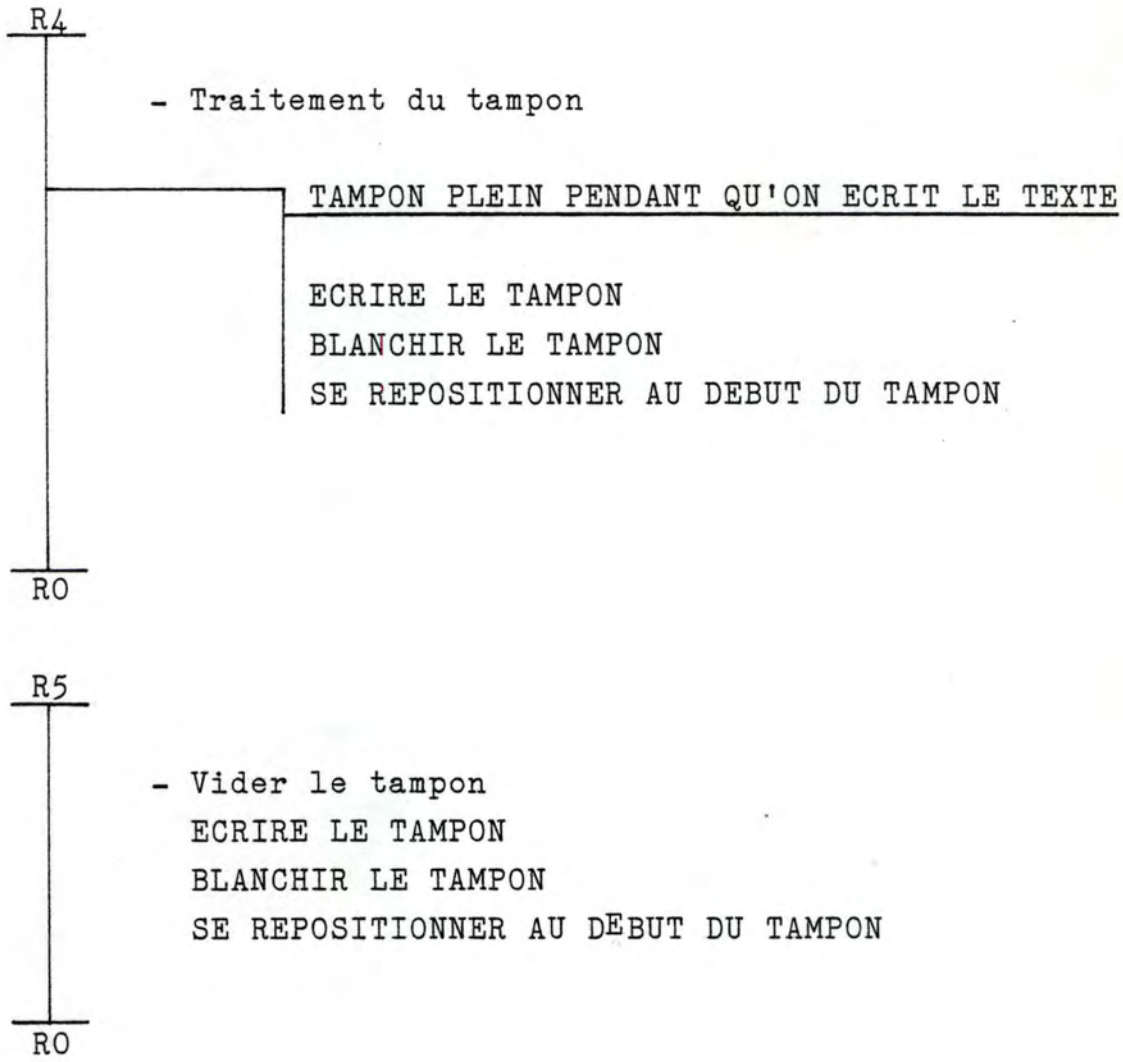
R6

- Traitement de ces cinq caractères

SI CES CINQ CARACTERES SONT DES BLANCS

DIMINUER LA LONGUEUR DU TEXTE
DE 5 BLANCS

R3


R4
- Traitement du tampon
TAMPON PLEIN PENDANT QU'ON ECRIT LE TEXTE
ECRIRE LE TAMPON
BLANCHIR LE TAMPON
SE REPOSITIONNER AU DEBUT DU TAMPON
R0
R5
- Vider le tampon
ECRIRE LE TAMPON
BLANCHIR LE TAMPON
SE REPOSITIONNER AU DÉBUT DU TAMPON
R0

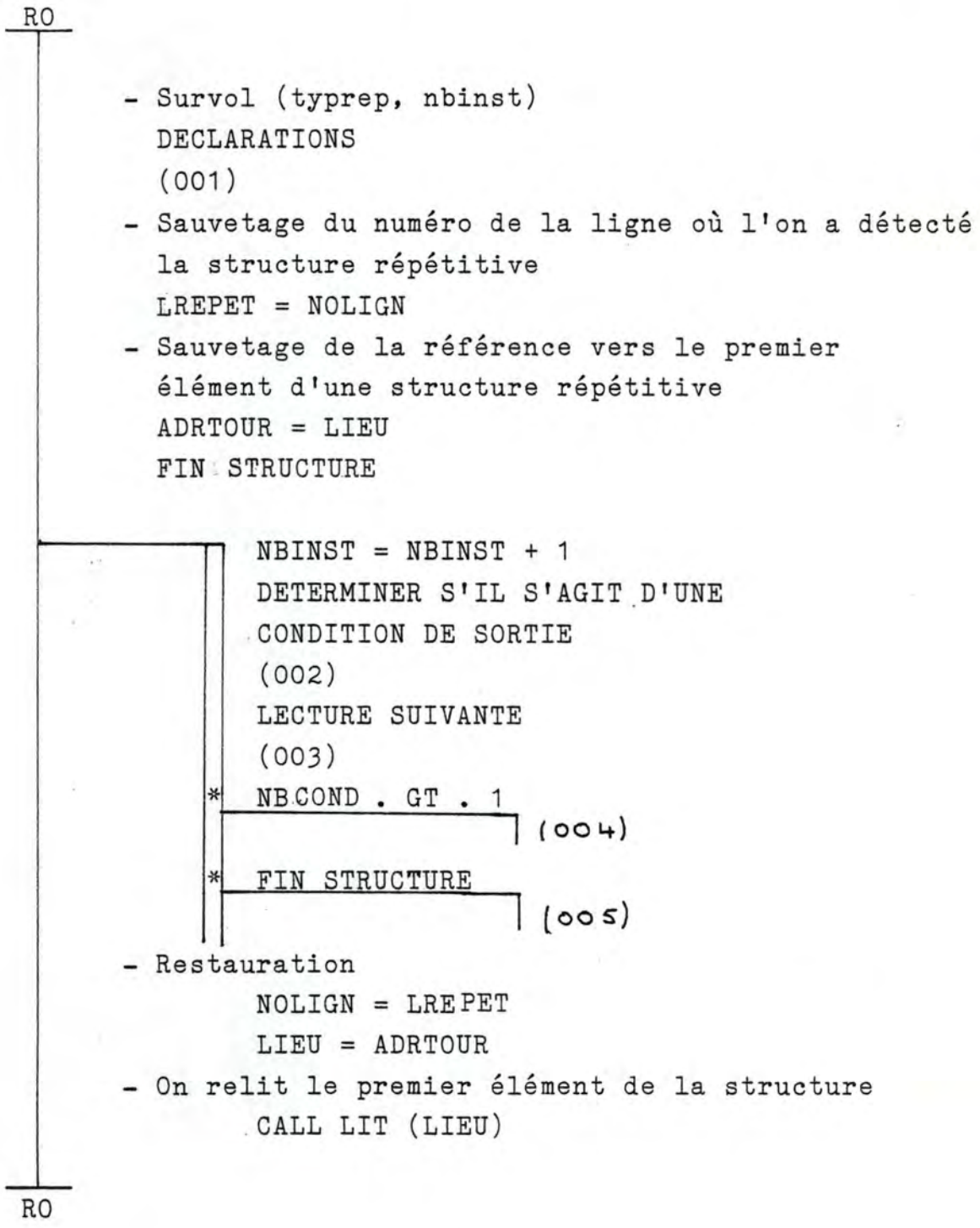
21. Conception de la routine IMPERR

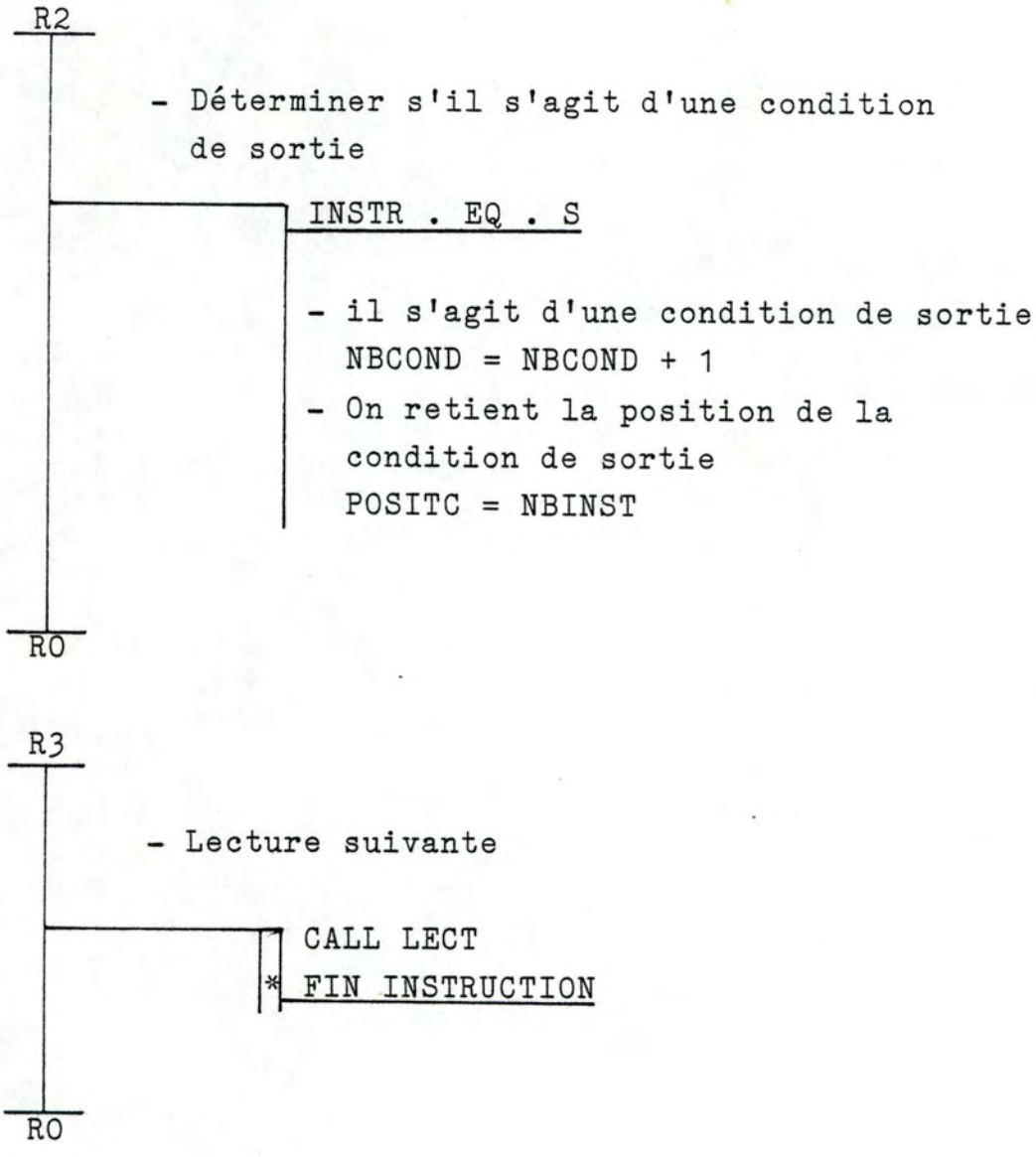
- Imperr (lerr, nomess)
  DECLARATIONS (tableaux de caractères contenant
  les messages d'erreur)
  (001)
  WRITE (LE MESSAGE ASSOCIE A NOMMES)
- Ecrire l'instruction où on a détecté l'erreur
  WRITE (LE CONTENU DE LERR)
- Ecrire où se trouve l'erreur
  WRITE ("raffinement numero :")
  WRITE (RAFF)
  WRITE ("ligne numero :")
  WRITE (NOLIGN

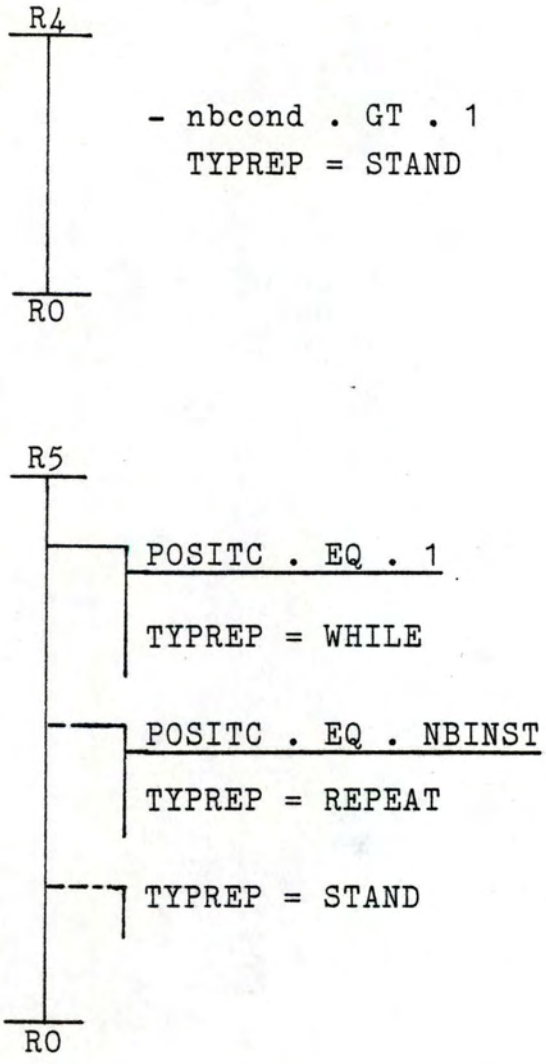
22. Conception de la routine IMPECR (nomerr)

- Impecr (nomerr)
  DECLARATIONS
  (001)
- Effacer l'écran
  CALL FPUTC
- Adresser l'écran
  CALL ADRE
  WRITE (message associé à NOMERR)
  WRITE ("arrêt de la traduction")

23. Conception de la routine SURVOL




R2
- Déterminer s'il s'agit d'une condition de sortie
INSTR . EQ . S
- il s'agit d'une condition de sortie
NBCOND = NBCOND + 1
- On retient la position de la condition de sortie
POSITC = NBINST
RO
R3
- Lecture suivante
CALL LECT
*
FIN INSTRUCTION
RO


R4
- nbcond . GT . 1
TYPREP = STAND
R0
R5
POSITC . EQ . 1
TYPREP = WHILE
POSITC . EQ . NBINST
TYPREP = REPEAT
TYPREP = STAND
R0

A N N E X E 3 :

MANUEL D'UTILISATEUR DU CODEUR PASCAL

3. MANUEL D'UTILISATEUR DU CODEUR PASCAL

Le codeur PASCAL est un des modules de Schémacode au même titre que l'éditeur, le module d'impression et le codeur FORTRAN.

On accède donc au codeur PASCAL via le mode moniteur de Schémacode.

Lorsque l'utilisateur a terminé l'édition de son programme en Schémacode orienté PASCAL, il quitte le mode éditeur et revient automatiquement au mode moniteur.
A ce moment là, il peut appeler le codeur PASCAL pour traduire son algorithme.
Pour obtenir le code PASCAL imprimé; la commande est PPASCA.
Pour obtenir le code PASCAL affiché à l'écran, la commande est LPASCA.
Pour faire la traduction de son algorithme sans imprimer ni afficher à l'écran, la commande est PASCA.

Quelle que soit la commande utilisée, trois types de résultats peuvent survenir :

(1) L'utilisateur obtient le code PASCAL.

Quelle que soit la commande utilisée, le code PASCAL se trouve dans le fichier S. nom du fichier SPC.

(2) L'utilisateur n'obtient pas le code PASCAL et le fichier S. nom du fichier SPC n'existe pas. Par contre, une liste d'erreurs apparaît à l'écran et l'utilisateur a un fichier ERR. PASCAL dans sa directory.

Ce fichier donne la liste des erreurs détectées dans le programme en pseudocode schématique et affichée à l'écran.
Le ou les messages d'erreur se compose de :
- La situation de l'erreur : Raff i, ligne j,
- Le texte de la ligne où le codeur a détecté l'erreur,
- Le type de l'erreur commise.

Ces erreurs sont appelées " erreurs de codage ".

Elles sont faciles à corriger puisque le codeur donne leur situation exacte.

(3) L'utilisateur n'obtient pas le code PASCAL et le fichier S. nom du fichier SPC n'existe pas. Un message à l'écran prévient l'utilisateur de " l'arrêt de la traduction ", et de la raison de cet arrêt.
Les erreurs qui provoquent l'arrêt de la traduction sont appelées " erreurs fatales ".

ERREURS DE CODAGE

1) " Les déclarations doivent être faites dans le raff 1 ou dans les raffinements référencés par le Raff 1 ".
L'utilisateur doit introduire les déclarations de son programme dans l'ordre prévu par le PASCAL standard et rassembler ces déclarations dans le Raff 1 ou dans les raffinements référencés par le Raff 1.
Si le codeur trouve une déclaration dans un autre raffinement (un raffinement qui développe un traitement), il émet un message d'erreur et le type de l'erreur est le message ci-dessus.

2) " le Raff 1 et les raffinements qu'il référence ne peuvent contenir que des déclarations
des commentaires
des références à d'autres raffinements"

Le Raff 1 et les raffinements qu'il référence doivent contenir toutes les déclarations mais uniquement les déclarations. Ils ne peuvent contenir aucun traitement.
Si le codeur trouve une affectation, une structure conditionnelle ou répétitive dans les raffinements de déclarations, il émet un message d'erreur et le type de l'erreur est le message ci-desus.

3) " Appel de procédure interdit dans les déclarations ".
Un appel de procédure est assimilé à un traitement, il ne peut donc pas y en avoir dans les raffinements de déclarations.

4) " Instruction en 1ère colonne interdite ".
L'instruction introduite par la commande " X " de Schémacode est spécialement conçue pour les programmes FORTRAN; elle est destinée au compilateur. La notion d'instruction destinée au compilateur n'existe pas en PASCAL; l'instruction introduite par la commande " X " de Schémacode est donc interdite.

Une fois ces erreurs corrigées dans le programme en pseudocode, le codeur doit être activé une nouvelle fois.

ERREURS FATALES

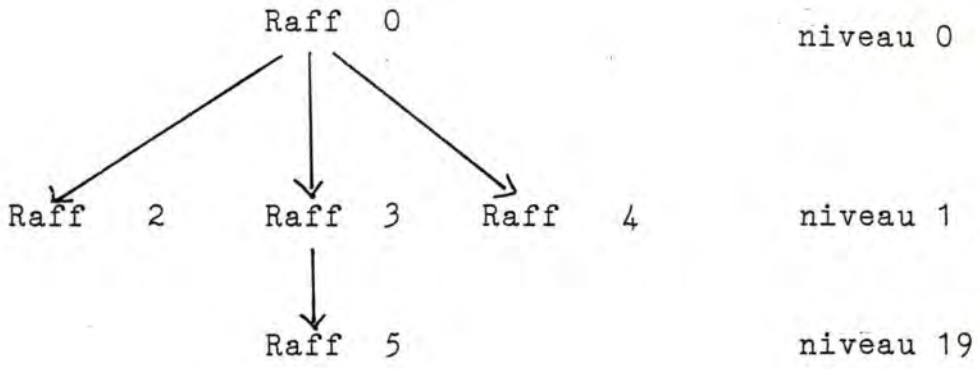
1) " Raff 1 n'est pas libéré ".

Le codeur ne sait pas traiter le raff 1 s'il n'est pas libéré. Il suffit d'éditer le raffinement qui référence le raff 1 et d'effacer par la commande " E " la référence au raff 1.

2) " Raff 1 n'existe pas ".
Le Raff 1 doit exister et contenir toutes les déclarations du programme (mais ne peut contenir aucun traitement).
Il faut donc créer le Raff 1 et ne pas oublier de le libérer.

3) " Il n'y a pas de déclarations, ni dans le Raff 1 ni dans les raffinements référencés par le Raff 1 ".
" Le Raff 1 ne contient ni déclarations, ni référence à un autre raffinement ".
Le Raff 1 doit contenir les déclarations et / ou référencer des raffinements qui contiennent les déclarations; le Raff 1 et / ou les raffinements qu'il référence doivent contenir TOUTES les déclarations.

4) " Plus de vingt références imbriquées à partir du niveau qui contient le raff 0 "



Dans le cas ou le programme s'étend sur plus de 20 niveaux, il faut soit le restructurer soit en faire plusieurs fichiers SPC; par exemple prendre une partie du programme et en faire une procédure externe.

5) " Plus de vingt références imbriquées à partir du niveau qui contient le Raff 1 "

Ce cas très rare peut survenir si les déclarations sont trop clairsemées. Il faut les restructurer.

6) " Les raffinements de votre programme sont référencés un trop grand nombre de fois ".

Cette limitation provient de l'implémentation du codeur.
Si un raffinement est référencé un grand nombre de fois, une façon d'éviter cette limite est d'en faire une procédure externe. En dernier recours, l'utilisateur peut s'adresser au responsable Schémacode pour qu'il recule cette limite.

7) " Détection d'un cycle dans les références successives aux raffinements ".

Toute forme de récursivité est interdite. Il faut la supprimer.

RECOMMANDATIONS

De ce qui précède, on peut tirer un certain nombre de recommandations :

1) Le Raff 1 et les raffinements qu'il référence doivent contenir TOUTES les déclarations à l'exclusion de tout traitement.
2) Le Raff 1 doit être libre.
3) La structure de l'algorithme doit être telle qu'il n'y ait pas plus de 20 niveaux à partir du Raff 0 et à partir du Raff 1.
4) Ne pas utiliser la commande " X ".
5) Aucune forme de récursivité n'est admise.

De plus, certaines facilités et certaines restrictions proviennent de Schémacode orienté PASCAL et du codeur PASCAL.

1) Les mots réservés " BEGIN " et " END " ne doivent pas être utilisés; ils sont tous générés par le codeur.
2) Pour développer un programme, une procédure ou une fonction : les entêtes " PROGRAM ", " PROCEDURE ", " FUNCTION " doivent être respectivement les premières instructions introduites dans le Raff 0.
3) L'instruction " VALUE " est interdite.
4) L'utilisateur ne peut pas créer et développer une procédure ou une fonction interne à un programme développé dans un fichier SPC.

A N N E X E 4 :

DEFINITION BNF DU PSEUDO-LANGAGE

RAF CODE

DEFINITION D'UN PSEUDO-LANGAGE.

Nous donnons ici la description complète du pseudo-langage défini au chapitre 5.

Nous nous intéressons à l'expression des traitements uniquement, pour lesquels le langage reprend les instructions suivantes :

- l'affectation
- l'instruction composée
- l'instruction conditionnelle
- l'instruction répétitive
- la référence à un raffinement
- le commentaire.

La grammaire est décrite au moyen d'un ensemble de règles formelles de type BNF :

- Un élément entre crochets < et > est un élément "non terminal" du pseudo-langage défini par une autre règle.
- Un élément qui n'est pas entre crochets est un élément "terminal" du pseudo-langage, qui se trouve donc tel quel dans le programme en pseudo-langage.
- Un groupe d'éléments entre accolades { et } peut être répété :
  un astérisque (*) derrière l'accolade fermante indique qu'il peut être répété zéro, une ou plusieurs fois;
  un signe plus (+) indique qu'il doit apparaître au moins une fois;
  un astérisque entouré (⊛) indique qu'il peut apparaître zéro ou une fois.

- Le symbole ::= indique que le non-terminal à sa gauche est défini par la suite d'éléments à sa droite, appelée production.
- La barre oblique / est utilisée pour séparer différentes productions correspondant à un même non-terminal (signifie donc "OU").

1 SYMBOLES DE BASE

Syntaxe :

⟨symbole de base⟩ ::= ⟨identificateur⟩ / ⟨constante⟩ / ⟨symbole spécial⟩

⟨identificateur⟩ ::= ⟨lettre⟩ / ⟨identificateur⟩ ⟨lettre⟩ / ⟨identificateur⟩ ⟨chiffre⟩

⟨lettre⟩ ::= a / b / c / d / e / f / g / h / i / j / k / l / m / n / o / p / q / r / s / t / u / v / w / x / y / z
A / B / C / D / E / F / G / H / I / J / K / L / M / N / O / P / Q / R / S / T / U / V / W / X / Y / Z

⟨chiffre⟩ ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9

⟨constante⟩ ::= ⟨entier⟩ / ⟨réel⟩ / '⟨caractère⟩' / ⟨valeur logique⟩

⟨entier⟩ ::= ⟨chiffre⟩ / ⟨entier⟩ ⟨chiffre⟩

⟨réel⟩ ::= ⟨entier⟩ ⟨exposant⟩ / ⟨nombre décimal⟩ / ⟨nombre décimal⟩ ⟨exposant⟩

⟨exposant⟩ ::= E ⟨entier⟩ / E ⟨opérateur additif⟩⟨entier⟩

⟨opérateur additif⟩ ::= +/-

⟨nombre décimal⟩ ::= ⟨entier⟩ . / . ⟨entier⟩ / ⟨entier⟩ .⟨entier⟩

⟨caractère⟩ ::= ⟨chiffre⟩ / ⟨lettre⟩ / ⟨caractère spécial⟩

⟨caractère spécial⟩ ::= + / - / * / = / < / > / (/) / ; / . / ' / / / : / , / ␣ / ! / ? / %

Remarque : " ␣ " est mis pour le caractère "espace".

2 LES CONSTRUCTIONS DU PSEUDO-LANGAGE

Parmi toutes les suites de symboles de base, certaines seulement ont une signification (sémantique) dans le cadre du pseudo-langage. Ces suites de symboles sont appelées constructions du pseudo-langage et leur sémantique correspond toujours à la spécification d'une suite d'action à effectuer.

Nous allons donc définir la syntaxe et la sémantique des constructions envisagées dans ce pseudo-langage :

- les expressions
- les instructions
- les raffinements
- les programmes.

2.1 Les expressions

Syntaxe :

⟨expression⟩ ::= ⟨expression simple⟩ / ⟨expression arithmétique⟩ / ⟨expression logique⟩

2.1.1 Les expressions simples.

Syntaxe :

⟨expression simple⟩ ::= ⟨expression de désignation⟩ / (⟨expression⟩) / ⟨constante⟩

⟨expression de désignation⟩ ::= ⟨nom de variable simple⟩ / ⟨nom de variable indicée⟩

⟨nom de variable simple⟩ ::= ⟨identificateur⟩

⟨nom de variable indicée⟩ ::= ⟨nom de tableau⟩ (⟨liste d'indice⟩)

⟨nom de tableau⟩ ::= ⟨identificateur⟩

⟨liste d'indices⟩ ::= ⟨indice⟩ / ⟨liste d'indices⟩ , ⟨indice⟩

⟨indice⟩ ::= ⟨expression arithmétique⟩

Sémantique des expressions simples

- Une expression de désignation identifie une variable. Sa valeur est la valeur de la variable qu'elle désigne.

- L'évaluation d'une constante produit toujours la même valeur :
  * Si la constante est une suite de chiffres, sa valeur est le nombre entier dont cette suite de chiffres est la représentation décimale.
  * Si la constante est de la forme 'C', sa valeur est le caractère C.
  * Si la constante est TRUE ou FALSE, sa valeur est vrai ou faux.

2.1.2 Les expressions arithmétiques

Syntaxe :

⟨expression arithmétique⟩ ::= ⟨terme⟩ ⟨opérateur additif⟩ ⟨terme⟩ / ⟨expression arithmétique⟩ ⟨opérateur additif⟩ ⟨terme⟩

⟨terme⟩ ::= ⟨facteur⟩ / ⟨terme⟩ ⟨opérateur multiplicatif⟩ ⟨facteur⟩

⟨opérateur additif⟩ ::= +/-

⟨facteur⟩ ::= ⟨expression simple⟩

⟨opérateur multiplicatif⟩ ::= * / DIV / MOD

Sémantique des expressions arithmétiques.

Une expression arithmétique est une règle pour calculer une valeur numérique. Cette valeur est obtenue en exécutant les opérations arithmétiques indiquées, sur les valeurs actuelles des éléments de l'expression.

Soit fact, un facteur
term, un terme
expr, une expression

Soit ω1, un opérateur multiplicatif
ω2, un opérateur additif

* fact est évalué comme une expression simple
* term ω1 fact : on évalue term et fact. Soit v1 et v2 les valeurs obtenues. La valeur de l'expression est v1 ω1 v2.
* expr ω2 term : on évalue expr puis term. Soit v1 et v2 les valeurs obtenues. La valeur de l'expression est v1 ω2 v2.
* ω2 term : on évalue term. Soit v1 la valeur obtenue. La valeur de l'expression est ω2 v1.

Les opérateurs +, -, * ont leur signification conventionnelle, respectivement addition, soustraction et multiplication.

Les opérateurs DIV et MOD expriment tous deux la division :
DIV donne le quotient par défaut, résultat entier de la division,

MOD donne le reste par défaut, reste de la division entière.

2.1.3 Les expressions logiques.

Syntaxe :

‹expression logique› ::= ‹conjonction› / ‹conjonction› OR ‹expression logique›

‹conjonction› ::= ‹négation› / ‹négation› AND ‹conjonction›

‹négation› ::= ‹proposition atomique› / NOT ‹proposition atomique›

‹proposition atomique› ::= ‹expression simple› / ‹expression arithmétique› ‹opérateur relationnel› ‹expression arithmétique› / ‹expression simple› ‹opérateur relationnel› ‹expression simple›

‹opérateur relationnel› ::= < / > / >= / <= / = / <>

Sématique des expressions logiques.

Une expression logique est une règle de calcul d'une valeur logique.

Soit sexpr, une expression simple,
aexpr1, aexpr2, deux expressions simples ou arithmétiques,
aprop, une proposition atomique,

nég, une négation,
conj, une conjonction,
lexpr, une expression logique,
w, un opérateur relationnel.

L'évaluation des expressions ci-dessous s'effectue de la façon suivante :

* sexpr : on évalue sexpr comme une expression simple. La valeur obtenue doit être une valeur logique.

* aexpr1 w aexpr2 : on évalue aexpr1, ensuite aexpr2; si les valeurs obtenues v1 et v2 sont comparables, la valeur de l'expression logique est v1 w v2.

* NOT aprop : on évalue aprop, soit v, sa valeur;
  si v est vrai, NOT aprop a pour valeur faux,
  si v est faux, NOT aprop a pour valeur vrai.

* nég AND conj : on évalue nég;
  si la valeur de nég est faux, c'est aussi la valeur de l'expression nég AND conj;
  si la valeur de nég est vrai, on évalue conj; la nouvelle valeur obtenue est aussi celle de l'expression nég AND conj.

* conj OR lexpr : on évalue conj;
  si la valeur de conj est vrai, c'est aussi la valeur de l'expression conj OR lexpr;
  si la valeur de conj est faux, on évalue lexpr; la nouvelle valeur obtenue est aussi celle de l'expression conj OR lexpr.

2.2 Les instructions

Une instruction est une construction du pseudo-langage qui spécifie une suite d'opérations à effectuer.

Syntaxe :

⟨instruction⟩ ::= ⟨affectation⟩ /
⟨instruction composée⟩ /
⟨instruction conditionnelle⟩ /
⟨instruction répétitive⟩ /
⟨référence à un raffinement⟩ /
⟨commentaire⟩

2.2.1 Affectation

Syntaxe :

⟨affectation⟩ ::= ⟨expression de désignation⟩ := ⟨expression⟩

Sémantique :

Soit expr, une expression,
dexpr, une expression de désignation;

l'exécution de l'instruction

dexpr := expr se déroule comme suit :

- on évalue expr, soit v sa valeur,
- on détermine la variable x, désignée par dexpr,
- on affecte à x la valeur v.

2.2.2 Instruction composée

Syntaxe :

‹instruction composée› ::= ‹instruction› /
‹instruction composée› ; ‹instruction›

Sémantique :

L'instruction composée est une liste d'instructions qui doivent être exécutées séquentiellement, dans d'ordre où elles apparaissent.

2.2.2 Instruction conditionnelle

Syntaxe :

‹instruction conditionnelle› ::=
‹clause SI› {‹clause OUSI›} * FINSI /
‹clause SI› {‹clause OUSI›} * ‹clause SINON› FINSI
‹clause SI› ::= SI ‹expression logique› ALORS ‹instruction›
‹clause OUSI› ::= OUSI ‹expression logique› ALORS ‹instruction›
‹clause SINON› ::= SINON ‹instruction›

Sémantique :

L'instruction conditionnelle provoque l'exécution de certaines instructions et le passage d'autres instructions, selon les valeurs spécifiées par les expressions logiques.

Soit cond1, cond2, ..., condi, ..., condn, des expressions logiques,
ins1, ins2, ..., insi, ..., ins(n + 1), des instructions.

Soit l'instruction conditionnelle

```
si cond1 ALORS ins1
OUSI cond2 ALORS ins2
.
.
.
OUSI condi ALORS insi
.
.
.
OUSI condn ALORS insn
SINON ins(n + 1)
```

L'exécution de l'instruction conditionnelle ci-dessus se déroule comme suit :

```
on évalue cond1, soit v1 sa valeur,
   si v1 = vrai, on exécute ins1 et l'exécution de l'instruction conditionnelle est terminée;
   si v1 = faux, alors
      on évalue cond2, soit v2 sa valeur,
         si v2 = vrai, on exécute ins2 et l'exécution
                       de l'instruction conditionnelle
                       est terminée;
         si v2 = faux, alors
         .
         .
         .
            on évalue condi, soit vi sa valeur,
               si vi = vrai, on exécute insi et l'exécution
                             de l'instruction conditionnelle est terminée;
```

si vi = faux, alors
.
.
.
on évalue condn, soit vn sa valeur,
si vn = vrai, on exécute insn et l'exécution de l'instruction conditionnelle est terminée;
si vn = faux, alors on exécute ins(n + 1) et l'exécution de l'instruction conditionnelle est terminée.

2.2.4 Instruction répétitive

Syntaxe :

‹instruction répétitive› ::=
DO {‹instruction› ;} ⊛ {‹bloc répétitif› ;} $^{+}$ OD

‹bloc répétitif› ::=
‹expression logique›==> {‹instruction›}⊛ // {‹exception›} ⊛

‹exception› ::= ‹instruction›

Sémantique :

Soit : cond1, cond2, condi, ..., condn, des expressions logiques, appelées conditions d'exécution,
ins0, ins1, ..., insi, ..., insn, des instructions,
exception 1, exception i, des instructions.

Soit l'instruction répétitive

```
DO ins0;
   cond1 ==> ins1    //  exception 1   ;
   cond2 ==> ins2    ;
   .
   .
   .
   condi ==> insi    //  exception i   ;
   .
   .
   .
   condn ==> insn    ;
OD
```

Pour tout i, insi et/ou exception i peuvent être l'instruction vide. Mais il existe au moins un j tel que insj soit différent de l'instruction vide.

L'exécution de l'instruction répétitive se déroule comme suit :

```
on exécute ins0
on évalue cond1, soit v1 sa valeur,
   si v1 = faux, on exécute exception 1 et l'exécution de l'
                 instruction répétitive est terminée;
   si v1 = vrai, alors
      on exécute ins1
      on évalue cond2, soit v2 sa valeur,
         si v2 = faux, l'exécution de l'instruction répétitive
                       est terminée;
         si v2 = vrai, alors
            on exécute ins2
            .
            .
            .
               on évalue condi, soit vi sa valeur,
                  si vi = faux, on exécute exception i et l'
                                exécution de l'instruction répétiti-
                                ve est terminée;
```

Le Raff 0 est exécuté en premier lieu dès le lancement de l'exécution du programme et la fin de l'exécution du Raff 0 coïncide avec la fin de l'exécution du programme.

2.4 Le programme

Syntaxe :

⟨programme⟩ ::= PROGRAM ⟨nom du programme⟩ ;
(déclaration)
⟨suite de raffinements⟩ .

⟨suite de raffinements⟩ ::= ⟨raffinement⟩ /
⟨raffinement⟩ ; ⟨suite de raffinements⟩

⟨nom du programme⟩ ::= ⟨identificateur⟩

Sémantique :

L'exécution d'un programme consiste à exécuter le Raff 0 de ce programme.

Remarque : nous avons mis déclaration entre parenthèses car elles sont à cette place là mais nous ne les traitons pas dans le cadre de ce pseudo-langage.

A N N E X E 5 :

ALGORITHMES DU CODEUR ASSOCIE AU SYSTEME ALTERNATIF

Dans cette annexe, nous donnons les algorithmes du codeur associé à RAFCODE; ces algorithmes sont écrits en RAFCODE. Mais quelques préambules sont nécessaires :

- Nous décrivons les types de données et les noms des variables utilisés dans les algorithmes; RAFCODE ne traite pas les déclarations, nous nous sommes donc basées sur les types PASCAL.
- La notion de procédure interne n'a pas été définie dans RAFCODE. Or, tout le monde connait l'intérêt des procédures avec paramètres; mais pour éviter toute confusion avec la notion de raffinement, il serait plus intéressant d'introduir les raffinements avec paramètre plutôt que les procédures internes. Vu que nous ne traitons pas les déclarations et que la RI utilisée empêche toute analyse syntaxique et sémantique en cours de codage, nous ne pouvons qu'esquisser le genre de programmation auquel l'ajoût des raffinements avec paramètres donnerait bien. C'est pourquoi, après chaque référence à un raffinement, nous avons mis entre parenthèses la référence avec paramètres.

Structure des données

```
TYPE    PTXT = ↑TEXTE
        TEXTE = RECORD
                   LGTXT : INTEGER;
                   CONTENU : PACKED ARRAY [1.. 100] OF CHAR;
                   TSVT : PTXT
                END;
        PCONS = ↑CONSTR;
        IDENCONSTR = (PRG, NIV, RAFF, AFFECT, COMMENT, REFER,
                      COMPOS, CONDIT, SIOUSI, REPET, BLOC );
        CONSTR = RECORD
           CASE IDCONS : IDENCONSTR OF
             PRG : ( NOMPRG : PTXT; PREMMIV : PCONS );
             NIV : ( NUMNIV : INTEGER; PREMRAF :PCONS,
                     SVTNIV : PCONS );
             RAFF : ( NUMRAF : INTEGER; NOMRAF : PTXT;
                      INSRAF, SVTRAF : PCONS );
             AFFECT : ( TEXRAFF : PTXT; SVTAFF : PCONS );
             COMMENT : ( TEXCOM : PTXT; SVTCOM : PCONS );
             REFER : ( NUMREF : INTEGER; SVTREF : PCONS );
             COMPOS : ( PREMINS : PCONS );
             CONDIT : ( INSSI, INSOUSI, INSINON, SVTCONDIT :
                        PCONS );
             SIOUSI : ( TEXCOND : PTXT; INSALORS, SVTOUSI :
                        PCONS );
             REPET  : ( PREMINS, BLOCREP, SVTREP : PCONS );
             BLOC   : ( CONDR, INSR, EXCEPTION, SVTBLOC :
                        PCONS)

                        END;
```

Définition des variables utilisées dans les algorithmes :

° Variables de type PCONS

PREMIER : pointeur initial vers le premier enregistrement de la RI d'un programme.

NIVEAU : pointeur vers l'enregistrement relatif au niveau qu'on traite.

RAFFINEMENT : pointeur vers l'enregistrement relatif au raffinement qu'on traite.

x, y, z, : dans chaque raffinement, représentent des pointeurs vers des instructions ou vers des morceaux d'instruction.

° Variables de type PTXT

T : pointeur vers un texte à écrire.

° Variables globales

NIVIND : niveau d'indentation de code produit

TYPEREPET : type de la répétitive à traduire (WHILE, REPEAT, ou STANDARD)

IMBRIC : niveau d'imbrication des répétitives standards.

```
PROGRAM TRADRAFCODE;
( Déclarations )
RAFF  000  % TRT PROGRAMME   RAFCODE %;
%  Initialisations %
%  niveau d'indentation du code produit %;
NIVIND := 1;
% degré d'imbrication de répétitives %;
IMBRIC := 1;
% Ecrire le nom du programme %;
T : = PREMIER ↑  . NOMPRG;
REF  701; % ECRITURE    T %;
NIVEAU := PREMIER↑  .   PREMNIV↑ .  SVTNIV;
% Traitement des niveaux 1 à N du programme %;
DO NIVEAU <> NIL ==> REF    101; % TRTNIV ( NIVEAU ) %;
                     NIVEAU := NIVEAU ↑    . SVTNIV
OD;
% Traitement du Raff 0 : programme principal %;
REF 701: % ECRITURE en-tête du programme principal %;
NIVEAU :=  PREMIER ↑  .  PREMNIV;
REFINSTR := NIVEAU ↑ . PREMRAF ↑  . INSRAF;
REF 301; TRTINS ( REFINSTR, RET ) %;
REF 701; % ECRITURE fin programme principal %
FAR;
RAFF 101 % TRTNIV (x) : traitement d'un niveau %;
% x : référence au niveau à traiter %;
REF 601; % COMECR en-tête niveau %;
RAFFINEMENT := x ↑ .  PREMRAF;
% Traitement des raffinements du niveau %;
DO RAFFINEMENT <>NIL  ==>
   REF 201;   % TRTRAF ( RAFFINEMENT ) %;
   RAFFINEMENT : = RAFFINEMENT ↑. SVTRAF
OD;
REF 701; % ECRITURE   fin niveau %
FAR;
```

```
RAFF 201  % TRTRAF (x) : Traitement d'un raffinement %;
% x : référence au raffinement à traiter %;
% Ecrire le nom de la procédure %;
T := x↑. NOMRAF;
REF 601; % COMECR (T) %;
% Ecrire l'en-tête de la procédure avec x↑. NUMRAF %;
% Traduction du corps de la procédure %;
% Incrémenter le niveau d'indentation de 1 %;
REFINSTR : = x↑.  INSRAF;
REF 301;   % TRTINS ( REFINSTR, RET ) %;
% décrémenter le niveau d'indentation de 1 %;
REF 701; % ECRITURE fin procédure %;
FAR ;

RAFF 301  % TRTINS ( x, RET ) : identification des
            instructions %;
% x : référence à l'instruction à identifier %;
% Isoler le type de l'instruction %;
TYPINS : = x↑. IDCONS;
SI  TYPINS = AFFECT
    ALORS  REF 402; % TRTAFFECT (x) %;
           RET : = x↑. SVTAFF
OUSI  TYPINS = COMMENT
    ALORS  REF 403; % TRTCOMMENT (x) %;
           RET : = x↑. SVTCOM
OUSI  TYPINS = REFER
    ALORS  REF 404; % TRTREFER (x) %;
           RET : = x↑. SVTREF
OUSI TYPINS = CONDIT
    ALORS  REF 405; % TRTCONDIT (x) %;
           RET : = x↑. SVTCONDIT
OUSI  TYPINS = REPET
    ALORS  REF 406; % TRTREPET (x) %;
           RET : = x↑. SVTREP
OUSI TYPINS = COMPOS
    ALORS  REF 401; % TRTCOMPOS (x) %;
FINSI;
FAR;
```

```
RAFF 401 % TRTCOMPOS (x) : Traitement d'une instruction
           composée %;
% x : référence à l'instruction composée à traiter %;
y : = x↑ . PREMINS;
DO    y <> NIL ==> REF  301; % TRTINS ( y, RET ) %;
                   RET  701; % ECRITURE ";" %;
                   y : = RET
OD
FAR;

RAFF 402 % TRTAFFECT (x) : Traitement des affectations  ;
% x : référence à l'affectation à traiter %;
T : = x↑ .  TEXAFF;
REF  602; % TXTECR  T %
FAR;

RAFF 403 % TRTCOMMENT (x) : Traitement des commentaires %;
% x : référence au commentaire à traiter %;
T : =  x↑ .   TEXCOM;
REF  601;   % COMECR    T  %
FAR;

RAFF 404  % TRTREFER (x) : Traitement d'une référence à
          raffinement %;
% x : référence à l'instruction à traiter %;
% Ecrire le nom de la procédure appelée %;


REF 601; % COMECR   T %;
% Ecrire l'appel de procédure %;
NUM : = x↑ .  NUMREF;
REF 701; % ECRITURE " p " NUM %;
FAR;
```

```
RAFF 405        % TRTCONDIT (x) : Traitement d'une instruction
                conditionnelle %;
% x : référence à l'instruction conditionnelle à traiter %;
% Traitement de la première alternative %;
y : = x↑ . INSSI;
REF 502; % SIOUSI (y) %;
% Traitement des alternatives OUSI %;
y : = x↑ . INSOUSI;
% Initialiser le nombre d'indentations successives dans
  la conditionnelle %;
NBRIND : = 0;

DO   y <> NIL ==>
     REF 701; % ECRITURE " ELSE " %;
     NIVIND : = NIVIND + 1;
     NBRIND : = NBRIND + 1;
     REF 502;  % SIOUSI (y) %;
     y : = y↑ .  SVTOUSI
OD;
% Traitement de l'alternative SINON %;
y : = x↑ . INSINON;
Si   y <> NIL
     ALORS    REF 701; % ECRITURE " ELSE " %;
              NIVIND : = NIVIND + 1;
              NBRIND : = NBRIND + 1;
              REF 604; % BEGINEND (y) %
FINSI;
NIVIND : = NIVIND - NBRIND;
FAR;

RAFF 502 % SIOUSI (x) : Traitement d'une alternative
           d'une conditionnelle %;
% x : référence à une alternative " SI... ALORS " ou
     à une alternative " OUSI ... ALORS " %;
% Ecriture de " IF condition " %;
REF 701; % ECRITURE " IF " %;
T : = x↑ . TEXCOND;
REF 602;  %  TXTECR ( T ) %;
% Traitement de l'instruction consécutive ALORS %;
```

```
REF 702;  %  ECRITURE " THEN " %;
y : = x↑. INSALORS;
REF 604;  % BEGINEND (y) %;
FAR;

RAFF 604  % BEGINEND (x) : encadre une instruction composée
          d'un BEGIN et d'un END %;
SI  x↑.  IDCONS = COMPOS
     ALORS     REF 701; % ECRITURE " BEGIN " %;
               NIVIND : = NIVIND + 1;
               REF 301; TRTINS (x,RET);
               NIVIND : = NIVIND - 1;
               REF 701; % ECRITURE " END " %;
     SINON     REF 301; % TRTINS (x, RET);
FINSI;
FAR;

RAFF 406 % TRTREPET (x) Traitement des instructions
           répétitives %;
% x : référence à l'instruction répétitive à traiter %;
% Identification du type de répétitive à produire %;
               REF 503; % SURVOL (x, TYPEREPET) %;
SI          TYPEREPET = S
     ALORS     REF 504; % REPS (x) %;
OUSI        TYPEREPET = R
     ALORS     REF 505; % REPR (x) %;
OUSI        TYPEREPET = W
     ALORS     REF 506; % REPW (x) %;
FINSI;
FAR;
```

```
RAFF 503   %   SURVOL ( x, typerepet ) : identification du
               type d'une instruction répétitive %;
% x : référence à l'instruction répétitive à identifier %;
SI    x↑ .  BLOCREP ↑ .  SVTBLOC  =  NIL
      ALORS  % il y a un seul bloc répétitif %;
             SI  x↑ .  PREMINS <> NIL   AND
                 x↑ .  BLOCREP ↑ .  INSR =  NIL
                 ALORS % il y a une première instruction
                         mais pas d'instruction dans le
                         bloc répétitif %;
                         typerepet : = R
             OUSI : x ↑ .  PREMINS = NIL   AND
                    x ↑ .  BLOCREP ↑ .  INSR <> NIL
                    ALORS % il n'y a pas d'instruction en
                    tête de la répétitive et il y a
                    une instruction dans le seul bloc
                    répétitif %;
                    typerepet : = W
             SINON  typerepet : = S
             FINSI;
      SINON  typerepet : = S
FINSI
FAR;
```

```
RAFF 505 % REPR (x) : Traitement des instructions
         répétitives ayant une instruction en tête
         de la répétitive et un seul bloc répétitif sans
         instruction %;
% x : référence à l'instruction répétitive à traiter %;
REF 701; % ECRITURE " REPEAT " %;
NIVIND : = NIVIND + 1;
y : = x↑ .  PREMINS;
REF 301;  % TRTINS (y, RET);
NIVIND : = NIVIND - 1;
REF 701; % ECRITURE " UNTIL (NOT "%;
T : = x↑ . BLOCREP ↑ . CONDR;
```

```
REF 602;  % TXTECR  T %;
REF 701; %  ECRITURE ")";
% Traitement du cas d'exception %;
y : = x↑ .  BLOCREP↑ .  EXCEPTION;
SI   y <> NIL
     ALORS    REF 301; TRTINS (y, RET) %;
FINSI;
FAR;

RAFF 506 % REPW (x) : Traitement des instructions
         répétitives n'ayant pas d'instruction en tête
         de la répétitive et ayant un seul bloc répétitif
         avec instruction %;
% x : référence à l'instruction répétitive à traiter %;
REF 701;  % ECRITURE " WHILE " %;
% Ecrire la condition %;
y : = x↑ . BLOCREP;
T : = y↑ .  CONDR;
REF 602; % TXTECR  T%;
REF 701;  % ECRITURE " DO " %;
NIVIND : = NIVIND + 1;
z : = y↑ .  INSR;
REF 604;  % BEGINEND ( z ) %;
NIVIND : = NIVIND - 1;
z : = y↑ .  EXCEPTION;
SI    z <> NIL
     ALORS    REF 30 ;  % TRTINS ( Z, RET ) %;
FINSI;FAR;

RAFF 504  %  REPS (x) : Traitement des instructions
          répétitives ayant une instruction en tête et
          au moins une instruction dans le premier bloc
          répétitif ou ayant au moins deux blocs
          répétitifs %;
% x : référence à l'instruction répétitive à traduire %;
IMBRIC : = IMBRIC + 1;
```

```
% Ecrire l'en-tête de la répétitive %;
REF 701; % ECRITURE " LITNU [NUMRAF, IMBRIC] : = TRUE; "%;
REF 701; % ECRITURE " REPEAT " %;
NIVIND : = NIVIND + 1;
y : = x↑ .  PREMINS;
SI    y <> NIL
     ALORS    REF 301; % TRTINS (y, RET) %;
FINSI;
% Traitement des blocs répétitifs %;
y : = x↑ . BLOCREP;
REF 603; % TRTBLOC (y) %;
% Ecrire la fin de la répétitive %;
REF 701; ECRITURE " UNTIL LITNU [NUMRAF, IMBRIC] = FALSE "%;
IMBRIC : = IMBRIC - 1
FAR;

RAFF 603 % TRTBLOC (x) : Traitement des blocs répétitifs %;
% x : référence à un bloc répétitif %;
% Ecrire la condition %.
T : = x↑ .  CONDR;
REF 701; % ECRITURE " IF " %;
REF 602; % TXTECR (T) %;
% Traitement des instructions du bloc %;
REF 701; % ECRITURE " THEN BEGIN " ;
NIVIND : = NIVIND + 1;
y : = x↑ .  INSR;
SI   y <> NIL
     ALORS    REF 301;  % TRTINS (y, RET )%;
FINSI;
% Traiter le bloc répétitif suivant, s'il existe %;
y : = x↑ .  SVTBLOC;
SI   y <> NIL
     ALORS    REF 603; % TRTBLOC(y) %;
FINSI;
% Clore le " THEN " %;
REF 701; % ECRITURE " END" %;
NIVIND : = NIVIND - 1;
% Traiter le cas d'exception %;
y : = x↑ . EXCEPTION;
```

```
SI          y <> NIL
     ALORS     REF 701;  % ECRITURE " ELSE BEGIN " %;
               REF 301;  % TRTINS (y, RET ) %;
               REF 701;  % ECRITURE fin traitement exception
                           standard %;
FINSI;
SI   x↑.  BLOCREP   NIL
     ALORS     NIVIND : = NIVIND - 1
FINSI;
FAR;

RAFF 601  %  COMECR (T) : Ecriture d'un texte en commentaire
             %;
% T : référence au premier morceau du texte à écrire %;
REF 701;  % ECRITURE " ( * " %;
REF 602;  % TXTECR (T) %;
REF 701;  % ECRITURE " *) %;
FAR;

RAFF 602  % TXTECR (T) : Ecriture d'un texte qui s'étend
          éventuellement sur plusieurs enregistrements %;
% T : référence au premier enregistrement du texte à
      écrire %;
DO  T <> NIL ==> TEXTE : = T↑ .  CONTENU;
               REF 701; % ECRITURE (TEXTE) %;
               T : = T↑ .  TSVT
OD
FAR;
```