Titre: Title:	Restructuration automatisée de programmes FORTRAN en pseudocode schématique
Auteur: Author:	Jean-Bernard Trouvé
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Trouvé, J.-B. (1989). Restructuration automatisée de programmes FORTRAN en pseudocode schématique [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/58289/

UNIVERSITÉ DE MONTRÉAL

Restructuration automatisée de programmes FORTRAN en pseudocode schématique

Par

Jean-Bernard TROUVÉ

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU GRADE DE MAITRE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)

Mars 1989

ISBN 0-315-52714-5

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Ce mémoire intitulé:

Restructuration automatisée de programmes FORTRAN en pseudocode schématique

présenté par Jean-Bernard Trouvé

en vue de l'obtention du grade de Maître ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de:

M. Jean Lavoie, M.Sc.A., président

M. Michel Soulié, D.Sc.A.

M. Pierre N. Robillard, Ph.D.

SOMMAIRE

Nous présentons dans ce travail une méthode de traduction de programmes FORTRAN 77 en pseudocode schématique basée sur la transformation de graphes. Le programme source est d'abord converti en graphe à l'aide d'un analyseur FORTRAN. Le graphe est ensuite simplifié itérativement en alternant l'application de transformées de réduction des formes structurées et, si nécessaire, de transformées de restructuration. Le pseudocode schématique est produit à partir du graphe final structuré dans un format compatible avec l'outil logiciel SCHEMACODE.

Les transformées de restructuration sont nécessaires lorsque le programme source n'est pas structuré selon la définition des structures du pseudocode schématique. Ces transformées impliquent soit la création de variables de contrôle, soit la duplication d'énoncés. L'utilisateur peut choisir le compromis le plus acceptable à chaque transformation; ce compromis peut être différent d'un programme à l'autre.

Un prototype d'outil logiciel visant à automatiser l'application de la méthode a été réalisé. Différents programmes FORTRAN ont été ainsi convertis avec succès. Le prototype cumule également un certain nombre de statistiques qui peuvent être utilisées comme métriques, autant sur le programme original que sur le programme résultant.

ABSTRACT

This paper describes a method to translate FORTRAN 77 into schematic pseudocode based on graph transforms. The source program is first converted into a labeled graph by a FORTRAN parser. The resulting graph is then iteratively simplified by alternating between two types of transforms: reduction of structured forms and, when necessary, restructuring transforms. The schematic pseudocode generated from the final structured graph is SCHEMACODE-compatible.

Restructuring transforms are necessary when the source program is not structured according to the definitions of the schematic pseudocode basic constructs. These transforms involve either the introduction of predicate flags or code duplication or both. The best compromise can be achieved by selecting an alternative when each transform occurs and has been shown to be case-dependent.

A working prototype of software tool has been designed to implement the method. Many FORTRAN programs have been successfully converted with the tool. The tool also computes some statistics that can be used as metrics on the original program as well as on the generated program.

REMERCIEMENTS

La réalisation d'un projet de recherche ne saurait être l'oeuvre d'une seule personne. A différents stades de ce projet, plusieurs gens ont contribué à son avancement en fournissant conseils pertinents et commentaires éclairés.

Merci tout d'abord à M. Pierre Robillard, directeur de recherche, pour m'avoir évité de nombreuses difficultés et avoir su garder sur ce projet une perspective globale.

Un merci particulier à Alain Grenier, ami et collègue, pour l'intérêt manifesté à ce projet, ses suggestions pertinentes et son aide inestimable. Son apport a contribué dans une très large mesure à la réalisation du prototype présenté ici.

Merci également à André Beaucage et à tous mes collègues du laboratoire de recherche en génie logiciel pour leur support, leurs suggestions et leur expérience.

Ce mémoire doit beaucoup à l'ambiance musicale dans laquelle il a été rédigé. Merci donc à Glenn Gould pour son jeu incomparable, et à mon lecteur de disques compacts pour sa fiabilité durant d'inlassables répétitions (trois mois, c'est beaucoup trop long).

Finalement, mes remerciements les plus spéciaux vont à Pascale qui, lors des moments difficiles, a su m'aider avec sa patience infinie, son support continuel et sa cuisine incomparable.

TABLE DES MATIÈRES

LISTE DES FIGURES

CHAPITRE 1

CHAPITRE 2

CHAPITRE 3

CHAPITRE 4

CHAPITRE 5

CHAPITRE 6

INTRODUCTION

Civilization advances by extending the number of important operations which we can do without thinking of them (Anonyme).

Le génie logiciel s'intéresse à la production et à la maintenance de systèmes informatiques de haute qualité, livrés dans les délais prescrits et aux coûts estimés [41]. Pour arriver à ces fins, trois composantes sont à considérer: les méthodes, les langages et les outils

Il y a quelques années, avant que les notions fondamentales du génie logiciel soient largement acceptées et exploitées, la production de logiciel se réalisait néanmoins, mais selon des techniques qui relevaient plus de l'intuition que de la procédure systématique. Jusqu'à aujourd'hui un nombre formidable de logiciels ont complètement disparu, souvent même avant d'avoir été mis en service, à cause de leur faiblesse à être entretenus.

Cette faiblesse d'entretien a des racines dans une ou plusieurs de ces composantes: méthodes, langages et outils. L'absence d'utilisation d'une méthode systématique de développement nuit à la qualité du produit; le choix d'un langage mal adapté entraîne des coûts trop élevés de mise au point; le manque d'outils logiciels ou leur utilisation insuffisante ajoute à la difficulté de production et aux pertes de temps.

Pour éliminer ces problèmes d'entretien, deux stratégies existent: la première consiste à faire table rase et à produire un nouveau logiciel à l'aide des méthodes, langages et outils appropriées; la seconde vise plutôt à «récupérer» la plus grande partie de l'investissement réalisé et à l'adapter à de nouveaux outils, langages et/ou méthodes afin d'en faciliter l'entretien.

C'est cette seconde stratégie appelée restructuration, qui nous intéresse dans le cadre de ce travail. Plus particulièrement, afin de pouvoir présenter des résultats concrets, nous nous concentrerons sur la restructuration du design détaillé de programmes écrits en langages FORTRAN. Nous viserons, de plus, à automatiser le plus possible cette restructuration.

Les principaux avantages de la restructuration largement automatisée sont les suivantes:

- elle permet de voir les résultats immédiatement et d'évaluer si une refonte complète est réellement nécessaire (à l'aide de mesure de diagnostics);
- elle est moins coûteuse qu'une refonte complète;
- elle produit des résultats corrects rapidement;
- les programmes résultants peuvent être entretenus à l'aide d'outils logiciels avancés.

Par contre, il faut garder en tête certaines limitations propres à la restructuration: d'une part le fait que le programme résultant peut être passablement différent de l'original, ce qui cause une confusion de la part du programmeur habitué à «son» programme; d'autre part le fait que la restructuration n'opère qu'au niveau du design détaillé et ne peut pas corriger de problème dû au design structurel.

But du travail

Essentiellement, il s'agit d'élaborer un mécanisme pour traduire des programmes FORTRAN en une représentation fonctionnellement équivalente dans un nouveau langage: le pseudocode schématique. Une fois le mécanisme de traduction connu, nous chercherons à l'automatiser le plus possible en construisant les outils logiciels appropriés.

Les programmes résultants pourront ensuite être entretenus à l'aide de SCHEMACODE[1], un outil logiciel d'aide à la programmation qui permet d'organiser la documentation des programmes et qui supporte l'édition directe en pseudocode schématique. Les avantages du pseudocode schématique résident dans une lisibilité des programmes largement accrue, dans une documentation interne hiérarchisée plutôt que linéaire et dans la normalisation de la représentation des structures de contrôle (séquentielles, conditionnelles, répétitives). Finalement. les programmes en pseudocode schématique sont toujours structurés.

Cette dernière caractéristique donne évidemment beaucoup d'attrait aux programmes résultants mais est effectivement la plus grande source de difficulté lorsque vient le temps de traduire des programmes non-structurés. En s'inspirant des travaux déjà effectués sur ce sujet, nous élaborerons une technique de restructuration

---

1 SCHEMACODE est une marque déposée de Schemacode International Inc.

basée sur la théorie des graphes qui présente l'avantage supplémentaire d'être indépendante des langages de programmation.

Organisation du texte

Le texte de ce mémoire est organisé en six chapitres.

Dans le premier, nous présentons la problématique de traduction de programmes FORTRAN en pseudocode schématique après avoir examiné les caractéristiques respectives de ces deux langages.

Le second chapitre décrit les études préliminaires qui ont été effectuées pour choisir une stratégie valable de résolution du problème. On y présente une première approche manuelle, puis une approche empirique semi-automatisée basée sur le graphe de contrôle et enfin une approche totalement automatisée, dans un autre langage, afin d'évaluer un produit fini typique.

Les explorations du second chapitre montrant de façon claire l'avantage du graphe de contrôle en tant que représentation intermédiaire, nous approfondissons ce sujet au troisième chapitre. Après une brève revision des concepts de la théorie des graphes pertinents à notre étude, trois méthodes d'implantation de graphe sont explorées: par base de données relationnelle, par matrice d'adjacence et par type abstrait de données. C'est cette dernière qui sera utilisée ultimement.

C'est dans le quatrième chapitre que sont présentés les détails de la méthode de restructuration. On y explique les diverses phases du processus: traduction du code source en graphe de contrôle, structuration du graphe puis traduction du graphe en pseudocode schématique.

La réalisation du système fait l'objet du cinquième chapitre. Les détails d'environnement, les composantes du système de même que ses possibilités et limitations y sont abordés.

Un prototype fonctionnel étant alors disponible, il est utilisé et, au sixième chapitre, on examine les résultats obtenus en traduisant divers types de programmes: structurés, non-structurés, de grandes et petites dimensions, etc.

Finalement, nous conclurons sur l'utilité du système proposé et ses performances et, à la lumière des résultats obtenus, sur des alternatives d'amélioration et d'extension de ses possibilités.

CHAPITRE 1

PROBLÉMATIQUE

Those who ignore history are doomed to repeat it (Anonyme).

Ce chapitre vise à poser le problème de la façon la plus précise possible en présentant ses composantes. Nous examinerons d'abord le langage FORTRAN (langage source) et ses caractéristiques, puis le pseudocode schématique (langage cible) et ses structures de contrôle, et enfin les caractéristiques souhaitées du processus de traduction.

1.1 Le langage FORTRAN

Le langage FORTRAN a été originalement conçu par la firme International Business Machines (IBM). Depuis sa mise en marché en 1957, il a subi deux étapes de normalisation par l'American National Standards Institute: ANSI 66 et ANSI 77. C'est cette dernière définition du langage FORTRAN qui est maintenant la plus utilisée et qui sera l'objet de notre travail.

1.1.1 Pourquoi FORTRAN?

Le choix du FORTRAN comme langage d'expérimentation dans ce travail découle de plusieurs facteurs convergents.

Tout d'abord, c'est un langage fortement répandu dans les milieux scientifiques et de génie. Une grande quantité de programmes et de programmathéques ont été réalisés en FORTRAN depuis sa création. Cela représente un nombre relativement élevé de lignes de code qui sont actuellement en phase de maintenance. Il existe donc un réel besoin de garder un plein contrôle sur cette masse de programmes et d'utiliser des outils logiciels appropriés pour ce faire.

Ensuite, il faut considérer le style de programmation favorisé par le folklore du FORTRAN (utilisation de trucs et astuces bien souvent obscurs), la présence d'étiquettes et la pauvreté de l'ensemble des structures de contrôle qui ont conduit, comme bien des gens l'ont décrié, à la mauvaise utilisation du GOTO ou de ses dérivés. D'ailleurs Dijkstra disait du FORTRAN: « [FORTRAN is] an infantile disorder [...] too risky and therefore too expensive to use » [15].

Bref, bien des programmes FORTRAN sont difficiles à lire en comparaison à leurs homologues écrits en pseudocode schématique.

L'existence du GOTO et de ses dérivés permet au programmeur FORTRAN de déborder du cadre délimité par les préceptes de la programmation structurée. En donnant facilement les moyens de ne pas se limiter aux structures conditionnelles ou répétitives à «une entrée/une sortie», le FORTRAN fournit ainsi la possibilité d'engendrer rapidement des programmes non-structurés. Cette caractéristiques hautement indésirable a toutefois des impacts très tangibles dans beaucoup de réalisations.

En résumé, le besoin de structuration est très présent dans plusieurs logiciels écrits en FORTRAN. et ce langage est assez général pour qu'une solution qui le concerne puisse s'appliquer aisément dans un autre langage.

Finalement, dans un autre ordre d'idées, la nécessité de disposer d'un analyseur syntaxique s'est manifestée assez rapidement dans ce projet. Comme un tel analyseur était déjà disponible au laboratoire de recherche en génie logiciel de l'Ecole Polytechnique de Montréal, nous en avons profité pour consacrer plutôt nos efforts sur le coeur du sujet. Notons que la réalisation d'un analyseur de ce type est considérée comme étant particulièrement difficile [2] [21] [43].

1.1.2 Caractéristiques et énoncés de contrôle

Une définition complète du langage FORTRAN ANSI 77 est donnée dans [18]. Certaines variantes d'implantation sont documentées dans [5] et [7]. Nous nous en tiendrons cependant à la norme ANSI 77.

De façon à mieux comprendre certains problèmes qui deviendront évidents plus loin dans ce travail, nous examinons ici certaines caractéristiques du langage FORTRAN, et plus spécifiquement ses énoncés de contrôle.

Etiquettes

Chaque énoncé exécutable peut être étiqueté de façon unique par un nombre positif inférieur à 100 000. L'étiquette sert de référence et identifie une «adresse» où le contrôle du programme peut éventuellement être transféré, à l'intérieur d'une même procédure[1]. Un exemple est donné à la figure 1.1.

---

[1] On désigne par le terme général de «procédure» tout sous-programme (SUBROUTINE) ou fonction (FUNCTION) FORTRAN.

```
        I = 10
1       J = SUB(I)                                          Enoncé étiqueté «1»
        K = J+I
```

Fig. 1.1 - Exemple d'étiquette.

Enoncé GOTO inconditionnel

Comme son nom l'indique, cet énoncé transfère inconditionnellement le contrôle à l'énoncé portant l'étiquette mentionnée (figure 1.2).

```
        I = 10
        GOTO 1                                              GOTO inconditionnel
à «1»

...

1       J = J+K
```

Fig. 1.2 - Exemple de GOTO

Notons que l'énoncé suivant le GOTO inconditionnel doit obligatoirement être étiqueté pour être exécuté (sinon, il s'agit de «code mort») [43].

Enoncé IF-GOTO

Cet énoncé est la forme la plus simple de transfert de contrôle conditionnel. La syntaxe est:

IF (expression-booléenne) GOTO étiquette

Si l'expression booléenne est vraie, le programme continue à l'étiquette indiquée. Sinon, l'exécution se poursuit à l'énoncé suivant. Voir la figure 1.3 pour un exemple.

```
        READ (5,*) IA
        IF (IA.EQ.0) GOTO 1                                    Enoncé IF-GOTO
        X = 1. / IA
1       J = J+X
```

Fig. 1.3 - Exemple d'IF-GOTO.

GOTO calculé

Le GOTO calculé est lui aussi un énoncé de transfert conditionnel de contrôle. Il suit la syntaxe:

GOTO (étiquette 1, étiquette 2,...étiquette n) expression-numérique-entière

L'expression numérique est évaluée une fois. Si sa valeur est K (K entre 1 et n inclusivement) il y a branchement à l'étiquette K. Sinon, l'exécution se poursuit à l'énoncé suivant. Voir la figure 1.4 pour un exemple.

```
10      WRITE (9,*) 'Entrer 1 pour calculer, 2 pour finir'
        READ (9,*) I
        GOTO (20,30) I   Enoncé GOTO calculé
        GOTO 10
20      CALL CALCUL
30      STOP
```

Fig. 1.4 - Exemple de GOTO calculé.

GOTO assigné

Le GOTO assigné est un énoncé de transfert inconditionnel dont la destination est variable. Il prend la forme :

GOTO variable-entière (liste-d'étiquettes)

où la liste d'étiquettes et les parenthèses sont optionnelles.

Le contrôle est transféré à l'étiquette dont la valeur est conservée dans la variable. Essentiellement, il s'agit d'un GOTO indirect. L'affectation d'une valeur à cette variable se fait cependant par l'intermédiaire d'un énoncé dédié (ASSIGN) et non pas par l'opérateur « = ».

Si aucune étiquette ne correspond à la valeur de la variable, l'exécution se poursuit à l'énoncé suivant. Voir la figure 1.5 pour un exemple.

```
10      WRITE (9,*) 'Entrer 10 pour calculer, 20 pour finir'
        READ (9,*) I
        ASSIGN I TO ETIQ                                    Enoncé ASSIGN
        GOTO ETIQ (20,30)                                   Enoncé GOTO assigné
        GOTO 10
20      CALL CALCUL
30      STOP
```

Fig. 1.5 - Exemple de GOTO assigné.

IF arithmétique

Le IF arithmétique est un autre énoncé de transfert conditionnel. Sa syntaxe est:

IF (expression-numérique) étiquette-1, étiquette-2, étiquette-3

L'expression numérique est évaluée une fois. Si sa valeur est négative l'exécution se poursuit à l'étiquette-1; si elle est nulle, à l'étiquette-2; si elle positive, à l'étiquette-3. Voir la figure 1.6 pour un exemple.

```
      WRITE (9,*) 'Entrer <= 1 pour calculer, >= 2 pour finir'
      READ (9,*) I
      IF (I-2) 10,20,20                                          IF arithmétique
10    CALL CALCUL
20    STOP
```

Fig. 1.6 - Exemple d'IF arithmétique.

IF énoncé

Le IF énoncé est aussi un énoncé de transfert conditionnel. Sa syntaxe est:

IF (expression-booléenne) énoncé

L'énoncé est exécuté seulement si l'expression booléenne est vraie. Autrement, l'exécution se poursuit à l'énoncé suivant. On remarquera que le IF-GOTO n'est qu'un cas particulier du IF énoncé.

Tout dépendant des compilateurs, certaines limitations existent concernant le type d'énoncé qui peut être utilisé avec le IF. Voir la figure 1.7 pour un exemple.

```
      WRITE (9,*) 'Entrer 1 pour calculer, 2 pour finir'
      READ (9,*) I
      IF (I.EQ.1) CALL CALCUL                          Enoncé IF-énoncé
      IF (I.EQ.2) STOP                                 Enoncé IF-énoncé
```

Fig. 1.7 - Exemple d'IF-énoncé.

Enoncés IF, ELSE IF, ELSE

Cet énoncé apparu avec la norme ANSI 77. Il est similaire à celui qu'on rencontre en C ou en Pascal et sert de bloc de base pour l'élaboration de conditionnelles structurées. Sa syntaxe est:

```
IF (expression-booléenne) THEN          ┐
        [énoncés]                       ┘ branche IF
ELSE IF (expression-booléenne) THEN     ┐
        [énoncés]                       ┘ branche ELSE IF
ELSE                                    ┐
        [énoncés]                       ┘ branche ELSE
ENDIF
```

Alors que la branche IF est obligatoire, la branche ELSE est optionnelle (zéro ou une présente) de même que la branche ELSE IF (zéro ou plusieurs présentes).

A l'exécution, l'expression booléenne du IF est testée. Si elle est vraie, les énoncés de la branche IF sont exécutés et l'exécution se poursuit après le END IF. Si elle est fausse, c'est l'expression du ELSE IF suivant (si elle existe) qui est testée. Si cette dernière est vraie, les énoncés de la branche ELSE IF sont exécutés et le contrôle est transféré après le END IF. Sinon, il y a test de tous les END IF présents en suivant la même procédure.

Si aucune condition n'est vraie, ce sont les énoncés de la branche ELSE (si elle existe) qui sont exécutés. Voir la figure 1.8 pour un exemple.

```
WRITE (9,*) 'Entrer 1 pour calculer, 2 pour finir'
READ (9,*) I
IF (I.EQ.1) THEN                                        Enoncé IF
   CALL CALCUL
ELSE IF (I.EQ.2) THEN                                   Enoncé ELSE IF
   CALL FINIR
ENDIF                                                   Enoncé ENDIF
```

Fig. 1.8 - Exemple d'IF-THEN et ELSE IF-THEN.

Enoncé DO

L'énoncé DO est le bloc de base pour créer des boucles comptées. L'énoncé suit la syntaxe:

DO étiquette, variable = valeur-initiale, valeur-finale, incrément

où l'incrément est optionnel (il vaut 1 s'il n'est pas spécifié)

Encore ici, l'implantation du modèle d'exécution de l'énoncé DO est variable d'un compilateur à l'autre. Un premier modèle initialise la variable à la valeur-initiale, exécute tous les énoncés jusqu'à atteindre l'étiquette spécifiée (l'énoncé étiqueté est aussi exécuté), augmente la variable de la valeur d'incrément puis compare la variable à la valeur finale. Si la valeur dans la variable est supérieure à la valeur finale, l'exécution se poursuit après l'énoncé étiqueté. Sinon, elle se poursuit à l'énoncé suivant le DO.

Un second modèle, implanté dans le compilateur VAX FORTRAN [7], effectue d'abord l'initialisation, puis fait le test avant d'exécuter les énoncés suivant le DO. Une fois l'énoncé exécuté, l'incrément est ajouté au contenu de la variable et on «remonte» au DO pour refaire le test.

Il y a également des variantes d'un compilateur à l'autre quant à la manipulation de la variable (et de la valeur finale si c'est aussi une variable). On conseille habituellement au programmeur d'éviter d'en modifier la valeur à l'exécution. Voir la figure 1.9 pour un exemple.

```
C       Imprime les nombres de 1 à 5 inclusivement

        DO 10, I=1,5      Enoncé DO
        WRITE (9,*) I
10      CONTINUE                                                  Etiquette de DO
```

Fig. 1.9 - Exemple de DO.

Autres énoncés de contrôle

Nous venons de couvrir les énoncés de contrôle tels que définis dans la norme ANSI 77. Même si certains compilateurs acceptent des énoncés supplémentaires, nous ne nous en préoccupons pas dans le cadre de notre expérimentation.

Ils ne représentent d'ailleurs pas de différences fondamentales par rapport aux énoncés couverts ici.

1.2 Le pseudocode schématique

Contrairement au FORTRAN, le pseudocode schématique est relativement jeune. Conçu au laboratoire de recherche en génie logiciel de l'Ecole Polytechnique [33] [36], ce langage a été utilisé avec succès dans plusieurs projets de développement [30] [32] [35] [37]. Nous allons en présenter les caractéristiques et avantages dans cette section.

1.2.1 Caractéristiques du pseudocode schématique

Plusieurs caractéristiques du pseudocode schématique en font un langage de choix lors de la phase d'entretien d'un logiciel et, par conséquent, un langage cible intéressant pour notre application.

Tout d'abord, les programmes écrits en pseudocode schématique sont toujours structurés (au sens du flux de contrôle). Ces programmes sont bâtis à l'aide de trois structures de base (séquentielle, conditionnelle, répétitive) qui ont chacune un seul point d'entrée et un seul point de sortie. Les avantages des programmes structurés sont de nos jours assez bien compris ou, du moins, bien acceptés [22] [49].

Ensuite, ces trois structures de base sont représentées sous forme graphique simple (d'où l'adjectif «schématique»). Ces schémas sont représentés à la figure 1.10.

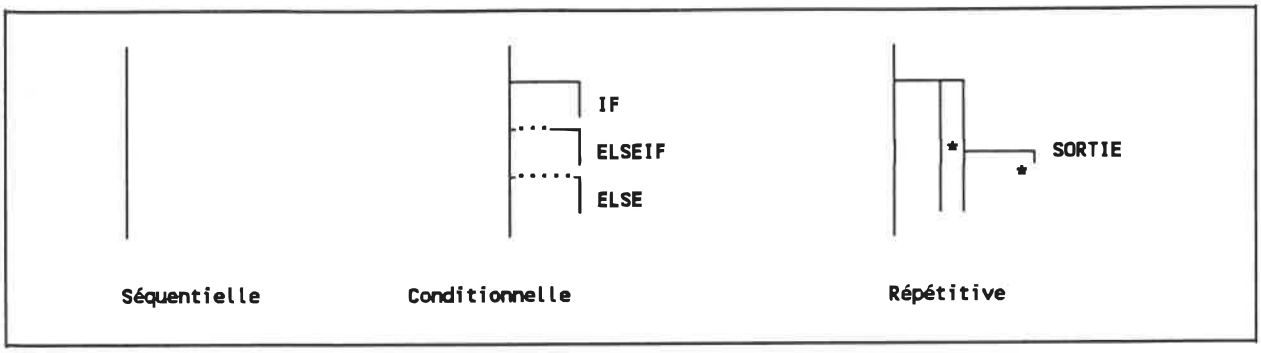


Fig. 1.10 - Structures du pseudocode schématique.

Les schémas servent donc à représenter le flux de contrôle du programme, aussi appelé logique du programme. Cependant, ils ne suffisent pas pour créer un programme car il leur manque la capacité d'exprimer les actions à effectuer lors des différents cheminements imposés par les décisions. Pour combler cette lacune, les actions sont représentées textuellement, de même que les expressions booléennes. Le langage textuel utilisé est habituellement un sous-ensemble d'un langage de programmation dépourvu de tout énoncé de contrôle (il contient des énoncés de déclaration, d'affectation, d'entrée-sortie, etc).

On voit donc que, contrairement à un langage de programmation purement textuel, le pseudocode schématique permet la visualisation directe de la dichotomie logique/contenu.

Examinons chaque structure séparément pour bien comprendre cette notation.

Structure séquentielle

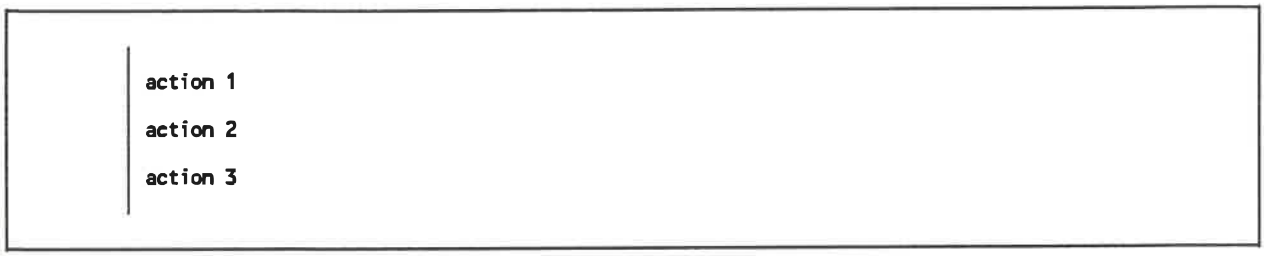


Fig. 1.11 - Structure séquentielle.

La structure séquentielle (figure 1.11) est la plus simple. Elle se lit de haut en bas et s'interprète de cette façon:

- exécuter l'action 1, puis
- exécuter l'action 2, puis
- exécuter l'action 3.

Voici un exemple de structure séquentielle avec son homologue FORTRAN (figure 1.12):

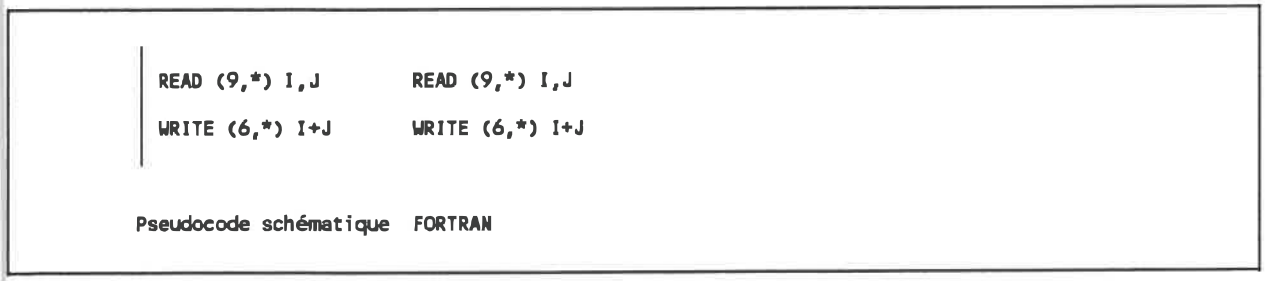


Fig. 1.12 - Exemple de structure séquentielle.

Structure conditionnelle

La structure conditionnelle implante les cheminements alternatifs vers l'avant. Elle est très similaire au IF-ELSE-IF-ELSE du FORTRAN et est composée de trois types de branches qui portent les mêmes noms (voir figure 1.13):

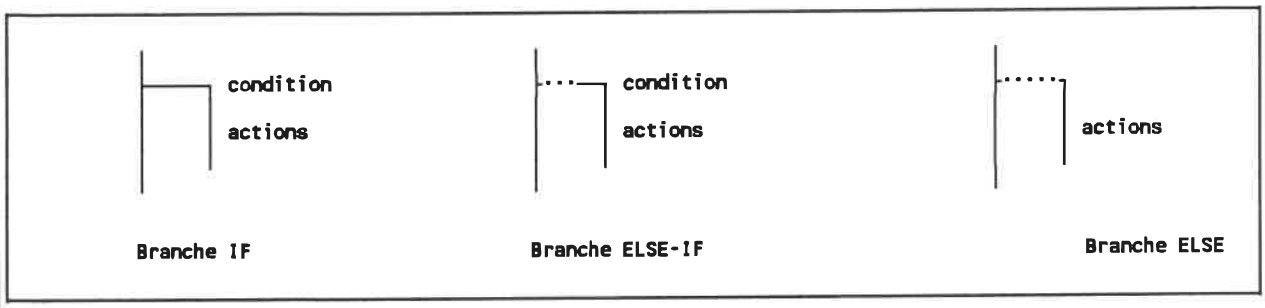


Fig. 1.13 - Structure conditionnelle.

Comme en FORTRAN, une structure conditionnelle possède une et une seule branche IF, zéro, une ou plusieurs branches ELSE IF, et zéro, ou une branche ELSE. Ces branches se concatènent verticalement dans l'ordre IF, ELSE IF (si présente) et ELSE (si présente). Diverses combinaisons sont présentées à la figure 1.14.

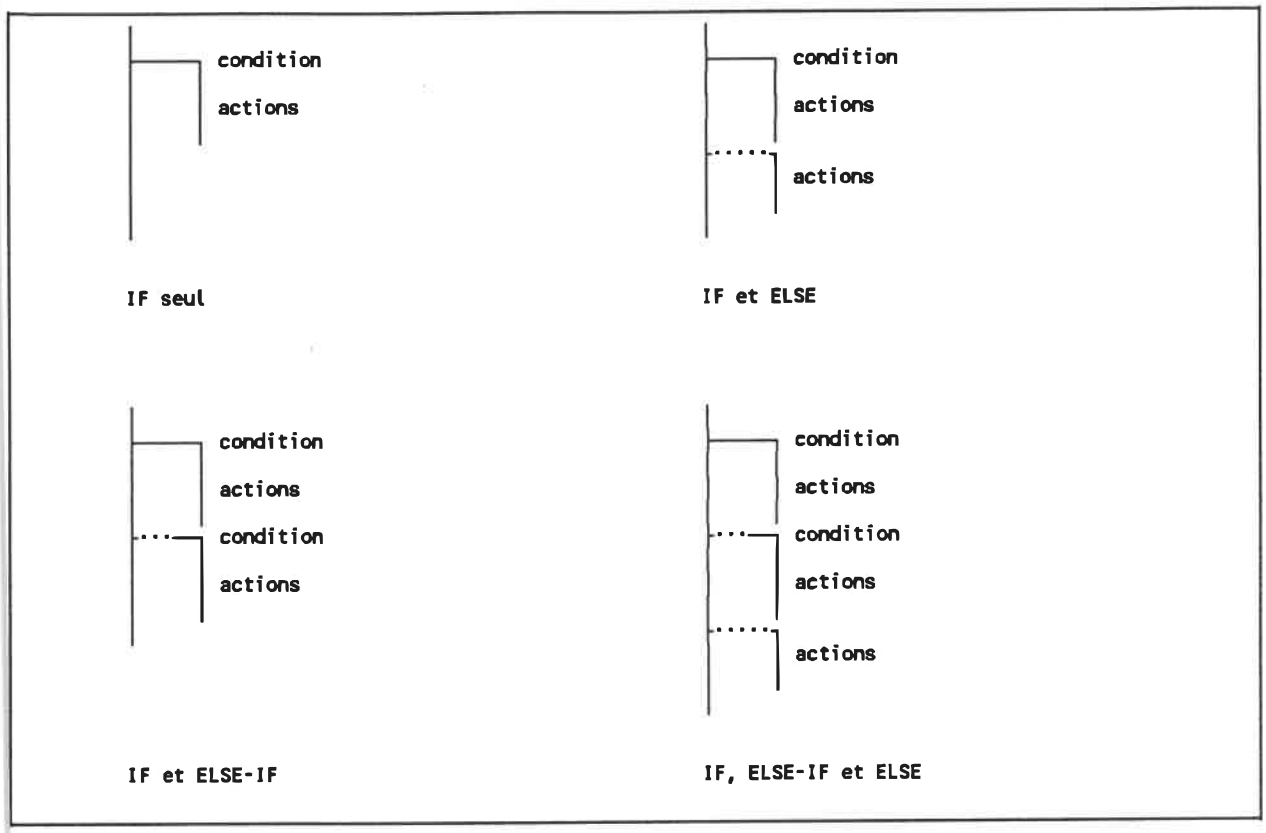


Fig. 1.14 - Différentes structures conditionnelles.

Bien que sa représentation soit différente, la structure conditionnelle en pseudocode schématique s'interprète de la même façon qu'en FORTRAN. L'exemple de la figure 1.15 illustre cet aspect:

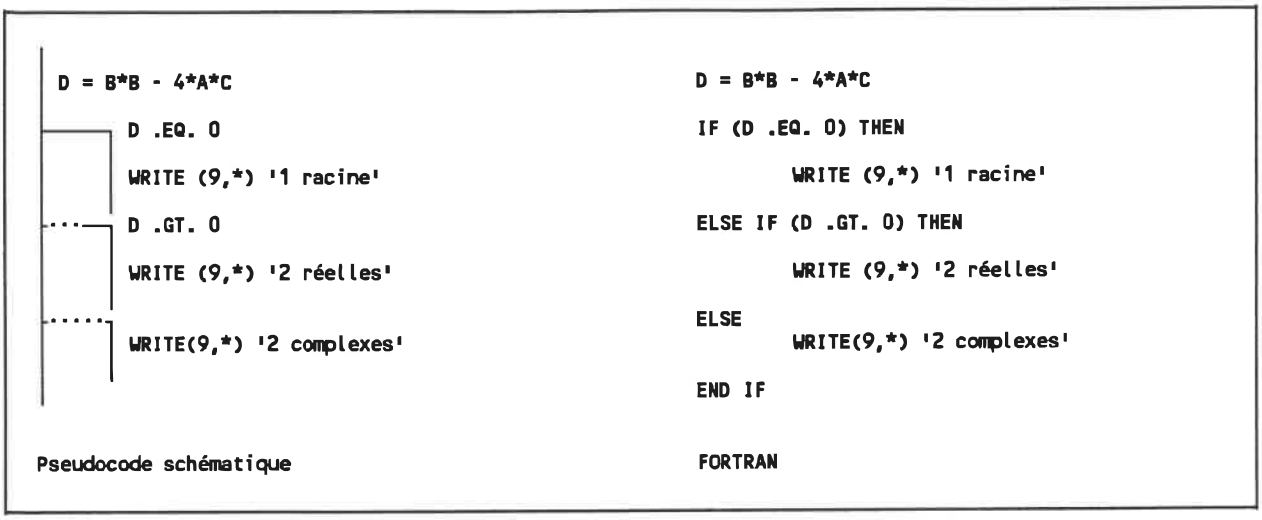


Fig. 1.15 - Exemple de structure conditionnelle.

Structure répétitive

La structure répétitive permet la la réalisation de toutes les formes de boucles en pseudocode schématique. C'est une structure qui, tout en se limitant à un seul point d'entrée et un seul point de sortie, permet l'utilisation de plusieurs conditions de sortie disposées à n'importe quel endroit dans la boucle. Cette caractéristique, beaucoup plus importante qu'elle ne le laisse paraître, évite bien souvent au programmeur de créer des variables de contrôle inutiles.

La structure répétitive est formée d'au moins une action et d'au moins une condition de sortie. De plus, à l'intérieur d'une branche de condition de sortie peut se retrouver d'autres actions qui sont exécutées avant que le contrôle ne soit transféré après la structure répétitive. Voici quelques exemples (figure 1.16):

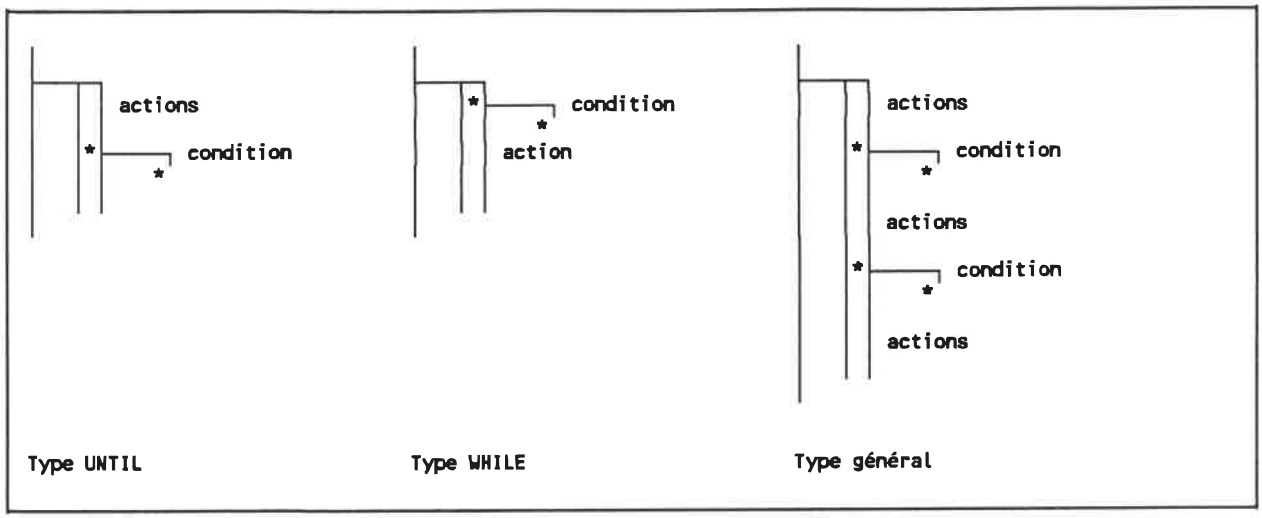


Fig. 1.16 - Différentes structures répétitives.

L'interprétation du schéma de la structure répétitive se fait comme suit. On entre dans cette structure par la ligne horizontale et on suit les deux lignes verticales de haut en bas. Lorsqu'on rencontre une condition de sortie (marquée par un astérisque), on teste sa valeur; si elle est vraie, on exécute les actions qui font partie de sa branche et on continue l'exécution après la répétitive; si elle est fausse, on continue après la fin de la condition de sortie (marquée par un second astérisque), tout en restant à l'intérieur de la répétitive. Lorsqu'on atteint le bas des lignes verticales, on pousuit l'exécution au point d'entrée (ligne horizontale). On «tourne» ainsi jusqu'à la vérification d'une condition de sortie.

L'exemple de la figure 1.17 montre la généralité des conditions de sortie multiples. il s'agit d'un segment de programme effectuant une recherche dans un tableau.

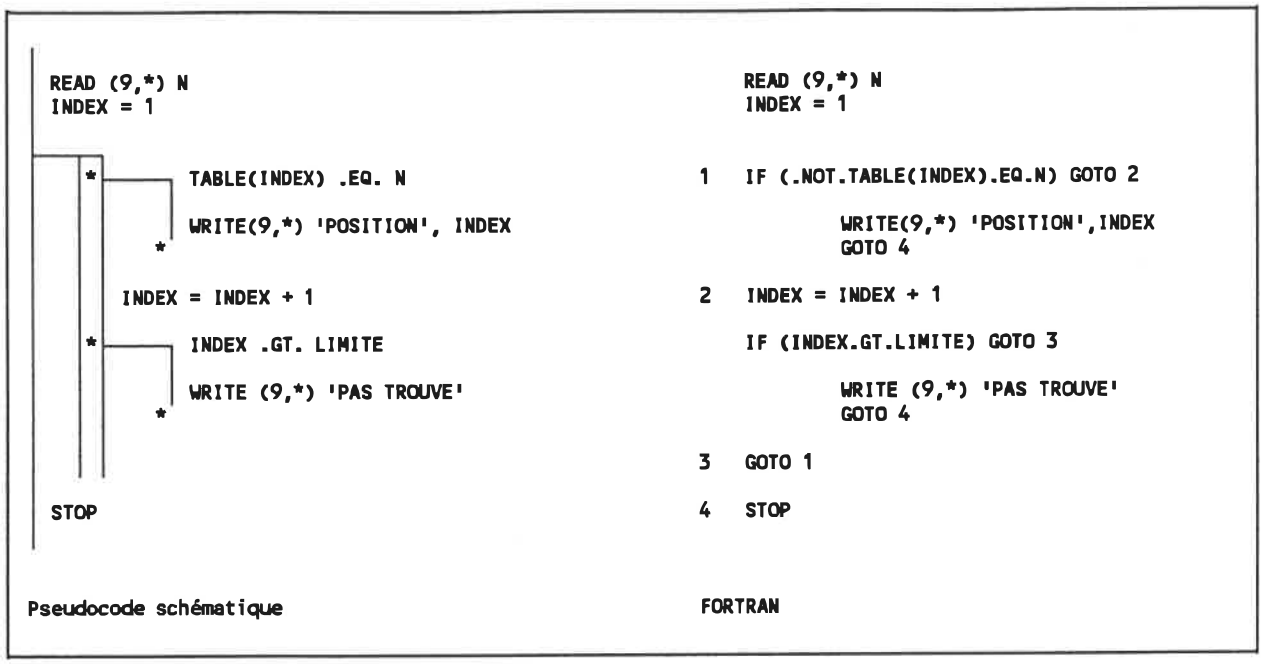


Fig. 1.17 - Exemple de structure répétitive.

1.2.2 Avantages du pseudocode schématique

L'assurance de toujours produire des programmes structurés et la simplicité de la représentation schématique sont deux avantages importants du pseudocode schématique. Ce ne sont toutefois pas les seuls.

L'uniformité assurée par les schémas de base est un facteur important de lisibilité. Par exemple, quelle que soit le genre de boucle que le programmeur veuille créer (WHILE, REPEAT, etc.), il utilise toujours le même schéma. Cette uniformité dans la notation favorise une meilleure communication entre programmeurs [19].

Un avantage supplémentaire est la réelle capacité de construire un programme selon une approche descendante [49]. En effet, un mécanisme appelé commentaire opérationnel [37] [31] permet de décrire succintement une sous-tâche à réaliser sans en élaborer les détails immédiatement. Un commentaire opérationnel est un texte libre

(numéroté pour référence) qui s'insère à la place d'une action dans toute structure du pseudocode schématique.

La présence de ce mécanisme permet d'énoncer d'abord un algorithme sous forme de commentaires très abstraits et de structures de contrôle schématisées. Puis, chaque commentaire opérationnel peut-être élaboré en d'autres commentaires opérationnels et/ou structures à l'intérieur de raffinements [34] . Chaque commentaire opérationnel capte alors une partie de la documentation relative à une étape de décomposition de l'algorithme.

Il en résulte une documentation qui est organisée de façon hiérarchique au lieu d'être séquentielle ou séparée lorsqu'on utilise un langage de programmation. Ceci favorise la compréhension, et ce au niveau d'abstraction désiré.

1.2.3 SCHEMACODE

Le pseudocode schématique est une notion qui comprend un langage (schématique) et une méthode de construction d'algorithmes (descendante). Toutefois, de façon à être vraiment utilisable dans un environnement de développement, le pseudocode schématique a besoin d'un outil pour le supporter et en automatiser l'utilisation. Cet outil existe et il s'appelle SCHEMACODE TM [6] [31].

SCHEMACODE comprend un éditeur de pseudocode schématique et plusieurs modules de traduction pour plusieurs langages de programmation Ces derniers sont nécessaires afin de transformer le pseudocode schématique en énoncés respectant la syntaxe d'un langage de programmation afin d'être compilé puis exécuté.

A l'aide de SCHEMACODE, l'entretien des programmes est plus facile et moins onéreux grâce aux avantages du pseudocode schématique précédemment cités.

1.3 Traduction du FORTRAN en pseudocode schématique (SPC[2])

Cette section présente le processus de traduction, ses intrants et ses extrants.

Au plus haut niveau d'abstraction, on peut considérer le système de traduction de cette façon (figure 1.18):

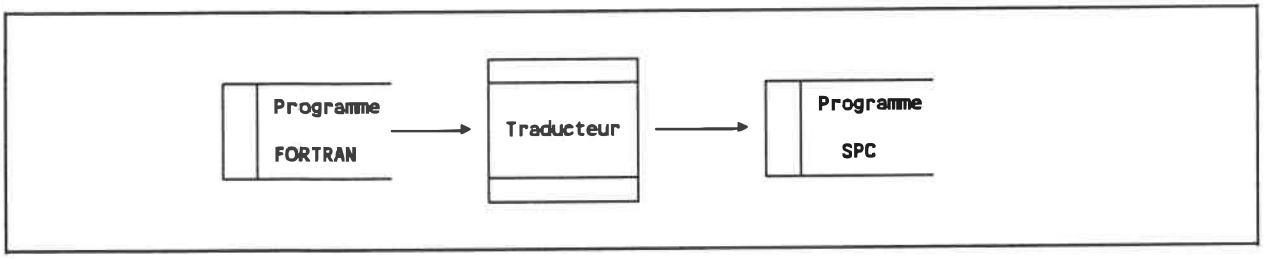


Fig. 1.18 - Vue abstraite du système de traduction.

1.3.1 Intrants

Les intrants sont des fichiers contenants des programmes FORTRAN satisfaisant les conditions suivantes:

- le langage respecte le norme ANSI 77;
- les fichiers sont codés en ASCII;
- tous les fichiers inclus (par INCLUDE, $INCLUDE ou tout autre énoncé équivalent) doivent accompagner les fichiers principaux;
- ils ont déjà été compilés sans erreurs.

---

2 A partir de maintenant, nous utiliserons plutôt l'abréviation «SPC», utilisée dans [31], pour désigner le pseudocode schématique.

A ce stade, nous ne désirons pas apporter d'autres limitations sur le contenu des fichiers d'entrée.

1.3.2 Extrants

Essentiellement, le système de traduction doit produire un fichier SPC contenant un programme en pseudocode schématique qui effectue la même tâche que le programme FORTRAN original. Plus précisément:

- chaque procédure SPC, une fois retraduite en FORTRAN par SCHEMACODE, peut être compilée et le fichier objet ainsi créé peut remplacer le fichier objet original en démontrant un comportment identique (vu des autres procédures);
- le fichier SPC est produit dans un format acceptable par SCHEMACODE;
- les actions (affectations, entrées/sorties, déclarations, appels à procédure, etc.) sont exprimées en FORTRAN et sont textuellement identiques à celles retrouvées dans le programme original;
- les expressions booléennes des conditions sont également exprimées en FORTRAN, mais résultent parfois d'une dérivation du texte original (quand elles ne sont pas écrites explicitement);
- les commentaires narratifs sont produits à partir des commentaires présents dans le fichier source;
- les commentaires opérationnels sont présents afin de permettre l'imbrication des structures (tel que l'exige la définition du pseudocode schématique), mais leur contenu n'est pas pertinent puisqu'il se situe à un niveau de sémantique beaucoup trop abstrait;

- tous les énoncés de contrôle sont convertis en structures schématiques, exceptés ceux qui terminent l'exécution (STOP) et ceux qui retournent le contrôle à la procédure appelante (RETURN), ces derniers étant conservés tels quels.

Le résultat est alors nécessairement un programme structuré auquel il ne manque que la description des commentaires opérationnels. Ces derniers doivent être spécifés par un usager à l'aide de l'outil SCHEMACODE.

1.3.3 Caractéristiques du processus de traduction

La traduction du FORTRAN en SPC n'est pas une opération banale. La présence de bris dans la structure du programme original crée en effet des difficultés qu'un système basé sur une simple translittération ne saurait résoudre [44]. Le processus entre plutôt dans la catégorie «abstraction avec ré-implantation», sans toutefois atteindre le degré d'abstraction décrit dans [44].

Il faut donc considérer l'utilisation d'une représentation intermédiaire abstraite pour conserver l'information sur la structure du programme. Il s'agit en fait d'un graphe de contrôle dont nous donnons les détails au chapitre 3. Comme nous l'exprimons au chapitre 2, les tentatives effectuées pour éviter une représentation intermédiaire n'ont pas été fructueuses. Il est évidemment bien important de s'en assurer puisqu'un processus basé sur la translittération est beaucoup plus simple à réaliser.

La phase de transformation du source FORTRAN en graphe de contrôle est dans notre cas réalisée en grande partie par le logiciel DATRIX[3] [10]. DATRIX est un outil de mesure du logiciel qui possède entre autres un module d'analyse lexicale et syntaxique adapté au FORTRAN. Une représentation du graphe de contrôle étant

---

3 DATRIX est une marque déposée de Bell Canada Inc.

produite puis conservée sur disque, il suffit d'en connaître l'organisation pour en extraire l'information.

Notons un avantage supplémentaire qu'offre la représentation intermédiaire: l'indépendance du langage de programmation. Ceci permet en effet de rendre indépendants l'analyseur (front end) et le générateur (back end) [2] [43] du système de traduction. Eventuellement, d'autres analyseurs pourront s'y greffer pour accepter différents langages comme source.

Comme il existe plusieurs stratégies pour convertir un programme non-structuré, le traducteur doit alors offrir à l'usager une liste la plus complète possible des stratégies de restructuration envisageables. De plus, le traducteur doit assister l'usager dans son choix de stratégies en quantifiant, si possible, chaque stratégie à l'aide de critères prédéfinis. Par exemple, ces critères peuvent être des mesures prises sur le programme SPC final, sur le graphe de contrôle résultant ou sur les opérations de modifications effectuées sur la représentation intermédiaire (mesure de variation).

Finalement, le temps de traduction, ne doit pas être trop grand, surtout si le système doit évaluer plusieurs stratégies. Un temps d'ordre O(n2) [2] [23] reste acceptable car il s'agit tout de même d'un prototype. Cependant, on cherchera à maximiser la performance pour obtenir un outil raisonnablement utilisable.

CHAPITRE 2

ETUDES PRELIMINAIRES

You can never tell which way the train went by looking at the track (Anonyme).

Nous présentons dans ce chapitre différentes solutions qui ont été envisagées. Premièrement, nous présentons une approche directe qui a servi à bien comprendre le problème et son étendue. Puis, nous examinons une approche empirique, partiellement automatisée et basée sur un graphe de contrôle rudimentaire. Nous présentons ensuite un système totalement automatisé traduisant en SPC un langage source plus simple que le FORTRAN, ceci afin de bien percevoir les avantages et les inconvénients de la translittération. Finalement, nous ferons une synthèse de ces études préliminaires pour converger vers un système adéquat et réalisable.

2.1 Traduction directe

Lorsqu'on aborde le problème de traduction en examinant certains programmes sources de dimensions réduites (i.e. quelques lignes ou quelques dizaines de ligne), il est tentant de croire qu'un opérateur est capable d'en analyser la structure et d'en synthétiser son équivalent en SPC par simple inspection visuelle. Très tôt, cependant, il devient tentant d'utiliser des repères écrits sur imprimé du programme afin d'identifier les structures et leur portée. Prenons par exemple le programme de la figure 2.1 (PRIME):

```
 1      C Determine si un nombre est premier.
 2      C
 3            PROGRAM PRIME
 4            INTEGER DIV,LIM
 5            REAL P,FLAG
 6            WRITE (0,*) 'Entrer un nombre entier'
 7            READ (0,*) P
 8            DIV = 2
 9            LIM = INT(SQRT(P))
10      20000    FLAG = P/DIV - INT(P/DIV)
11               IF (FLAG.EQ.0 .OR. DIV.EQ.LIM) GOTO 20001
12               DIV=DIV+1
13               GOTO 20000
14      20001 IF (FLAG.NE.0 .OR. P.LT.4) THEN
15               WRITE(0,*) INT(P),' est premier.'
16            ELSE
17               WRITE(0,*) DIV,' est le plus petit facteur de ',INT(P)
18            END IF
19            END
```

Fig. 2.1 - Programme PRIME en FORTRAN.

Après une inspection relativement rapide, on peut identifier une structure répétitive précédant une structure conditionnelle à 2 cas (figure 2.2):

```
1          C Determine si un nombre est premier.
2          C
3                PROGRAM PRIME
4                INTEGER DIV,LIM
5                REAL P,FLAG
6                WRITE (0,*) 'Entrer un nombre entier'
7                READ (0,*) P
8                DIV = 2
9                LIM = INT(SQRT(P))
10         20000    FLAG = P/DIV - INT(P/DIV)
11                  IF (FLAG.EQ.0 .OR. DIV.EQ.LIM) GOTO 20001
12                  DIV=DIV+1
13                  GOTO 20000
14         20001 IF (FLAG.NE.0 .OR. P.LT.4) THEN
15                  WRITE(0,*) INT(P),' est premier.'
16               ELSE
17                  WRITE(0,*) DIV,' est le plus petit facteur de ',INT(P)
18               END IF
19               END
```

Fig. 2.2 - Programme PRIME avec structures identifiées visuellement.

Dans ce cas précis, il est effectivement très simple de produire un SPC manuellement à partir du source. Il n'y a qu'à récupérer le texte original à l'aide de SCHEMACODE et d'y remplacer les énoncés de contrôle par des structures schématiques. Le diagramme de la figure 2.3 illustre ce procédé:

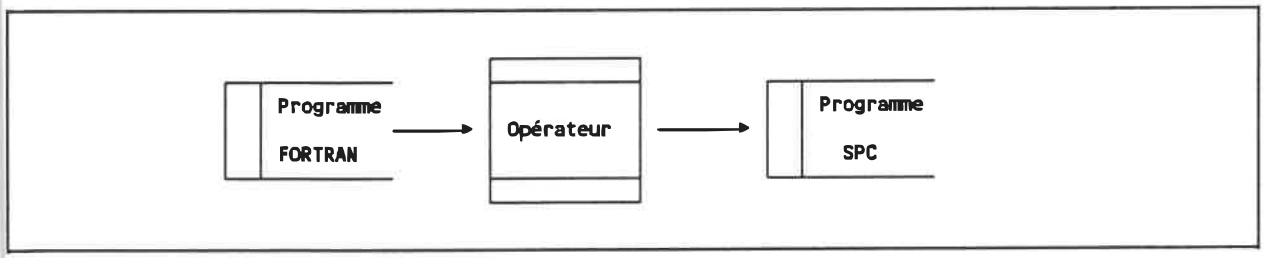


Fig. 2.3 - Système de traduction par inspection visuelle.

Ce qui donne (figure 2.4):

```
-Determine si un nombre est premier.
-
 PROGRAM PRIME
 INTEGER DIV,LIM
 REAL P,FLAG
 WRITE (0,*) 'Entrer un nombre entier'
 READ (0,*) P
 DIV = 2
 LIM = INT(SQRT(P))
       FLAG = P/DIV - INT(P/DIV)
     * FLAG.EQ.0 .OR. DIV.EQ.LIM
       *
       DIV=DIV+1
       FLAG.NE.0 .OR. P.LT.4
       WRITE(0,*) INT(P),' est premier.'
.....
       WRITE(0,*) DIV,' est le plus petit facteur de ',INT(P)
 END
```

Fig. 2.4 - Programme PRIME en SPC.

Examinons maintenant un autre programme (STRNG2) ayant environ le même nombre d'énoncés (figure 2.5):

```
        SUBROUTINE STRNG2
        DIMENSION IASCI(89),JLOWR(26)
        K = 1
        IERR = 0
        IERR2 = 0
        JCNT = 0
        IHOLD = IN(IPTR1)
15      IPTR1 = IPTR1+1
        INCHR = IN(IPTR1)
        IF(IPTR1.GE.ILONE) GO TO 910
        DO 20 I=1,63
        IF(JCHAR(I).NE.INCHR) GO TO 20
        GO TO 43
20      CONTINUE
        DO 30 I=1,26
        IF(JLOWR(I).NE.INCHR) GO TO 30
        GO TO 40
30      CONTINUE
        IERR = 1
        GO TO 15
40      I = I+63
43      IF(I-51) 45,55,45
45      IF(I-52) 15,60,15
55      IF(INEXT-JQUOT) 65,62,65
60      IF(INEXT-JDQOT)  65,62,65
62      IPTR1 = IPTR1+1
        GO TO 15
65      IF(JCNT.EQ.1) GO TO 990
910     IERR = 1
990     IPTR1 = IPTR1+1
        RETURN
        END
```

Fig. 2.5 - Procédure STRNG2.

La compréhension du programme, et donc sa traduction directe, est beaucoup plus difficile à cause des facteurs suivants:

- il y a plus d'imbrications de structures;
- il y a plus d'énoncés de contrôle;
- il y a plus de points de référence (étiquettes);

- le programme n'est pas documenté (notons que même si on ne peut pas se fier entièrement à elle, la documentation des énoncés aide la plupart du temps à la compréhension);
- le programme n'est pas structuré.

Ce dernier facteur est critique au niveau de la traduction car il n'existe pas de représentation SPC pour des programmes non-structurés. Conséquemment, une étape intermédiaire de restructuration est nécessaire mais n'est pas banale à implanter au niveau du texte (source) du programme.

Conclusions

Après avoir «attaqué» de cette façon plus de deux mille lignes de code réparties dans une quinzaine de procédures, nous en tirons les conclusions suivantes:

- la méthode de traduction directe est mal adaptée aux programmes de grande dimension;
- elle est insuffisante pour des programmes non-structurés;
- sa forte dépendance de l'opérateur la rend très sujette aux erreurs humaines;
- son temps de traduction est essentiellement imprévisible, et devient rapidement prohibitif en termes de coût.

Il est maintenant clair qu'on doit mettre au point une méthode qui soit non seulement systématique, mais aussi automatisée le plus possible.

2.2 Appproche semi-automatisée avec graphe de contrôle

La tentative suivante est approchée par prototypage. Il s'agit d'éliminer en grande partie l'intervention humaine au niveau du code source et d'introduire une représentation intermédiaire (graphe de contrôle). L'architecture du système prototype est présentée à la figure 2.6.

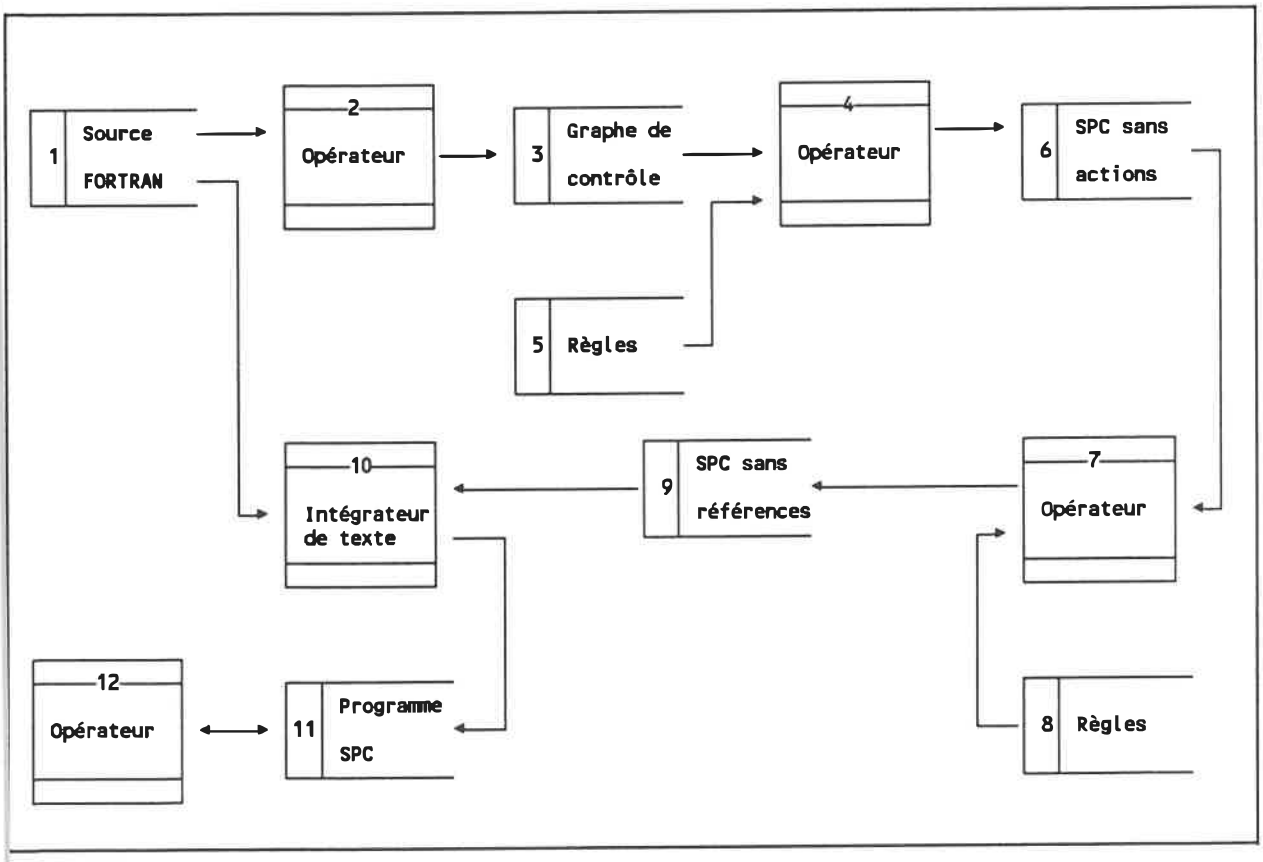


Fig. 2.6 - Système de traduction semi-automatisé.

Nous décrivons ici les onze composantes de ce système de traduction.

Le source FORTRAN (1) est lu par un analyseur lexical et syntaxique (2) écrit en langage C[1]. Cet analyseur, assez primitif, ne traite pas tous les cas permis par la grammaire de FORTRAN 77. Il convient cependant pour une étude de prototypage.

L'analyseur effectue deux phases de lecture du source FORTRAN: la première identifie la position des étiquettes tandis que la seconde produit un graphe de contrôle imprimé (3). Par exemple, à partir de la procédure illustrée illustré à la figure 2.7, l'analyseur crée le graphe de contrôle de la figure 2.8.

```
 1            FUNCTION LOOKUP(ITEM,TABLE,LONG)
 2            INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
 3            START=1
 4            FINISH=LONG
 5       1    I = (START+FINISH)/2
 6            WRITE(0,*) I
 7            IF (ITEM .EQ. TABLE(I)) GOTO 2
 8            IF (TABLE(I) .LT. ITEM) START = I+1
 9            IF (TABLE(I) .GT. ITEM) FINISH = I-1
10            IF (FINISH-START .GT.1) GOTO 1
11            IF (TABLE(START) .EQ. ITEM) GOTO 2
12            IF (TABLE(FINISH) .EQ. ITEM) GOTO 2
13            LOOKUP = 0
14            GOTO 3
15       2    LOOKUP = 1
16       3    RETURN
17            END
```

Fig. 2.7 - Procédure LOOKUP.

[1] Il est toujours sous-entendu que les programmes développés dans le cadre de ce projet l'ont été avec l'outil logiciel SCHEMACODE, avec le langage cible indiqué.

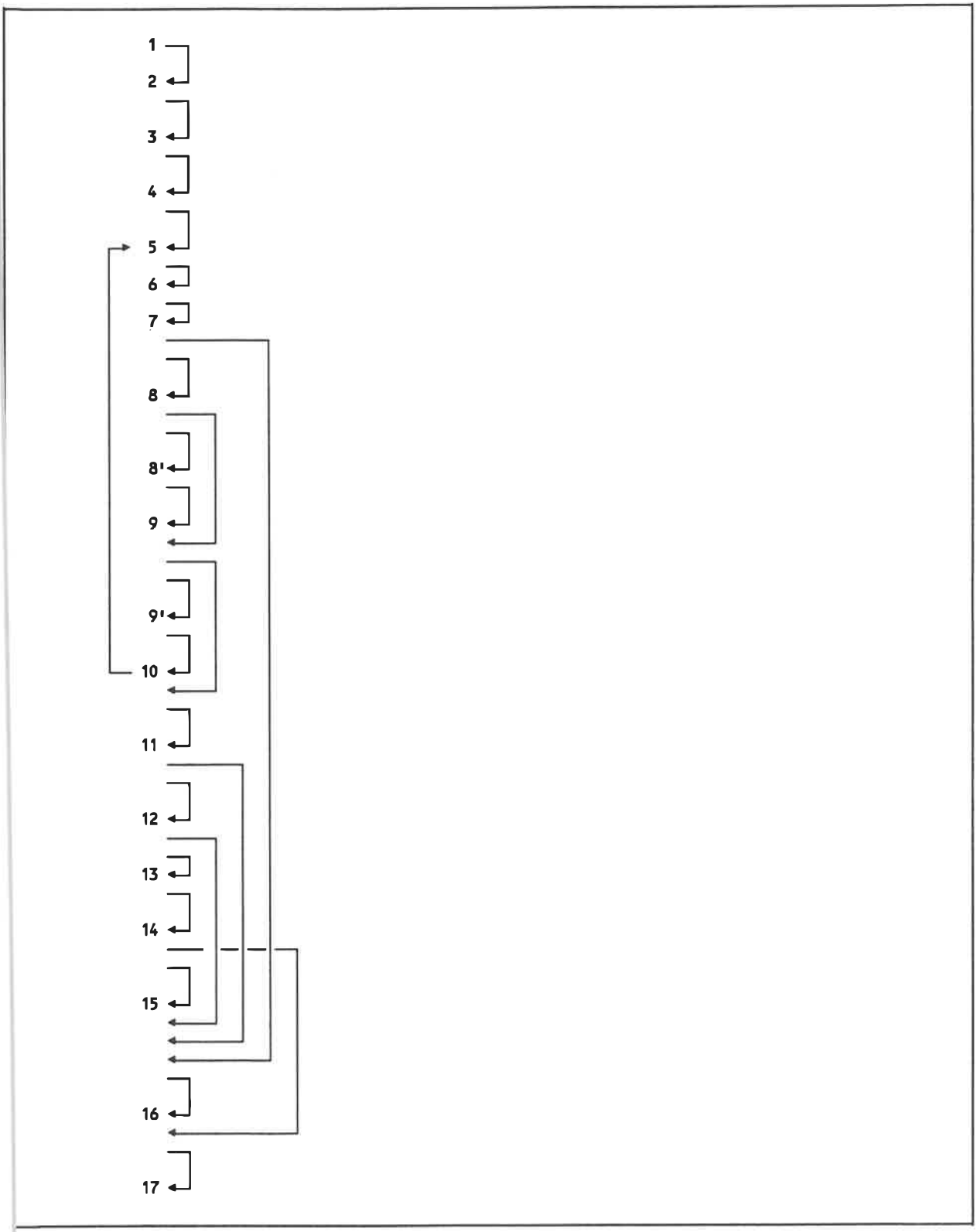


Fig. 2.8 - Graphe de contrôle de LOOKUP.

Le graphe de contrôle est constitué de sommets et d'arcs. Chaque arc représente une direction et le sens que le flux de contrôle du programme peut emprunter. Chaque sommet est un point de référence (arrivée) ou de départ d'un arc. Chaque énoncé FORTRAN produit un seul sommet sauf l'énoncé DO qui en produit plusieurs. Le nombre d'arcs dépend du type d'énoncé (séquentiel ou décisionnel).

A partir de ce graphe, un opérateur (4) applique des règles empiriques (5) pour créer un squelette de SPC (6). Ce SPC n'est pas véritablement un programme puisque ses actions ne sont que des références (numéros de ligne) au programme source original. De plus, sa logique reflète exactement celle du programme source; toutes les structures sont schématisées mais tous les bris de structure, prenant la forme de références (GOTO) à des énoncés, sont aussi représentés par un symbolisme textuel.

L'étape suivante, qui est aussi la plus difficile, consiste à éliminer les bris de structures du SPC sans actions(6). Cette tâche est réalisée par un opérateur (7) qui, encore une fois, applique un ensemble de règles (8). Ces règles sont dites de restructuration et se basent généralement sur trois techniques.

- la duplication de code qui, comme son nom l'indique, consiste à copier une ou plusieurs actions et ainsi éliminer des GOTOs;
- la mise en sous-programme qui, couplée à la duplication, évite l'augmentation prohibitive de la dimension du programme en regroupant le code dupliqué en une nouvelle procédure;
- l'introduction de variables de contrôle (boolean flags) qui permettent de «créer un chemin» entre un GOTO et sa destination.

Il en résulte un programme SPC exempt de toute référence interne (9), donc structuré (le flux de contrôle est représenté exclusivement par les schémas). L'étape qui suit est effectuée par un programme intégrateur (10) développé en C, qui remplace les numéros de ligne contenus dans le SPC sans actions (6) par les énoncés appropriés

pour former le SPC avec actions (11). Finalement, le SPC subit quelques corrections et modifications cosmétiques, effectuées par un opérateur (12).

Conclusions

La faisabilité de cette méthode a été démontrée sur les procédures cobayes de la première méthode (plus de deux mille lignes de code), puis sur d'autres procédures qui lui avaient résisté. Les avantages qui en ressortent sont les suivants;

- fiabilité accrue d'analyse du source par son automatisation;
- facilité de manipulation du programme à un niveau plus abstrait grâce au support fourni par le graphe de contrôle;
- clarté d'opération dans la structuration et dans le raffinement du SPC grâce à sa représentation intégrée [6];
- élimination d'erreurs humaines grâce à l'utilisation d'outils logiciels;
- adaptation facile des algorithmes suivis par l'opérateur à cause de la base de règles aisément modifiable.

Cependant, le système possède également des inconvénients non-négligeables:

- l'analyseur est trop simpliste et ne supporte pas la syntaxe complète du FORTRAN 77; l'utilisation de techniques et d'outils générateurs d'analyseurs syntaxiques est nécessaire;
- la présence du facteur humain, bien que réduite, est encore source d'erreurs;
- les règles restent empiriques et informelles, bien qu'elles soient raisonnables;
- l'intégrateur de texte est lui aussi limité dans ses capacités, ce qui rend l'intervention de l'opérateur (11) obligatoire;

- finalement, le processus, bien que toujours réalisable, reste très long.

2.3 Approche automatisée pour un langage structuré

Afin de se donner une bonne idée du fonctionnement d'un système totalement automatisé, nous avons décidé d'expérimenter sur un prototype. Le but de cette expérience est de vérifier qu'il est effectivement possible de produire des programmes SPC automatiquement, et que ces programmes restent «humainement» lisibles et reconnaissables.

Comme il s'agit surtout de tester les capacités de production de code, nous avons décidé d'effectuer le processus de traduction sur un langage source qui ne comporte pas les difficultés d'analyse du FORTRAN. Le choix s'est porté sur le langage dBASE III[2] [4], et ce pour les raisons suivantes:

- c'est un langage qui possède des énoncés de structures conditionnelles (IF-THEN-ELSE et DO CASE) et repétitives (DO-WHILE);
- sa syntaxe est très simple à cause de l'utilisation de mots réservés [2];
- il ne comporte pas d'étiquettes donc pas d'énoncés de référence;
- il est aussi supporté par SCHEMACODE.

Conséquement, il est possible de produire un analyseur dBASE III simple et complet, et ce dans un temps raisonnable.

L'architecture du prototype réalisé est la suivante (figure 2.9):

---

2 dBASE III est une marque déposée d'Ashton-Tate Inc.

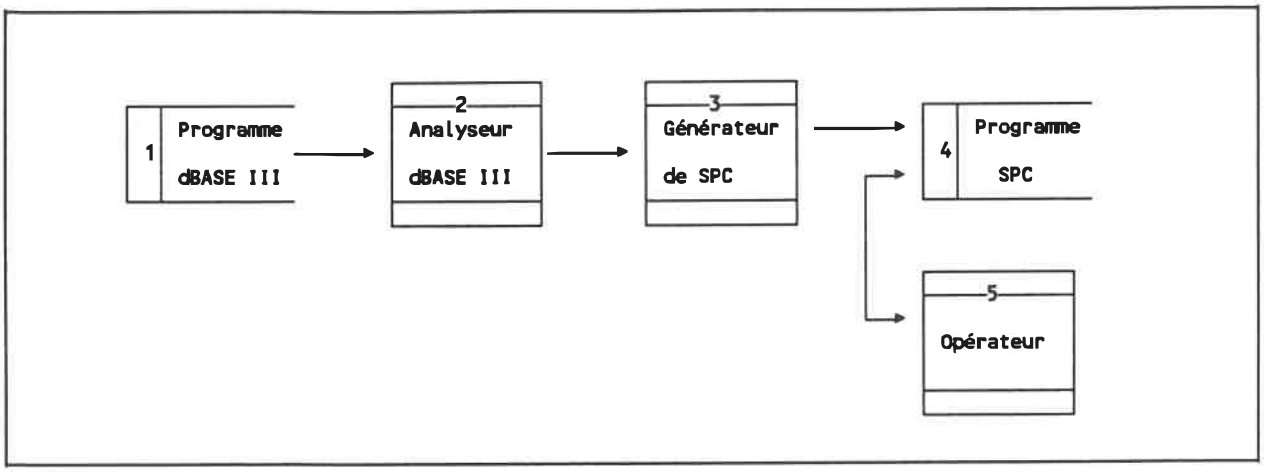


Fig. 2.9 - Système de traduction automatique pour dBASE III.

Le programme source (1) est lu une seule fois par un analyseur lexical (2). Ce dernier est produit par LEX [21] [2], un outil logiciel générateur d'analyseur lexical. A partir d'une description du vocabulaire (terminal symbols) d'un langage, LEX produit un analyseur lexical rapide en langage C, qui est par la suite compilé. Les symboles du vocabulaire y sont décrits à l'aide d'expressions régulières [2] [21], ce qui réduit beaucoup la tâche de programmation.

A chaque fois, qu'un symbole est reconnu par l'analyseur, une action spécifique est déclenchée. Ecrites en langage C, chacune de ces actions est, dans notre cas, dédiée à la production de SPC et constitue le générateur de code du système (3). Ces actions servent également à maintenir des informations relatives au programme global.

La figure 2.10 illustre un exemple de programme dBASE III; La figure 2.11 montre le SPC produit automatiquement.

```
* Open the data file
USE MYDATA
* Display available commands
clear
set talk off
set bell off
@ 7,20 say "Available commands :"
@ 9,20 say "(A)dd, (D)isplay, (E)nd"
* Ask for the desired operation
DO WHILE .T.
* Ask for the user's choice
choice= " "
@ 13,20 say "Your choice : (A/D/E)" get choice
read
IF upper(choice) = "E"
EXIT
ENDIF
* Determine operation
DO CASE
CASE upper(choice) = "A"
* Add a new record
append blank
@ 15,20 say "NAME........." GET NAME
@ 16,20 say "ADDRESS. . . ." GET ADDRESS
READ
CASE upper(choice) = "D"
* Display all records
CLEAR
? "**** LIST OF RECORDS ****"
GOTO TOP
DO WHILE .NOT. (eof())
? "Name    : ",name
? "Address: ",address
SKIP
ENDDO
? "*** Press <Enter> to continue ***"
wait " "
OTHERWISE
* Display an error message
@ 20,20 SAY "Bad choice. Try again."
@ 21,20 SAY "Press any key to continue"
wait " "
ENDCASE
ENDDO
* Close the data file
CLOSE DATABASES
```

Fig. 2.10 - Programme en dBASE III.

```
0  1 |-SCHEMACODE 1.5 - Jean-Bernard Trouve
   2 |-Open the data file
   3 |USE MYDATA
   4 |-Display available commands
   5 |clear
   6 |set talk off
   7 |set bell off
   8 |@ 7,20 say "Available commands :"
   9 |@ 9,20 say "(A)dd, (D)isplay, (E)nd"
  10 |-Ask for the desired operation
  11 |-01 .T. ----
1  2 |
   3 |     *         .NOT.(.T.)
   4 |            *
   5 |      -Ask for the user's choice
   6 |      choice= " "
   7 |      @ 13,20 say "Your choice : (A/D/E)" get choice
   8 |      read
   9 |      -02 upper(choice) = "E"
2  2 |               upper(choice) = "E"
  3  |             |-1     $$$---$E$---$$$---$E$---$$$---$E$-< EXIT >
   4 |             |EXIT
   5 |
1 10 |      -Determine operation
  11 |      -03 upper(choice) = "A"
3  2 |               upper(choice) = "A"
   3 |             |-Add a new record
   4 |             |append blank
   5 |             |@ 15,20 say "NAME.........." GET NAME
   6 |             |@ 16,20 say "ADDRESS. . . ." GET ADDRESS
   7 |             |READ
   8 |      ...     upper(choice) = "D"
   9 |             |-Display all records
  10 |             |CLEAR
  11 |             |? "***** LIST OF RECORDS *****"
  12 |             |GOTO TOP
  13 |             |-04 .NOT. (eof())
4  2 |             |
   3 |             |     *         .NOT.(.NOT. (eof()))
   4 |             |            *
   5 |             |       ? "Name    : ",name
   6 |             |       ? "Address: ",address
   7 |             |       SKIP
   8 |             |
3 14 |             |? "**** Press <Enter> to continue ***"
  15 |             |wait " "
  16 |      .....
  17 |             |-Display an error message
  18 |             |@ 20,20 SAY "Bad choice. Try again."
  19 |             |@ 21,20 SAY "Press any key to continue"
  20 |             |wait " "
  21 |
1 12 |
0 12 |-Close the data file
  13 |CLOSE DATABASES
  14 |---------------------------------- RDB3 --------- >>>>
  15 |- Fichier SPC produit par RDB3.  Source: EXDB3.PRG.
  16 |- Conversion completee avec succes...
  17 |- 1 enonces EXIT restants (0 difficiles)
  18 |- 58 lignes SPC produites
  19 |- 5 raffinements produits
  20 |---------------------------------- RDB3 --------- >>>>
```

Fig. 2.11 - Programme résultant en SPC.

Le programme SPC (4) produit est, en théorie directement utilisable par SCHEMACODE sans intervention humaine. Tous les détails cosmétiques sont traités par le générateur.

Il existe cependant des limites à ce système, basé sur une simple translittération, qui concernent deux énoncés de contrôle: EXIT et LOOP.

Ces deux énoncés ne sont permis seulement à l'intérieur d'une structure répétitive DO-WHILE. L'énoncé EXIT transfère le contrôle à l'énoncé END-DO suivant, i.e. il agit comme sortie de boucle. A l'inverse, l'énoncé LOOP redonne le contrôle au dernier énoncé DO-WHILE rencontré sans END-DO, ce qui crée un retour au début de la boucle.

Si un EXIT ou un LOOP est imbriqué dans une structure conditionnelle à plus d'un niveau, le programme n'est pas structuré et il devient impossible de générer du SPC directement. Dans notre système, c'est l'opérateur (5) qui, finalement, décide à l'aide de règles si chaque EXIT ou LOOP peut être enlevé. Dans les cas où cela n'est pas possible, une restructuration manuelle du SPC s'impose.

La figure 2.12 montre le même programme (modifié par un opérateur) dont les commentaires opérationnels ont été remaniés et l'énoncé EXIT éliminé et remplacé par une condition de sortie.

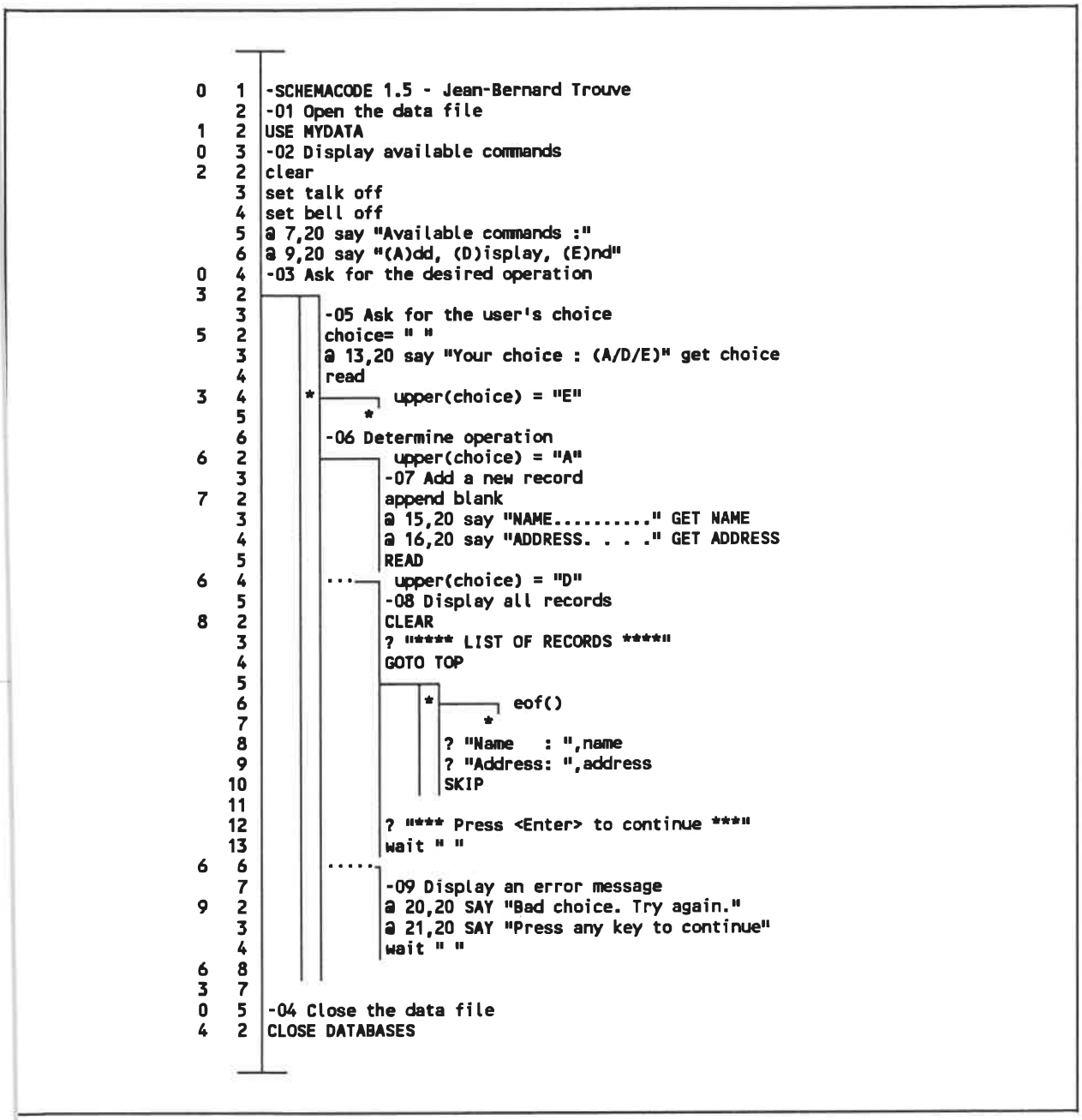


Fig. 2.12 - Programme SPC final tel que modifié par un opérateur.

Conclusions

Sans aller plus loin dans notre exploration, nous avons converti selon cette méthode plus de deux mille lignes de programmation dBASE III. Après examen des résultats nous tirons quelques conclusions:

- la translittération par énoncés est une technique simple et rapidement réalisable, mais qui possède des limites inhérentes au niveau de la «perception» du programme: trop réduite, manque de puissance de manipulation globale;
- la restructuration est hors de portée de cette méthode;
- la simplicité de réalisation de l'analyseur dBASE III montre à quel point est importante la difficulté reliée à l'analyse du FORTRAN, mais aussi comment des outils logiciels comme LEX peuvent nous aider à cette fin;
- on constate qu'il est effectivement possible de générer des programmes SPC qui ressemblent à ce qu'un programmeur aurait pu produire, et ce automatiquement;
- l'automatisme total permet d'éliminer complètement les erreurs dues à l'opérateur.

Notons finalement qu'une étude sur un système d'architecture similaire mais convertissant plutôt un sous-ensemble du langage C au lieu du dBASE III, a été réalisé par [39]. Les conclusions de cette étude concordent avec les nôtres.

2.4 Synthèse

A la lueur des études préliminaires décrites précédemment, il est possible de dégager les caratéristiques positives que chaque système a pu mettre en évidence:

- utilisation d'outils logiciels avancés pour réduire la difficulté du développement;
- fiabilité et complétude de l'analyseur FORTRAN;
- automatisation du processus (le moins d'intervention humaine possible);
- utilisation d'une approche par abstraction et ré-implantation basée sur le graphe de contrôle (forme intermédiaire abstraite);
- formalisation des transformations effectuées (sur le graphe ou ailleurs).

CHAPITRE 3

LE GRAPHE DE CONTROLE

Nature is indifferent towards the difficulties it causes to mathematicians (Fourier).

Les études préliminaires ont montré les avantages, voire la nécessité d'utiliser une forme intermédiaire abstraite, et plus spécifiquement le graphe de contrôle. Ce chapitre élabore ce thème afin de formaliser cette représentation.

Nous définissons la terminologie utilisée en survolant les bases de la théorie des graphes, puis nous présentons une forme normalisée et schématique du graphe de contrôle. Finalement, nous étudions trois formes de représentations pratiques de ce graphe: par base de données relationnelle, par matrice d'adjacence et par type abstrait de donnée.

3.1 Théorie des graphes

Avant de décrire le graphe de contrôle, définissons d'abord précisément quelques termes qui sont utilisés dans ce chapitre. Ces définitions sont tirées de [20], [11] et [26].

Graphe orienté

Un graphe orienté, G=(S,A), est un couple formé de deux ensembles: un ensemble fini non-vide S, appelé l'ensemble des sommets de G, et un ensemble A de paires de sommets (inclus dans le produit cartésien S X S), appelé l'ensemble des arcs de G.

Arc

Un arc est un couple (x,y) où x S et y S. Si a = (x,y) A, alors on dit que x est l'extrémité initiale de a et que y est l'extrémité finale de a. On dit que a est l'arc (ou flèche) de x à y. Notons que dans un graphe orienté, les arcs (x,y) et (y,x) sont distincts.

On peut représenter un graphe orienté par une figure dans le plan. Par exemple, la figure 3.1 représente le graphe orienté G= {S,A} où S= {1, 2, 3, 4, 5} et A= { (1,2), (2,3), (3,2), (3,4), (4,5) }.

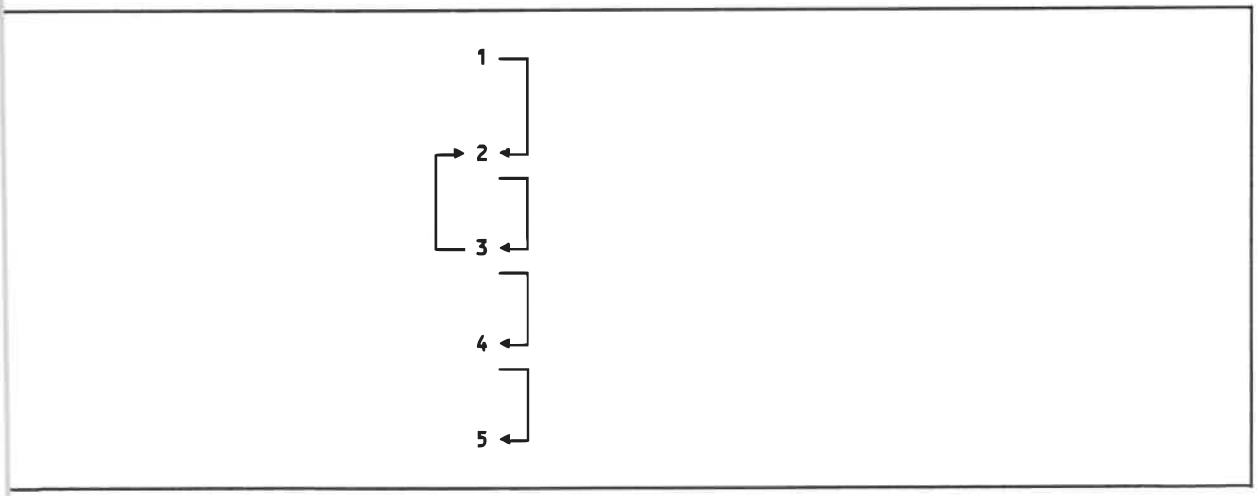


Fig. 3.1 - Exemple de graphe orienté.

Chemin

On appelle chemin [u,v] une séquence d'arcs (s0,s1)(s1,s2)...(sn-1,sn) où u=s0 et v=sn. Un chemin élémentaire est un chemin dans lequel un sommet apparaît au plus une fois. Un chemin simple est un chemin dans lequel un arc apparaît au plus une fois.

Demi-degré intérieur

Soit s S un sommet du graphe orienté G=(S,A), on définit le demi-degré intérieur (ou in-degree) d'un sommet:

d-(s)= Card { y | (y,s) A },

où «Card» est le cardinal, i.e. le nombre d'éléments de l'ensemble. Autrement dit, il s'agit, pour un sommet donné, du nombre d'arcs se terminant à ce sommet.

Demi-degré extérieur

Soit s S un sommet du graphe orienté G = (S,A), on définit le demi-degré extérieur (ou out-degree) d'un sommet:

d+(s) = Card { y | (s,y) A }.

Autrement dit, il s'agit, pour un sommet donné, du nombre d'arcs débutant à ce sommet.

Multigraphe orienté

Un multigraphe orienté est un couple (S,U) où U est une famille d'éléments de S X S. Autrement dit. c'est un graphe orienté qui admet plus d'un arc possédant les mêmes extrémités (sommets) initiales et finales. La figure 3.2 illustre un multigraphe orienté:

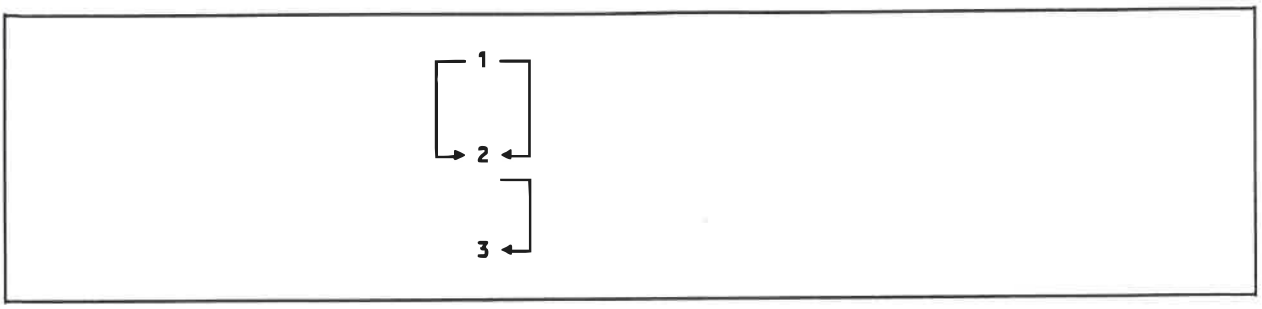


Fig. 3.2 - Exemple de multigraphe orienté.

Notons que les définitions des demi-degrés intérieur et extérieur s'étendent aux multigraphes orientés.

3.2 Représentation graphique du flux de contrôle

On appelle graphe de contrôle (ou flowgraph) le multigraphe orienté qui représente le flux de contrôle d'un programme.

Le graphe de contrôle montre la structure de contrôle d'un programme sans définir en détails les actions qu'il effectue. Un graphe de contrôle représente donc une famille de programmes distincts possédant la même structure de contrôle.

Plusieurs auteurs utilisent les graphes[1] pour représenter le flux de contrôle. Citons [25] (qui l'utilisa pour tenter de mesurer la complexité des programmes), [13], [26], [27], [45], [46], [47] et [14]. Mentionnons [12] qui a grandement contribué à leur application dans ce domaine.

Le graphe est habituellement produit à partir d'un organigramme (flowchart), qui représente le flux de contrôle. Une fois l'organigramme construit, on assigne un sommet à chaque traitement (rectangle), décision (losange) et connecteur (cercle), puis on relie les sommets par des arcs de la même façon que les symboles sont reliés par des flèches. La figure 3.3 illustre un exemple de programme, accompagné de son organigramme (figure 3.4) et du graphe résultant (figure 3.5). Ce programme, PRIME, est inspiré de [46].

---

1 Dorénavant, lorsque dans le texte nous utiliserons le terme «graphe», nous sous-entendrons «multigraphe orienté» (comme la plupart des auteurs le font).

```
1       C Determine si un nombre est premier.
2       C Exemple tire de Whitty, Fenton et Kaposi.
3             PROGRAM PRIME
4             INTEGER DIV,LIM
5             REAL P,FLAG
6             WRITE (0,*) 'Entrer un nombre entier'
7             READ (0,*) P
8             DIV = 2
9             LIM = INT(SQRT(P))
10      20000    FLAG = P/DIV - INT(P/DIV)
11               IF (FLAG.EQ.0 .OR. DIV.EQ.LIM) GOTO 20001
12               DIV=DIV+1
13               GOTO 20000
14      20001 IF (FLAG.NE.0 .OR. P.LT.4) THEN
15               WRITE(0,*) INT(P),' est premier.'
16            ELSE
17               WRITE(0,*) DIV,' est le plus petit facteur de ',INT(P)
18            END IF
19            END
```

Fig. 3.3 - Programme PRIME en FORTRAN.

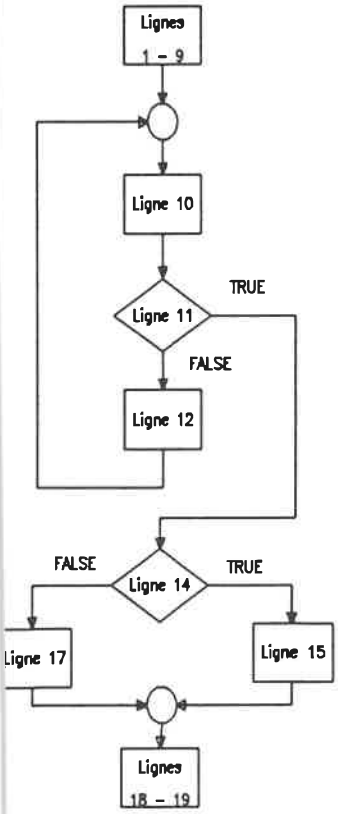


Fig. 3.4 - Organigramme de PRIME.

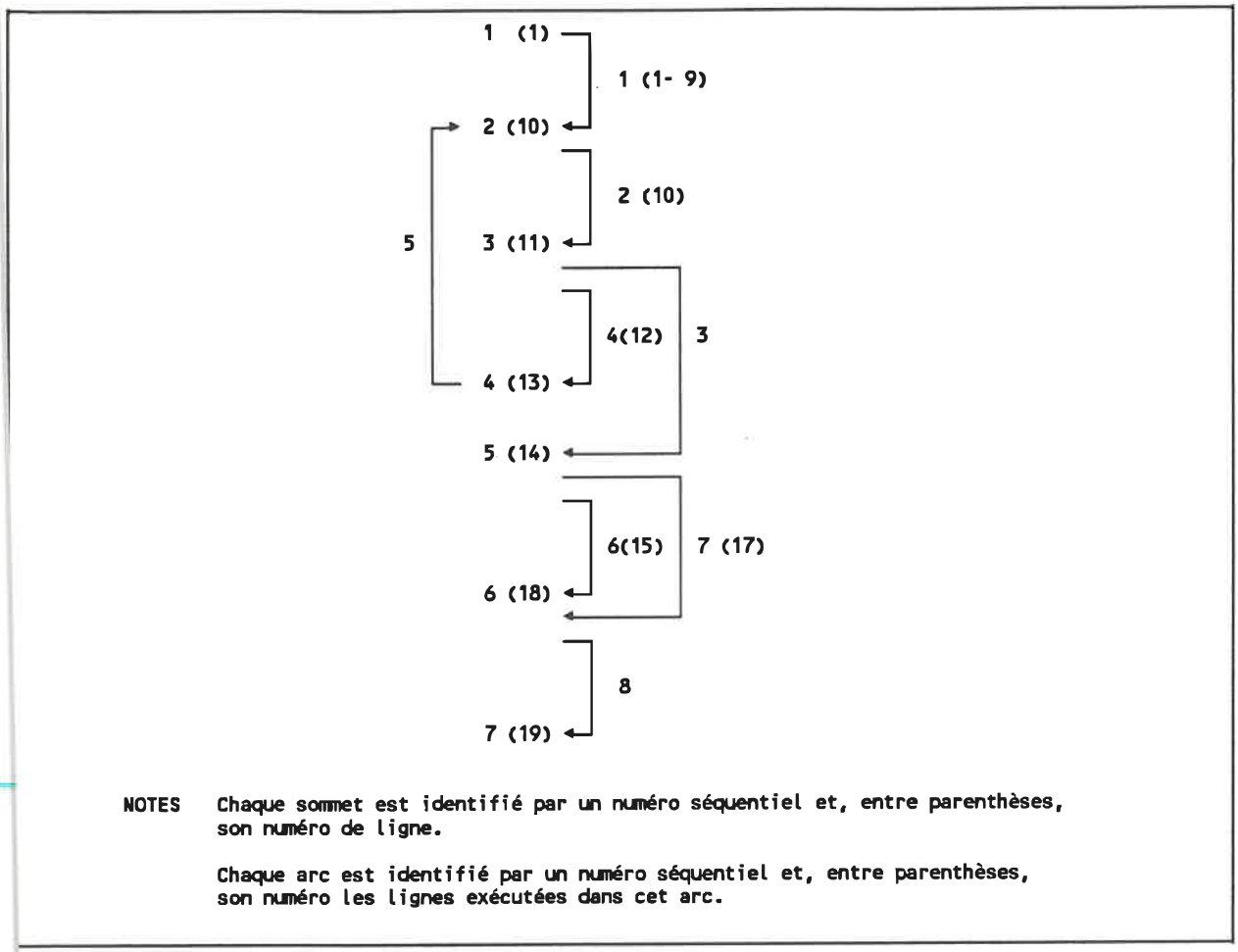


Fig. 3.5 - Graphe de contrôle de PRIME.

Une représentation légèrement différente est produite par le logiciel DATRIX [10]; elle reste cependant tout-à-fait équivalente dans le cadre de notre étude. L'avantage de cette représentation réside dans la systématisation de sa production. En voici les caractéristiques générales:

- à chaque énoncé est assigné un sommet;
- chaque sommet possède un numéro d'identification séquentiel qui correspond à sa position dans le texte du programme source (1 pour le premier énoncé, 2 pour le suivant, etc);

- les sommets sont alignés verticalement par ordre croissant de numéro d'identification;
- les sommets sont représentés par des lignes simples horizontales;
- les arcs descendants sont représentés par des lignes verticales simples;
- les arcs ascendants sont représentés par des lignes verticales doubles.

La figure 3.6 montre le graphe DATRIX équivalent au graphe précédent;

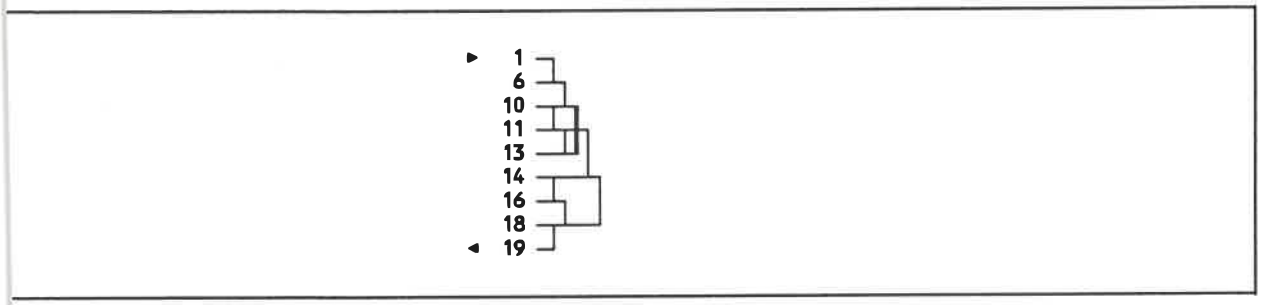


Fig. 3.6 - Graphe de contrôle de PRIME (format DATRIX).

Cette représentation est utilisée dans le reste de l'ouvrage.

3.3 Structure de données et application

[27], [45], [49] et d'autres, ont démontré que la restructuration d'un programme s'effectue à l'aide d'opérations sur le graphe, qu'on retraduit ensuite sous forme de texte en langage de haut niveau. Ces opérations, ainsi que toute autre, doivent cependant se produire sur une réalisation particulière de structure de données de graphe.

Dans notre démarche, trois techniques sont envisagées: la base de données relationnelle, la matrice d'adjacence et le type abstrait de données. Chacune de ces techniques est examinée avec ses avantages et ses inconvénients.

3.3.1 Graphe par base de données relationnelle

Une base de données contenant un ensemble de sommets et un ensemble d'arcs est construite en utilisant directement la définition du graphe. Le diagramme entité-association suivant illustre cette construction:

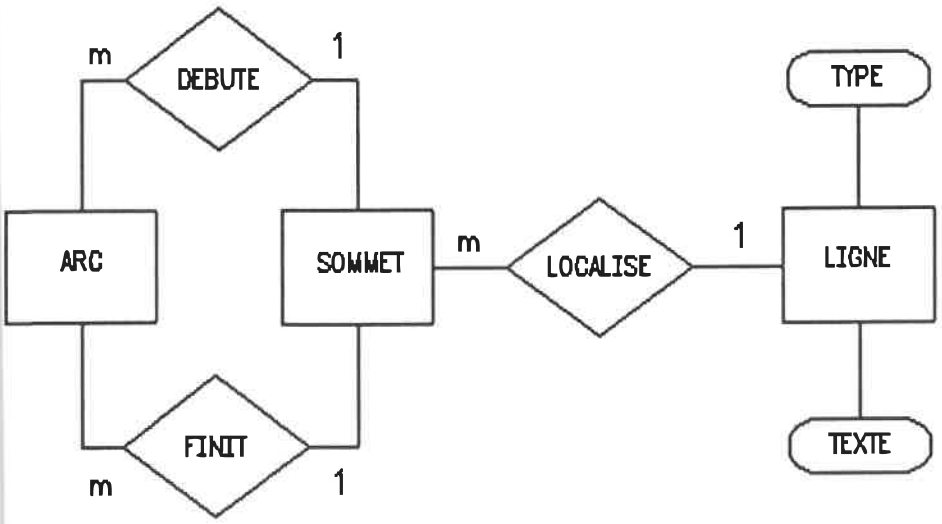


Fig. 3.7 - Diagramme entité-association de la base de données.

L'entité SOMMET est au coeur du diagramme. C'est simplement l'ensemble de tous les sommets du graphe. L'entité ARC est l'ensemble des arcs. Chaque arc est relié à deux sommets: un par l'association DEBUTE (qui indique le sommet initial) et un pour l'association FINIT (qui indique le sommet final). Le degré 1:m de l'association montre qu'un arc ne possède qu'un seul sommet initial et final, mais qu'un sommet peut être le début ou la fin de plusieurs arcs.

L'entité LIGNE représente le texte du programme source. Chaque sommet est lié à une ligne unique par l'association ASSOCIE-A, tandis qu'une ligne de programme peut avoir produit plusieurs sommets, tout dépendant de son TYPE (qui est un attribut de LIGNE). Le texte de la ligne est représenté par l'attribut TEXTE.

Afin d'expérimenter avec ce schéma conceptuel, il faut l'implanter dans une forme moins abstraite. Des trois formes les plus classiques (hiérarchique, réseau, relationnelle) [42], nous retenons l'approche relationnelle à cause de sa simplicité d'utilisation et de sa rapidité d'implantation (rappelons-nous qu'il s'agit d'un prototype). La décomposition du schéma en tables s'effectue de la façon suivante (troisième forme normale):

ARC (NO-ARC, NO-SOMMET-INITIAL, NO-SOMMET-FINAL)
LOCALISE (NO-SOMMET, NO-LIGNE)
LIGNE (NO-LIGNE, TYPE, TEXTE)

Chaque rangée dans la table ARC représente un arc du graphe de contrôle. Un arc est identifié par un numéro unique (clé NO-ARC) et possède un sommet de départ (NO-SOMMET-INITIAL) et d'arrivée (NO-SOMMET-FINAL) identifiés par des numéros de sommet uniques. La table ARC contient donc l'entité ARC et les associations (DEBUTE et FINIT).

La table LOCALISE assigne un numéro de ligne de la table LIGNE à chaque sommet (initial ou final) qui existe dans la table ARC. C'est l'implantation de l'association LOCALISE-A.

Finalement, la table LIGNE contient, pour chaque ligne de source, identifiée par NO-LIGNE, le texte issu du programme original (TEXTE) de même que le type d'énoncé (séquentiel, IF-GOTO, IF arithmétique, etc.).

Un fait intéressant à noter: il n'y a pas d'information particulière attachée aux sommets. Les sommets peuvent être perçus comme étant une conséquence des arcs (et de leur topologie) et non pas des entités indépendantes. Ceci est dû au fait qu'un sommet ne peut exister sans qu'au moins un arc y soit relié, puisque dans le flux de contrôle, il y a toujours une instruction suivante à exécuter. Les figures 3.8 à 3.10 illustrent la représentation sous forme de tables du programme PRIME déjà étudié.

ARC	NO-ARC	NO-SOMMET-INITIAL	NO-SOMMET-FINAL
	1	1	2
	2	2	3
	3	3	5
	4	3	4
	5	4	2
	6	5	6
	7	6	6
	8	6	7

Fig. 3.8 - Table ARC.

LOCALISE	NO-SOMMET	NO-LIGNE
	1	1
	2	10
	3	11
	4	13
	5	14
	6	18
	7	19

Fig. 3.9 - Table LOCALISE.

LIGNE	NO-LIGNE	TYPE	TEXTE
	1	COMM	Determine si un nombre est premier.
	2	COMM	Exemple tire de Whitty, Fenton et Kaposi.
	3	ACTION	PROGRAM PRIME
	4	ACTION	INTEGER DIV,LIM
	5	ACTION	REAL P,FLAG
	6	ACTION	WRITE (0,*) 'Entrer un nombre entier'
	7	ACTION	READ (0,*) P
	8	ACTION	DIV = 2
	9	ACTION	LIM = INT(SQRT(P))
	10	ACTION	FLAG = P/DIV - INT(P/DIV)
	11	IFGOTO	IF (FLAG.EQ.0 .OR. DIV.EQ.LIM) GOTO 20001
	12	ACTION	DIV=DIV+1
	13	GOTO	GOTO 20000
	14	IFTHEN	IF (FLAG.NE.0 .OR. P.LT.4) THEN
	15	ACTION	WRITE(0,*) INT(P),' est premier.'
	16	ELSE	ELSE
	17	ACTION	WRITE(0,*) DIV,' est le plus petit facteur de ',INT(P)
	18	ENDIF	END IF
	19	ACTION	END

Fig. 3.10 - Table LIGNE.

Le logiciel dBASE III est utilisé pour créer les tables de relations; il ne reste qu'à les remplir. Au lieu de procéder par analyse humaine du source pour en extraire le graphe, nous utilisons l'analyseur du logiciel DATRIX et réalisé un programme de conversion du graphe de contrôle. Ce programme lit et interprète un fichier de données DATRIX et produit une description équivalente en ASCII, que dBASE III peut accepter.

Une fois les tables remplies, quelques programmes sont réalisés en langage dBASE III pour évaluer la performance générale de cette structure de données. Après avoir traité quelques dizaines de graphes de contrôle de dimensions variables nous en tirons les observations suivantes:

- le traitement est très lent, même lorsque les programmes de traitement sont compilés sous forme de modules exécutables [3], la base de données étant conservée sur disque;

- les opérations implantées, même si elles sont conceptuellement très simples, exigent des algorithmes complexes étant donné les limitations du langage de programmation;

Pour ces raisons, l'approche par base de données est rapidement mise de côté. La lenteur et la complexité du traitement entraînent un coût prohibitif, d'autant plus que la vraie difficulté de ce traitement, la restructuration comme telle, n'est même pas encore abordée.

3.3.2 Graphe par matrice d'adjacence

Afin d'éviter les problèmes rencontrés avec l'approche précédente, nous effectuons une autre tentative en changeant la structure de données et le langage de programmation.

La nouvelle structure de donnée est connue sous le nom de matrice d'adjacence [20] [23]. Cette matrice est carrée: pour chaque sommet existant dans le graphe, on crée une rangée et une colonne dont la position (1,2,3...) est assignée à un et un seul sommet, identifié par un numéro (1,2,3...). Finalement, chaque élément mij de la matrice est initialisé au nombre d'arcs (i,j) qui existent dans le graphe. Un exemple, tiré du graphe du programme PRIME de la figure 3.5, est fourni à la figure 3.11.

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	1	0	0
4	0	1	0	0	0	0	0
5	0	0	0	0	0	2	0
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

Fig. 3.11 - Matrice d'adjacence du graphe de PRIME.

Cette forme est mathématiquement intéressante car des opérations matricielles permettent d'en extraire, par exemple, l'existence de chemins entre deux sommets, leur longueur, etc. [20]. Cependant, elle est insuffisante parce qu'elle ne permet pas de conserver d'information distincte pour chaque arc lorsque plusieurs arcs joignent le même sommet (par exemple, à la cinquième rangée de la matrice de la figure 3.11).

La solution à ce problème consiste à ajouter une dimension à la matrice, une «profondeur». Chaque élément m i j k peut alors contenir l'information particulière à chaque arc. Nous verrons au chapitre suivant pourquoi cela est important.

Malheureusement, cette représentation, bien que simple conceptuellement, est difficilement réalisable en pratique. En effet, un graphe ayant 300 sommets (ce qui est assez courant) et pas plus de 10 arcs joignant les mêmes sommets (cas habituel) exige une matrice de 300 X 300 X 10, ce qui représente 900 000 éléments. Si on veut conserver ne serait-ce qu'un pointeur (4 octets) comme information (ce qui est déjà insuffisant), il faut disposer de plus de 3,4 Moctets de mémoire, ce qui dépasse largement les limites imposées par le système d'exploitation utilisé (MS-DOS).

Evidemment, comme la matrice est habituellement creuse (la plupart de ses éléments sont nuls), différentes techniques peuvent être utilisées pour limiter les exigences d'espace mémoire [23]. Cependant, cela augmente substantiellement la complexité des algorithmes, d'autant plus qu'il y a trois dimensions.

Malgré ces inconvénients, nous réalisons les mêmes opérations que sur la première implantation (base de données) afin d'évaluer les gains de rapidité et de facilité de programmation. Le langage C est utilisé ainsi qu'une matrice à deux dimensions du type illustré précédemment. Notons quelques observations:

- l'utilisation du langage C et la conservation des informations en mémoires centrale accroissent de façon considérable la vitesse de traitement;

- les algorithmes sont généralement plus simples avec cette structure de données (matrice) et ce langage de programmation (C);
- la forme complète de représentation exige cependant trop de mémoire pour notre environnement de développement.

La section suivante présente une solution alternative qui consiste à conserver le langage C pour sa rapidité et sa puissance mais à utiliser une autre représentation pour le graphe de contrôle.

3.3.3 Graphe par type abstrait de données

Une programmathèque (library) de types abstraits de données a été développée par [17] pour permettre la manipulations de piles, files, listes et graphes en langage C. Basé sur les principes d'information hiding, cet ensemble de sous-programmes définit des opérations élémentaires de manipulation de ces types de données (créations, ajout, retrait, etc). Il est très versatile et permet. à l'instar des tables relationnelles, de repésenter tout type d'objet.

Les avantages de cette approche sont immédiatement évidents: les données sont conservées en mémoire centrale pour assurer une rapidité d'accès maximale, les fonctions d'opération sont simples d'utilisation et font abstraction de l'organisation interne des données, et l'interface au langage C assure la rapidité et la puissance de programmation.

D'une façon similaire à la base de données relationnelle, les graphes de cette progammathèque sont constitués de deux ensembles: un pour les sommets, une pour les arcs. Chacun de ces ensembles est construit à l'aide d'un autre type abstrait: la liste. Les fonctions de manipulation de graphe opèrent sur ces deux listes d'une façon tout-à-faite transparente.

La liste de sommets contient un élément par sommet du graphe, l'information sur un sommet comprend un numéro d'identification, le numéro de ligne de source correspondante, le type de ligne ainsi que les demi-degrés intérieur et extérieur.

La figure 3.12 illustre cette liste de sommets pour le graphe du programme PRIME (figures 3.3 et 3.5).

NO-SOMMET	NO-LIGNE	TYPE	D+	D-
1	1	DEPART	1	0
2	10	ETIQ.	1	2
3	11	IFGOTO	2	1
4	13	GOTO	1	1
5	14	IFTHEN	2	1
6	18	ENDIF	1	2
7	19	TERM.	0	1

Fig. 3.12 - Liste des sommets du programme PRIME.

La liste des arcs contient un élément par arc. On y retrouve, pour chaque arc, un numéro d'identification, le texte des actions exécutées dans cet arc ainsi que la condition d'exécution (plus de détails au chapitre suivant). Les sommets initial et final sont conservés dans une liste invisible que seules accèdent les fonctions de manipulation de graphe.

La figure 3.13 illustre la liste d'arcs pour le programme PRIME.

NO-ARC	CONDITION	TEXTE
1	.TRUE.	T Determine si un nombre est premier. T Exemple tire de Whitty, Fenton et Kaposi. A PROGRAM PRIME A INTEGER DIV,LIM A REAL P,FLAG A WRITE (0,*) 'Entrer un nombre entier' A READ (0,*) P A DIV = 2 A LIM = INT(SQRT(P))
2	.TRUE.	A FLAG = P/DIV - INT(P/DIV)
3	FLAG.EQ.0 .OR. DIV.EQ.LIM	<nil>
4	.NOT.(FLAG.EQ.0 .OR. DIV.EQ.LIM)	DIV=DIV+1
5	.TRUE.	<nil>
6	(FLAG.NE.0 .OR. P.LT.4)	A WRITE(0,*) INT(P),' est premier.'
7	.NOT.(FLAG.NE.0 .OR. P.LT.4)	A WRITE(0,*)DIV,'est petit facteur de',INT(P)
8	.TRUE.	A END

Fig. 3.13 - Liste des sommets du programme PRIME.

Les fonctions de manipulation de listes suivantes sont disponibles:

- créer une liste (LSTOPEN);
- ajouter un élément à la liste (LSTADD);
- retirer un élément de la liste (LSTREMOV);
- accéder à l'information d'un élément de la liste (LSTVIEW);
- déplacer le pointeur d'élément courant en absolu ou en relatif (LSTGO);
- détruire la liste (LSTCLOSE).

On crée un graphe en liant les deux liste et on le manipule à l'aide des fonctions suivantes:

- créer un graphe à partir de deux listes (GPHOPEN);
- ajouter un arc (GPHADD);

- retirer un arc (GPHDELBC);
- retirer un sommet (GPHDELNO);
- obtenir l'adresse d'un arc (GPHGET);
- positionner le pointeur d'arc courant (GPHGO);
- obtenir l'adresse du sommet final de l'arc (GPHGETNO2);
- détruire le graphe (GPHCLOSE).

Plus précisement, on crée d'abord la liste de sommets (LSTOPEN), qu'on remplit (LSTADD). Puis, on remplit la liste d'arcs (GPHADD). Pour naviguer dans le graphe, on déplace le pointeur de la liste de sommets au sommet désiré (LSTGO); à partir de ce sommet, on accède aux arcs entrants ou sortants (GPHGO). Pour modifier le graphe, on peut ajouter un sommet (LSTADD), détruire un sommet (GPHDELNO), ajouter un arc (GPHADD) ou détruire un arc (GPHDELBC).

L'implantation du graphe n'est ni visible ni accessible autrement que par les fonctions mentionnées. Cependant, la figure 3.14 montre que le graphe est basé sur une liste de liens contenant un sommet de départ et un sommet d'arrivée (tous deux dans la liste de sommets), et un identificateur d'arc servant de repère dans la liste d'arcs. Cette liste est indépendante de la liste d'arcs car elle est entièrement invisible par l'application.

NO-ARC	NO-SOMMET-INITIAL	NO-SOMMET-FINAL
1	1	2
2	2	3
3	3	5
4	3	4
5	4	2
6	5	6
7	6	6
8	6	7

Fig. 3.14 - Représentation du graphe par liste de liens.

Des programmes de mise en forme standard (similaires à ceux réalisés lors des deux approches précédentes) sont développés et appliqués aux mêmes programmes sources (plus de 2000 lignes de code). Les conclusions sont claires: l'implantation du graphe par type abstrait de données, à l'aide de la programmathèque mentionnée, est très efficace tant au point de vue espace mémoire et rapidité d'exécution que facilité de programmation.

C'est donc cette structure de données que nous avons utilisée lors de la réalisation.

CHAPITRE 4

METHODE DE TRADUCTION

Mi par di veder un gran lume (Fermat).

Maintenant que les concepts de base ont été présentés, nous décrivons la méthode de traduction elle-même. Elle se décompose en trois étapes successives:

- la traduction du code source en graphe de contrôle;
- la structuration du graphe de contrôle;
- la traduction du graphe de contrôle en pseudocode schématique

Nous examinons chacune de ces étapes successivement et nous les illustrons par un exemple.

4.1 Traduction du code source en graphe de contrôle

La première étape est l'analyse du programme source et sa traduction en graphe de contrôle. Ce processus est basé sur celui décrit dans [10].

Il s'agit premièrement d'identifier quel type de sommet va produire chaque énoncé. On distingue trois types de sommets, caractérisés par leurs demi-degré extérieur (ces types constituent un sous-ensemble des types définis par [26]):

- les sommets de terminaison ou halt nodes (d+ = 0)
- les sommets séquentiels ou chain nodes (d+ = 1)
- les sommets de décision ou decision nodes (d+ > 1)

Une procédure ne contient par définition qu'un seul sommet de terminaison. C'est une façon de localiser de façon unique, dans le graphe, le retour à la procédure appelante ou l'arrêt de programme.

Chaque énoncé FORTRAN produit un sommet qui représente le point de référence (ou l'adresse) de l'énoncé. A partir de ce sommet peuvent débuter un ou plusieurs arcs: un seul dans le cas des énoncés séquentiels, plusieurs dans le cas des énoncés de décision.

Chacun de ces arcs contient deux informations particulières:

- une condition d'exécution (expression booléenne);
- des actions (calculs, entrées/sorties), effectuées losque la condition d'exécution est vérifiée;

NOTES L'arc sortant d'un sommet séquentiel étant toujours exécuté, nous disons que sa condition d'exécution est toujours vraie.

Il est possible qu'aucune action ne soit exécutée dans l'arc; on dit alors que l'action est nulle et le symbole la représente.

Ces deux informations constituent une étiquette qui est apposée à chaque arc [26] [27]. La forme de l'étiquette est p.a, où p est la condition d'exécution et a l'action effectuée. Si l'action est exécutée inconditionnellement, la forme de l'étiquette est simplement .a .

Une fois le graphe construit, l'arrangement des arcs fait apparaître une autre caractéristique des sommets, basée cette fois sur le demi-degré intérieur (d-). On distingue alors:

- les sommets de départ ou start nodes (d- = 0);
- les sommets de convergence ou collector nodes (d- > 1).

Notons que cette caractéristique est indépendante du type de sommet et est une conséquence de la disposition des énoncés, et non pas des énoncés eux-mêmes. Mentionnons également que les sommets ayant d- = 1 n'ont pas de signification particulière dans notre cas.

Examinons maintenant les énoncés FORTRAN et observons quels sous-graphes sont produits par eux.

Enoncés séquentiels

Chaque énoncé séquentiel (i.e. affectation, commentaire, énoncé d'entrée/sortie, appel à procédure, etc.) produit un sommet et un arc sortant. Ce dernier, qui se termine au sommet correspondant à l'énoncé suivant, possède une condition vraie et une action qui est l'énoncé lui-même.

Notons que DATRIX utilise une variante: il produit un seul arc pour tout bloc d'énoncés séquentiels exempt d'étiquette. On considère alors que l'action est décrite par la séquence complète d'énoncés.

Voici un exemple d'énoncé séquentiel (figure 4.1):

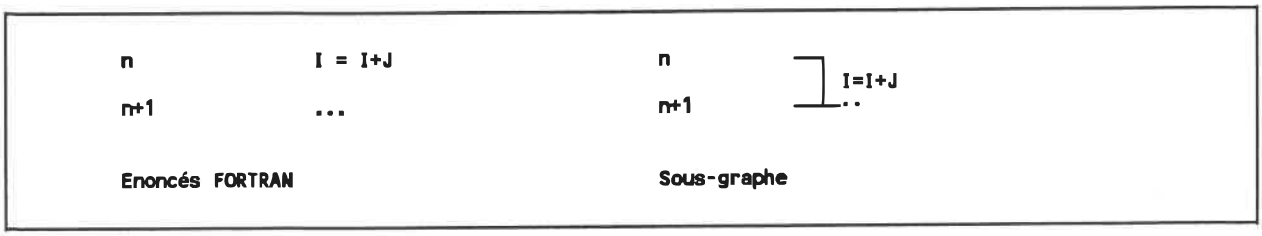


Fig. 4.1 - Sous-graphe d'un énoncé séquentiel.

Enoncé GOTO

L'énoncé GOTO produit un sommet séquentiel. L'arc sortant est exécuté inconditionnellement et son action est nulle.

Voici deux exemples d'énoncés GOTO (figure 4.2):

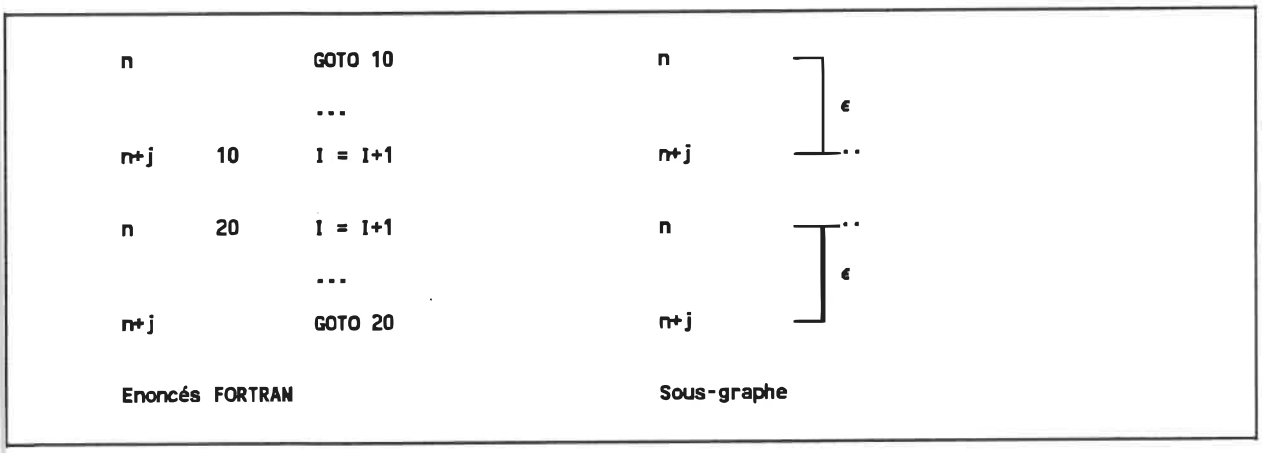


Fig. 4.2 - Sous-graphes d'énoncés GOTO.

Enoncé IF-GOTO

Cet énoncé produit un sommet conditionnel avec deux arcs sortants. Bien que ces arcs aient des actions nulles, chacun possède une condition d'exécution qui est la négation de l'autre.

Voici deux exemples d'énoncé IF-GOTO (figure 4.3):

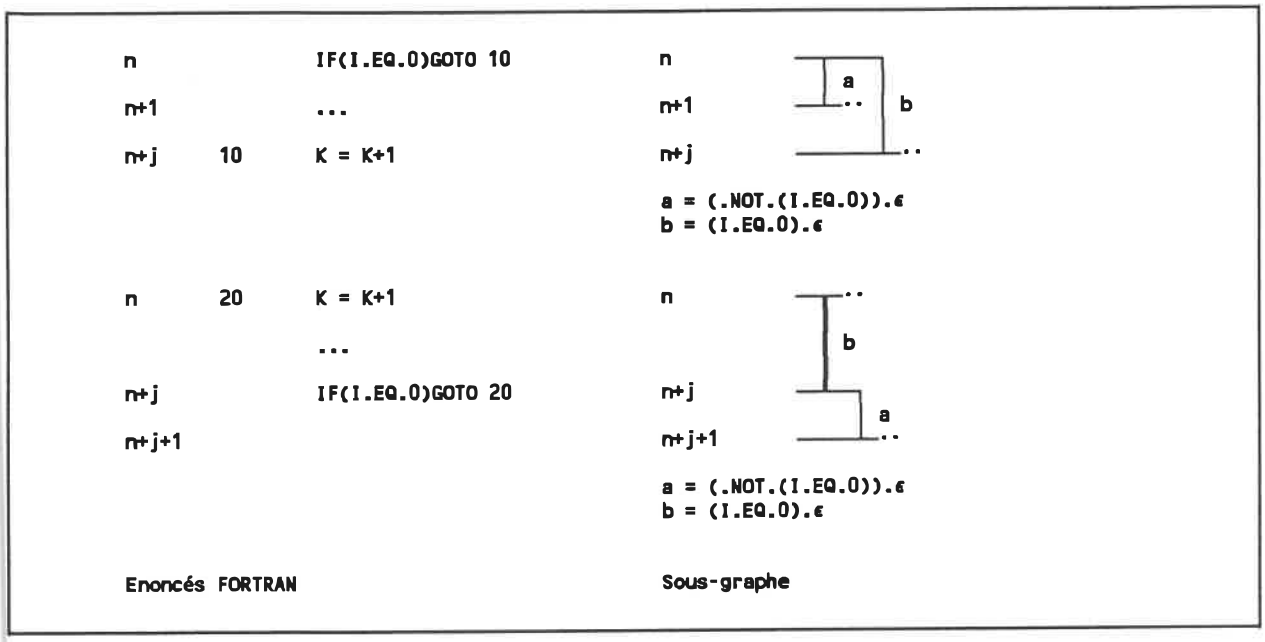


Fig. 4.3 - Sous-graphe d'énoncés IF-GOTO.

Enoncé GOTO calculé

Cet énoncé produit un sommet conditionnel avec N+1 arcs sortant, N étant le nombre d'étiquettes présentes dans l'énoncé. Tous les arcs ont des actions nulles. Les conditions d'exécution sont de la forme expression.EQ.valeur, où expression est l'expression numérique de l'énoncé, et valeur varie de 1 à N. L'arc supplémentaire va à l'énoncé suivant et a une condition de la forme (expression .LT.1).OR.(expression .GT.N).

Voici un exemple d'énoncé GOTO calculé (figure 4.4):

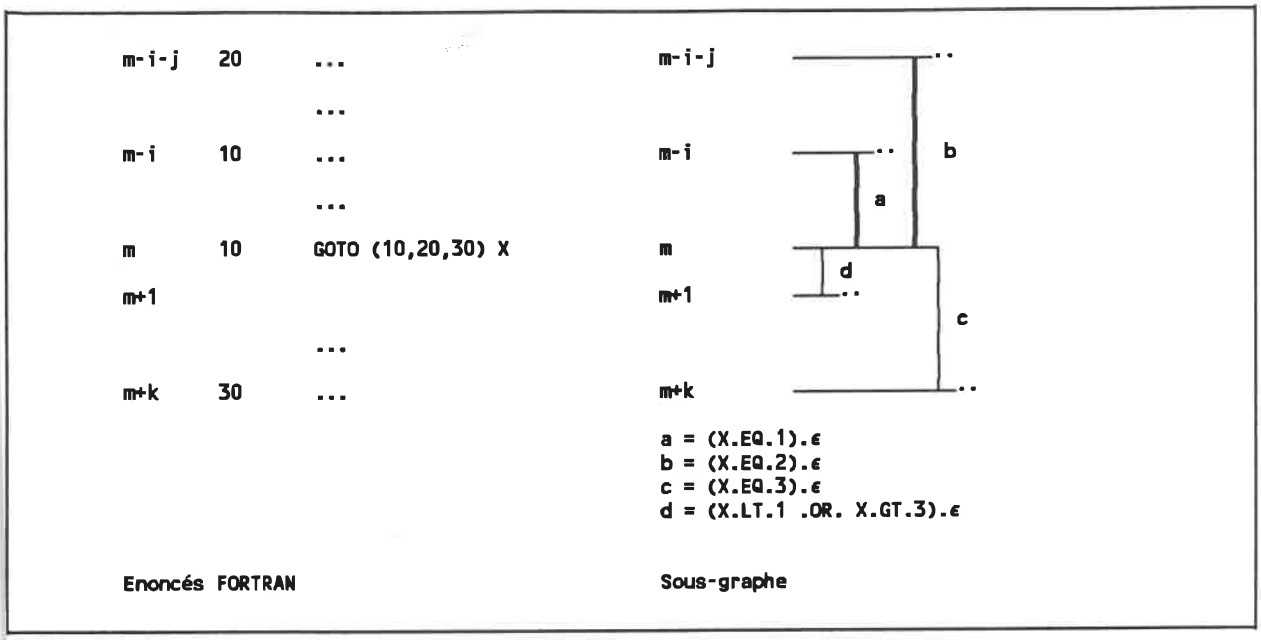


Fig. 4.4 - Sous-graphe d'un énoncé GOTO calculé.

Enoncé IF arithmétique

Cet énoncé produit un sommet conditionnel avec trois arcs sortants. Les conditions d'exécution sont expression.LT.0, expression.EQ.0 et expression.GT.0, où expression est l'expression numérique de l'énoncé. Tous les arcs ont des actions nulles.

Voici un exemple d'énoncé IF-arithmétique (figure 4.5:

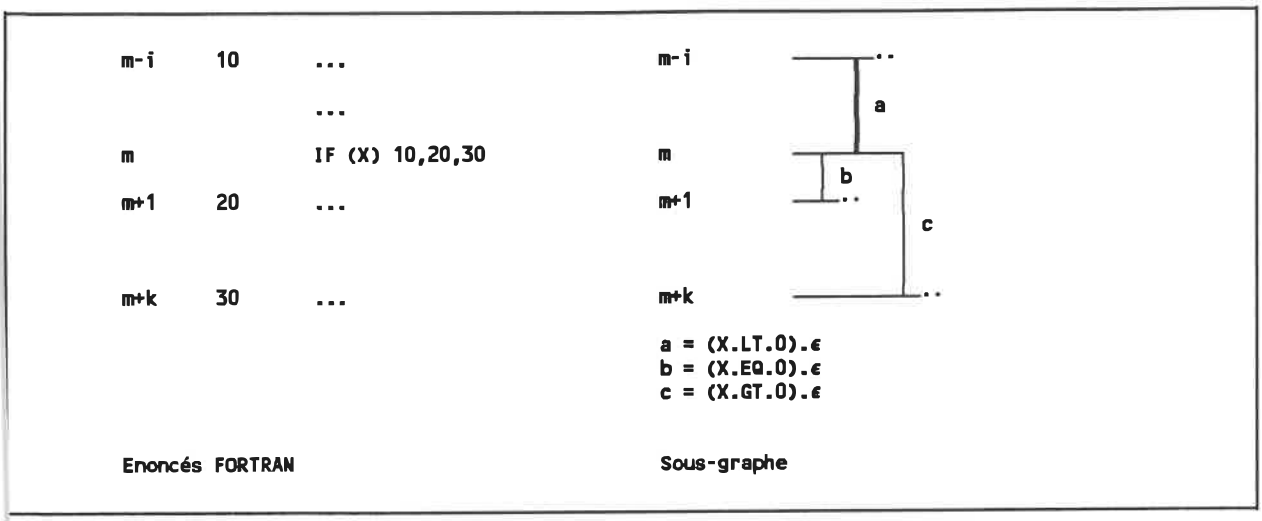


Fig. 4.5 - Sous-graphe d'un énoncé IF arithmétique.

Enoncé IF-énoncé

Cet énoncé produit d'abord un sommet conditionnel identique à celui du IF-GOTO avec la même paire d'arcs sortants. Un de ces arcs va à l'énoncé suivant, l'autre va à un sommet correspondant à l'énoncé présent dans l'IF-énoncé.

Voici un exemple d'IF-énoncé (figure 4.6):

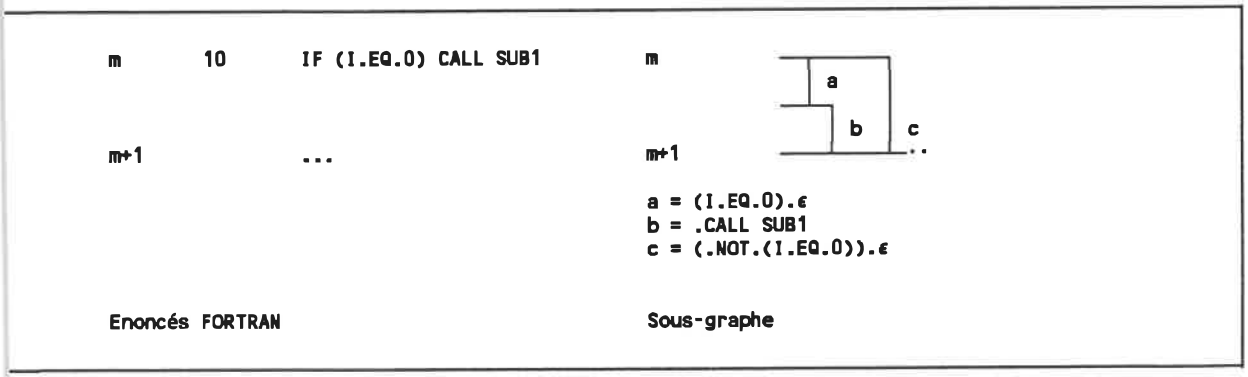


Fig. 4.6 - Sous-graphe d'un énoncé IF-énoncé.

Enoncé IF-THEN

Cet énoncé produit un sommet conditionnel avec deux arcs sortants. Les deux arcs possèdent des actions nulles. L'un est exécuté si expression est vraie, l'autre si .NOT.expression est vraie, expression étant l'expression booléenne du IF. Le premier mène à l'énoncé suivant, tandis que l'autre va au ELSE IF suivant s'il existe, au ELSE s'il existe et qu'il n'y a pas d'ELSE IF, ou au END IF s'il n'y a ni ELSE IF ni ELSE.

Enoncé ELSE IF THEN

L'énoncé ELSE IF THEN produit exactement la même résultat que l'énoncé IF THEN.

Voici un exemple d'énoncés IF THEN et ELSE IF THEN (figure 4.7):

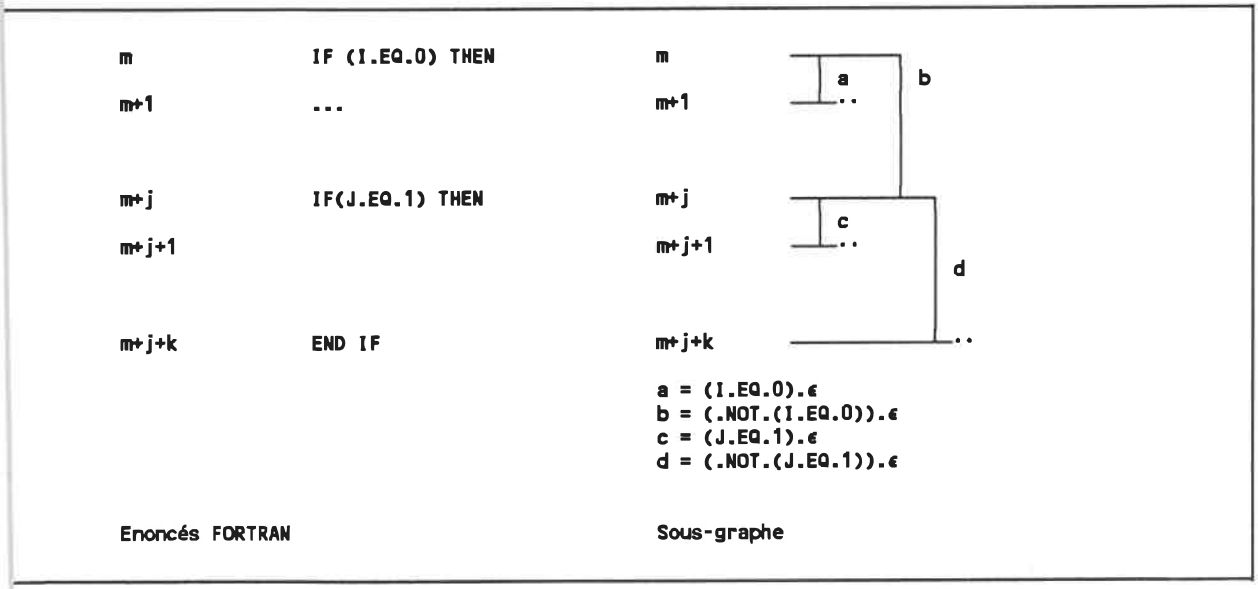


Fig. 4.7 - Sous-graphes des énoncé IF THEN et ELSE IF THEN.

Enoncé DO

L'énoncé DO ne peut être considéré seul; on doit aussi tenir compte de l'énoncé possédant l'étiquette mentionnée dans l'énoncé DO (appelé énoncé étiqueté).

L'énoncé DO lui-même produit un sommet séquentiel dont l'arc sortant contient l'initialisation variable = valeur-initiale. Cet arc se termine sur un sommet conditionnel dont le degré extérieur vaut 2. Un de ces arcs sortants, de condition variable .GT. valeur-finale et ayant une action nulle, branche au sommet correspondant à l'énoncé suivant l'énoncé étiqueté. L'autre arc, de condition variable .LE. valeur finale et ayant une action nulle, branche à l'énoncé suivant le DO.

L'énoncé étiqueté crée un ou plusieurs sommets, tout dépendant de son type. L'arc sortant qui pointe normalement sur l'énoncé suivant cet énoncé étiqueté est alors intercepté et se termine plutôt sur un sommet séquentiel. L'arc sortant contient l'incrément de la forme variable = variable + incrément et se termine sur le sommet conditionnel de l'énoncé DO correspondant (arc ascendant).

Voici un exemple d'énoncé DO (figure 4.8):

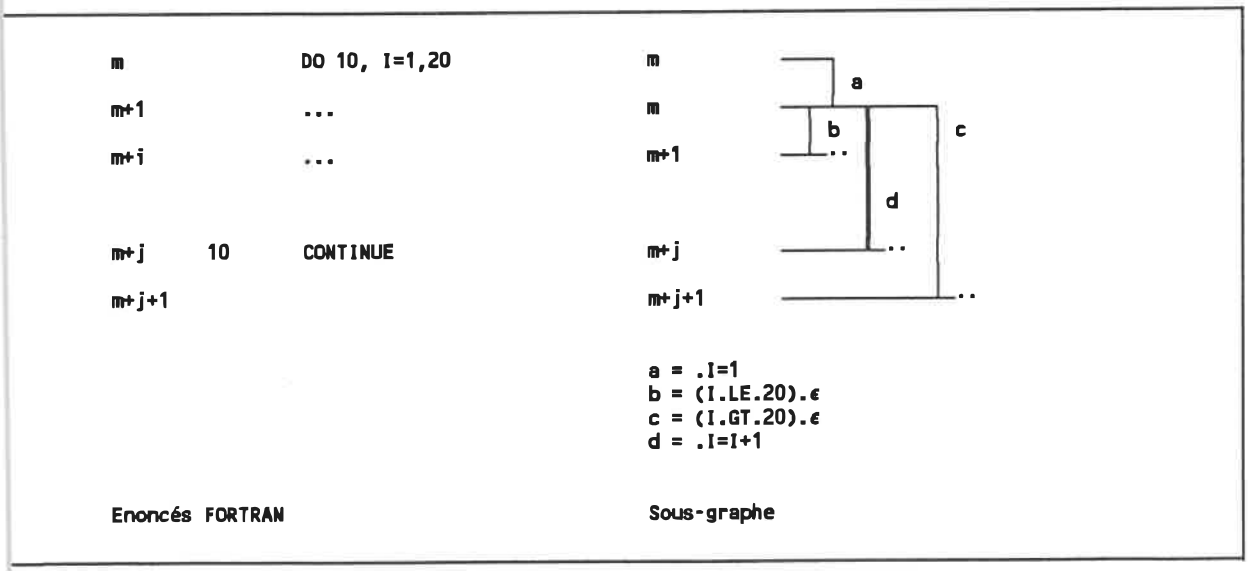


Fig. 4.8 - Sous-graphe d'un énoncé DO.

4.2 Structuration du graphe de contrôle

Comme il est mentionné dans [26], le but de l'agorithme de transformation du graphe de contrôle est de réduire ce graphe à un graphe structuré. Quelques termes doivent cependant être définis afin de bien comprendre ce processus.

On dit qu'un graphe est en forme réduite si:

- il possède exactement un sommet de départ d ;
- il possède exactement un sommet de terminaison t ;
- pour tous les autres sommets, il existe un chemin [d,s] et un chemin [s,t], c'est-à-dire qu'il n'y a aucun code mort ni aucune boucle infinie.

On dit qu'un graphe est en forme standard si:

- il ne possède aucun sommet s ayant d-(s) = 1 et d+(s) = 0 (i.e. aucun sommet procédural);
- pour toute paire de sommets {u,v}, il existe au plus un arc (u,v) (i.e. pas d'arcs parallèles)
- il n'existe aucun arc (u,u), c'est-à-dire aucun arc réflexif;
- il n'existe aucun chemin [u,v] formé d'une séquence d'au moins deux arcs (u0,u1) (u1,u2)...(un,u0) dont les sommets intermédiaires u1,u2...un ont des arcs sortants se terminant tous vers le même sommet.

NOTE Cette dernière condition est une généralisation de celle mentionnée dans [26]. Elle est basée sur les structures répétitives du pseudocode schématique.

On dit qu'un graphe en forme standard est structuré s'il ne comprend qu'un seul arc allant du sommet de départ au sommet de terminaison.

4.2.1 Description générale de l'algorithme

La méthode de restructuration utilisée dans notre travail s'applique sur des graphes en forme réduite. Cette condition n'est pas très restrictive en pratique puisque:

- une procédure FORTRAN ne comprend qu'un sommet de départ (les énoncés ENTRY sont considérés séquentiels);
- par définition, une procédure ne comprend qu'un seul sommet de sortie;
- le code mort peut être éliminé sans problème;
- on peut insérer une condition de sortie qui n'est jamais vérifiée dans une boucle infinie.

L'algorithme général est le suivant (figure 4.9):

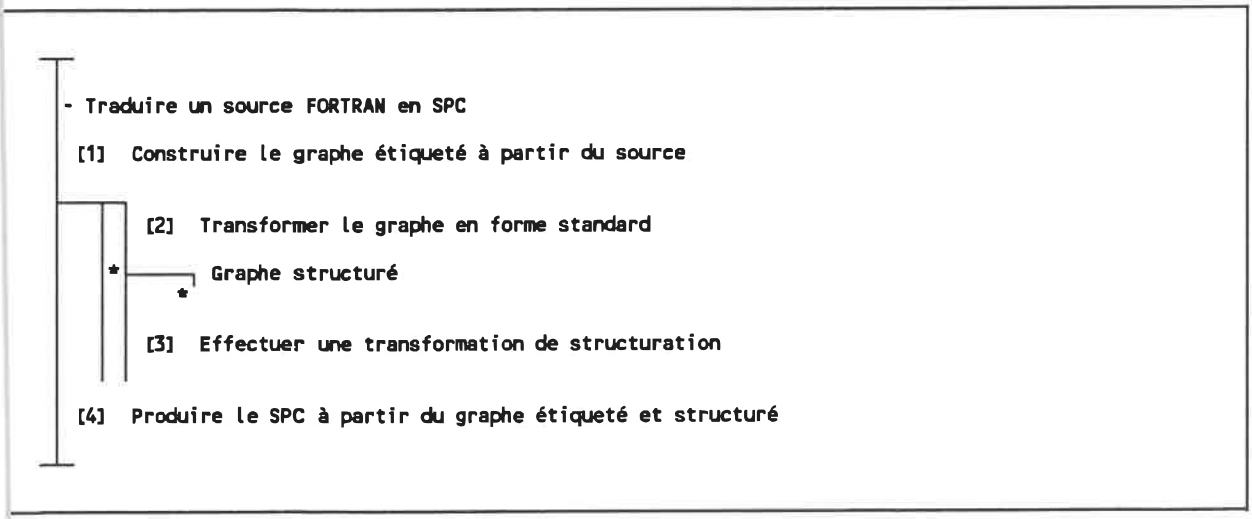


Fig. 4.9 - Algorithme 4.1: traduction du FORTRAN en SPC.

Examinons maintenant en détail les étapes 2 et 3.

4.2.2 Transformation du graphe en forme standard

La transformation du graphe en forme standard consiste à éliminer les sommets procéduraux, les arcs parallèles, les arcs réflexifs et les boucles généralisées en les remplaçant par des sous-graphes plus simples (un seul arc). Ces cas correspondent à des structures de base séquentielle, conditionnelle et répétitive. Nous les examinons séparément.

1. Élimination des sommets procéduraux

Les sommets procéduraux représentent des actions séquentielles. Représentée graphiquement, la transforamtion est la suivante (figure 4.10):

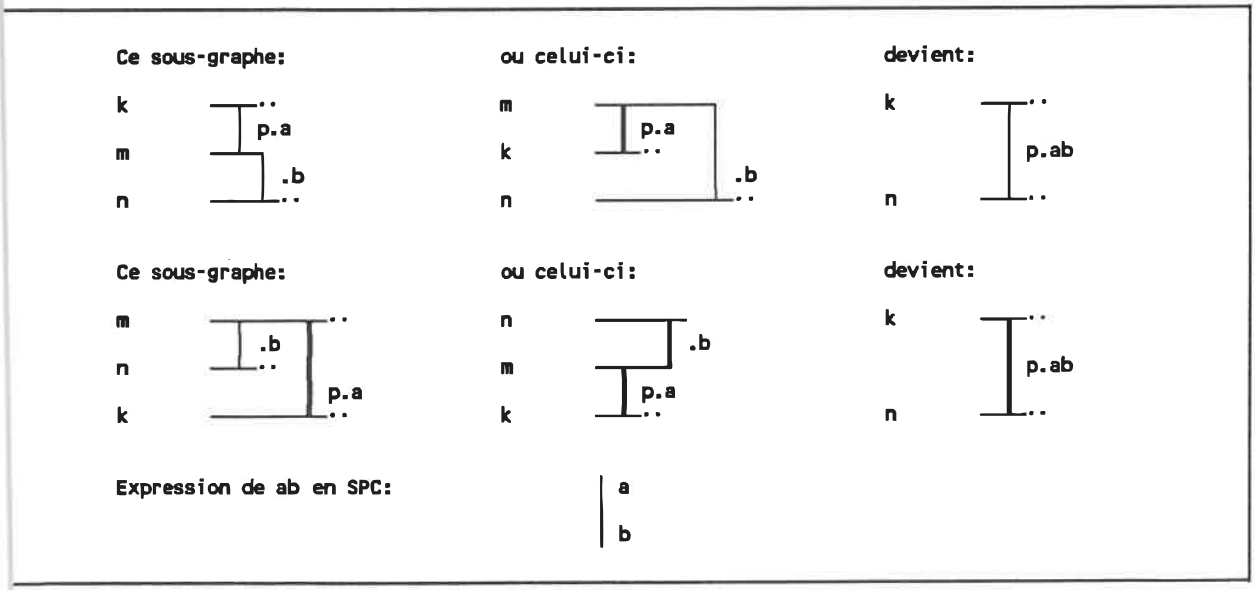


Fig. 4.10 - Élimination des sommets procéduraux.

Il s'agit simplement d'éliminer tout sommet procédural en concaténant en un seul arc les actions effectuées dans les deux arcs adjacents. La condition d'exécution est la même que celle de l'arc entrant.

2. Élimination des arcs parallèles

Les arcs parallèles représentent des structures conditionnelles. On les remplace par un seul arc dont l'action contient une structure conditionnelle. Graphiquement, cela est représenté à la figure 4.11:

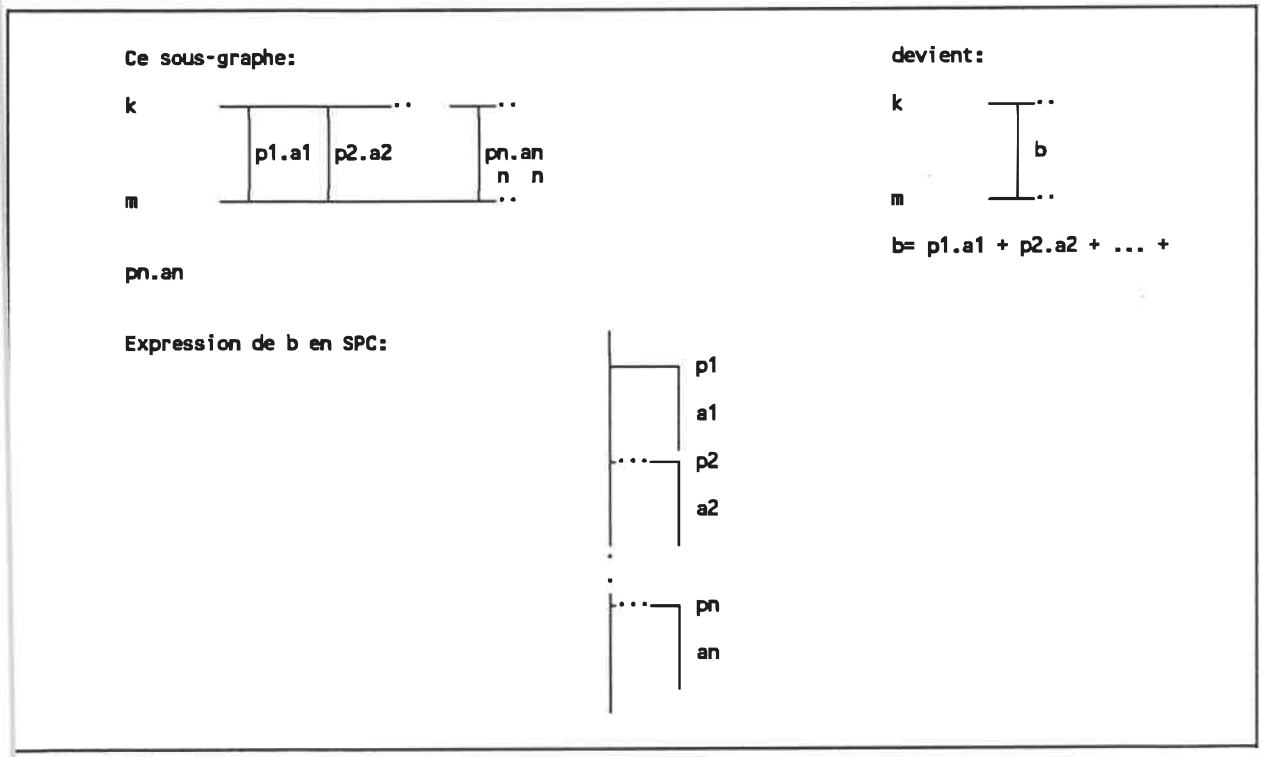


Fig. 4.11 - Élimination d'arcs parallèles.

NOTE La condition d'exécution q est vraie si le sommet k devient séquentiel après la transformation. S'il ne le devient pas, q vaut alors ~(q1 + q2 + ...qn), les q1, q2...qn étant les conditions d'exécution des autres arcs sortants de k (~ représente la négation logique).

3. Élimination des arcs réflexifs

Ces arcs représentent la forme la plus simple de boucle: le WHILE DO. Il apparaît lorsqu'un des arcs sortants d'un sommet conditionnel se termine sur ce même sommet. La simplification est présentée graphiquement à la figure 4.12:

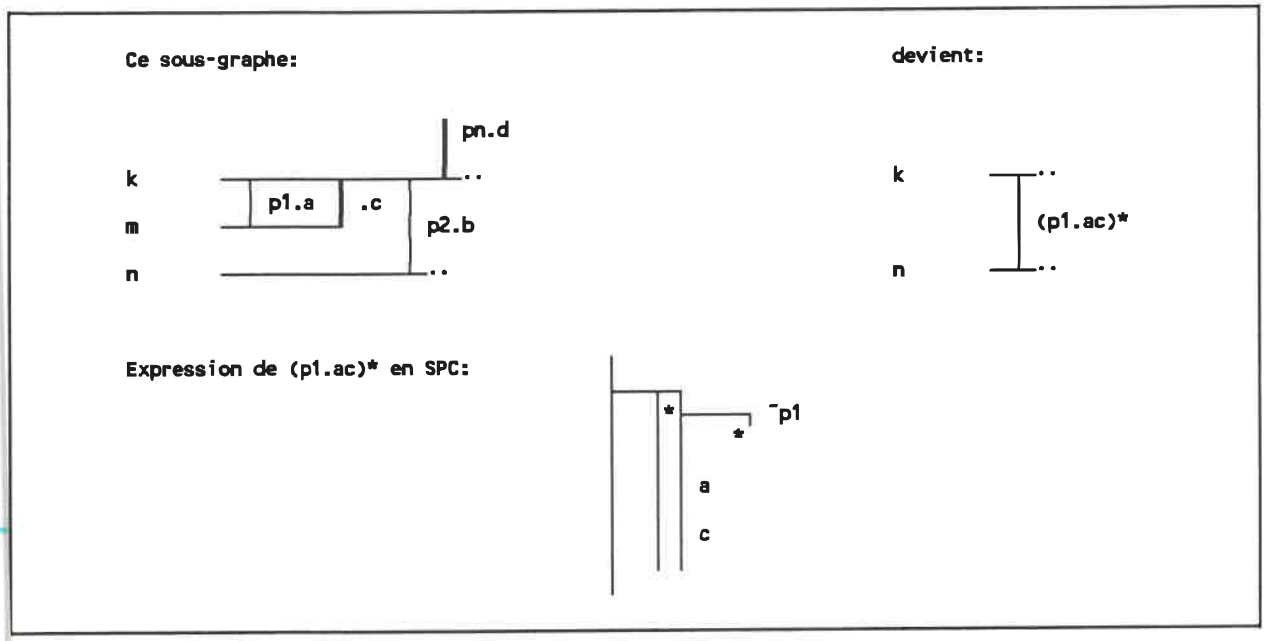


Fig. 4.12 - Élimination d'arc réflexif.

NOTE le symbole * dans l'expression (p1.a)* représente la fermeture-étoile de Kleene et signifie exécuter 0,1 ou plusieurs fois.

4. Élimination des boucles généralisées

En plus des boucles WHILE, le pseudocode schématique supporte une forme très générale de structure répétitive. Il s'agit de reconnnaître le sous-graphe comprenant la structure répétitive entière. Graphiquement, la transformation est présentée à la figure 4.13:

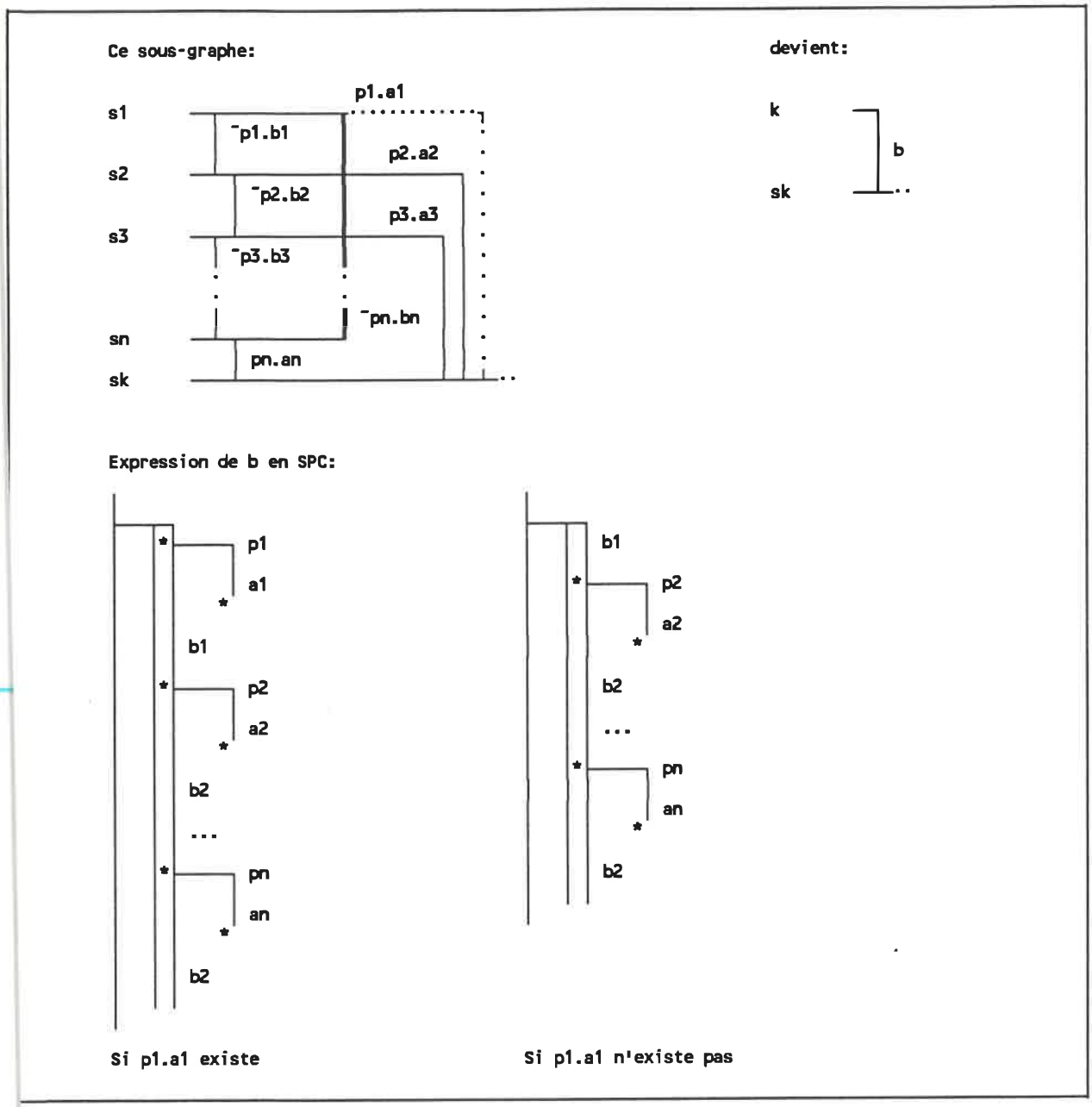


Fig. 4.13 - Élimination des boucles généralisées.

Il existe cependant deux conditions d'application à l'élimination des boucles généralisées:

- Pour tout si, i=2,n , on a d+(si) = 2

- 1 d+(s1) 2 (i.e. l'arc p1.a1 peut exister ou non).

Ces quatre transformations sont appliquées itérativement au graphe. Pour diminuer le temps de calcul, on effectue d'abord les deux premières jusqu'à ce qu'aucune modification ne soit faite, on applique les deux autres transformations et on recommence. L'algorithme est illustré à la figure 4.14.

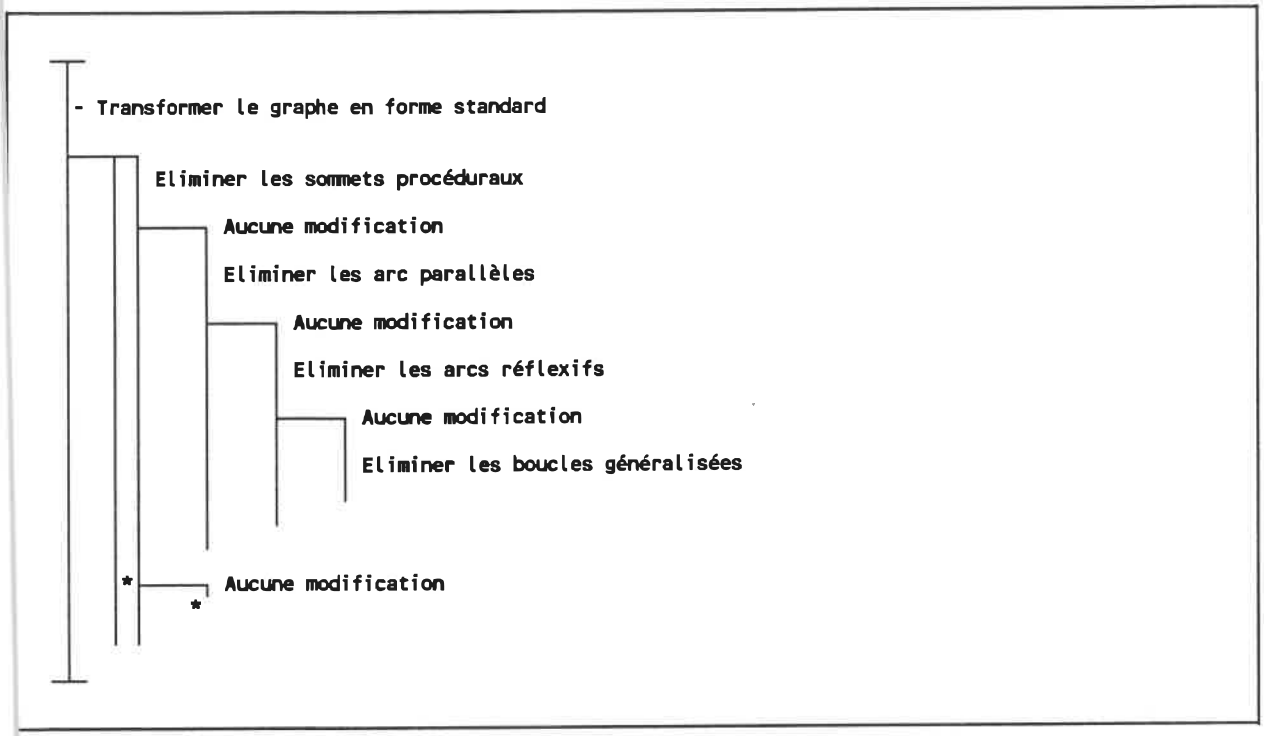


Fig. 4.14 - Algorithme 4.2: transformation du graphe en forme standard.

A la suite de l'application de cet algorithme, on peut obtenir ou non un graphe structuré. Si c'est le cas, le processus est terminé et on peut alors produire le SPC, comme le montre l'algorithme 4.1 (figure 4.9). Sinon, il faut effectuer des opérations de structuration, qui sont décrites à la section suivante.

4.2.3 Structuration du graphe par transformation

Les opérations de structuration ont pour but de transformer un graphe de forme standard non-structuré en un graphe équivalent mais structuré. Ces opérations sont décrites dans [26] et [8]; les formes non-structurées ont été examinées également par [25] et [48].

Essentiellement, il existe six formes non-structurées de base qui peuvent apparaître dans un graphe de contrôle:

- saut dans une décision (ID: in decision);
- saut hors d'une décision (OD: out of decision);
- saut dans le chemin descendant d'une boucle (IL: in loop);
- saut hors du chemin descendant d'une boucle (OL: out of loop);
- saut dans le chemin ascendant d'une boucle (IB: in backward path);
- saut hors du chemin ascendant d'une boucle (OB: out of backward path);

Ces formes non-structurées de base sont illustrées à la figure 4.15:

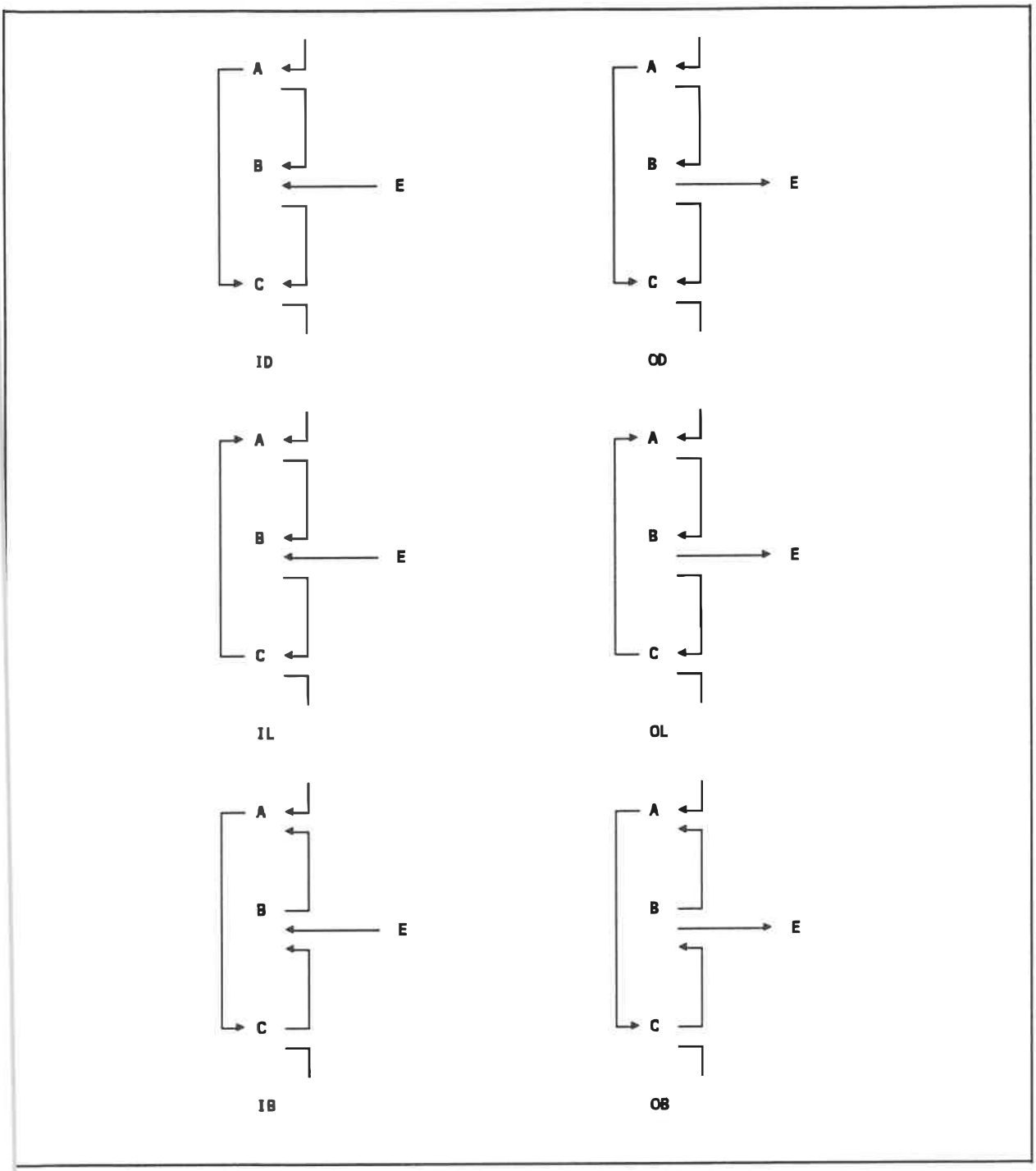


Fig. 4.15 - Les six formes non-structurées de base.

En se référant à la figure précédente, il existe trois positions possibles pour le sommet E:

- dans le chemin allant du sommet de départ au sommet A:
- dans un chemin allant du sommet C au sommet de terminaison;
- dans un chemin allant de D à T qui ne contient ni A, ni B, ni C.

En analysant les trois positions possibles du sommet E pour chacune des six formes non-structurées de base, on obtient les dix-huit formes illustrées aux figures 4.16 à 4.21.

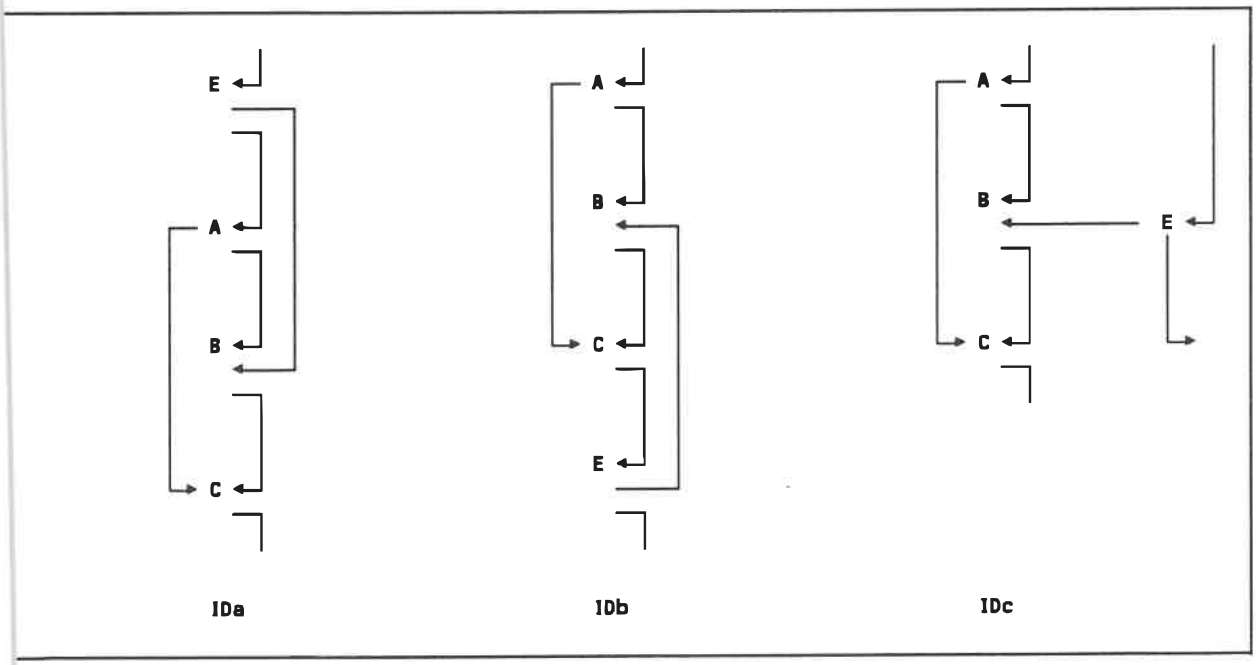


Fig. 4.16 - Les trois formes de non-structure ID.

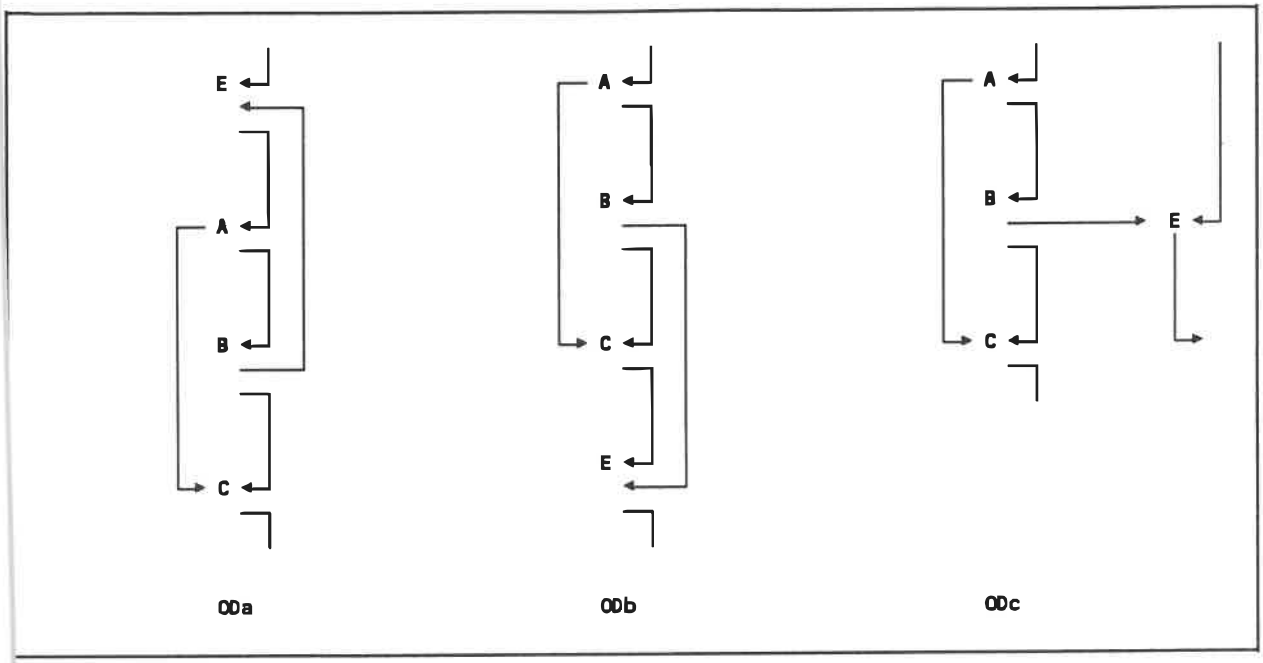


Fig. 4.17 - Les trois formes de non-structure OD.

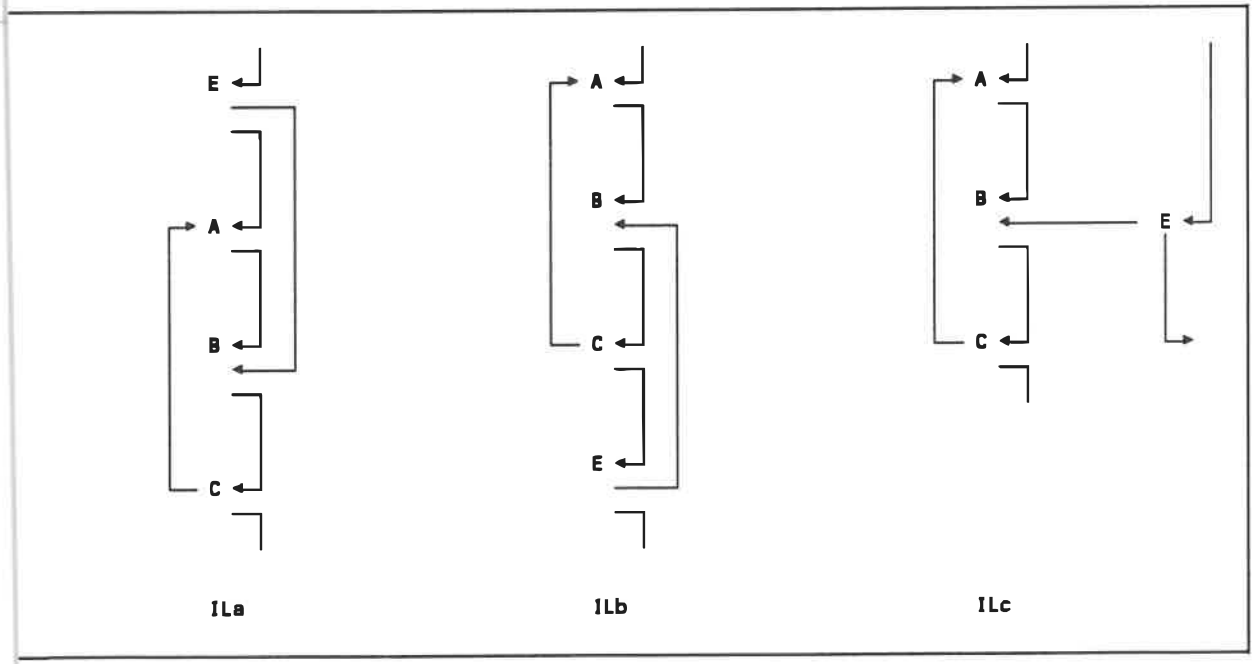


Fig. 4.18 - Les trois formes de non-structure IL.

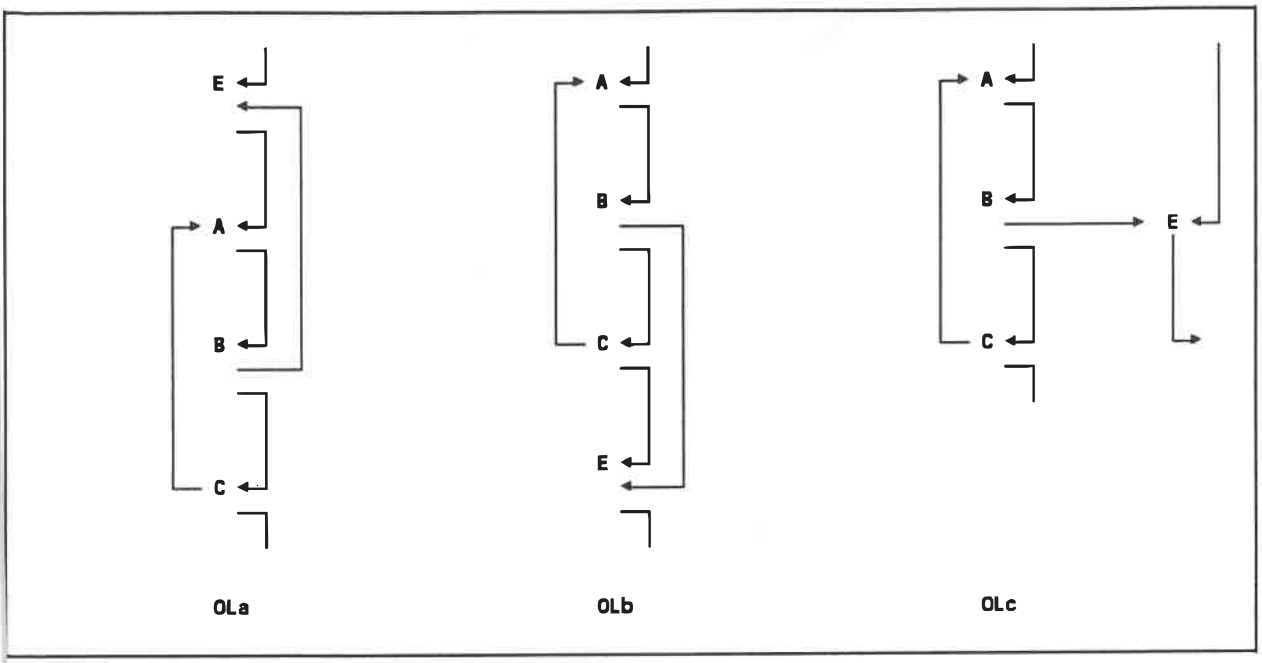


Fig. 4.19 - Les trois formes de non-structure OL.

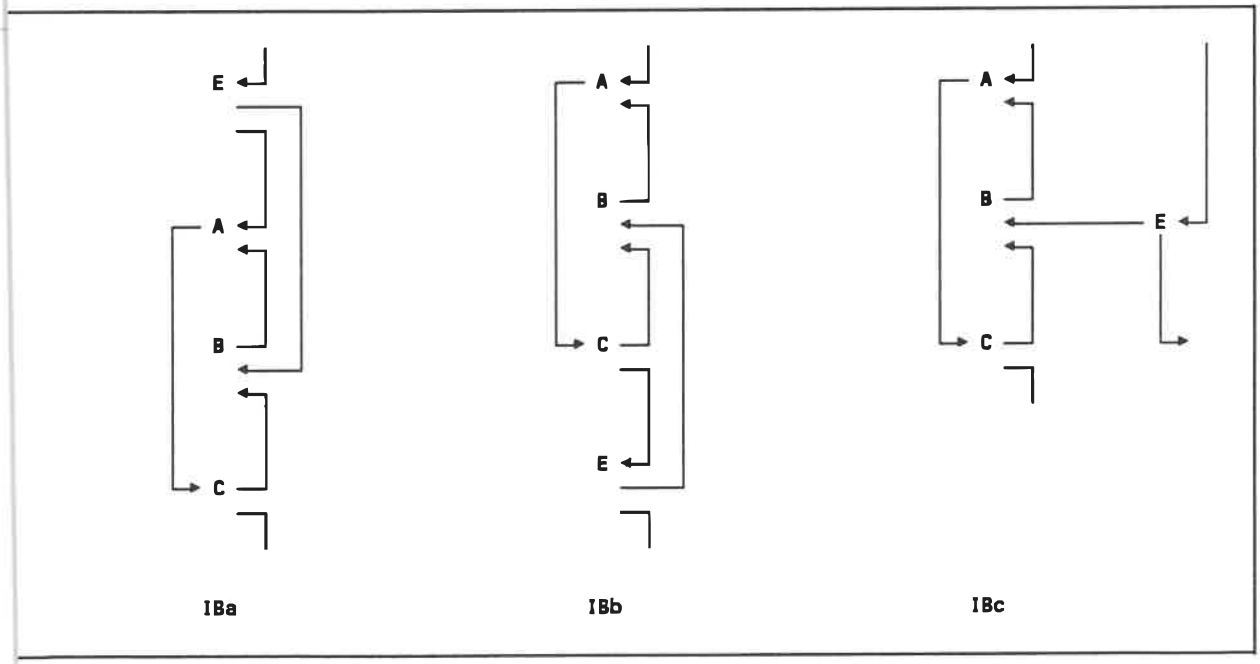


Fig. 4.20 - Les trois formes de non-structure IB.

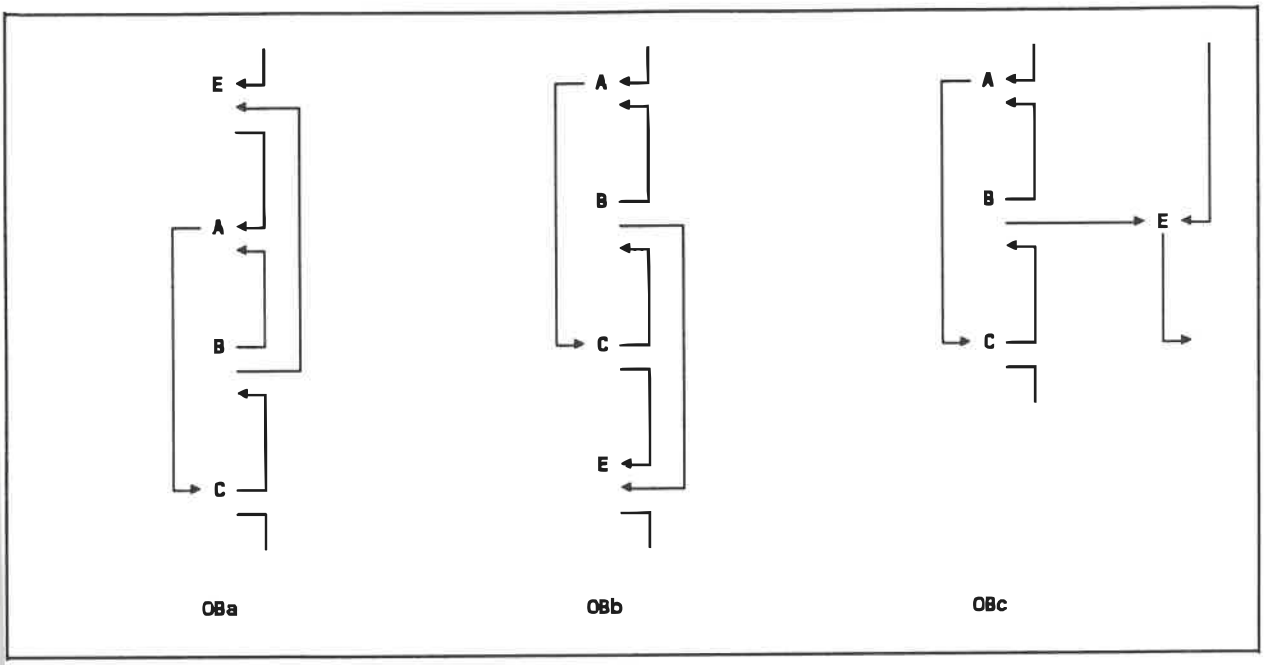


Fig. 4.21 - Les trois formes de non-structure OB.

On constate que certaines formes sont équivalentes: IDa et ODb, IDb et ILa, ILb et OLa, ODa et OLb. De plus, en inversant la position des sommets intérieurs de IBa et OBb, on découvre deux équivalences supplémentaires: IDb et IBa, ODa et OBb.

Notons que la forme ODa est structurée au sens du pseudocode schématique puisqu'il s'agit d'une boucle généralisée. En conséquence, elle ne peut pas être présente dans un graphe en forme standard et n'est donc pas considérée.

Il reste finalement les cinq formes de non-structure illustrées à la figure 4.22, présentées selon la notation utilisée par [27]. Notons que les formes IDc, ODc, ILc, OLc, IBc et OBc peuvent s'exprimer comme des combinaisons de ces cinq formes.

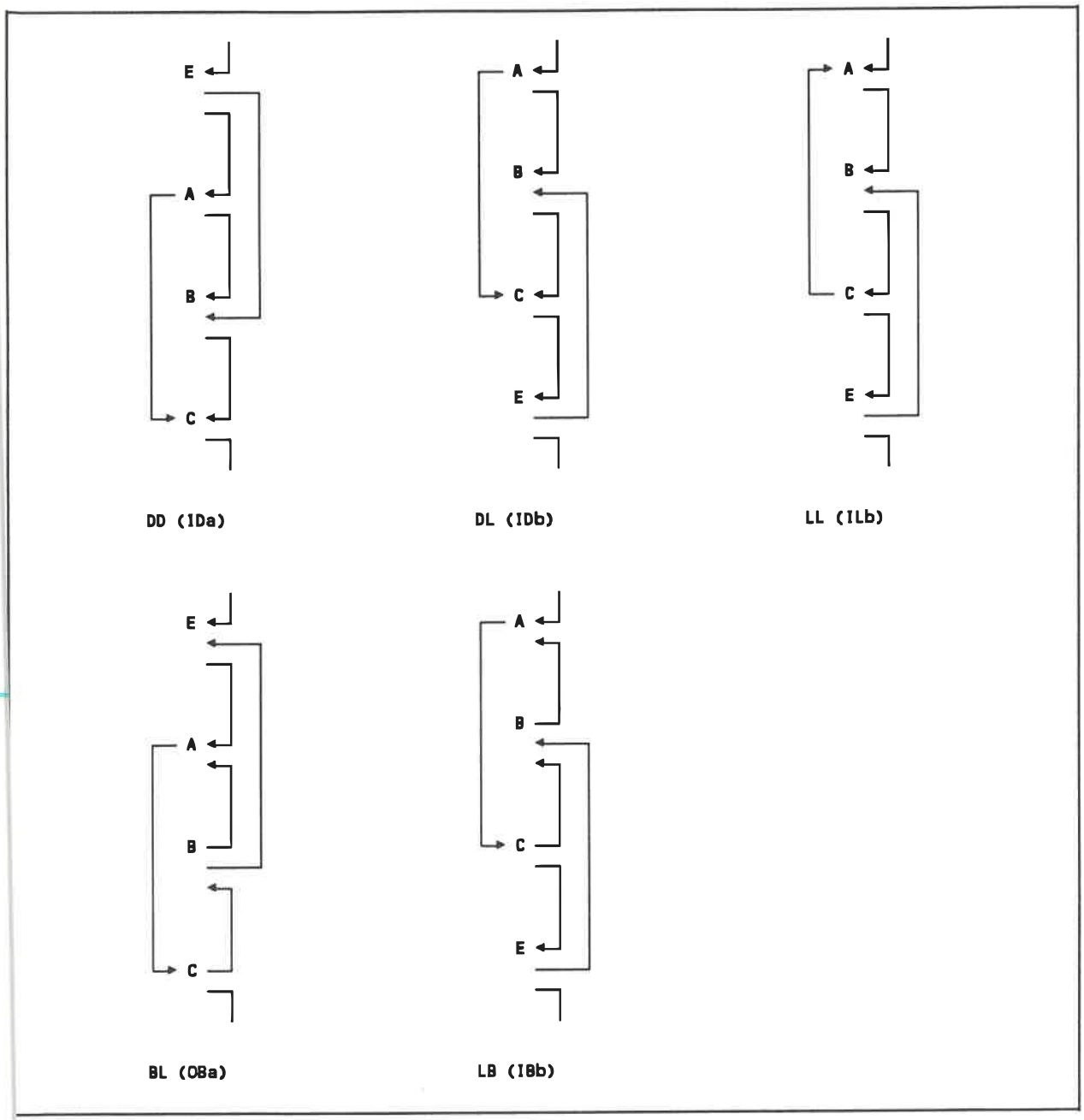


Fig. 4.22 - Les cinq formes de base de non-structure.

L'examen de ces cinq formes de non-structures montre qu'elles sont construires en combinant les formes non-structurées de base suivantes: ID, OD, IL et OL. Voici ces combinaisons:

- DD vient de ID et OD;
- DL vient de ID et IL;
- IL vient de II et OL;
- BL vient de ID et OL;
- LB vient de OD et IL.

Conséquemment, il suffit de déterminer les transformations permettant d'éliminer des formes ID, OD, IL et OL pour structurer un graphe. Ces transformations sont décrites dans [26], [27] et [49].

Elimination de la forme ID

Deux transformations sont possibles pour éliminer la forme ID. La première, ID-0, implique la duplication d'un arc et conséquemment des actions qui y sont effectuées, tandis que la seconde, ID-1, implique la duplication d'un test (prédicat) et la création d'une variable de contrôle (flag). Le suffixe 0 ou 1 qui suit le nom d'une transformation indique le nombre de variables créées par cette transformation. Les deux transformations sont illustrées à la figure 4.23:

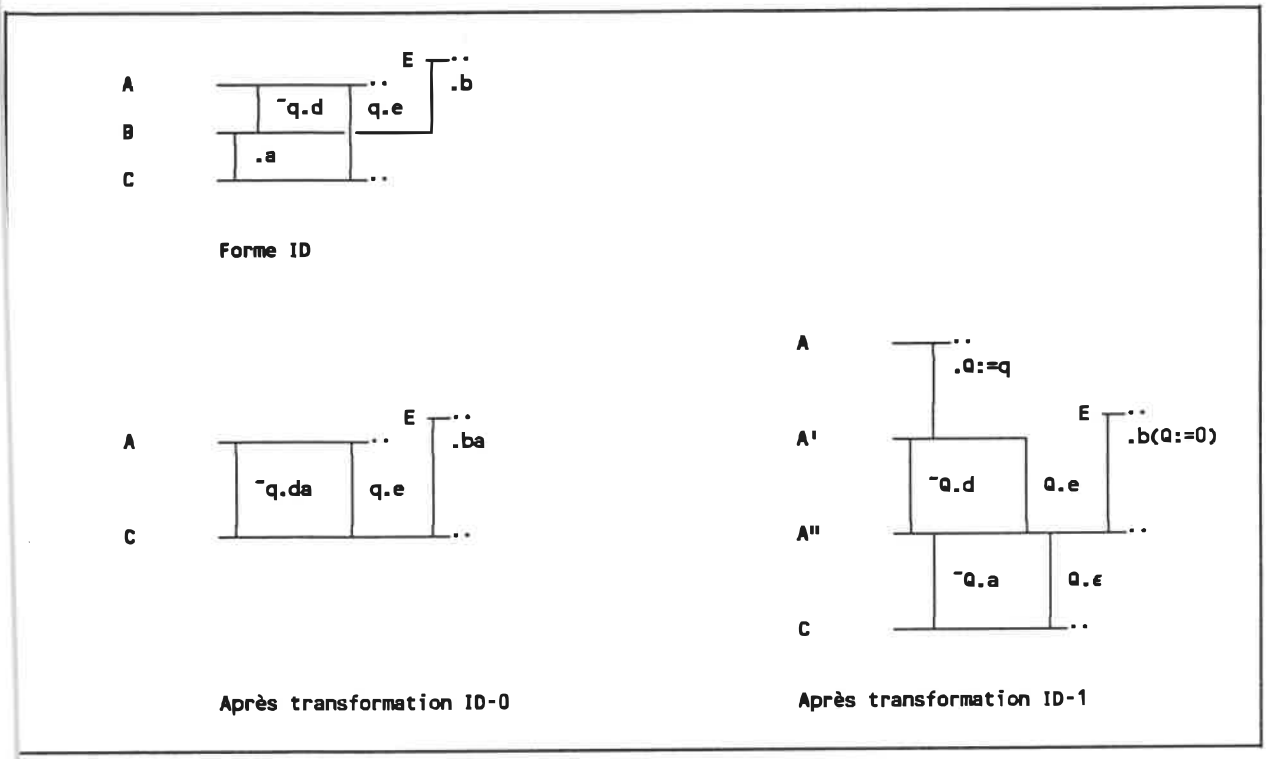


Fig. 4.23 - Elimination de ID par les transformations ID-0 ou ID-1.

Notons l'introduction d'un arc entrant au sommet A qui préserve la valeur de l'expression q dans la variable Q. Une variable différente devra être utilisée pour chaque transformation afin d'éviter de perdre leur valeur.

L'expression Q := 0 représente l'initialisation de la variable Q à FAUX

Le coût de ID-0 est la duplication de l'arc a, tandis que celui de ID-1 est l'introduction d'une variable de contrôle.

Elimination de la forme IL

Comme pour la forme ID, il existe deux transformations possibles pour la forme IL: IL-0, qui utilise la duplication d'arc, et IL-1, qui utilise une variable de contrôle. Les deux transformations sont illustrées à la figure 4.24:

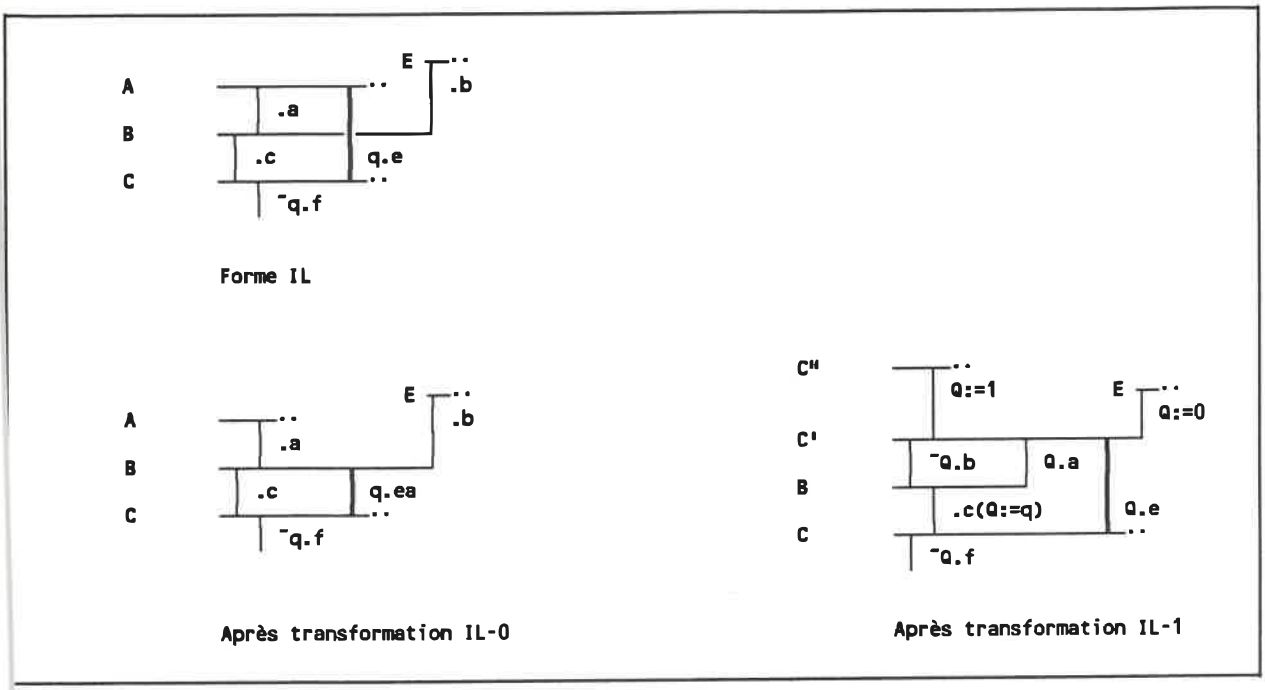


Fig. 4.24 - Elimination de IL par les transformations IL-0 ou IL-1.

L'expression Q := 0 représente l'initialisation de la variable Q à FAUX

L'expression Q := 1 représente l'initialisation de la variable à VRAI.

Elimination de la forme OD

La forme OD s'élimine par la transformation OD-1 qui implique l'utilisation d'une variable de contrôle. La transformation est montrée à la figure 4.25:

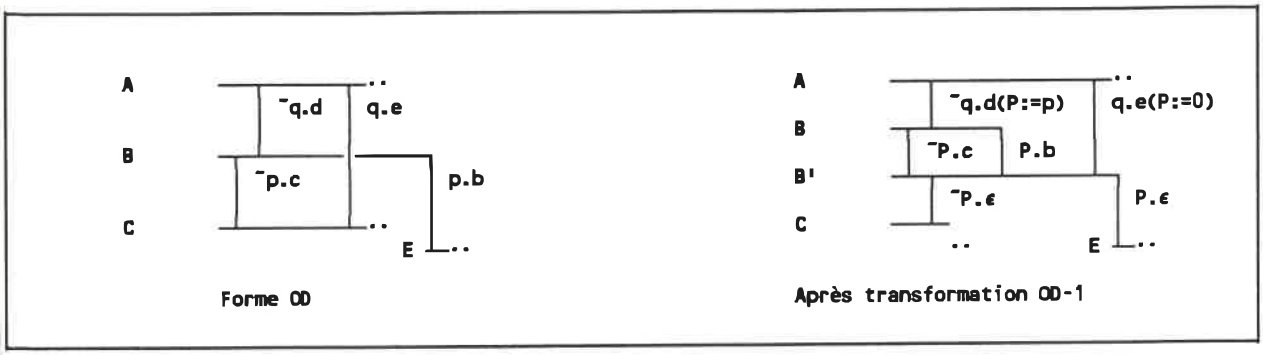


Fig. 4.25 - Elimination de OD par la transformation OD-1.

Elimination de la forme OL

La forme OL est éliminée par la transforamtion OL-1 qui, elle aussi, implique l'utilisation d'une variable de contrôle. La figure 4.26 illustre cette transformation.

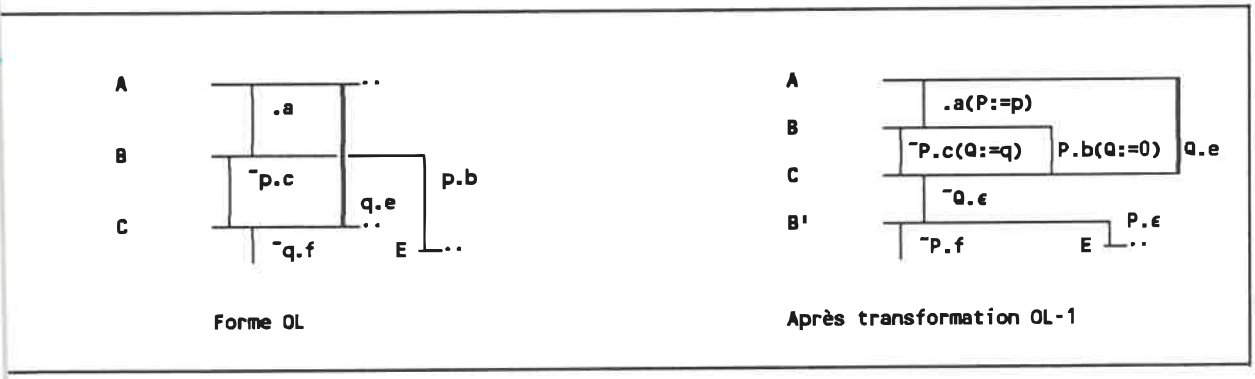


Fig. 4.26 - Elimination de OL par la transformation OL-1.

Ensemble minimal de transformations

Commme le prouve Oulsnam dans [26], il est toujours possible de convertir un graphe de contrôle non-structuré en un graphe structuré en utilisant seulement les transformations ID-0, ID-1, OD-1, IL-0 et IL-1. On pourrait même se limiter à ID-1, OD-1 et Il-1 si la production de variables de contrôle ne représentait pas

d'inconvénients. Cependant, le compromis lisibilité/dimension du programme résultant favorise souvent ID-0 et IL-0 et c'est pour cela que nous les conservons ici.

Ordre d'application des transformations

L'ordre d'application des transformations suit l'algorithme 4.3 (figure 4.27), qui est un raffinement de l'algorithme 4.1 (figure 4.9).

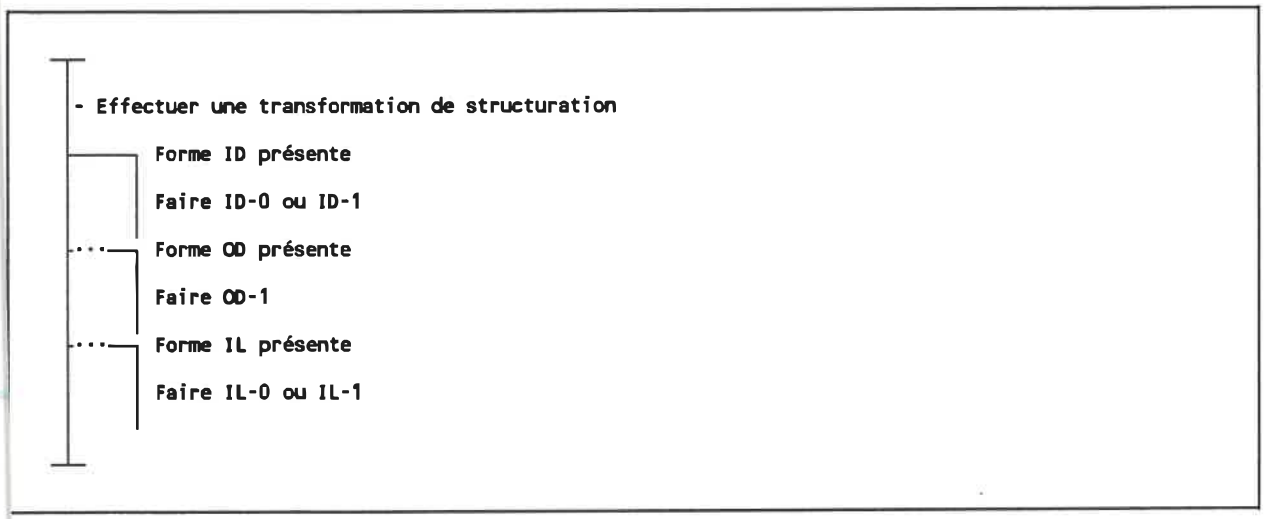


Fig. 4.27 - Algorithme 4.3: transformations de restructuration.

4.3 Traduction du graphe de contrôle en pseudocode schématique

Une fois que le graphe est en forme structurée, il ne comprend qu'un seul arc dont l'action est la procédure entière. Chaque action est représentée par l'énoncé FORTRAN correspondant et chaque condition est représentée par une expression booléenne qui, elle aussi, respecte la syntaxe du FORTRAN. Ce que nous n'avons pas encore défini est la représentation des structures conditionnelles et répétitives sous forme textuelle.

Afin d'éviter l'invention d'une nouvelle notation, nous avons préféré utiliser la notation RTF [6] pour représenter les structures du pseudocode schématique. Nous allons donc d'abord examiner cette notation et voir comment on l'intègre dans le graphe de contrôle.

4.3.1 Notation RTF du pseudocode schématique

La notation RTF permet de représenter textuellement (i.e. sans schéma) un algorithme en pseudocode schématique. Le sous-ensemble de cette notation que nous utilisons dans notre travail est décrit par la grammaire BNF suivante (figure 4.28):

```
Algorithme RTF          ::=    structure *

structure               ::=    séquentielle
                             | conditionnelle
                             | répétitive

séquentielle            ::=    A énoncé
                             | T commentaire

énoncé                  ::=  ENONCE-SEQUENTIEL-FORTRAN  <NL>

commentaire             ::=  TEXTE-LIBRE  <NL>

conditionnelle          ::=  si
                             sinon-si *
                             sinon ?
                             F  <NL>

si                      ::=  C condition
                             structure +

sinon-si                ::=  L condition
                             structure +

sinon                   ::=  O  <NL>
                             structure +

condition               ::=  EXPRESSION-BOOLEENNE-FORTRAN  <NL>

répétitive              ::=  R  <NL>
                                      structure +   sortie *
                                    | sortie +   structure *
                             structure | sortie
                             F  <NL>

sortie                  ::=  E condition
                             structure *
                             F  <NL>
```

Fig. 4.28 - Grammaire du sous-ensemble du RTF utilisé.

NOTES Les variables syntaxiques sont en minuscules.

Les symboles du vocabulaire (terminal symbols) sont en majuscules. Le symbole <NL> représente un changement de ligne (new line).

Les symboles *, +, et ? sont des facteurs de répétition.

* signifie: présent 1 ou plusieurs fois ou absent

+ signifie: présent 1 ou plusieurs fois

? signifie: présent 1 fois ou absent.

L'exemple de la figure 4.29 illustre un algorithme en SPC et la figure 4.30, son homologue RTF:

```
-Auteur: Bob McSchem
-Programme evaluant les racines d'un polynome du second degre.
PROGRAM POLY
REAL a,b,c,discri,racin1,racin2,r1,i1
CHARACTER repons*1
WRITE(5,*) 'CALCUL DES POLYNOMES DE DEGRE 2.Forme:ax2 +bx +c =0'

      WRITE(5,*) 'Entrez les valeurs de ''a'',''b'' et ''c'' : '
      READ(5,*) a,b,c

      discri = (b*b) - (4*a*c)

          a .EQ. 0

          x=(-c/b)
          WRITE(5,*) 'Racine unique, cas lineaire :',x
      ...discri .LT. 0

          r1 = (-b) / (2*a)
          i1 =  sqrt(-discri) / (2*a)
          WRITE(5,*) 'Deux racines complexes:(',r1,'+j',i1,')et(',r1,'-j',i1,')'
      ...discri .EQ. 0

          x=(-b)/2/a
          WRITE(5,*) 'Une seule racine reelle :',x
      .....

          racin1 = ( (-b) + sqrt(discri) ) / (2*a)
          racin2 = ( (-b) - sqrt(discri) ) / (2*a)
          WRITE(5,*) 'Deux racines reelles : ',racin1,' et ',racin2


      WRITE(5,*) 'Un autre ? (o/n) :'
      READ(5,1) repons
      1  FORMAT(A1)
    *     repons .EQ. 'N'
        *

END
```

Fig. 4.29 - Programme en SPC.

```
T Auteur: Bob McSchem
T Programme evaluant les racines d'un polynome du second degre.
A PROGRAM POLY
A REAL a,b,c,discri,racin1,racin2,r1,i1
A CHARACTER repons*1
A WRITE(5,*) 'CALCUL DES POLYNOMES DE DEGRE 2.Forme:ax2 +bx +c =0'
R
A WRITE(5,*) 'Entrez les valeurs de ''a'',''b'' et ''c'' : '
A READ(5,*) a,b,c
A discri = (b*b) - (4*a*c)
C a .EQ. 0
A x=(-c/b)
A WRITE(5,*) 'Racine unique, cas lineaire :',x
I discri .LT. 0
A r1 = (-b) / (2*a)
A i1 =  sqrt(-discri) / (2*a)
A WRITE(5,*) 'Deux racines complexes:(',r1,'+j',i1,')et(',r1,'-j',i1,')'
I discri .EQ. 0
A x=(-b)/2/a
A WRITE(5,*) 'Une seule racine reelle :',x
O
A racin1 = ( (-b) + sqrt(discri) ) / (2*a)
A racin2 = ( (-b) - sqrt(discri) ) / (2*a)
A WRITE(5,*) 'Deux racines reelles : ',racin1,' et ',racin2
F
A WRITE(5,*) 'Un autre ? (o/n) :'
A READ(5,1) repons
A 1         FORMAT(A1)
E repons .EQ. 'N'
F
F
A END
```

Fig. 4.30 - Programme équivalent en RTF.

4.3.2 Expressions des actions lors de la mise en forme standard

Lorsque le graphe est créé à partir du programme source, les arcs ne contiennent que des structures séquentielles. C'est lors de la transformation du graphe en forme standard que d'autres sructures apparaissent (voir la section 4.2.2).

- L'élimination d'un sommet procédural ne fait que concaténer des structures préalablement existantes.

- L'élimination d'arcs parallèles produit une structure conditionnelle formée d'un si et d'au moins un sinon-si (les branches sinon ne sont jamais créées à ce stade).
- L'élimination d'un arc réflexif produit une structure répétitive débutant par une sortie et dont la corps est formé des structures préalablement existantes. La condition de sortie est unique dans ce type de répétitive.
- L'élimination d'une boucle généralisée produit une structure répétitive à une ou plusieurs conditions de sortie. Chaque sortie peut contenir différentes structures préalablement existantes.

On voit donc que le processus suit une approche ascendante (bottom-up): les structures (sous-graphes) les plus internes sont d'abord reconnues puis abstraites en arcs simples contenant des structures représentées en RTF. Lorsque le graphe est structuré, l'arc restant contient la totalité de la procédure et il ne reste qu'à transférer le contenu de cet arc dans un fichier pour que SCHEMACODE puisse le traiter.

4.3.3 Amélioration du code RTF produit

Le processus de production de code lors de la mise en forme strandard est, comme, nous venons de le voir, relativement direct. L'avantage est, bien sûr, la simplicité de l'algorithme de production. Cependant, la qualité du code produit est loin d'être optimale, surtout si on considère que ce code est lu par un programmeur. Nous en déduisons, après examen de plusieurs cas réels, qu'une étape d'amélioration[1] du code RTF doit être incluse.

---

[1] Nous préférons utiliser le terme «amélioration» (ou enhancement) plutôt qu'«optimisation» car il est très difficile dans notre cas de définir, et encore plus d'atteindre, ce qui est optimal.

Les figures 4.31 à 4.40 illustrent les types de transformations qu'on peut effectuer pour améliorer le pseudocode schématique produit. Chaque transformation est montrée en représentation schématique pour favoriser la lisibilité.

NOTES Le symbole s, indicé au non, représente une ou plusieurs structures synthétisées (abstraites).

représente l'absence de structure.

Le symbole p, indicé au non, représente une expression boolénne.

~ est l'opérateur de négation booléenne; ^ est l'opérateur ET logique.

Les facteurs de répétition * , + , et ? ont la même signification que précédemment (voir section précédente).

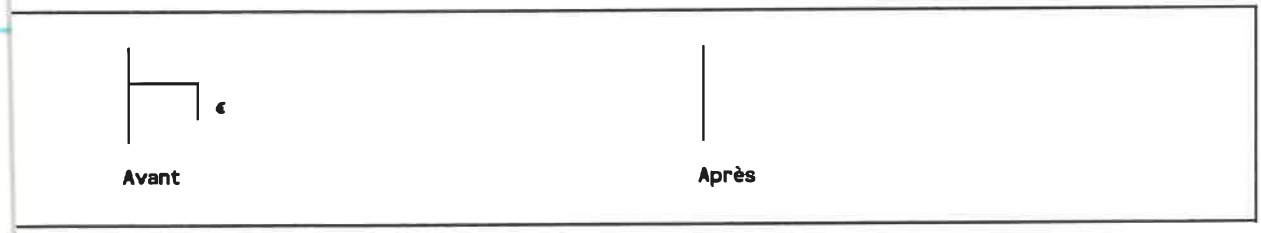


Fig. 4.31 - Elimination de SI vide.

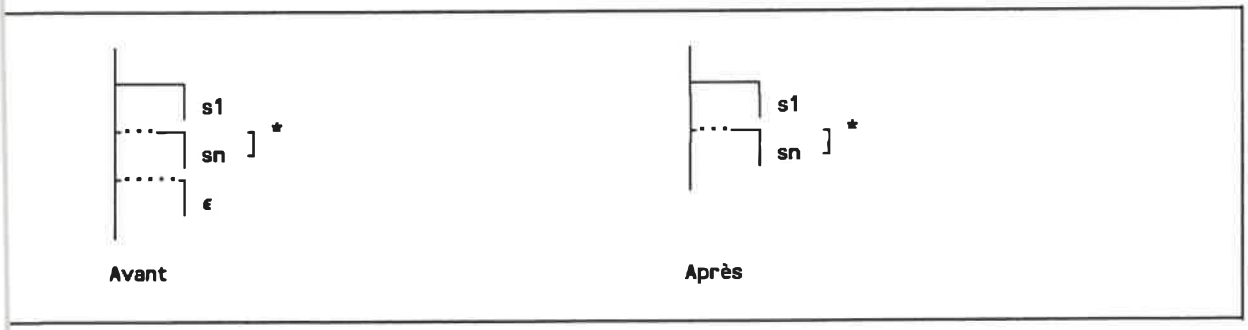


Fig. 4.32 - Elimination de SINON vide.

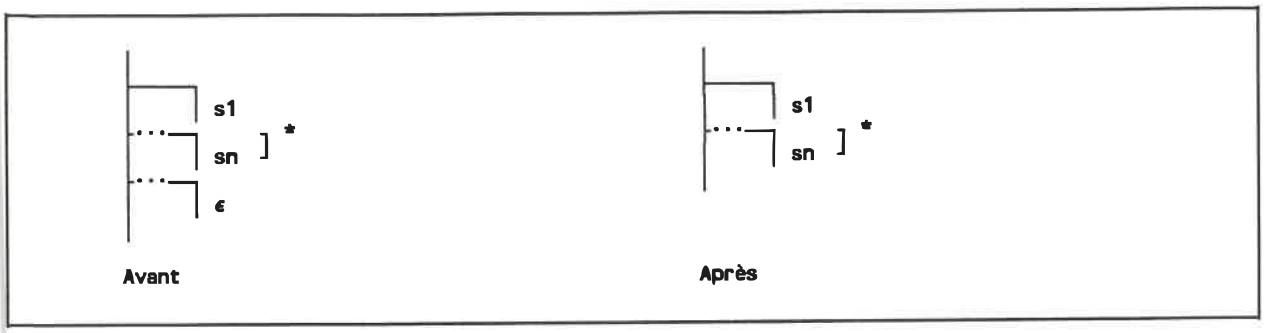


Fig. 4.33 -Elimination de SINON-SI final vide

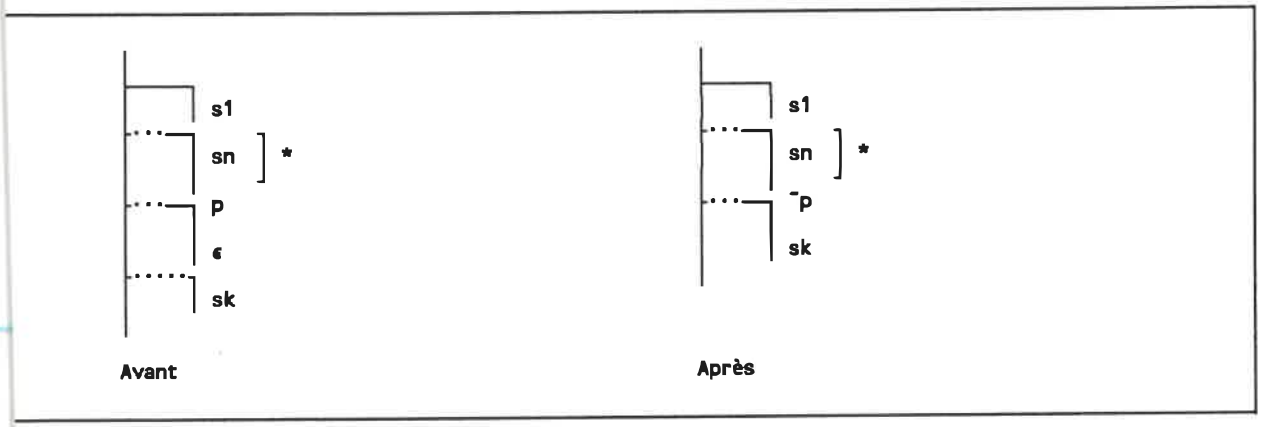


Fig. 4.34 - Elimination de SINON-SI final vide et conversion de SINON en SINON-SI.

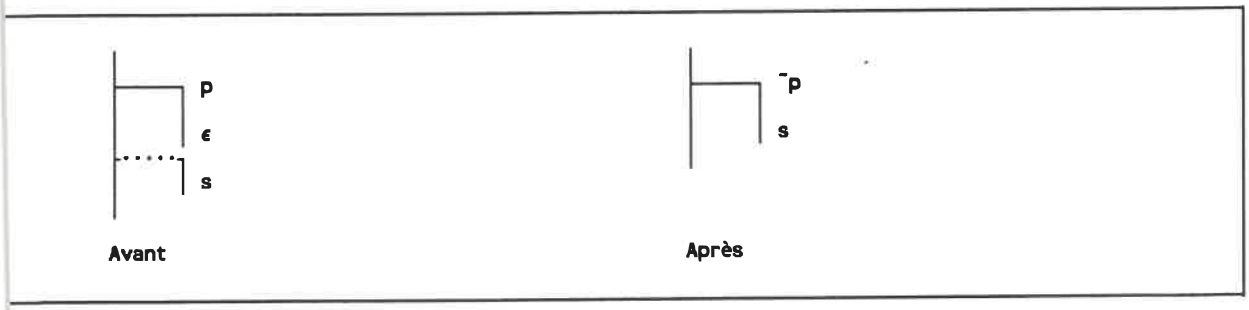


Fig. 4.35 - Elimination de SI vide et conversion de SINON en SI.

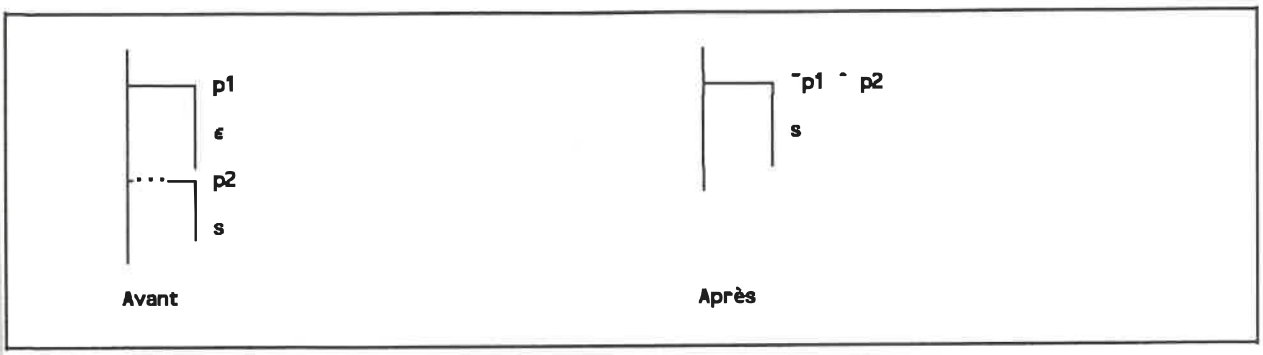


Fig. 4.36 - Elimination de SI vide et conversion de SINON-SI en SI.

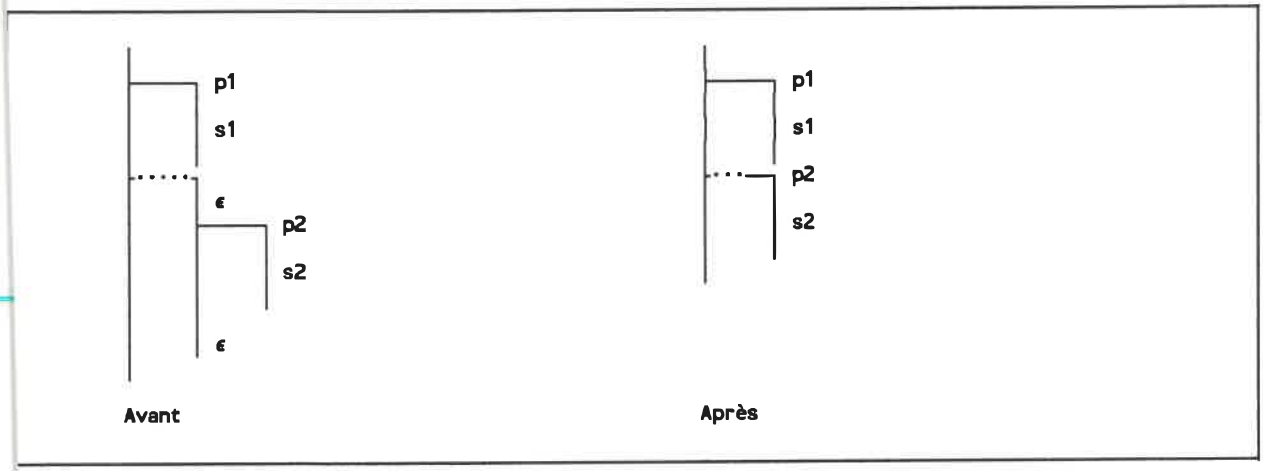


Fig. 4.37 - Conversion de SINON et SI en SINON-SI

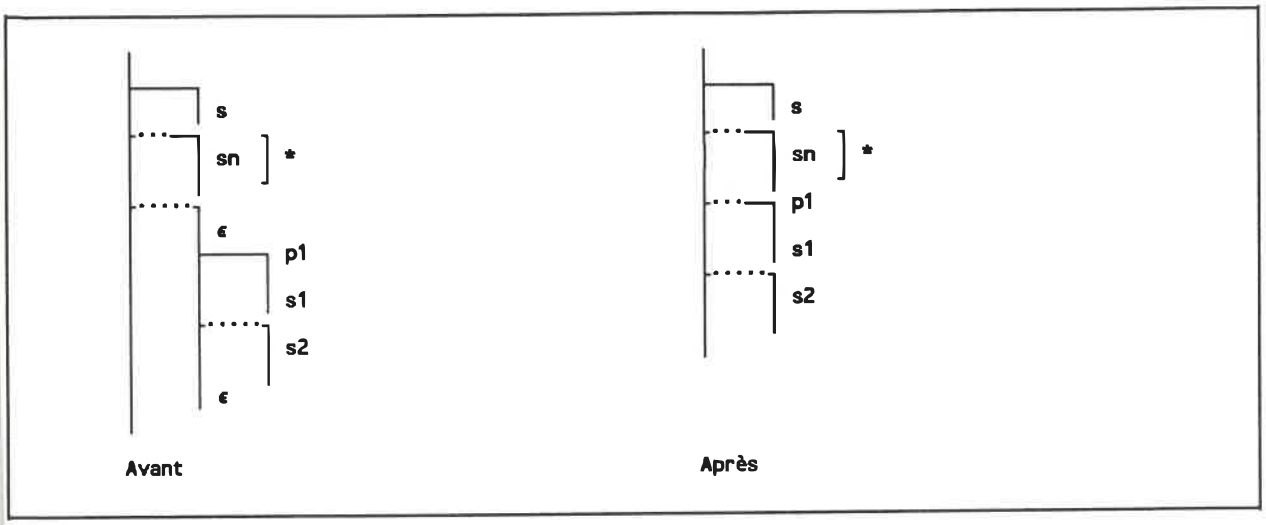


Fig. 4.38 - Création de SINON-SI

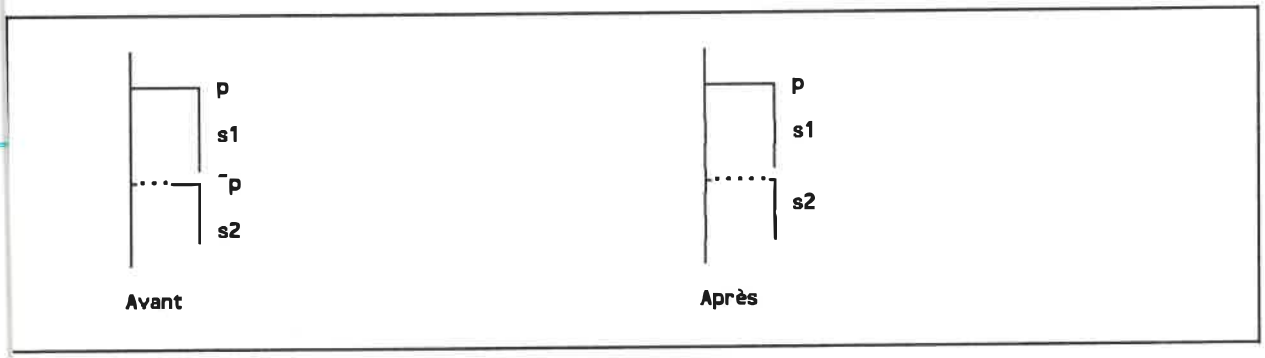


Fig. 4.39 - Création de SINON

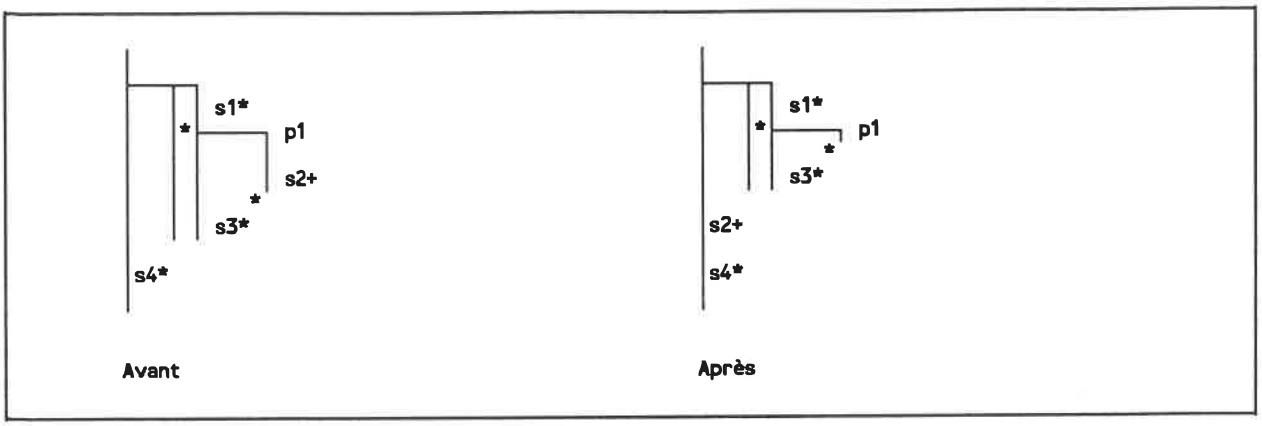


Fig. 4.40 - Déplacement de structures après une répétitive.

On s'aperçoit assez vite, cependant, que le graphe de contrôle n'est pas le support idéal pour cette étape car il ne met pas en évidence les structures et leur portée. Une représentation plus appropriée est l'arbre de syntaxe [2] [43] [50].

Chaque sommet représente une structure (ou une branche de structure) et possède une étiquette et un texte. L'étiquette peut prendre les valeurs suivantes:

/ : racine de l'arbre
S : structure séquentielle (actions ou commentaires)
K : structure conditionnelle
C : branche SI
I : branche SINON-SI
O : branche SINON
R : structure répétitive
E : sortie de boucle

Quant au texte, il est une expression booléenne dans le cas des sommets C, I et E, et les énoncés RTF dans le cas des structures séquentielles S. Dans les autres cas (/, K, O, R) le texte est nul car il n'est pas pertinent.

De plus, les arcs sortant d'un même sommet sont numérotés séquentiellement pour conserver l'ordre d'exécution. La figure 4.41 illustre un exemple en SPC et la figure 4.42, son arbre de syntaxe correspondant:

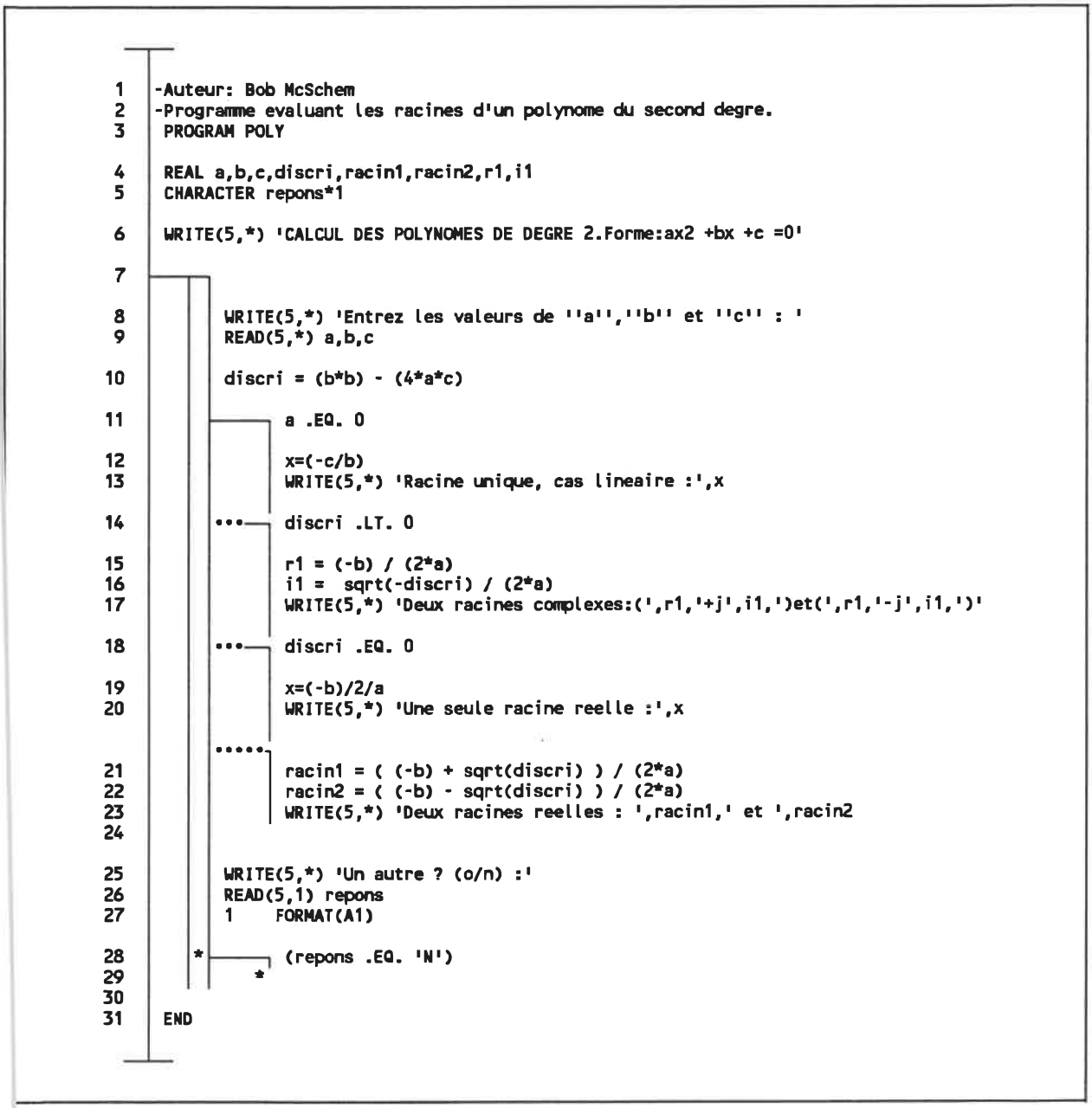


Fig. 4.41 - Programme en SPC.

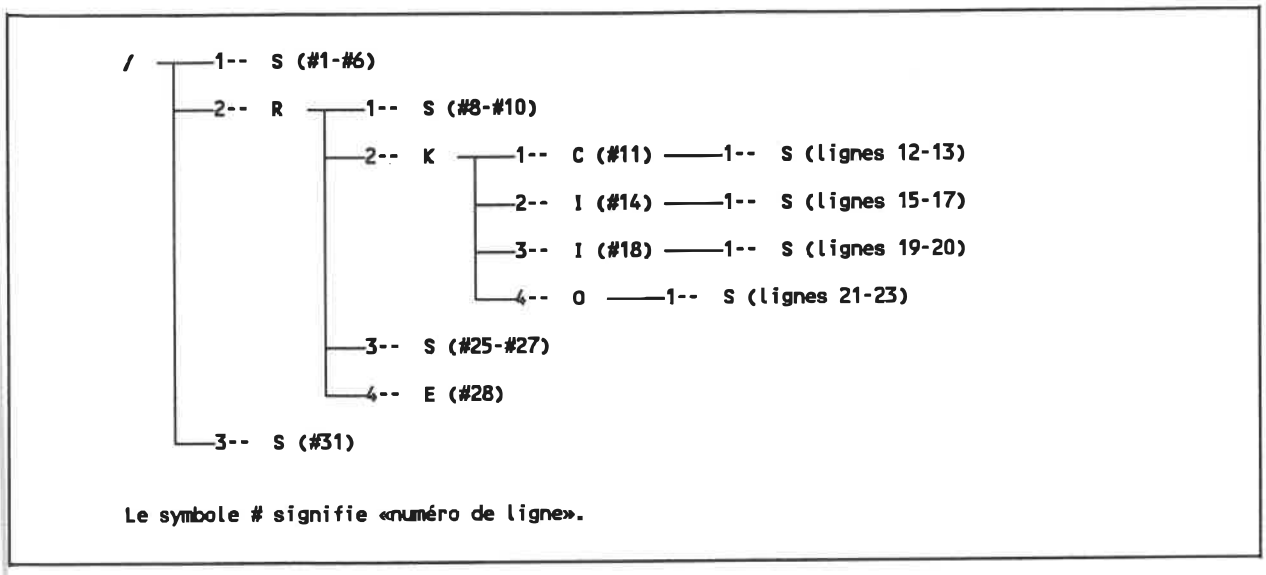


Fig. 4.42 - Arbre de syntaxe correspondant au SPC.

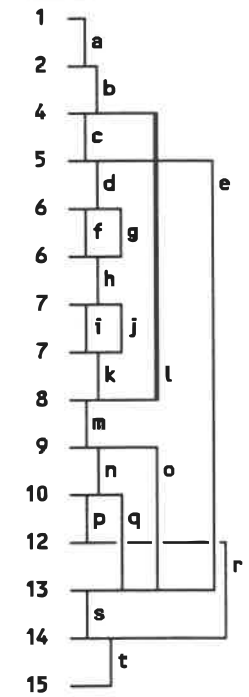
L'étape d'amélioration consiste donc à créer un arbre de syntaxe à partir du RTF produit par les transformations de structuration, à simplifier cet arbre en effectuant les transformées d'amélioration illustrées précédemment, puis à regénérer une suite d'instructions RTF dans un fichier récupérable par SCHEMACODE.

4.4 Exemple d'application

La méthode que nous venons de présenter comprend plusieurs étapes que nous avons examinées au point de vue théorique. Nous l'illustrons maintenant à l'aide d'un exemple aux figures 4.43 à 4.56. La procédure LOOKUP utilisée à cette fin est tirée d'un programme réel et modifiée afin de montrer le plus possible le mode d'opération de la méthode.

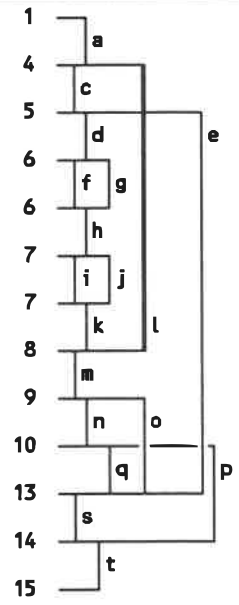
```
 1            INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH)
 2            INTEGER ITEM TABLE(1),FINISH,I
 3            START=1
 4       1    I = (START+FINISH)/2
 5            IF (ITEM .EQ. TABLE(I)) GOTO 2
 6            IF (TABLE(I) .LT. ITEM) START = I+1
 7            IF (TABLE(I) .GT. ITEM) FINISH = I-1
 8            IF (FINISH-START .GT.1) GOTO 1
 9            IF (TABLE(START) .EQ. ITEM) GOTO 2
10            IF (TABLE(FINISH) .EQ. ITEM) GOTO 2
11            LOOKUP = 0
12            GOTO 3
13       2    LOOKUP = 1
14       3    RETURN
15            END
```

Fig. 4.43 - Procédure originale en FORTRAN.



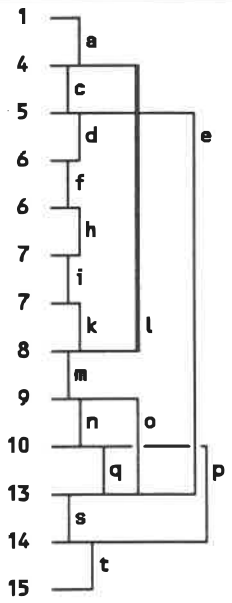
	Conditions d'exécution	Contenu des arcs
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH)
b		A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	ϵ
e	ITEM.EQ.TABLE(I)	ϵ
f	TABLE(I).LT.ITEM	A START = I+1
g	.NOT.(TABLE(I).LT.ITEM)	ϵ
h		ϵ
i	TABLE(I).GT.ITEM	A FINISH = I-1
j	.NOT.(TABLE(I).GT.ITEM)	ϵ
k		ϵ
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	ϵ
o	TABLE(START).EQ.ITEM	ϵ
p	.NOT.TABLE(FINISH).EQ.ITEM)	A LOOKUP = 0
q	TABLE(FINISH).EQ.ITEM	ϵ
r		ϵ
s		A LOOKUP = 1
t		A RETURN

Fig. 4.44 - Graphe de contrôle correspondant.



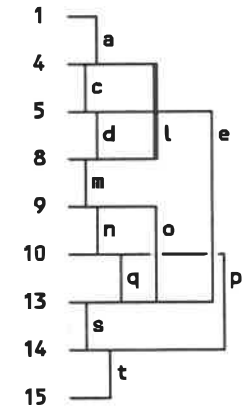
	Conditions d'exécution	Contenu des arcs
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	ϵ
e	ITEM.EQ.TABLE(I)	ϵ
f	TABLE(I).LT.ITEM	A START = I+1
g	.NOT.(TABLE(I).LT.ITEM)	ϵ
h		ϵ
i	TABLE(I).GT.ITEM	A FINISH = I-1
j	.NOT.(TABLE(I).GT.ITEM)	ϵ
k		ϵ
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	ϵ
o	TABLE(START).EQ.ITEM	ϵ
p	.NOT.TABLE(FINISH).EQ.ITEM)	A LOOKUP = 0
q	TABLE(FINISH).EQ.ITEM	ϵ
s		A LOOKUP = 1
t		A RETURN

Fig. 4.45 - Graphe après élimination du sommets procéduraux 2 et 12.



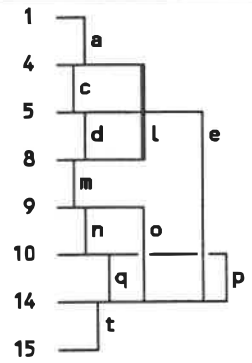
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	ϵ
e	ITEM.EQ.TABLE(I)	ϵ
f		C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F
h		ϵ
i		C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
k		ϵ
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	ϵ
o	TABLE(START).EQ.ITEM	ϵ
p	.NOT.TABLE(FINISH).EQ.ITEM)	A LOOKUP = 0
q	TABLE(FINISH).EQ.ITEM	ϵ
s		A LOOKUP = 1
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.46 - Graphe après élimination des arcs parallèles g et j.



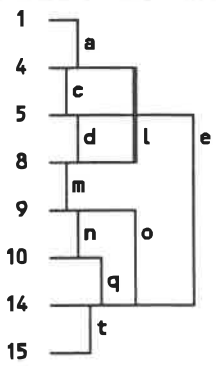
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	€
l	FINISH-START.GT.1	€
m	.NOT.(FINISH-START.GT.1)	€
n	.NOT.(TABLE(START).EQ.ITEM)	€
o	TABLE(START).EQ.ITEM	€
p	.NOT.TABLE(FINISH).EQ.ITEM)	A LOOKUP = 0
q	TABLE(FINISH).EQ.ITEM	€
s		A LOOKUP = 1
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.47 - Graphe après élimination du sommets procéduraux 6 et 7: forme standard non-structurée.



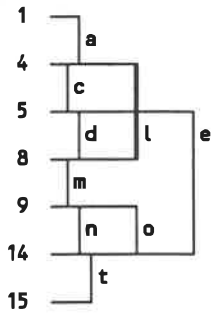
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	A LOOKUP = 1
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	ϵ
o	TABLE(START).EQ.ITEM	A LOOKUP = 1
p	.NOT.TABLE(FINISH).EQ.ITEM)	A LOOKUP = 0
q	TABLE(FINISH).EQ.ITEM	A LOOKUP = 1
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.48 - Graphe après deux transformations ID-0 au sommet 13 (arc s dupliqué dans e, o et q).



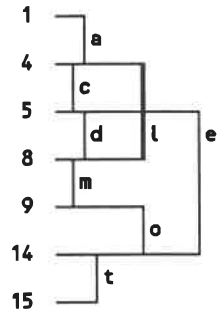
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	A LOOKUP = 1
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	ϵ
o	TABLE(START).EQ.ITEM	A LOOKUP = 1
q		C TABLE(FINISH).EQ.ITEM A LOOKUP = 1 I .NOT.TABLE(FINISH).EQ.ITEM) A LOOKUP = 0 F
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.49 - Graphe après élimination de l'arc parallèle p.



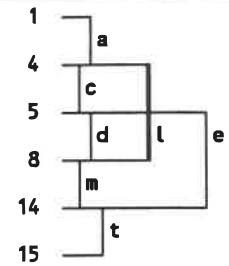
a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	A LOOKUP = 1
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
n	.NOT.(TABLE(START).EQ.ITEM)	C TABLE(FINISH).EQ.ITEM A LOOKUP = 1 I .NOT.TABLE(FINISH).EQ.ITEM) A LOOKUP = 0 F
o	TABLE(START).EQ.ITEM	A LOOKUP = 1
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.50 - Graphe après élimination du sommet procédural 10.



a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	A LOOKUP = 1
l	FINISH-START.GT.1	ϵ
m	.NOT.(FINISH-START.GT.1)	ϵ
o		C TABLE(START).EQ.ITEM A LOOKUP = 1 I .NOT.(TABLE(START).EQ.ITEM) C TABLE(FINISH).EQ.ITEM A LOOKUP = 1 I .NOT.TABLE(FINISH).EQ.ITEM) A LOOKUP = 0 F F
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.51 - Graphe après élimination de l'arc parallèle n.



a		A INTEGER FUNCTION LOOKUP(ITEM,TABLE,FINISH) A INTEGER ITEM TABLE(1),FINISH,I A START=1
c		A I = (START+FINISH)/2
d	.NOT.(ITEM.EQ.TABLE(I))	C TABLE(I).LT.ITEM A START = I+1 I .NOT.(TABLE(I).LT.ITEM) F C TABLE(I).GT.ITEM A FINISH = I-1 I .NOT.(TABLE(I).GT.ITEM) F
e	ITEM.EQ.TABLE(I)	A LOOKUP = 1
l	FINISH-START.GT.1	6
m	.NOT.(FINISH-START.GT.1)	C TABLE(START).EQ.ITEM A LOOKUP = 1 I .NOT.(TABLE(START).EQ.ITEM) C TABLE(FINISH).EQ.ITEM A LOOKUP = 1 I .NOT.TABLE(FINISH).EQ.ITEM) A LOOKUP = 0 F F
t		A RETURN
	Conditions d'exécution	Contenu des arcs

Fig. 4.52 - Graphe après élimination du sommet procédural 9.

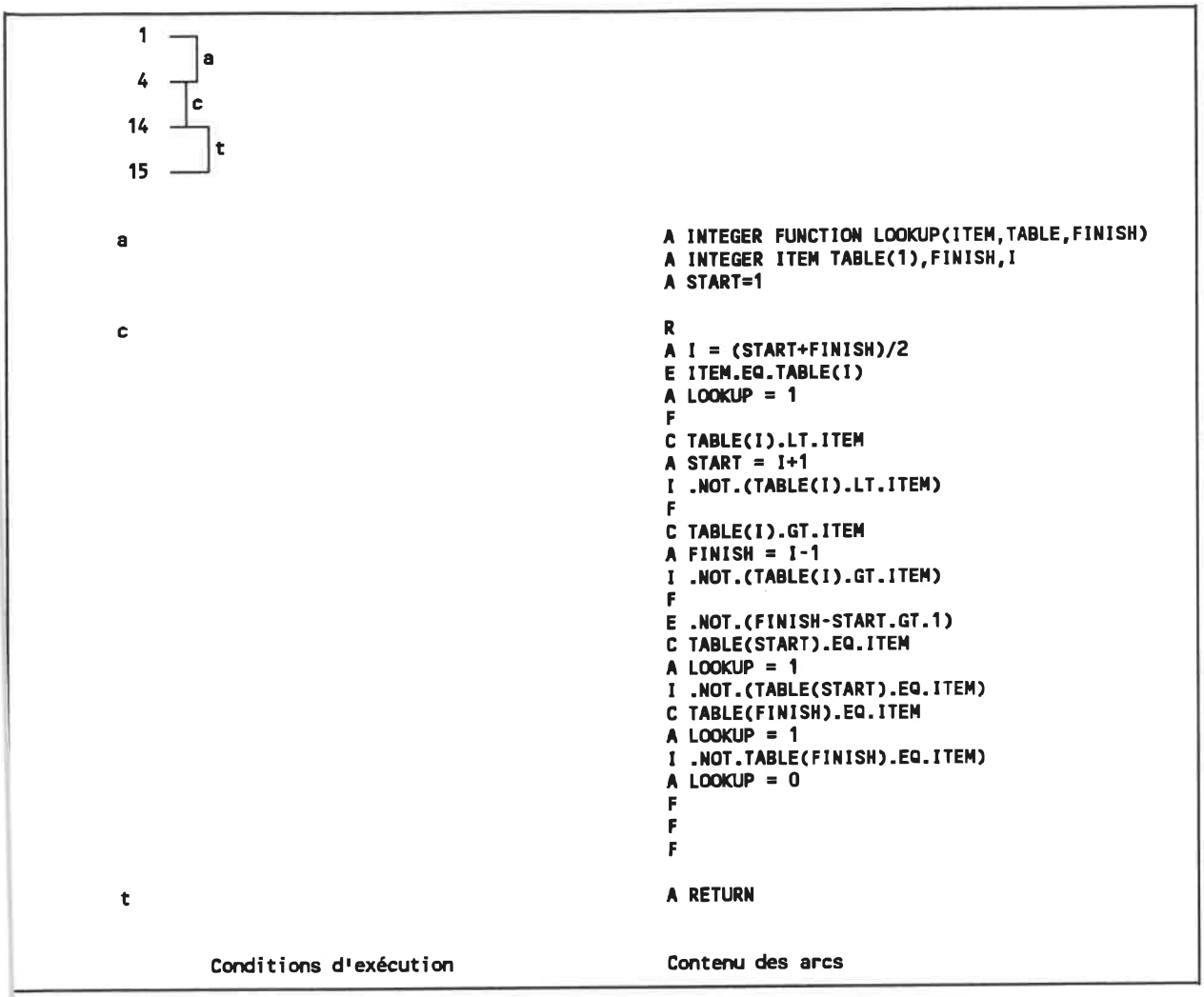


Fig. 4.53 - Graphe après élimination de la boucle généralisée entre les sommets 4 et 14.

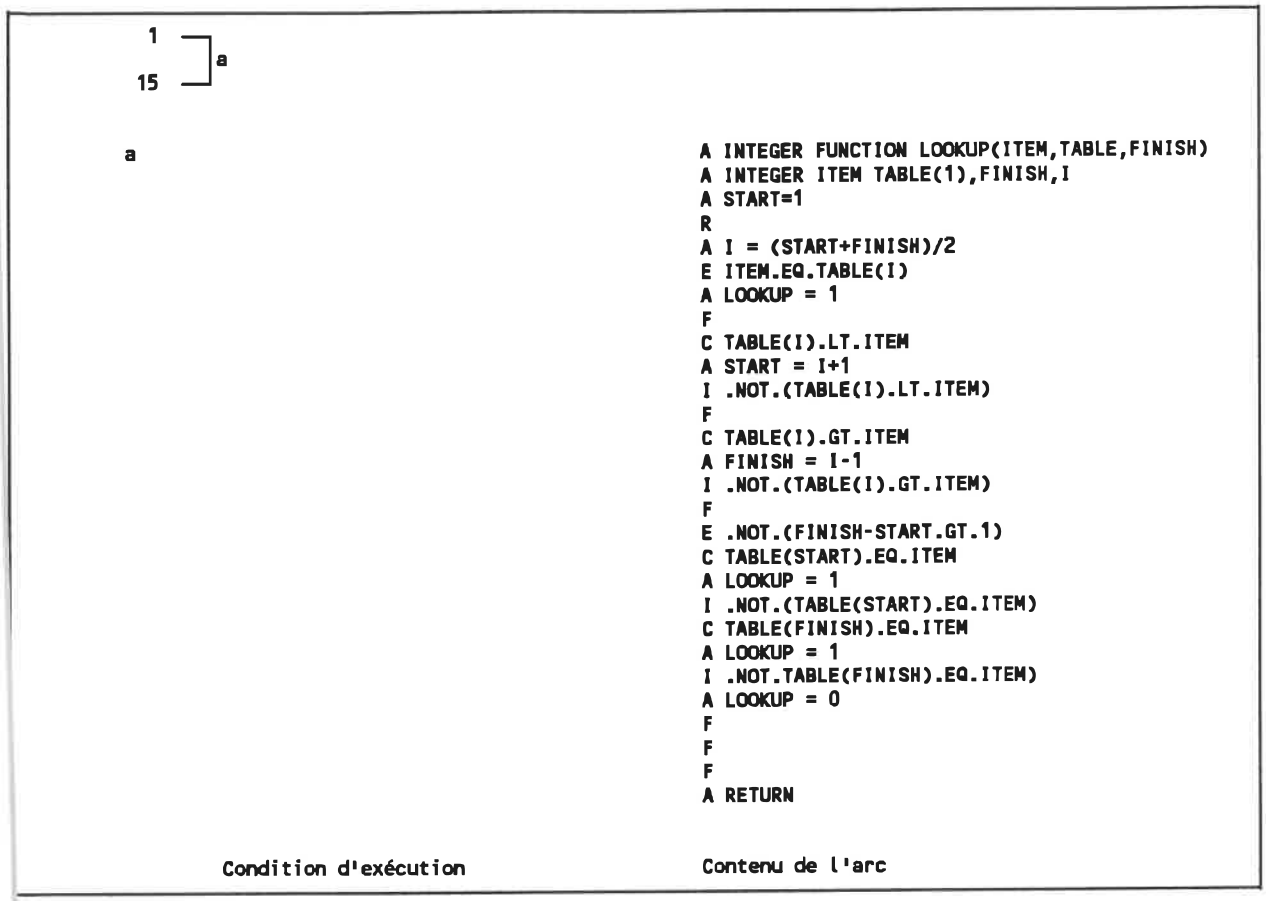


Fig. 4.54 - Graphe après élimination des sommets procéduraux 4 et 14 (graphe structuré).

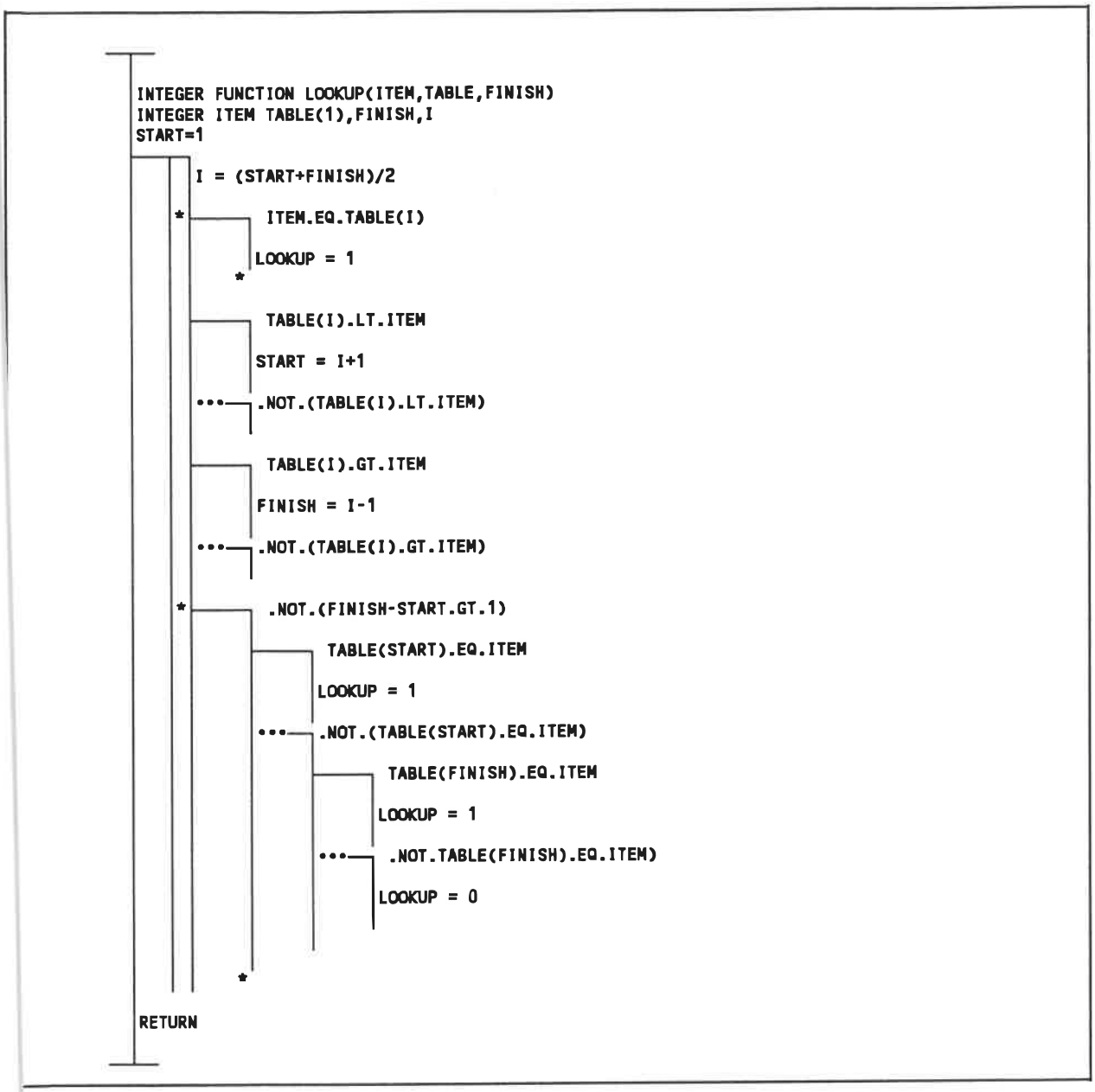


Fig. 4.55 - SPC correspondant au RTF produit.

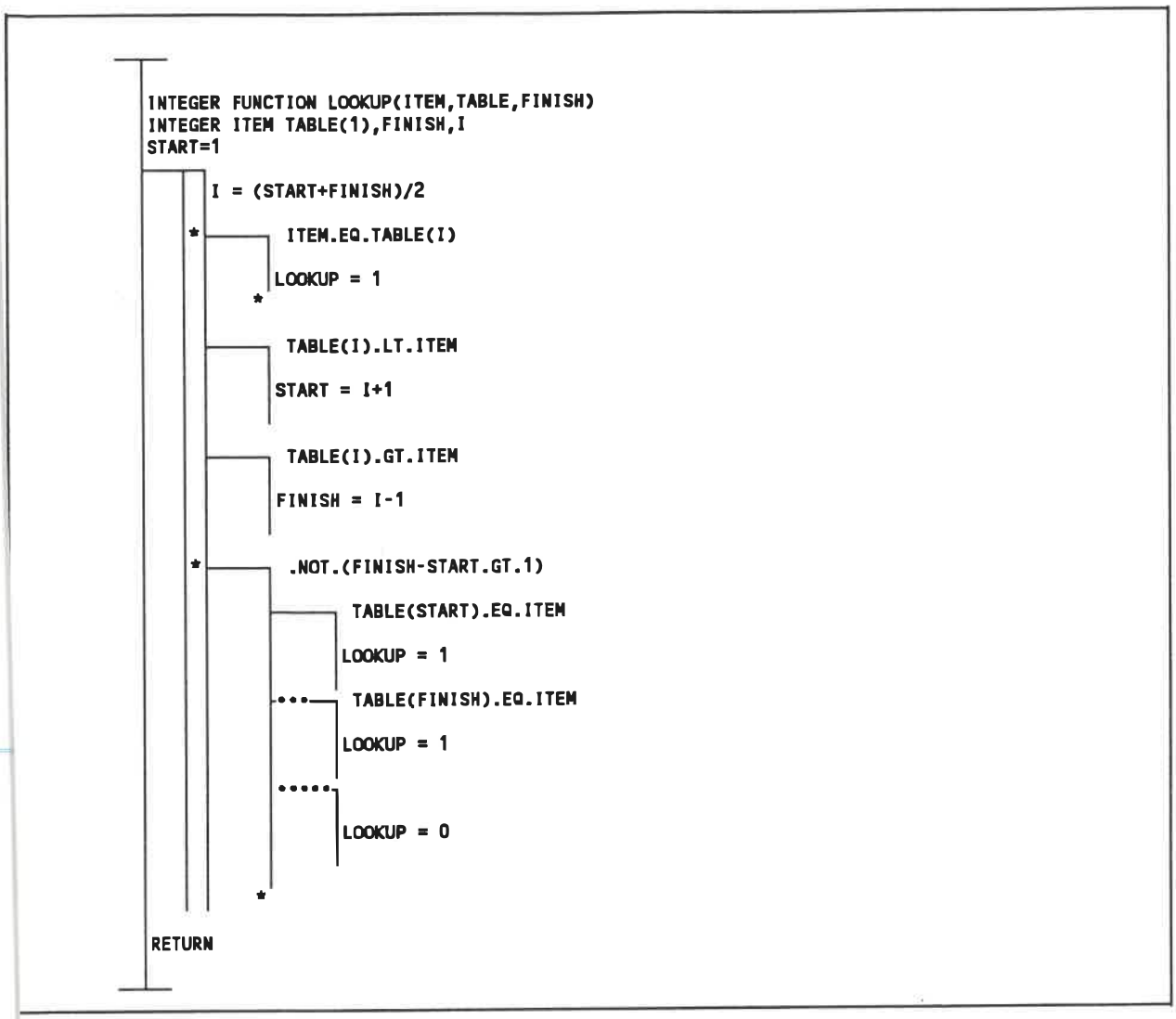


Fig. 4.56 - SPC après amélioration (final).

CHAPITRE 5

RÉALISATION DU PROTOTYPE

The essence of knowledge is, after having acquired it, to apply it (Confucius, 450 B.C.)

La méthode de traduction étant maintenant définie de façon systématique, nous présentons ici un prototype de système de traduction automatisée. Ce prototype a été réalisé dans le cadre de notre recherche et a permis, dans une certaine mesure, de perfectionner la méthode en général en fournissant des résultats concrets.

Nous décrivons d'abord brièvement l'environnement de développement du système, puis nous examinons chacune de ses composantes en exposant ses fonctions. Nous terminons par un résumé de performances et quelques métriques sur les programmes réalisés.

5.1 Environnement de développement

Pour des raisons de disponibilité et de coût, le système complet a été réalisé dans un environnement micro-ordinateur, sous un système d'exploitation PC-DOS.

Le matériel consiste en une unité centrale compatible IBM basée sur un micropocesseur 80286 (12 MHz), avec 640K octets de mémoire centrale et 2 Moctets de mémoire étendue (EMS). La mémoire auxiliaire est un disque rigide de 30 Moctets avec 65 ms de temps moyen d'accès.

Cet environnement s'est avéré tout-à-fait adéquat à l'usage car il offre une complète autonomie et une puissance de calcul suffisante.

Du côté logiciel, nous avons utilisé les produits suivants:

- système d'exploitation PC-DOS 3.3 (Microsoft);
- SCHEMACODE 1.5 avec traducteur C (Schémacode International);
- DATRIX 1.6 avec analyseur FORTRAN (Bell Canada);
- compilateur C 5.1 (Microsoft);
- interpréteur AWK (Mortice Kern System).

Le développement des programmes a été réalisé exclusivement à l'aide de SCHEMACODE. Pour tous les programmes, le pseudocode schématique a été traduit en langage C, sauf pour GBOOL qui a été traduit en AWK.

Le compilateur C a été configuré avec l'option large model de façon à profiter de toute la mémoire disponible. Cette caractéristique est importante à cause de l'utilisation intensive de fonctions d'allocation dynamique de mémoire.

5.2 Composantes du système

Le diagramme de la figure 5.1 illustre les composantes du prototype réalisé. Chaque composante est ensuite décrite en détail. Le contenu des fichiers est ensuite illustré à l'aide de l'exemple du chapitre précédent (LOOKUP).

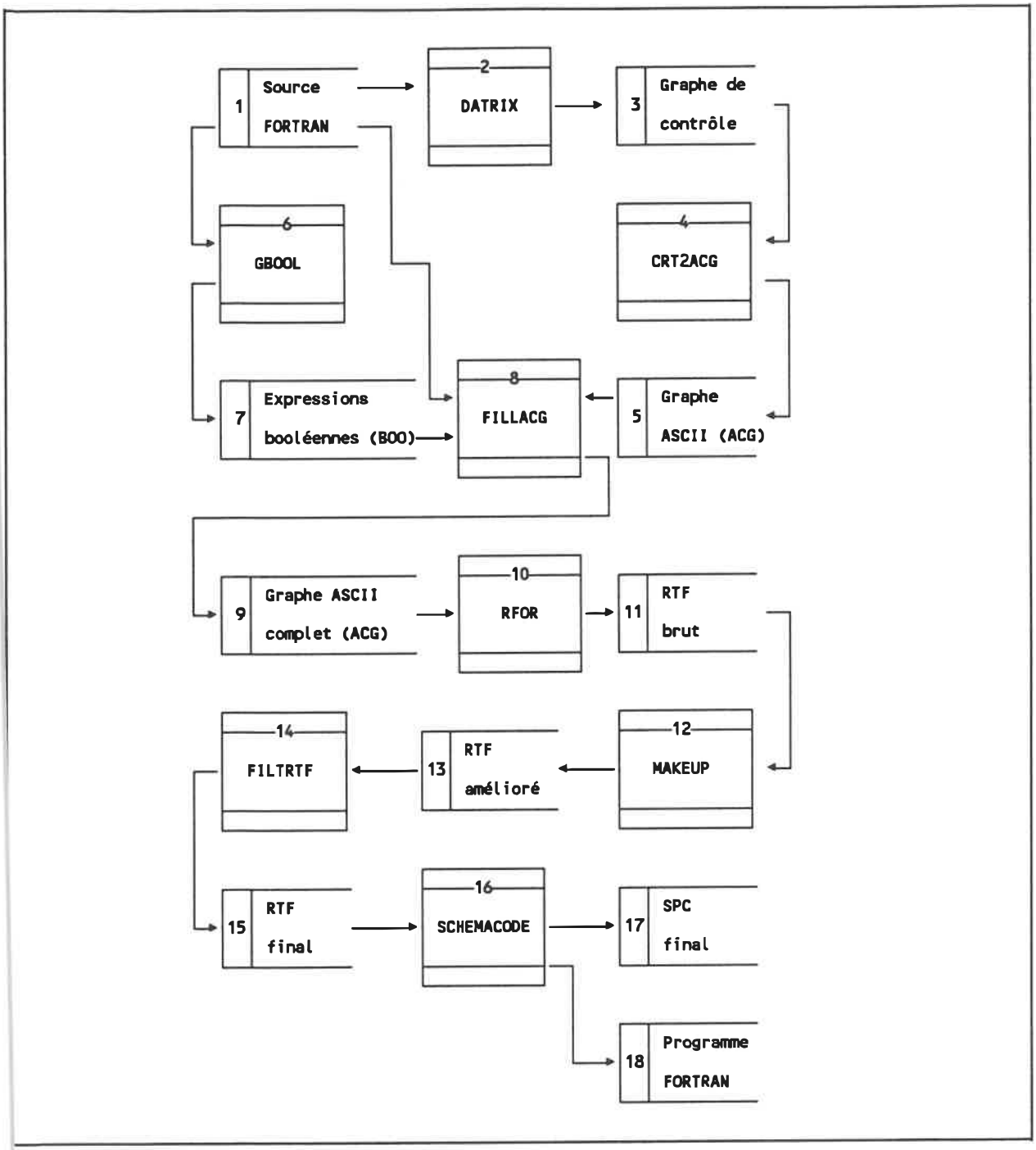


Fig. 5.1 - Diagramme de flux de données du système automatisé.

5.2.1 Fichier source FORTRAN

Tel que nous l'avons déjà mentionné, le fichier d'entrée (1) contient un programme écrit en FORTRAN 77 et codé en ASCII. Notons que ceci inclut implicitement le FORTRAN 66.

Au moment de la rédaction de ce texte, certaines limitations de l'analyseur FORTRAN de DATRIX imposent cependant des contraintes:

- le type BYTE, spécifique au compilateur VAX FORTRAN, n'est pas reconnu et provoque des erreurs d'analyse;
- des énoncés FORMAT ne peuvent apparaître dans la section de déclaration des variables;
- certains types d'énoncés DATA ne sont pas reconnus.

Ces énoncés peuvent être placés en commentaires avant l'analyse pour éviter les erreurs; ceci n'a pas de conséquence sur la construction du graphe de contrôle.

D'autres contraintes existent toutefois à cause des limitations des programmes développés. Ces contraintes peuvent être facilement respectées en modifiant les fichiers FORTRAN originaux à l'aide d'un éditeur de texte et/ou de programmes particuliers. Voici la liste de ces contraintes:

- le fichier ne doit pas contenir de marques de tabulation (TAB); il faut les remplacer par des espaces;
- les lignes vides sont interdites;
- chaque fichier ne peut contenir qu'une seule procédure; le nom du fichier et de la procédure doivent être les mêmes;

- aucun "code mort" ne doit être présent, i.e. tous les énoncés doivent pouvoir être atteignables à partir du premier;
- les énoncés INCLUDE ne sont pas acceptés; tout le code doit être présent dans un seul fichier;
- les énoncés IF THEN, ELSE IF THEN, ELSE et ASSIGN ne sont pas supportés;
- les énoncés DO ne peuvent être étiquetés; pour corriger ceci, insérer un énoncé CONTINUE avant le DO et y déplacer l'étiquette.

La figure 5.2 illustre un exemple de programme FORTRAN.

```
      INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
      INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
      START=1
      FINISH=LONG
1     I = (START+FINISH)/2
      IF (ITEM .EQ. TABLE(I)) GOTO 2
      IF (TABLE(I) .LT. ITEM) START = I+1
      IF (TABLE(I) .GT. ITEM) FINISH = I-1
      IF (FINISH-START .GT.1) GOTO 1
      IF (TABLE(START) .EQ. ITEM) GOTO 2
      IF (TABLE(FINISH) .EQ. ITEM) GOTO 2
      LOOKUP = 0
      GOTO 3
2     LOOKUP = 1
3     RETURN
      END
```

Fig. 5.2 - Fichier FORTRAN (procédure LOOKUP)

5.2.2 Programme DATRIX

DATRIX (2) est un logiciel développé au laboratoire de recherche en génie logiciel de l'Ecole Polytechnique de Montréal. Son but est d'évaluer un certain nombre de métriques du logiciel, tant sur le code source brut que sur son graphe de contrôle correspondant.

Ce qui rend DATRIX intéressant pour nous, c'est son analyseur FORTRAN et son générateur de graphe de contrôle. Comme ses capacités nous sont disponibles, nous les avons utlisées.

L'information concernant la structure interne du graphe et des fichiers de DATRIX (3) étant toutefois confidentielle nous ne pouvons pas en discuter ici. Néanmoins, la figure 5.3 illustre comment le graphe est présenté à l'usager:

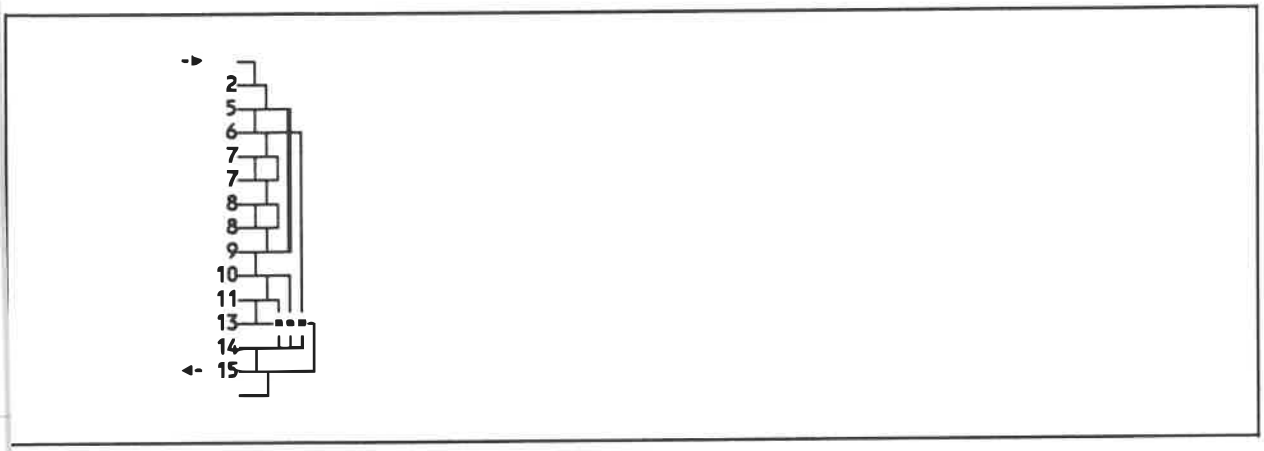


Fig. 5.3 - Graphe de contrôle produit par DATRIX.

5.2.3 Programme CRT2ACG

Ce programme (4) sert à convertir le graphe de contrôle (3) stocké dans un fichier par DATRIX en un graphe équivalent mais non-compressé (5). Il sert d'interface afin de rendre les autres programmes indépendants de la représentation interne utlisée par DATRIX. Pour des raisons de confidentialité, le processus de conversion ne peut cependant être décrit ici

5.2.4 Fichier de graphe de contrôle ASCII (ACG)

Le fichier ACG (5) contient le graphe de controle d'une procédure. On y retrouve de l'information concernant chaque sommet:

- un numéro d'identification de sommet;
- le numéro de la ligne qui a produit le sommet;
- le type de sommet, relié à un type d'énoncé FORTRAN.

On y retrouve également de l'information concernant chaque arc:

- un numéro d'identification d'arc;
- un numéro de séquence identifiant chaque cas d'arc sortant dans un sommet de décision;
- un poids, indiquant si l'arc contient des énoncés ou s'il est vide;
- une condition d'exécution représentée par une expression booléenne FORTRAN;
- le nombre de lignes de code associées à l'arc.

Cette information est conservée en format ASCII, et non pas en binaire, afin d'en faciliter l'examen visuel et, lors de la mise au point, d'en permettre la modification directe à l'aide d'un éditeur de texte.

Le contenu du fichier est défini par une syntaxe simple, illustrée à la figure 5.4:

```
Fichier_ACG                 ::=  sommet+

sommet                      ::=      info sommet
                                |  arcs_sortants*

info-sommet                 ::=    no_sommet . no_ligne . type <NL>

no_sommet                   ::=    nombre

no_ligne                    ::=    nombre

type                        ::=    [0-9] | 99

arcs_sortants               ::=    no_arc no_seq . poids . condition_exec <NL>
                                   nb_lignes <NL>
                                   (lignes_de_code <NL> )*
                                   <NL>

no_arc                      ::=    nombre

no_seq                      ::=    nombre

poids                       ::=    nombre

condition_exec              ::=    EXPRESSION BOOLEENNE FORTRAN

nb_ligne                    ::=    nombre

lignes de code              ::=    LIGNE EN FORMAT RTF
```

Fig. 5.4 - Syntaxe du contenu d'un fichier ACG.

Il est important de noter qu'à cause de certaines lacunes dans le fichier de graphe de DATRIX, le programme CRT2ACG ne peut évaluer les éléments d'information suivants:

- le type d'un sommet;
- la condition d'exécution d'un arc;
- les lignes de code RTF d'un arc.

Conséquemment, l'information contenue dans le fichier ACG produit par CRT2ACG est incomplète.

5.2.5 Programme GBOOL et fichier d'expressions booléennes (BOO)

Ce programme (6) sert à extraire, à partir du fichier source FORTRAN, l'information nécessaire pour identifier les types de sommets et les conditions d'exécution d'arcs. Cette information est conservée dans un fichier BOO (7), dont voici la syntaxe (figure 5.5):

```
Fichier_BOO          ::=    ligne+

ligne                ::=    no_ligne . type . nb_cont . offset . condition

no.ligne             ::=    nombre

nombre               ::=    [0 9]+

type                 ::=    -? ( [0-9] | 99 )

nb_cont              ::=    nombre

offset               ::=    nombre

condition            ::=    EXPRESSION BOOLEENNE FORTRAN
```

Fig. 5.5 - Syntaxe du contenu d'un fichier BOO.

Chaque ligne correspond à un sommet du graphe. L'information stockée comprend:

- le numéro de la ligne FORTRAN visée par le sommet;
- le type d'énoncé (GOTO, IF GOTO, etc.);
- le nombre de lignes de continuation;

- la position de l'énoncé dans un IF_énoncé;
- le texte nécessaire à la construction des expressions booléennes des arcs sortants du sommet.

A partir du fichier FORTRAN présenté précédemment, GBOOL produit le fichier suivant (figure 5.6):

```
-1       0  0
      1 99  0
      2 99  0
      3 99  0
      4 99  0
      5 99  0
      6  7  0[IFGO]ITEM .EQ. TABLE(I)
      7  6  0 30   TABLE(I) .LT. ITEM
      8  6  0 30   TABLE(I) .GT. ITEM
      9  7  0[IFGO]FINISH-START .GT.1
     10  7  0[IFGO]TABLE(START) .EQ. ITEM
     11  7  0[IFGO]TABLE(FINISH) .EQ. ITEM
     12 99  0
     13  2  0[GO  ]
     14 99  0
     15 99  0
     16 99  0
      0  0  0
```

Fig. 5.6 - Fichier BOO produit par GBOOL.

Notons que le programme GBOOL a été réalisé en pseudocode schématique codé en AWK. Le langage AWK [1] a été choisi à cause de la puissance à reconnaître des séquences de caractères, de la concision des expressions régulières et de la rapidité de réalisation que son interpréteur procure.

5.2.6 Programme FILLACG

Ce programme (8) la représente la dernière partie du front end de notre système. FILLACG intègre le graphe produit par CRT2ACG (5) au texte du source FORTRAN (1) et construit les conditions d'exécution à partir du texte (7) produit par GBOOL.

Le résultat est un fichier ACG (9) qui contient toute l'information nécessaire à sa traduction en pseudocode schématique. Un exemple est illustré à la figure 5.7:

```
1.-1.0
        2.2.0.
1
A INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)

2.2.99
        3.2.0.
3
A INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
A START=1
A FINISH=LONG

3.5.99
        4.1.0.
1
A I = (START+FINISH)/2

4.6.7
        5.0.2..NOT.(ITEM .EQ. TABLE(I))
1
X       CONTINUE
        13.0.1.ITEM .EQ. TABLE(I)
1
X       CONTINUE

5.7.6
        6.1.1.TABLE(I) .LT. ITEM
1
A START = I+1
        6.0.2..NOT.(TABLE(I) .LT. ITEM)
1
X       CONTINUE

6.7.6
        7.0.0..NOT.(TABLE(I) .LT. ITEM)
1
X       CONTINUE

7.8.6
        8.1.1.TABLE(I) .GT. ITEM
1
A FINISH = I-1
        8.0.2..NOT.(TABLE(I) .GT. ITEM)
1
X       CONTINUE

8.8.6
        9.0.0..NOT.(TABLE(I) .GT. ITEM)
1
X       CONTINUE

9.9.7
        10.0.2..NOT.(FINISH-START .GT.1)
1
X       CONTINUE
        3.0.1.FINISH-START .GT.1
1
X       CONTINUE

10.10.7
        11.0.2..NOT.(TABLE(START) .EQ. ITEM)
1
X       CONTINUE
        13.0.1.TABLE(START) .EQ. ITEM
1
X       CONTINUE
```

```
11.11.7
        12.1.2..NOT.(TABLE(FINISH) .EQ. ITEM)
1
A LOOKUP = 0
   .    13.0.1.TABLE(FINISH) .EQ. ITEM
1
X       CONTINUE

12.13.2
        14.0.0.
1
X       CONTINUE

13.14.99
        14.1.0.
1
A LOOKUP = 1

14.15.99
        15.0.0.
2
A RETURN
A END

15.0.0
```

Fig. 5.7 - Fichier ACG produit par FILLACG.

Notons que pour simplifier le traitement subséquent, les arcs vides sont remplis par des instructions RTF spéciales: X CONTINUE. Ces instructions sont éliminées subséquemment.

5.2.7 Programme RFOR

Ce programme (10) est essentiellement le coeur du système de traduction. Il accepte comme intrant un fichier ACG complet (9) et la stocke en mémoire centrale, à l'aide d'un type de donnée abstrait, pour manipuler le graphe rapidement. L'algorithme utilisé est essentiellement celui que nous avons décrit au chapitre précédent.

RFOR lit d'abord le graphe et le met en forme standard. S'il est structuré, un fichier RTF est produit. Sinon, une succession de transformations de restructuration

(ID-0, ID-1, OD-1, IL-0, et IL-1) et de mises en forme standard est effectuée. A la fin du processus, un fichier de résultats (11) est normalement toujours produit; toutefois, le système s'auto-vérifie constamment et s'arrête si une condition anormale et imprévue survient.

Notons que le système possède un paramètre de contrôle de duplication de code lors des transformations ID-0 et IL-0. Si une transformation demande plus de lignes dupliquées que le paramètre ne le spécifie, les transformations ID-1 et IL-1 sont utilisées au lieu de ID-0 et IL-0 respectivement.

Le diagramme hiérarchique de la figure 5.8 illustre les procédures les plus importantes de ce programme.

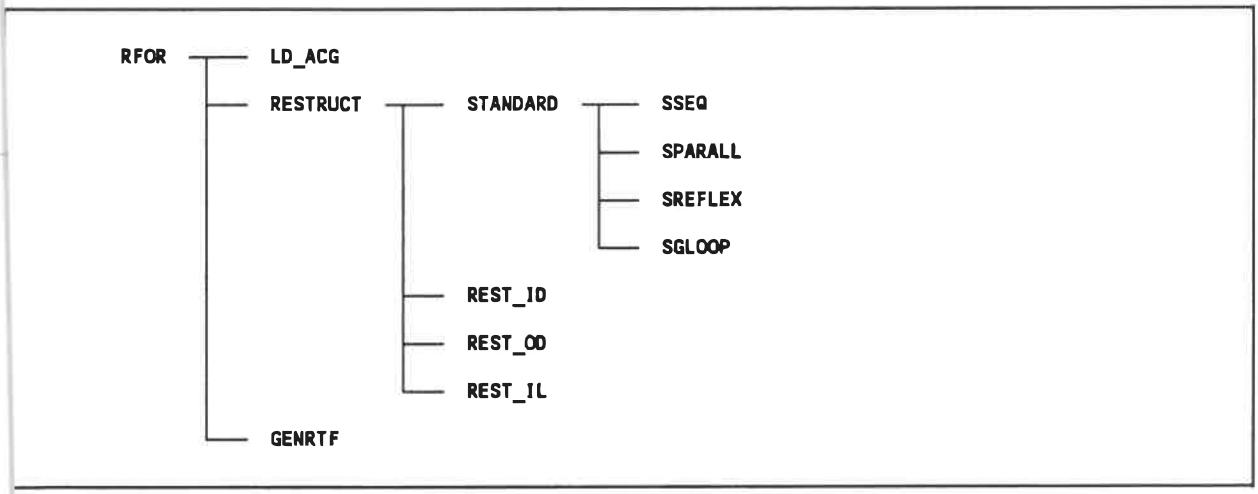


Fig. 5.8 - Diagramme hiérarchique de RFOR.

En résumé, on peut présenter l'algorithme implanté par les diagrammes en pseudocode schématique suivant (figures 5.9 à 5.11).

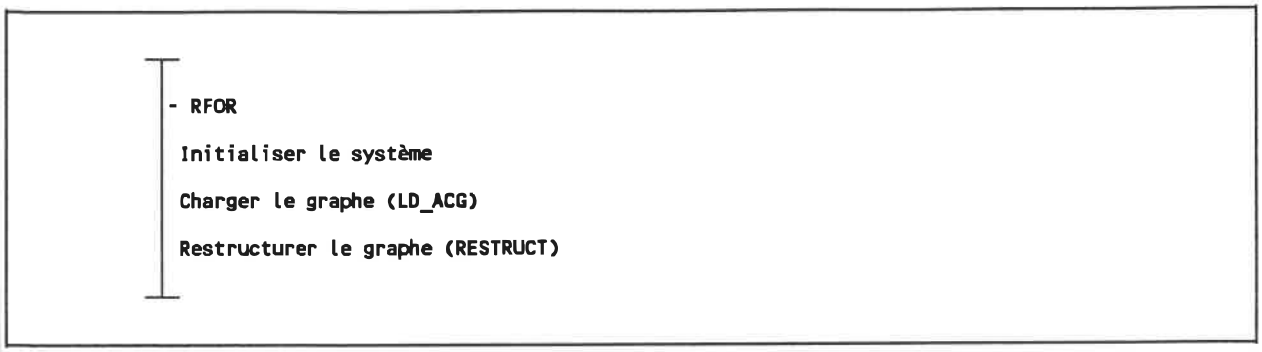


Fig. 5.9 - Algorithme de RFOR.

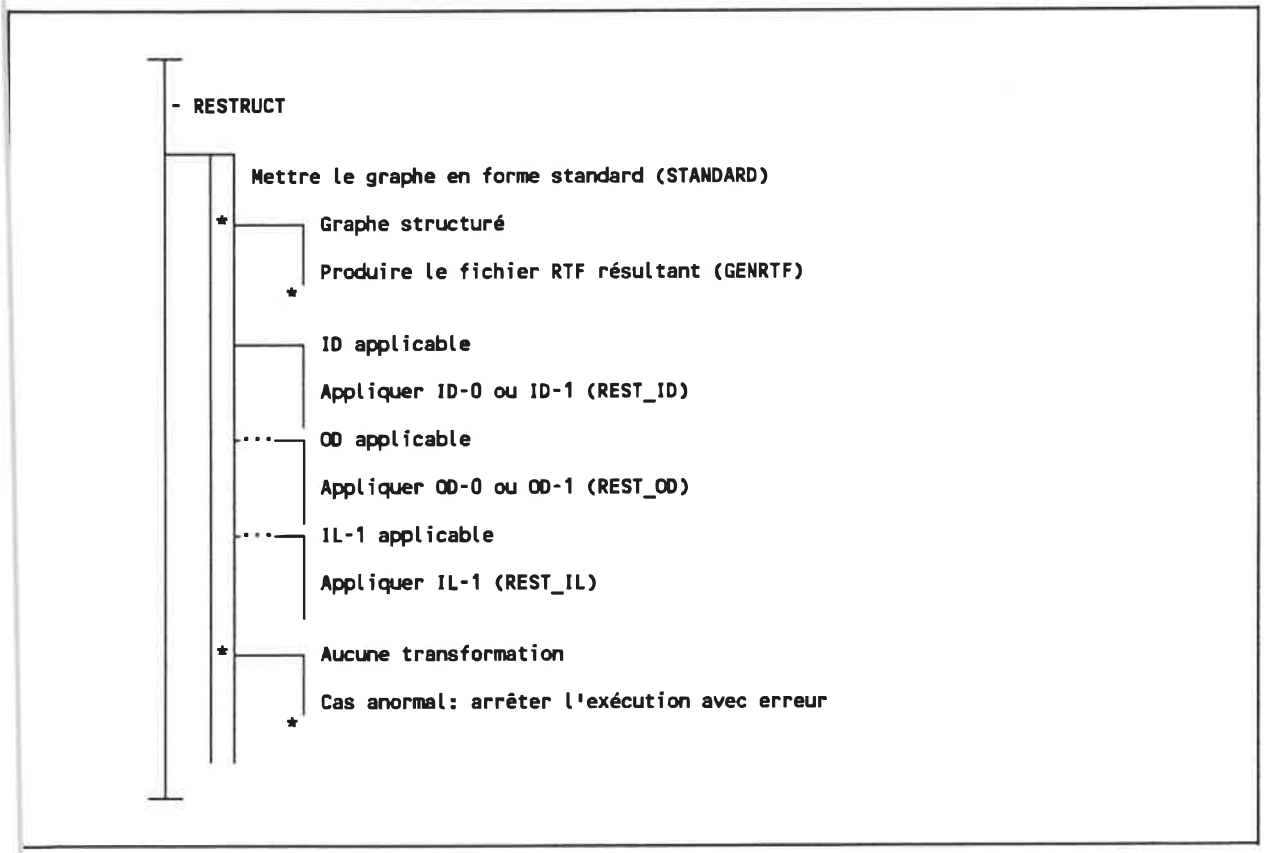


Fig. 5.10 - Algorithme de RESTRUCT.

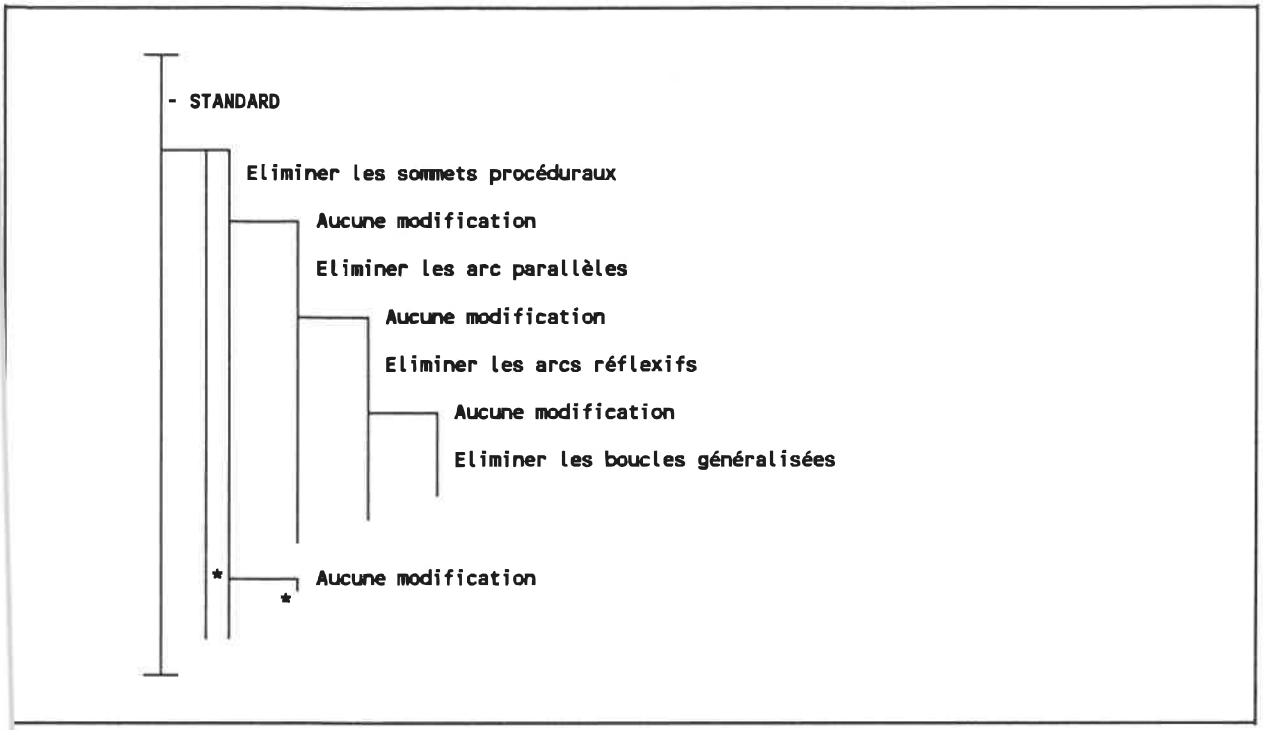


Fig. 5.11 - Algorithme de STANDARD.

Pour chaque transformation de restructuration, RESTRUCT cherche toujours la forme non-structurée ayant la portée (i.e. la longueur de chemin) la plus petite possible. Cela assure la production de variables de contrôle ayant l'impact le plus local possible dans le code. Ces variables ont d'ailleurs un nom qui débute toujours par «VAR_» suivi d'un numéro. Les initialisations et tests de ces variables sont produits au bon endroit dans le programme généré. Cependant leurs déclarations sont ajoutées automatiquement au début du programme d'où elles pourront facilement être relocalisées à l'endroit approprié.

A cause de l'utilisation exclusive de mémoire allouée dynamiquement, il n'y a pas de limites précises sur les dimensions du graphe, du texte qu'il inclut, etc, sauf la limite imposée par le compilateur (65535 caractères pour le texte d'un arc). Des graphes possédant jusqu'à 300 sommets et 500 arcs ont été traités avec succès.

Une fois la transformation terminée, un certain nombre de statistiques sont produites afin d'évaluer le processus. Un exemple de statistiques est donné à la figure 5.12.

```
RFOR 4.1

Fichier                          :  LOOKUP
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   20
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :    6
Arcs paralleles                  :    2
Arcs reflexifs                   :    0
Boucles generalisees             :    0
                                   ----
Total                            :    8

MESURES

% arcs elimines avant restruct.  :   42 %
% op. standard. avant restruct.  :   53 %

STANDARDISATION (total)

Sommets proceduraux              :   10
Arcs paralleles                  :    4
Arcs reflexifs                   :    0
Boucles generalisees             :    1
                                   ----
Total                            :   15

RESTRUCTURATION

Transformations ID-0             :    1
Transformations ID-1             :    0
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    1

Nombre de variables creees       :    0
Nombre de lignes dupliquees      :    2
```

Fig. 5.12 - Statistiques produites par RFOR concernant la traduction.

Le fichier produit respecte la syntaxe du sous-ensemble de RTF que nous avons décrite au chapitre précédent. Un exemple de fichier est illustré à la figure 5.13:

```
T ********************************************************
T Programme produit par           : RFOR 4.1
T Nombre de variables introduites : 0
T ********************************************************
A INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
A INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
A START=1
A FINISH=LONG
R
A I = (START+FINISH)/2
E ITEM .EQ. TABLE(I)
X       CONTINUE
A LOOKUP = 1
F
X       CONTINUE
C TABLE(I) .LT. ITEM
A START = I+1
I .NOT.(TABLE(I) .LT. ITEM)
X       CONTINUE
F
X       CONTINUE
C TABLE(I) .GT. ITEM
A FINISH = I-1
I .NOT.(TABLE(I) .GT. ITEM)
X       CONTINUE
F
X       CONTINUE
E .NOT.(FINISH-START .GT.1)
X       CONTINUE
C TABLE(START) .EQ. ITEM
X       CONTINUE
A LOOKUP = 1
I .NOT.(TABLE(START) .EQ. ITEM)
X       CONTINUE
C .NOT.(TABLE(FINISH) .EQ. ITEM)
A LOOKUP = 0
X       CONTINUE
I TABLE(FINISH) .EQ. ITEM
X       CONTINUE
A LOOKUP = 1
F
F
F
X       CONTINUE
F
A RETURN
A END
```

Fig. 5.13 - Fichier RTF produit par RFOR.

Ce fichier est à l'état «brut» et demande à être amélioré. C'est le but du programme MAKEUP.

5.2.8 Programme MAKEUP

Comme nous l'avons mentionné au chapitre précédent une amélioration du RTF est souhaitable pour rehausser la présentation du programme. Celle-ci est effectuée par le programme MAKEUP (12).

La figure 5.14 montre le fichier RTF amélioré (13).

```
T ********************************************************
T Programme produit par          : RFOR 4.1
T Nombre de variables introduites : 0
T ********************************************************
A INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
A INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
A START=1
A FINISH=LONG
R
A I = (START+FINISH)/2
E ITEM .EQ. TABLE(I)
A LOOKUP = 1
F
C TABLE(I) .LT. ITEM
A START = I+1
F
C TABLE(I) .GT. ITEM
A FINISH = I-1
F
E .NOT.(FINISH-START .GT.1)
C TABLE(START) .EQ. ITEM
A LOOKUP = 1
I .NOT.(TABLE(FINISH) .EQ. ITEM)
A LOOKUP = 0
I TABLE(FINISH) .EQ. ITEM
A LOOKUP = 1
F
F
F
A RETURN
A END
```

Fig. 5.14 - Fichier RTF amélioré par MAKEUP.

5.2.9 Programme FILTRTF

L'avant-dernière étape consiste à insérer des marques de commentaires opérationnels à l'intérieur du fichier RTF (13) afin qu'il soit accepté par SCHEMACODE (16). En effet, ce dernier doit absolument utiliser des commentaires

opérationnels pour imbriquer des structures et c'est la tâche de FILTRTF (14) de créer ces commentaires lorsqu'ils sont nécessaires. Le texte de ces commentaires n'a toutefois aucune valeur sémantique; c'est à l'opérateur ou au programmeur de les remplacer adéquatement.

Le fichier RTF final (15) est présenté à la figure 5.15:

```
T ***********************************************************
T Programme produit par           : RFOR 4.1
T Nombre de variables introduites : 0
T ***********************************************************
A INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
A INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
A START=1
A FINISH=LONG
R
A I = (START+FINISH)/2
E ITEM .EQ. TABLE(I)
A LOOKUP = 1
F
G .....
Z
C TABLE(I) .LT. ITEM
A START = I+1
F
C TABLE(I) .GT. ITEM
A FINISH = I-1
F
Y
E .NOT.(FINISH-START .GT.1)
G .....
Z
C TABLE(START) .EQ. ITEM
A LOOKUP = 1
I .NOT.(TABLE(FINISH) .EQ. ITEM)
A LOOKUP = 0
I TABLE(FINISH) .EQ. ITEM
A LOOKUP = 1
F
Y
F
F
A RETURN
A END
```

Fig. 5.15 - Fichier RTF final produit par FILTRTF

Notons que FILTRTF prévient l'opérateur si le fichier produit dépasse les limites de SCHEMACODE (trop de commentaires opérationnels ou trop de lignes).

5.2.10 SCHEMACODE

La dernière étape consiste à lire le fichier RTF final à l'aide de la commande d'entrée ASCII de SCHEMACODE (16). Cette opération est simple et directe: elle charge le RTF et le transforme en SPC (17). Une fois chargé, le fichier peut être édité et son code FORTRAN produit (18), tel qu'illustré aux figures 5.16 et 5.17.

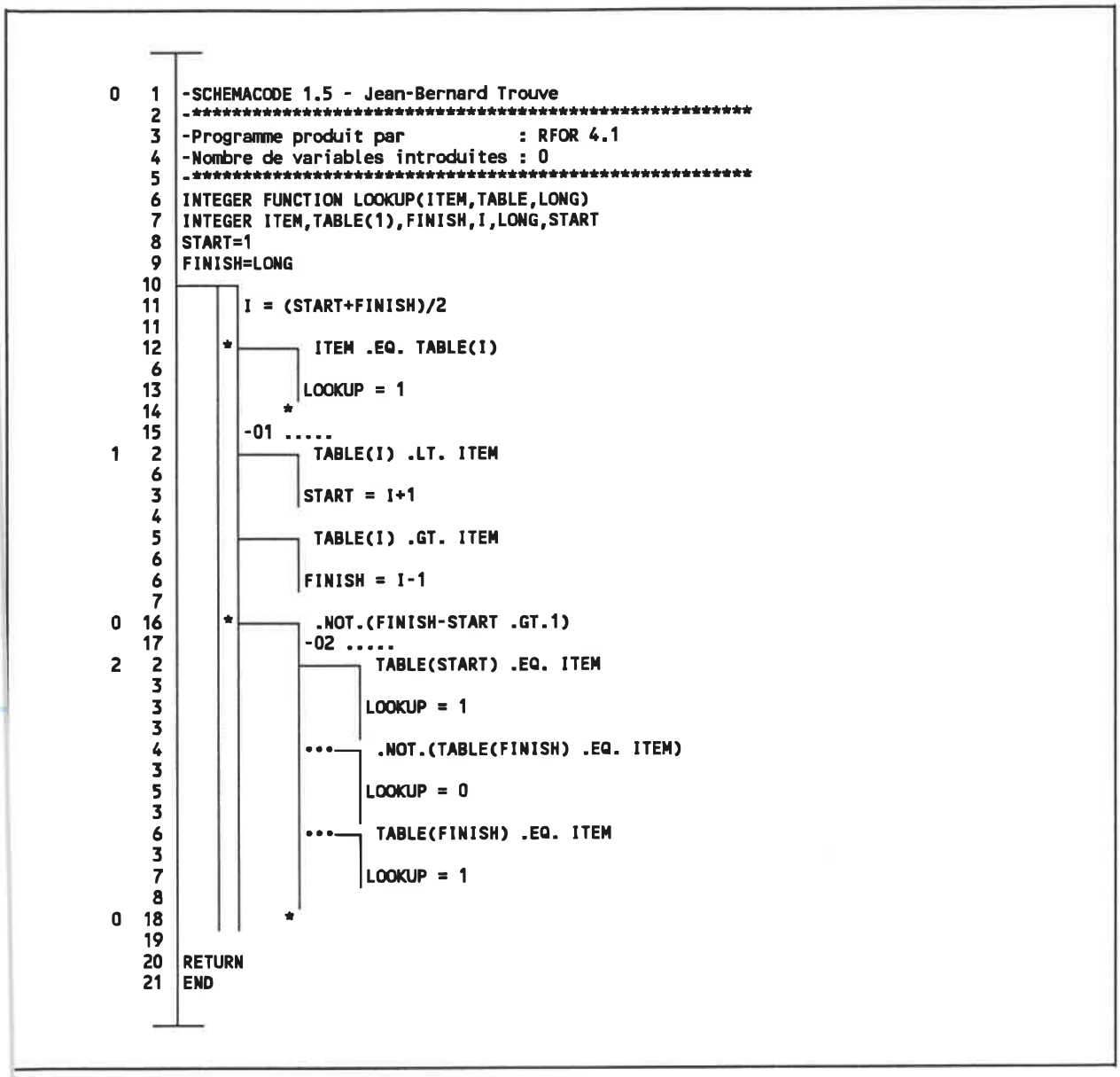


Fig. 5.16 - Programme SPC final.

```
C ------------------------------------------------------------------------
C SCHEMACODE 1.5         Jean-Bernard Trouve        1989-03-08  10:17:03
C
C LOOKUP (FORTRAN 77) created 1989-03-08
C
C Last saved: 1989-03-08  10:16:59
C ------------------------------------------------------------------------
C 00 SCHEMACODE 1.5 - Jean-Bernard Trouve
C *********************************************************
C Programme produit par           : RFOR 4.1
C Nombre de variables introduites : 0
C *********************************************************
      INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
      INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
      START=1
      FINISH=LONG
20000    I = (START+FINISH)/2
         IF (ITEM .EQ. TABLE(I)) THEN
            LOOKUP = 1
            GOTO 20001
         END IF
C 01 .....
         IF ( TABLE(I) .LT. ITEM) THEN
            START = I+1
         END IF
         IF ( TABLE(I) .GT. ITEM) THEN
            FINISH = I-1
         END IF
         IF (.NOT.(FINISH-START .GT.1)) THEN
C 02 .....
            IF ( TABLE(START) .EQ. ITEM) THEN
               LOOKUP = 1
            ELSE IF (.NOT.(TABLE(FINISH) .EQ. ITEM)) THEN
               LOOKUP = 0
            ELSE IF (TABLE(FINISH) .EQ. ITEM) THEN
               LOOKUP = 1
            END IF
            GOTO 20001
         END IF
         GOTO 20000
20001 RETURN
      END
```

Fig. 5.17 - Fichier FORTRAN 77 produit par SCHEMACODE.

5.3 Analyse de performances

Afin d'évaluer les performances réelles du système réalisé, nous présentons à la figure 5.18 un tableau de temps d'exécution de chacun des programmes.

Chaque rangée correspond à un des programmes du système, tandis que les colonnes représentent différentes dimensions de fichier FORTRAN: la première donne les temps d'exécution pour un fichier d'environ 100 lignes et est une somme de temps pour 29 fichiers totalisant environ 5000 lignes.

	1 fichier (100 lignes)	29 fichiers (5000 lignes)
DATRIX	8	530
CRT2ACG	2	42
GBOOL	4	163
FILLACG	3	228
RFOR	4	144
MAKEUP	2	45
FILTRTF	1	34

Fig. 5.18 - Temps d'exécution des programmes du système (en secondes).

NOTES Tous les temps sont en secondes.

Tous les fichiers ont été conservés sur disque dur.

CHAPITRE 6

ANALYSE DES RÉSULTATS

Tell a man there are 300 billion stars in the universe and he'll believe you. Tell him a bench has wet paint on it and he'll have to touch to be sure (Anonyme).

Le prototype décrit au chapitre précédent a été utilisé pour traduire plusieurs programmes FORTRAN en pseudocode schématique. Nous présentons ici quelques uns de ces programmes, chacun ayant des caractéristiques particulières. Les résultats sont ensuites analysés et comparés.

6.1 Présentation des résultats

Il est difficile d'évaluer les résultats produits par le système de restructuration en utilisant un seul programme ou plusieurs programmes pris au hasard. Nous avons donc préféré isoler, parmi tous les programmes analysés, ceux qui présentaient des caractéristiques particulières, mettant ainsi en relief les propriétés de la méthode de transformation.

Dans chaque sous-section suivante, nous présentons d'abord chaque programme en spécifiant son origine, puis nous fournissons son code source original ainsi que les métriques évaluées par DATRIX sur ce source. Nous présentons ensuite le programme traduit en SPC, les métriques évaluées par DATRIX sur le source produit par

SCHEMACODE à partir de ce SPC et finalement les statistiques de traduction produites par le système.

6.1.1 Procédure BOUND

La procédure BOUND, tirée de [44], fait partie du IBM FORTRAN Scientific Subroutine Package. Bien que cela ne saute pas aux yeux, cette procédure écrite en FORTRAN 66 est structurée; l'utilisation d'IF arithmétiques, d'IF GOTO et d'étiquettes la rendent toutefois peu lisible.

La structure est mise en évidence par le programme SPC produit avec le système de traduction. On peut facilement comparer la procédure originale et le SPC pour voir que les deux sont fonctionnellement identiques.

La figure 6.1 illustre le programme source original en FORTRAN.

```
      SUBROUTINE BOUND (A,S,BLO,BHI,UNDER,BETW,OVER,NO,NV,IER)
      DIMENSION A(1),S(1),BLO(1),BHI(1),UNDER(1),BETW(1),OVER(1)
      IER=0
      DO 10 I=1,NV
      IF (BLO(I)-BHI(I)) 10,10,11
11    IER=1
      GO TO 12
10    CONTINUE
      DO 1 K=1,NV
      UNDER(K)=0.0
      BETW(K)=0.0
1     OVER(K)=0.0
      DO 8 J=1,NO
      IJ = J-NO
      IF (S(J)) 2,8,2
2     CONTINUE
      DO 7 I=1,NV
      IJ=IJ+NO
      IF (A(IJ)-BLO(I)) 5,3,3
3     IF (A(IJ)-BHI(I)) 4,4,6
4     BETW(I) = BETW(I) + 1.0
      GO TO 7
5     UNDER(I) = UNDER(I) + 1.0
      GO TO 7
6     OVER(I) = OVER(I) +1.0
7     CONTINUE
8     CONTINUE
12    RETURN
      END
```

Fig. 6.1 - Procédure BOUND originale en FORTRAN.

La figure 6.2 illustre les métriques évaluées par DATRIX sur le programme source original. Les métriques à surveiller, indiquées par une flèche, sont les suivantes:

- le nombre de bris de structure, qui est zéro si le programme est structuré;
- le nombre de chemins indépendants que le flux de contrôle peut emprunter;
- le nombre total de lignes dans le programme, en incluant les commentaires;
- le nombre de lignes exécutables.

```
Routine:  bound
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          4
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)        9.7
Somme des pondérations          (Vw)         20
Vol. de com. des declarations (Vcd)           0
Vol. de com. des structures     (Vcs)         0
Ratio du vol. de commentaires (RVc)           0

Nombre de bris de structure     (Bs)          0   <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)          12
Nombre cyclomatique             (Vg)         13
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants (Np)         111   <--

Ratio structurel de boucle      (Rls)       1.2
Ratio pondéré de boucle         (Rlw)      0.95
Niveau d'imbrication moyen      (Nel)       4.1
Nv d'imbrication maximal    (Nelmax)          8
Nv d'imbrication pondéré      (Nelw)        3.2
Nombre de sommets                (v)         24
Nombre d'arcs                    (e)         35
Nb de sommets Conditionnels     (Ncn)         8
Complexité Moy. des Cond.       (Nc)        9.4
Complexité Max. des Cond.    (Ncmax)         19
Portée moyenne des cond.        (Psc)       5.4
Portée maximum des cond.     (Pscmax)         13
Longueur moy. des noms de var. (Ls)         2.4
Ratio des arcs commentés        (Rac)         0
Ratio des noeuds commentés      (Rnc)         0

Nombre total de lignes          (Nbt)        29   <--
Nombre de lignes exécutables    (Nbe)        29   <--
```

Fig. 6.2 - Métriques évaluées par DATRIX sur le source original.

La figure 6.3 illustre les statistiques produites par RFOR sur la transformation de BOUND en SPC. Elles sont regroupées en plusieurs sections:

- la première section donne le nom du fichier et indique que la structuration est réussie;
- la section GRAPHE donne une mesure du volume du graphe traité;

- la section STANDARDISATION (avant restructuration) donne le détail des opérations de mise en forme standard effectuées avant restructuration;
- la section MESURES donne le pourcentage du nombre d'arcs éliminés avant restructuration par rapport au nombre total d'arcs à éliminer, ce qui évalue le degré de structuration du programme original (ce pourcentage est 100 pour des programmes structurés);
- la section STANDARDISATION (total) détaille les opérations de mise en forme standard effectuées durant le processus complet;
- la section RESTRUCTURATION indique toutes les transformations de restructuration effectuées, de même que le nombre de variables créées et le nombre de lignes dupliquées.

Les valeurs à surveiller sont indiquées par une flèche.

```
RFOR 4.1
Fichier                          :  BOUND
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   35
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :   24
Arcs paralleles                  :    7
Arcs reflexifs                   :    3
Boucles generalisees             :    1
                                   ----
Total                            :   35

MESURES

% arcs elimines avant restruct.  :  100 %   <--
% op. standard. avant restruct.  :  100 %

STANDARDISATION (total)

Sommets proceduraux              :   24
Arcs paralleles                  :    7
Arcs reflexifs                   :    3
Boucles generalisees             :    1
                                   ----
Total                            :   35

RESTRUCTURATION

Transformations ID-0             :    0
Transformations ID-1             :    0
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    0   <--

Nombre de variables creees       :    0   <--
Nombre de lignes dupliquees      :    0   <--
```

Fig. 6.3 - Statistiques sur la transformation de BOUND.

La figure 6.4 illustre le programme SPC résultant.

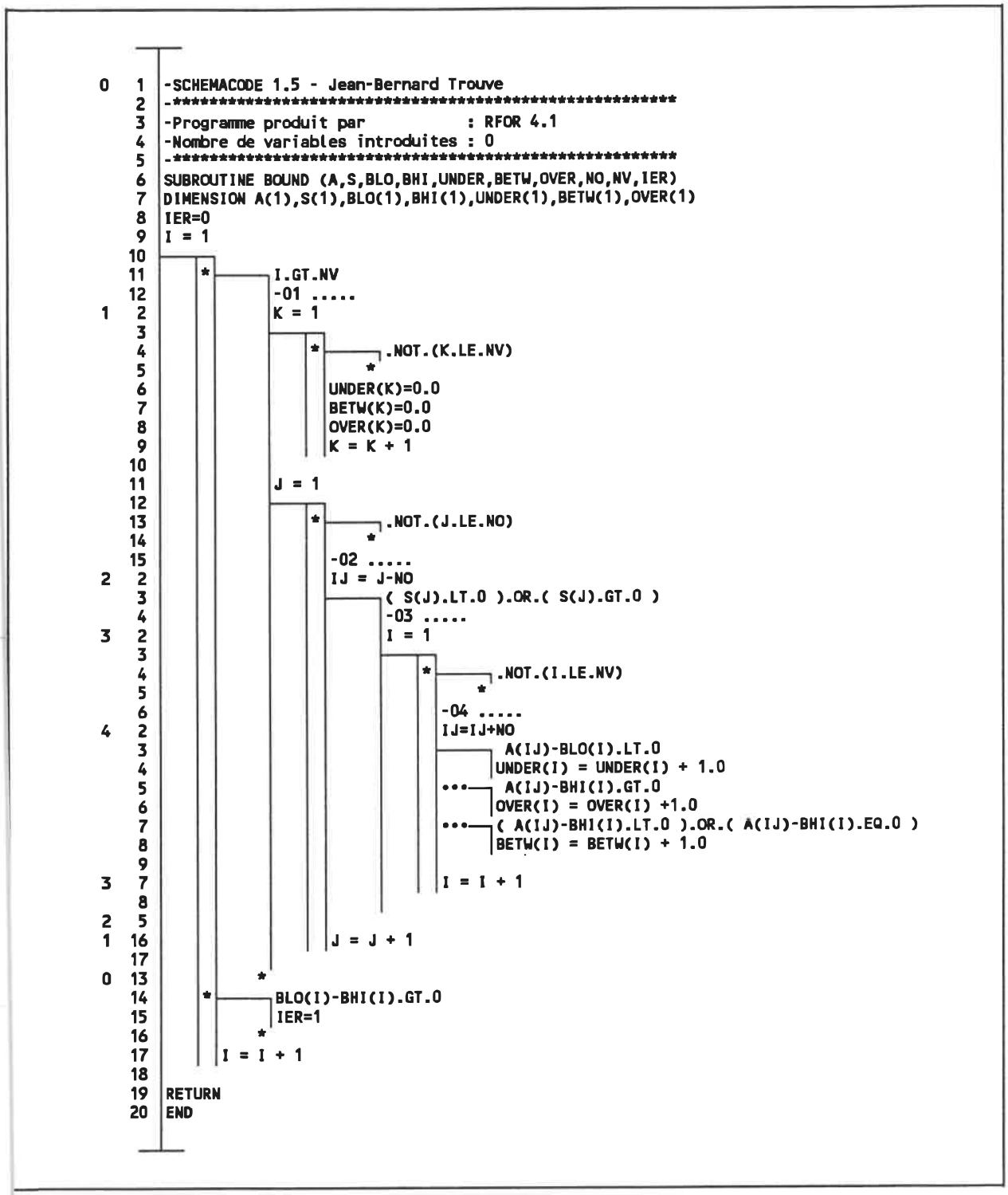


Fig. 6.4 - Procédure BOUND transformée en SPC.

La figure 6.5 illustre évaluées par DATRIX sur le source produit par SCHEMACODE. Remarquer la diminution importante du nombre de chemins indépendants malgré la légère augmentation du nombre de lignes exécutables.

```
Routine:  bound
                                   DATRIX 1.5.4(β)
Tableau récapitulatif              Projet:     ETC


Nom de la métrique                            Valeur

Nombre de sommets d'entrée      (Ni)            1
Nombre de sommets de sortie     (Ne)            1
Nombre de boucles               (Nl)            4
Nombre de sommets récursifs     (Nr)            0
Volume structurel               (Vs)          9.5
Somme des pondérations          (Vw)           20
Vol. de com. des declarations   (Vcd)         180
Vol. de com. des structures     (Vcs)           8
Ratio du vol. de commentaires   (RVc)        0.84

Nombre de bris de structure     (Bs)            0  <--

Nb de bris de str. pondéré      (Bsp)           0
Nombre de croisements des arcs  (K)             7
Nombre cyclomatique             (Vg)           10
Nombre de sommets pendants      (Vp)            0

Nombre de chemins indépendants  (Np)           30  <--

Ratio structurel de boucle      (Rls)         1.7
Ratio pondéré de boucle         (Rlw)         1.6
Niveau d'imbrication moyen      (Nel)         3.5
Nv d'imbrication maximal     (Nelmax)           8
Nv d'imbrication pondéré        (Nelw)        3.3
Nombre de sommets                 (v)          25
Nombre d'arcs                     (e)          33
Nb de sommets Conditionnels     (Ncn)           9
Complexité Moy. des Cond.       (Nc)          4.9
Complexité Max. des Cond.    (Ncmax)           12
Portée moyenne des cond.        (Psc)         5.7
Portée maximum des cond.     (Pscmax)          16
Longueur moy. des noms de var.  (Ls)          2.4
Ratio des arcs commentés        (Rac)         6.7
Ratio des noeuds commentés      (Rnc)           0

Nombre total de lignes          (Nbt)          59  <--
Nombre de lignes exécutables    (Nbe)          42  <--
```

Fig. 6.5 - Métriques évaluées par DATRIX sur le source produit par SCHEMACODE.

6.1.2 Programme POLY

Ce programme FORTRAN a été produit par SCHEMACODE à partir d'une description en pseudocode schématique. Conséquemment, il est intéressant de comparer le programme SPC original et le programme SPC reconstruit. On constate que le système est effectivement capable de retrouver la structure originale du programme.

La figure 6.6 illustre le programme original en SPC.

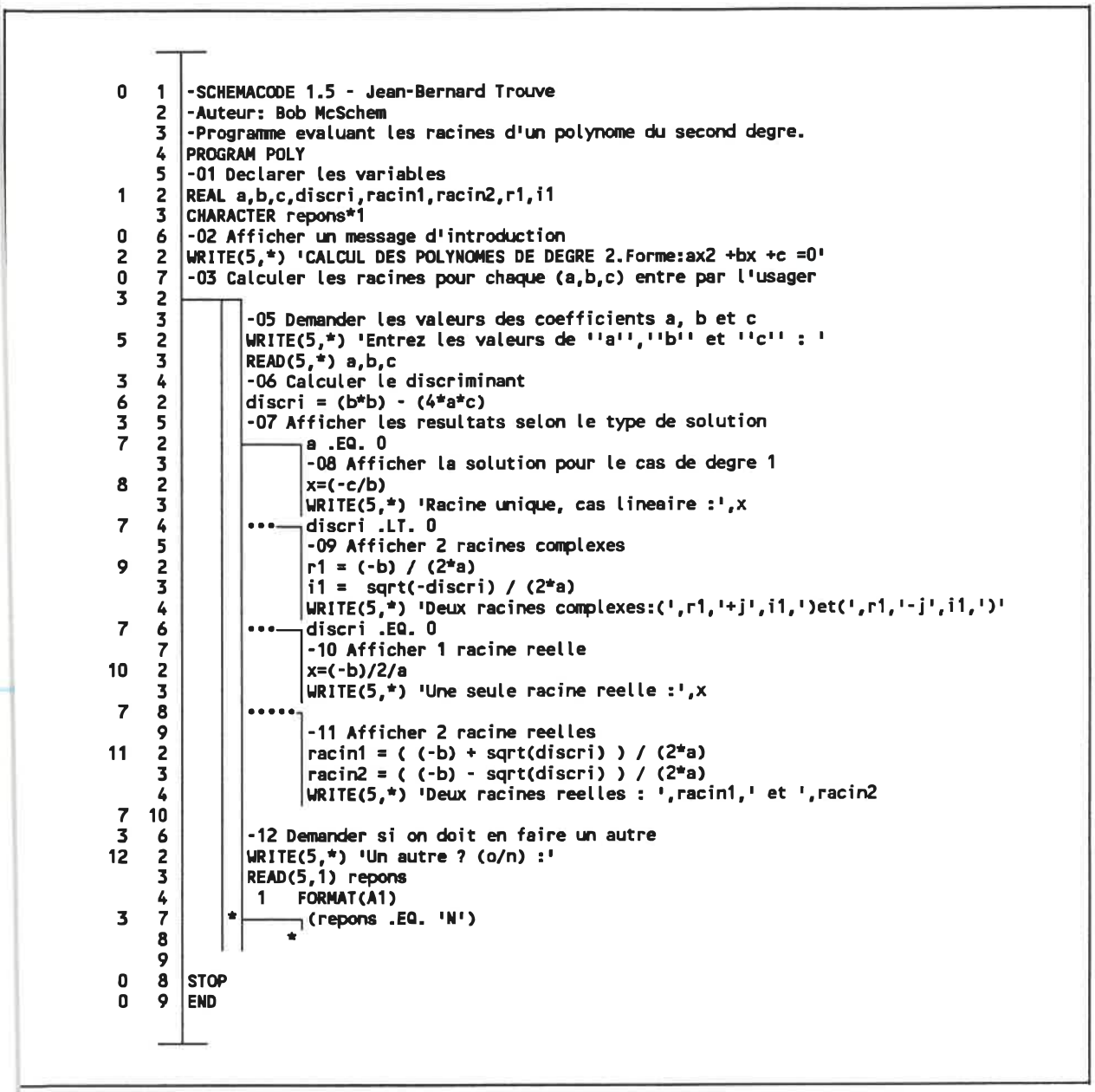


Fig. 6.6 - Programme POLY original en SPC.

La figure 6.7 illustre le programme POLY codé en FORTRAN par SCHEMACODE.

```
C ---------------------------------------------------------------------
C SCHEMACODE 1.5         Jean-Bernard Trouve         1989-02-09  20:28:11
C
C POLY (FORTRAN 66) created 1989-02-08
C
C Last saved: 1989-02-08  18:48:32
C ---------------------------------------------------------------------
C 00 SCHEMACODE 1.5 - Jean-Bernard Trouve
C Auteur: Bob McSchem
C Programme evaluant les racines d'un polynome du second degre.
      PROGRAM POLY
C 01 Declarer les variables
      REAL a,b,c,discri,racin1,racin2,r1,i1
      CHARACTER repons*1
C 02 Afficher un message d'introduction
      WRITE(5,*) 'CALCUL DES POLYNOMES DE DEGRE 2.Forme:ax2 +bx +c =0'
C 03 Calculer les racines pour chaque (a,b,c) entre par l'usager
C 05 Demander les valeurs des coefficients a, b et c
20000 WRITE(5,*) 'Entrez les valeurs de ''a'',''b'' et ''c'' : '
      READ(5,*) a,b,c
C 06 Calculer le discriminant
      discri = (b*b) - (4*a*c)
C 07 Afficher les resultats selon le type de solution
      IF(.NOT.(a .EQ. 0))GOTO20003
C 08 Afficher la solution pour le cas de degre 1
         x=(-c/b)
         WRITE(5,*) 'Racine unique, cas lineaire :',x
         GOTO20002
20003 IF(.NOT.(discri .LT. 0))GOTO20004
C 09 Afficher 2 racines complexes
         r1 = (-b) / (2*a)
         i1 =  sqrt(-discri) / (2*a)
         WRITE(5,*) 'Deux racines complexes:(',r1,'+j',i1,')et(',r1,'-j'
     +,i1,')'
         GOTO20002
20004 IF(.NOT.(discri .EQ. 0))GOTO20005
C 10 Afficher 1 racine reelle
         x=(-b)/2/a
         WRITE(5,*) 'Une seule racine reelle :',x
         GOTO 20002
C 11 Afficher 2 racine reelles
20005    racin1 = ( (-b) + sqrt(discri) ) / (2*a)
         racin2 = ( (-b) - sqrt(discri) ) / (2*a)
         WRITE(5,*) 'Deux racines reelles : ',racin1,' et ',racin2
C 12 Demander si on doit en faire un autre
20002 WRITE(5,*) 'Un autre ? (o/n) :'
      READ(5,1) repons
1        FORMAT(A1)
      IF((repons .EQ. 'N'))GOTO20001
      GOTO 20000
20001 STOP
      END
```

Fig. 6.7 - Programme POLY codé en FORTRAN par SCHEMACODE.

La figure 6.8 illustre programme POLY retransformé en SPC. Remarquer la similitude avec celui de la figure 6.6.

```
0   1  -SCHEMACODE 1.5 - Jean-Bernard Trouve
    2  -************************************************************
    3  -Programme produit par           : RFOR 4.1
    4  -Nombre de variables introduites : 0
    5  -************************************************************
    6  - ---------------------------------------------------------------------
    7  - SCHEMACODE 1.5         Jean-Bernard Trouve        1989-02-09  20:28:11
    8  -
    9  - POLY (FORTRAN 66) created 1989-02-08
   10  -
   11  - Last saved: 1989-02-08  18:48:32
   12  - ---------------------------------------------------------------------
   13  - 00 SCHEMACODE 1.5 - Jean-Bernard Trouve
   14  - Auteur: Bob McSchem
   15  - Programme evaluant les racines d'un polynome du second degre.
   16  PROGRAM POLY
   17  - 01 Declarer les variables
   18  REAL a,b,c,discri,racin1,racin2,r1,i1
   19  CHARACTER repons*1
   20  - 02 Afficher un message d'introduction
   21  WRITE(5,*) 'CALCUL DES POLYNOMES DE DEGRE 2.Forme:ax2 +bx +c =0'
   22  - 03 Calculer les racines pour chaque (a,b,c) entre par l'usager
   23  - 05 Demander les valeurs des coefficients a, b et c
   24  ┌──┐
   25  │  │ WRITE(5,*) 'Entrez les valeurs de ''a'',''b'' et ''c'' : '
   26  │  │ READ(5,*) a,b,c
   27  │  │ - 06 Calculer le discriminant
   28  │  │ discri = (b*b) - (4*a*c)
   29  │  │ - 07 Afficher les resultats selon le type de solution
   30  │  │ -01 .....
 1 2  │  │ ┌──────a .EQ. 0
   3  │  │ │      - 08 Afficher la solution pour le cas de degre 1
   4  │  │ │         x=(-c/b)
   5  │  │ │         WRITE(5,*) 'Racine unique, cas lineaire :',x
   6  │  │ ••••─discri .LT. 0
   7  │  │ │      - 09 Afficher 2 racines complexes
   8  │  │ │         r1 = (-b) / (2*a)
   9  │  │ │         i1 =  sqrt(-discri) / (2*a)
  10  │  │ │         WRITE(5,*) 'Deux racines complexes:(',r1,'+j',i1,')et(',r1,'-j',i1,')'
  11  │  │ ••••─discri .EQ. 0
  12  │  │ │      - 10 Afficher 1 racine reelle
  13  │  │ │         x=(-b)/2/a
  14  │  │ │         WRITE(5,*) 'Une seule racine reelle :',x
  15  │  │ •••••┐
  16  │  │ │    │    racin1 = ( (-b) + sqrt(discri) ) / (2*a)
  17  │  │ │    │    racin2 = ( (-b) - sqrt(discri) ) / (2*a)
  18  │  │ │    │    WRITE(5,*) 'Deux racines reelles : ',racin1,' et ',racin2
  19  │  │ │    └─ - 12 Demander si on doit en faire un autre
  20  │  │ │
  21  │  │ WRITE(5,*) 'Un autre ? (o/n) :'
  22  │  │ READ(5,1) repons
  23  │  │ 1         FORMAT(A1)
 0 31 │  * ┌──────(repons .EQ. 'N')
  32  │    *
  33  │
  34  STOP
  35  END
```

Fig. 6.8 - Programme POLY retransformé en SPC.

La figure 6.9 illustre les statistiques sur la transformation de POLY. Noter qu'aucune restructuration n'est nécessaire car le source original était produit à partir d'un programmme en SPC.

```
RFOR 4.1
Fichier                          :  POLY
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   18
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :   12
Arcs paralleles                  :    3
Arcs reflexifs                   :    0
Boucles generalisees             :    1
                                   ----
Total                            :   16

MESURES

% arcs elimines avant restruct.  :  100 %   <--
% op. standard. avant restruct.  :  100 %

STANDARDISATION (total)

Sommets proceduraux              :   12
Arcs paralleles                  :    3
Arcs reflexifs                   :    0
Boucles generalisees             :    1
                                   ----
Total                            :   16

RESTRUCTURATION

Transformations ID-0             :    0
Transformations ID-1             :    0
Transformations OO               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    0   <--

Nombre de variables creees       :    0   <--
Nombre de lignes dupliquees      :    0   <--
```

Fig. 6.9 - Statistiques sur la transformation de POLY.

6.1.3 Procédure LOOKUP

Cette procédure, qui effectue une recherche binaire classique dans un vecteur de nombre entiers, est tirée de [49]. Elle possède la caractéristique d'être non-structurée, quoique faiblement (la valeur de la métrique Bs n'est pas zéro).

La source de non-structure réside dans une forme ID dont le sommet central se situe à l'énoncé LOOKUP=1. Deux possibilités s'offrent alors: restructurer par ID-1, c'est à dire sans dupliquer de code mais en créant des variables de contrôle, ou par ID-0 en dupliquant du code. Ces deux possibilités sont explorées en spécifiant au système d'utiliser exclusivement ID-1 dans le premier cas, puis exclusivement ID-0 dans le second.

Les résultats tendent, dans ce cas précis, à favoriser la duplication de code car cette solution produit un programme plus lisible et de plus faibles dimensions.

La figure 6.10 illustre la procédure originale en FORTRAN.

```
      INTEGER FUNCTION LOOKUP(ITEM,TABLE,LONG)
      INTEGER ITEM,TABLE(1),FINISH,I,LONG,START
      START=1
      FINISH=LONG
1     I = (START+FINISH)/2
      IF (ITEM .EQ. TABLE(I)) GOTO 2
      IF (TABLE(I) .LT. ITEM) START = I+1
      IF (TABLE(I) .GT. ITEM) FINISH = I-1
      IF (FINISH-START .GT.1) GOTO 1
      IF (TABLE(START) .EQ. ITEM) GOTO 2
      IF (TABLE(FINISH) .EQ. ITEM) GOTO 2
      LOOKUP = 0
      GOTO 3
2     LOOKUP = 1
3     RETURN
      END
```

Fig. 6.10 - Procédure LOOKUP originale en FORTRAN.

La figure 6.11 illustre les métriques évaluées par DATRIX sur le source original. Remarquer que ce dernier n'est pas structuré (il y a 3 bris de structures).

```
Routine:  lookup
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          1
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)        5.7
Somme des pondérations          (Vw)          9
Vol. de com. des declarations (Vcd)           0
Vol. de com. des structures     (Vcs)         0
Ratio du vol. de commentaires (RVc)           0

Nombre de bris de structure     (Bs)          3   <--

Nb de bris de str. pondéré     (Bsp)          0
Nombre de croisements des arcs  (K)           5
Nombre cyclomatique             (Vg)          7
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants (Np)          65   <--

Ratio structurel de boucle     (Rls)       0.47
Ratio pondéré de boucle        (Rlw)       0.33
Niveau d'imbrication moyen     (Nel)          3
Nv d'imbrication maximal    (Nelmax)          4
Nv d'imbrication pondéré      (Nelw)        2.2
Nombre de sommets                (v)         15
Nombre d'arcs                    (e)         20
Nb de sommets Conditionnels    (Ncn)          6
Complexité Moy. des Cond.       (Nc)          3
Complexité Max. des Cond.    (Ncmax)          3
Portée moyenne des cond.       (Psc)        3.7
Portée maximum des cond.    (Pscmax)          9
Longueur moy. des noms de var. (Ls)         4.4
Ratio des arcs commentés       (Rac)          0
Ratio des noeuds commentés     (Rnc)          0

Nombre total de lignes         (Nbt)         16   <--
Nombre de lignes exécutables   (Nbe)         16   <--
```

Fig. 6.11 - Métriques évaluées par DATRIX sur le source original.

La figure 6.12 illustre la procédure LOOKUP transformée en SPC sans duplication de code.

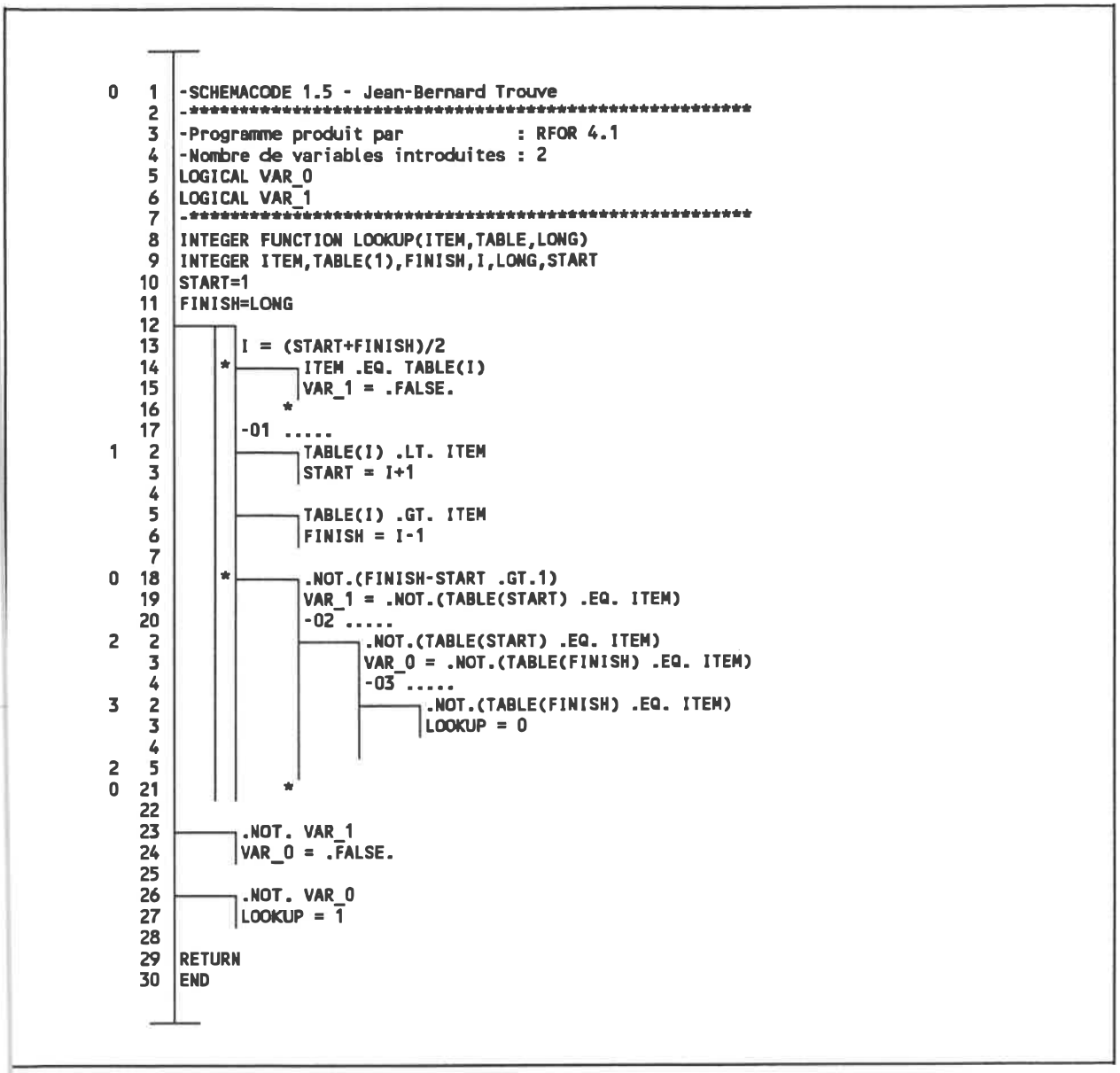


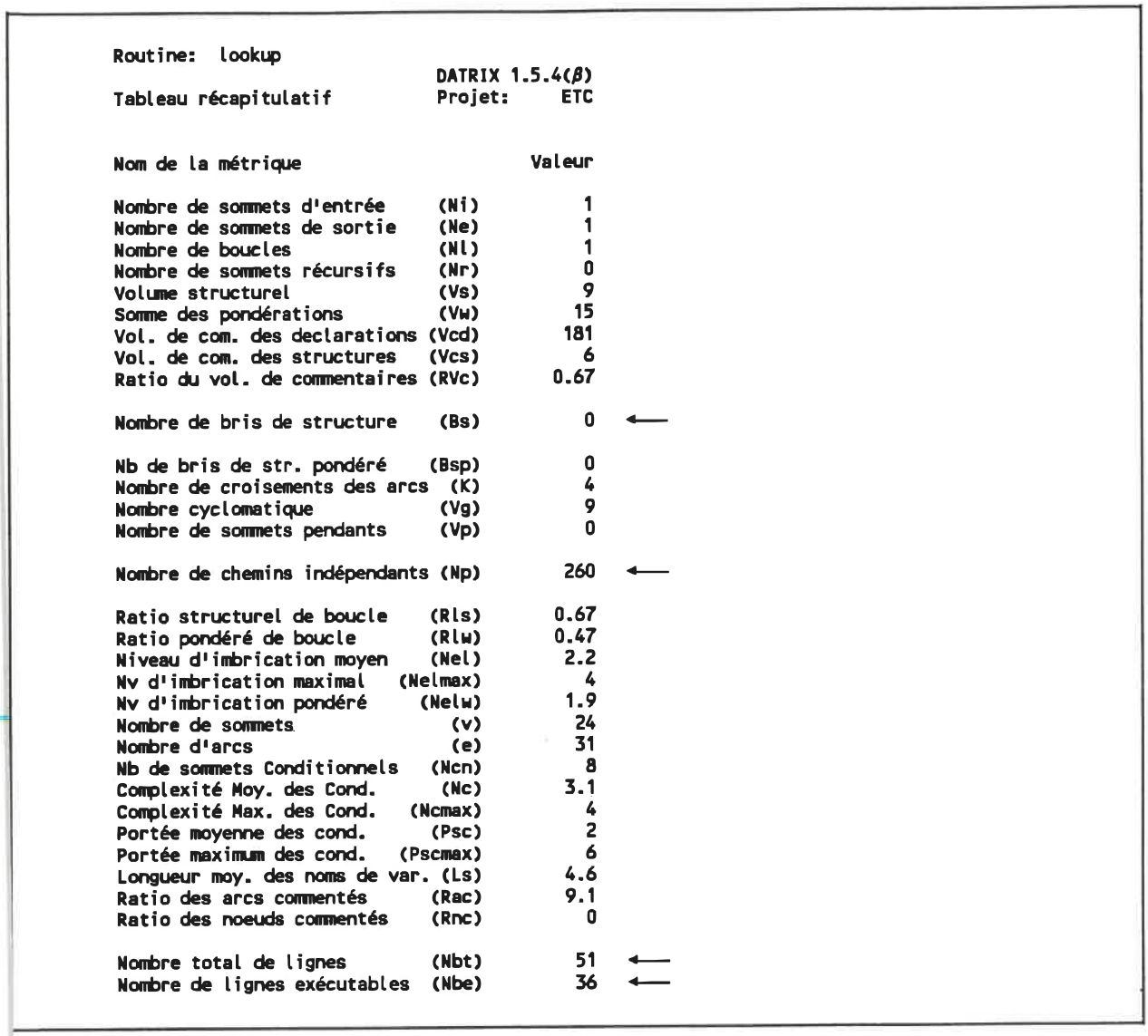
Fig. 6.12 - Procédure LOOKUP transformée en SPC (sans duplication de code).

La figure 6.13 illustre les statistiques sur la transformation de LOOKUP sans duplication de code. Remarquer le nombre de variables de contrôle créées (2).

```
RFOR 4.1

Duplication de code interdite.

Fichier                          :  LOOKUP
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   20
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :    6
Arcs paralleles                  :    2
Arcs reflexifs                   :    0
Boucles generalisees             :    0
                                   ----
Total                            :    8

MESURES

% arcs elimines avant restruct.  :   42 %   <----
% op. standard. avant restruct.  :   42 %

STANDARDISATION (total)

Sommets proceduraux              :   12
Arcs paralleles                  :    6
Arcs reflexifs                   :    0
Boucles generalisees             :    1
                                   ----
Total                            :   19

RESTRUCTURATION

Transformations ID-0             :    0
Transformations ID-1             :    2
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    2   <----

Nombre de variables creees       :    2   <----
Nombre de lignes dupliquees      :    0   <----
```

Fig. 6.13 - Statistiques sur la transformation de LOOKUP (sans duplication de code).

La figure 6.14 illustre les métriques évaluées par DATRIX sur le source produit par SCHEMACODE (sans duplication). Remarquer qu'il n'y a plus de bris de structure mais que le nombre de chemins indépendants et d'énoncés exécutables a augmenté (comparer avec la figure 6.11).



```
Routine:  lookup
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          1
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)          9
Somme des pondérations          (Vw)         15
Vol. de com. des declarations   (Vcd)       181
Vol. de com. des structures     (Vcs)         6
Ratio du vol. de commentaires   (RVc)      0.67

Nombre de bris de structure     (Bs)          0   <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)           4
Nombre cyclomatique             (Vg)          9
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants  (Np)        260   <--

Ratio structurel de boucle      (Rls)      0.67
Ratio pondéré de boucle         (Rlw)      0.47
Niveau d'imbrication moyen      (Nel)       2.2
Nv d'imbrication maximal        (Nelmax)      4
Nv d'imbrication pondéré        (Nelw)      1.9
Nombre de sommets               (v)          24
Nombre d'arcs                   (e)          31
Nb de sommets Conditionnels     (Ncn)         8
Complexité Moy. des Cond.       (Nc)        3.1
Complexité Max. des Cond.       (Ncmax)       4
Portée moyenne des cond.        (Psc)         2
Portée maximum des cond.        (Pscmax)      6
Longueur moy. des noms de var.  (Ls)        4.6
Ratio des arcs commentés        (Rac)       9.1
Ratio des noeuds commentés      (Rnc)         0

Nombre total de lignes          (Nbt)        51   <--
Nombre de lignes exécutables    (Nbe)        36   <--
```

Fig. 6.14 - Métriques évaluées par DATRIX sur le source produit par SCHEMACODE (sans duplication).

La figure 6.15 illustre la procédure LOOKUP transformée en SPC avec duplication de code.

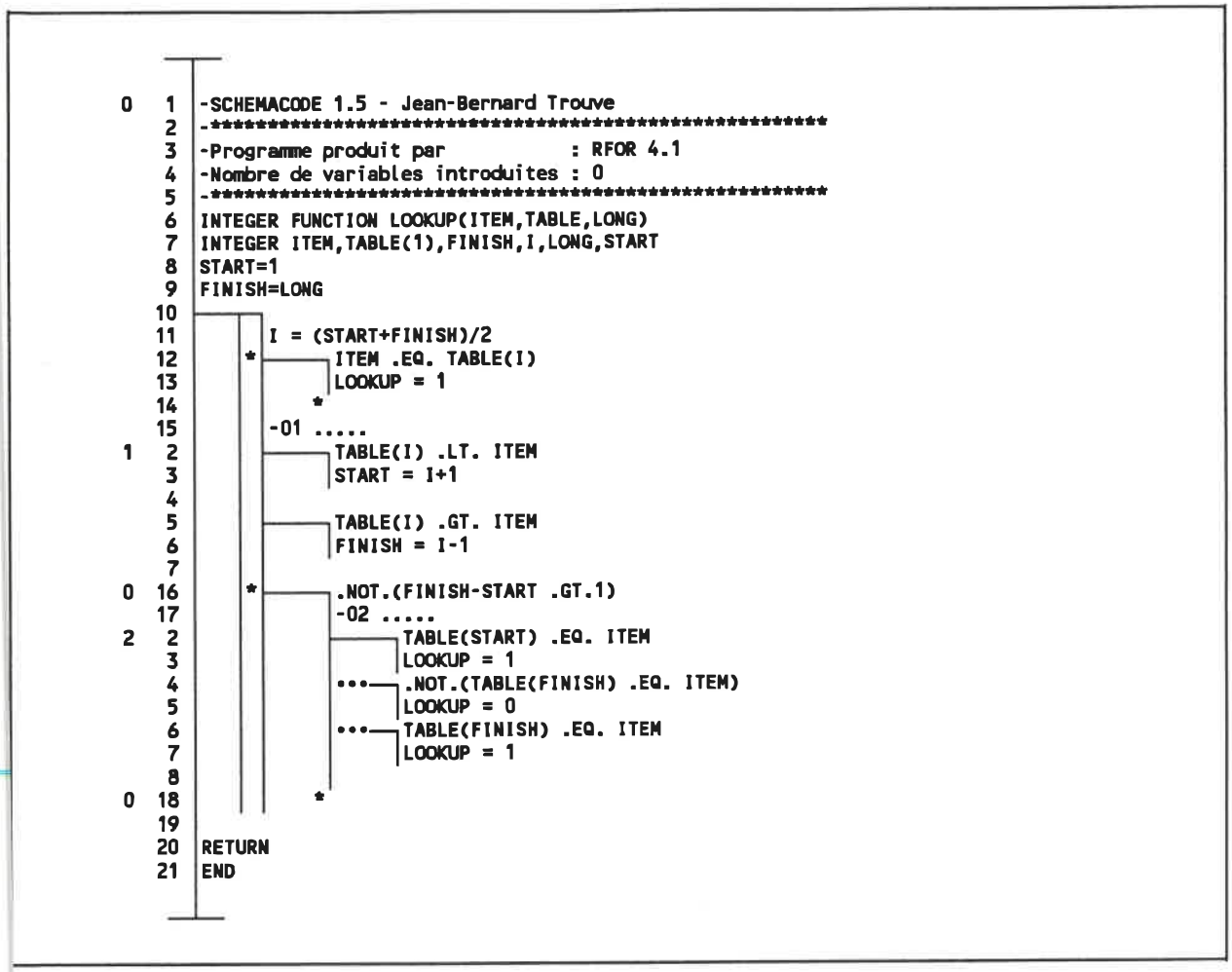


Fig. 6.15 - Procédure LOOKUP transformée en SPC (avec duplication de code).

La figure 6.16 illustre les statistiques sur la transformation de LOOKUP avec duplication de code. Remarquer le nombre de lignes dupliquées (2).

```
RFOR 4.1

Duplication de code toujours utilisee si possible.

Fichier                          :  LOOKUP
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   20
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :    6
Arcs paralleles                  :    2
Arcs reflexifs                   :    0
Boucles generalisees             :    0
                                   ----
Total                            :    8

MESURES

% arcs elimines avant restruct.  :   42 %   <----
% op. standard. avant restruct.  :   53 %

STANDARDISATION (total)

Sommets proceduraux              :   10
Arcs paralleles                  :    4
Arcs reflexifs                   :    0
Boucles generalisees             :    1
                                   ----
Total                            :   15

RESTRUCTURATION

Transformations ID-0             :    1
Transformations ID-1             :    0
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    1   <----

Nombre de variables creees       :    0   <----
Nombre de lignes dupliquees      :    2   <----
```

Fig. 6.16 - Statistiques sur la transformation de LOOKUP (avec duplication de code).

La figure 6.17 illustre les métriques évaluées par DATRIX sur le source produit par SCHEMACODE (avec duplication). Remarquer que les bris de structure sont éliminés; le nombre de chemins indépendants et le nombre d'énoncés ont augmenté, mais beaucoup moins que lorsqu'on évite la duplication de code (comparer avec la figure 6.14).

```
Routine:  lookup
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          1
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)        7.5
Somme des pondérations          (Vw)         11
Vol. de com. des declarations (Vcd)         181
Vol. de com. des structures     (Vcs)         4
Ratio du vol. de commentaires (RVc)        0.53

Nombre de bris de structure     (Bs)          0  <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)           4
Nombre cyclomatique             (Vg)          8
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants (Np)          85  <--

Ratio structurel de boucle      (Rls)       0.82
Ratio pondéré de boucle         (Rlw)       0.64
Niveau d'imbrication moyen      (Nel)        2.5
Nv d'imbrication maximal     (Nelmax)          5
Nv d'imbrication pondéré       (Nelw)        2.4
Nombre de sommets                 (v)         20
Nombre d'arcs                     (e)         26
Nb de sommets Conditionnels     (Ncn)          7
Complexité Moy. des Cond.       (Nc)         3.3
Complexité Max. des Cond.    (Ncmax)           4
Portée moyenne des cond.        (Psc)        2.3
Portée maximum des cond.     (Pscmax)          6
Longueur moy. des noms de var. (Ls)         4.4
Ratio des arcs commentés        (Rac)         11
Ratio des noeuds commentés      (Rnc)          0

Nombre total de lignes          (Nbt)         42  <--
Nombre de lignes exécutables    (Nbe)         28  <--
```

Fig. 6.17 - Métriques évaluées par DATRIX sur le source produit par SCHEMACODE (avec duplication).

6.1.4 Procédure PRINTEST

Cette procédure, comme la précédente, est aussi non-structurée. Dans un premier temps, on évite la duplication de code, puis on la permet sans limiter le

nombre de lignes dupliquées. Les résultats ont pour but de montrer que la stratégie de duplication n'est pas toujours la meilleure.

La figure 6.18 illustre le programme original en FORTRAN.

```
      PROGRAM PRINTEST
C  PRINTER/PLOTTER DRIVER TEST PROGRAM
C
      COMMON /IOCOM/ IUNIT,LUNIT,LREC,IOTYPE
C
C
      INTEGER*2 IPRINT(448),IPLOT(896)
      INTEGER FORMFD,EOT
C
C     DATA IPRINT/448*'AZ'/,IPLOT/896*-256/
C     DATA FORMFD/0/,EOT/2/
C     DATA LUNIT/5/
C
    1 FORMAT (' Printer/Plotter type?'//,
     X          ' Printer only = PR'/,
     X          ' Plotter only = PL'/,
     X          ' Printer/Plotter = PP'//,
     X          '$Enter Printer/Plotter type: ')
    2 FORMAT (A2)
    3 FORMAT ('$Enter columns/line = ')
    4 FORMAT (I3)
    5 FORMAT ('$Enter number of bytes/scan = ')
   10 FORMAT (' END OF DRIVER TEST')
C
      IOTYPE = 0
      CALL ATTACH
      TYPE 1
      ACCEPT 2,TYPE
      IF (TYPE.EQ.'PL') GO TO 200
C
C...  GET PRINTER CHARACTERISTICS
      TYPE 3
      ACCEPT 4,NCHARS
      IF (TYPE.EQ.'PR') GO TO 300
C
C...  GET PLOTTER CHARACTERISTICS
  200 TYPE 5
      ACCEPT 4,NBYTES
      IF (TYPE.EQ.'PL') GO TO 500
C
C...  PRINT TEST
  300 IOTYPE = 0
      DO 400 I=1,10
      CALL OUTPUT (IPRINT,NCHARS)
  400 CONTINUE
      CALL OUTPUT (FORMFD,0)
      IF (TYPE.EQ.'PR') GO TO 999
C
C...  PLOT TEST
  500 IOTYPE = 1
      DO 600 I=1,200
      CALL OUTPUT (IPLOT,NBYTES)
  600 CONTINUE
      CALL OUTPUT (FORMFD,0)
      IF (TYPE.EQ.'PL') GO TO 999
C
C...  PRINT/PLOT TEST
      DO 700 I=1,10
      IOTYPE = 0
      CALL OUTPUT (IPRINT,NCHARS)
      IOTYPE = 1
      DO 700 J=1,20
      CALL OUTPUT (IPLOT,NBYTES)
  700 CONTINUE
      CALL OUTPUT (FORMFD,0)
C
  999 CALL DETACH
```

```
      STOP
      END
```

Fig. 6.18 - Procédure PRINTEST originale en FORTRAN.

La figure 6.19 illustre les métriques évaluées par DATRIX sur le source original.

```
Routine:  printest
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          4
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)        8.2
Somme des pondérations          (Vw)         38
Vol. de com. des declarations   (Vcd)        82
Vol. de com. des structures     (Vcs)        80
Ratio du vol. de commentaires   (RVc)       9.8

Nombre de bris de structure     (Bs)          3   <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)           7
Nombre cyclomatique             (Vg)         10
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants  (Np)         70   <--

Ratio structurel de boucle      (Rls)      0.43
Ratio pondéré de boucle         (Rlw)      0.34
Niveau d'imbrication moyen      (Nel)       2.8
Nv d'imbrication maximal    (Nelmax)          5
Nv d'imbrication pondéré       (Nelw)       2.2
Nombre de sommets                (v)        21
Nombre d'arcs                    (e)        29
Nb de sommets Conditionnels     (Ncn)         9
Complexité Moy. des Cond.        (Nc)         1
Complexité Max. des Cond.     (Ncmax)         1
Portée moyenne des cond.        (Psc)       3.8
Portée maximum des cond.     (Pscmax)         9
Longueur moy. des noms de var.  (Ls)        4.6
Ratio des arcs commentés        (Rac)        11
Ratio des noeuds commentés      (Rnc)        11

Nombre total de lignes          (Nbt)        70   <--
Nombre de lignes exécutables    (Nbe)        44   <--
```

Fig. 6.19 - Métriques évaluées par DATRIX sur le source original.

La figure 6.20 illustre la procédure PRINTEST transformée en SPC sans duplication de code.

```
    1  -SCHEMACODE 1.5 - Jean-Bernard Trouve
    2  -*********************************************************
    3  -Programme produit par           : RFOR 4.1
    4  -Nombre de variables introduites : 3
    5  LOGICAL VAR_0
    6  LOGICAL VAR_1
    7  LOGICAL VAR_2
    8  -*********************************************************
    9  PROGRAM PRINTEST
   10  -  PRINTER/PLOTTER DRIVER TEST PROGRAM
   11  -
   12  COMMON /IOCOM/ IUNIT,LUNIT,LREC,IOTYPE
   13  -
   14  -
   15  INTEGER*2 IPRINT(448),IPLOT(896)
   16  INTEGER FORMFD,EOT
   17  -
   18  -     DATA IPRINT/448*'AZ'/,IPLOT/896*-256/
   19  -     DATA FORMFD/0/,EOT/2/
   20  -     DATA LUNIT/5/
   21  -
   22      1 FORMAT (' Printer/Plotter type?'//,          '
   23  ... Printer only = PR'/,          ' Plotter only = PL'/,
   24  ...       ' Printer/Plotter = PP'//,          '$Enter Printer/Plotter type: ')
   25      2 FORMAT (A2)
   26      3 FORMAT ('$Enter columns/line = ')
   27      4 FORMAT (I3)
   28      5 FORMAT ('$Enter number of bytes/scan = ')
   29     10 FORMAT (' END OF DRIVER TEST')
   30  -
   31  IOTYPE = 0
   32  CALL ATTACH
   33  -
   34  TYPE 1
   35  ACCEPT 2,TYPE
   36  ------ TYPE.EQ.'PL'
   37        VAR_2 = .FALSE.
   38  *****-
   39        -
   40        -...  GET PRINTER CHARACTERISTICS
   41        TYPE 3
   42        ACCEPT 4,NCHARS
   43        VAR_2 = TYPE.EQ.'PR'
   44        -01 .....
    2  ------TYPE.EQ.'PR'
    3  ------VAR_1 = .FALSE.
    4
   45
   46  ------ .NOT. VAR_2
   47        TYPE 5
   48        ACCEPT 4,NBYTES
   49        VAR_1 = TYPE.EQ.'PL'
   50        -02 .....
    2  ------TYPE.EQ.'PL'
    3  ------VAR_0 = .FALSE.
    4
   51
   52  ------ .NOT. VAR_1
   53        IOTYPE = 0
   54        I = 1
   55        -03 .....
    2  ------
    3  ------*------.NOT.(I.LE.10)
    4  ------      *
    5  ------CALL OUTPUT (IPRINT,NCHARS)
```

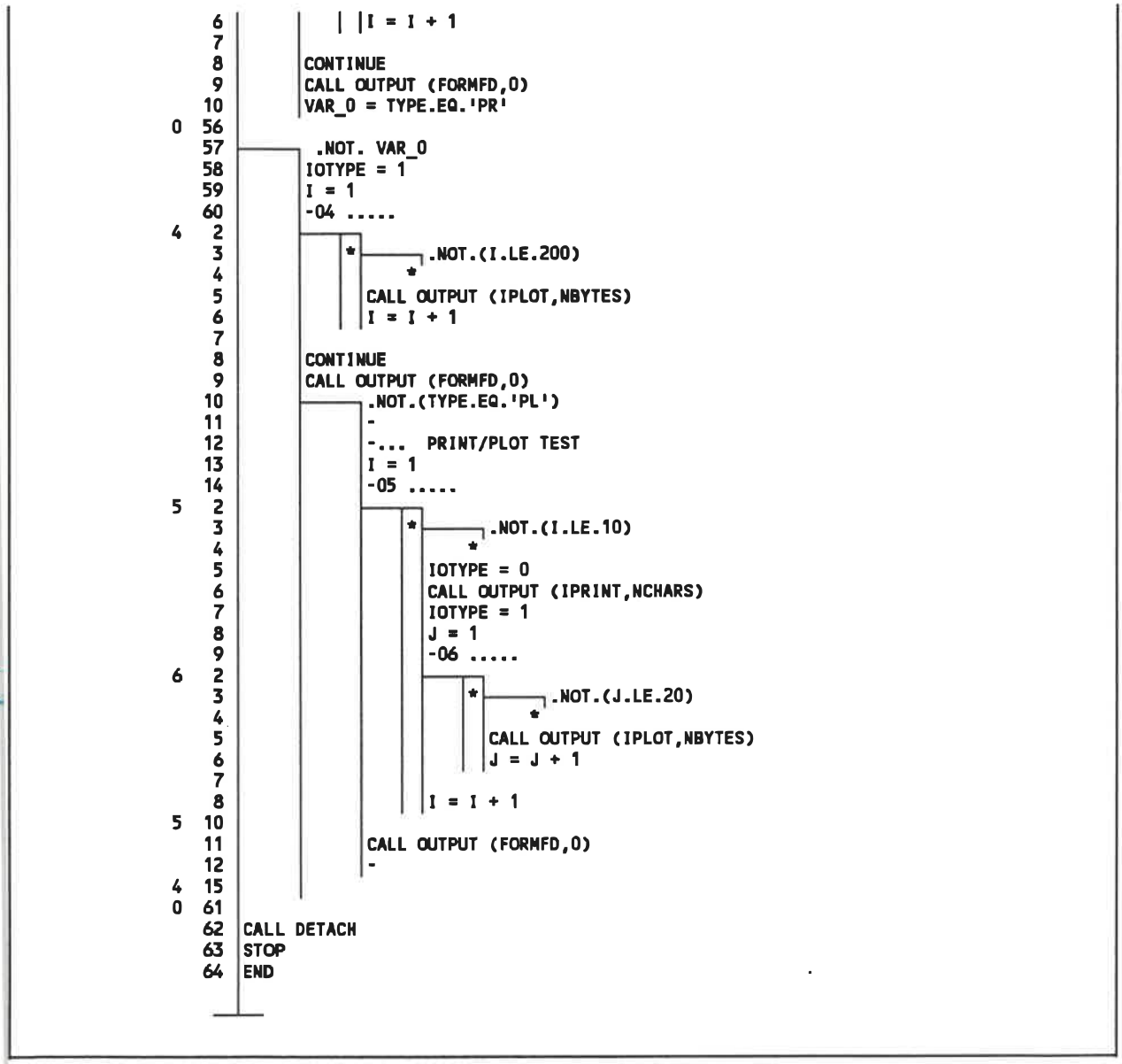


Fig. 6.20 - Procédure PRINTEST transformée en SPC sans duplication.

La figure 6.21 illustre les statistiques sur la transformation de PRINTEST sans duplication de code. Remarquer le nombre de variables de contrôle créées (3).

```
RFOR 4.1

Duplication de code interdite.

Fichier                          :  PRINTEST
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   29
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :   15
Arcs paralleles                  :    1
Arcs reflexifs                   :    4
Boucles generalisees             :    0
                                   ----
Total                            :   20

MESURES

% arcs elimines avant restruct.  :   57 %   <--
% op. standard. avant restruct.  :   57 %

STANDARDISATION (total)

Sommets proceduraux              :   23
Arcs paralleles                  :    8
Arcs reflexifs                   :    4
Boucles generalisees             :    0
                                   ----
Total                            :   35

RESTRUCTURATION

Transformations ID-0             :    0
Transformations ID-1             :    3
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    3   <--

Nombre de variables creees       :    3   <--
Nombre de lignes dupliquees      :    0   <--
```

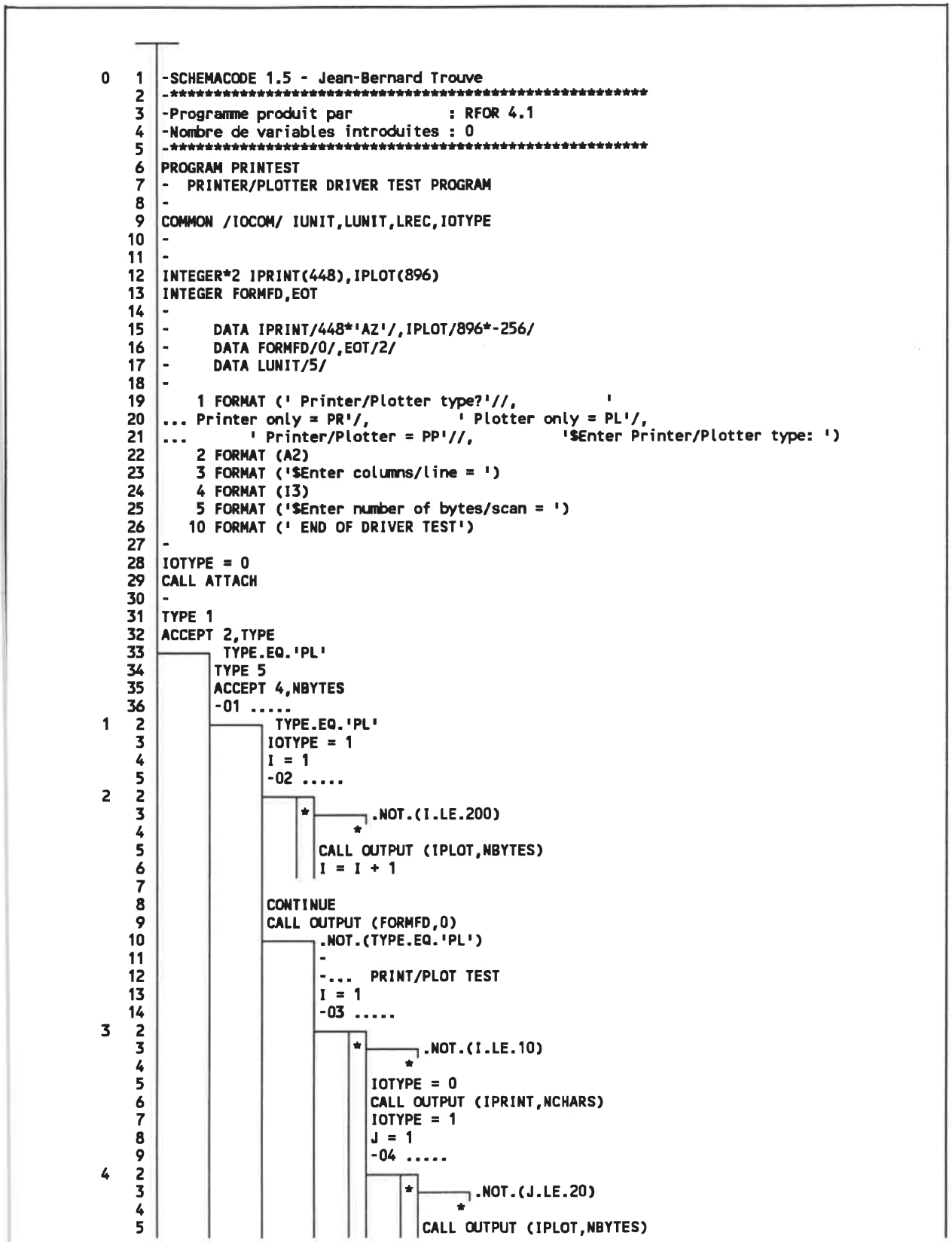
Fig. 6.21 - Statistiques sur la transformation de PRINTEST (sans duplication de code).

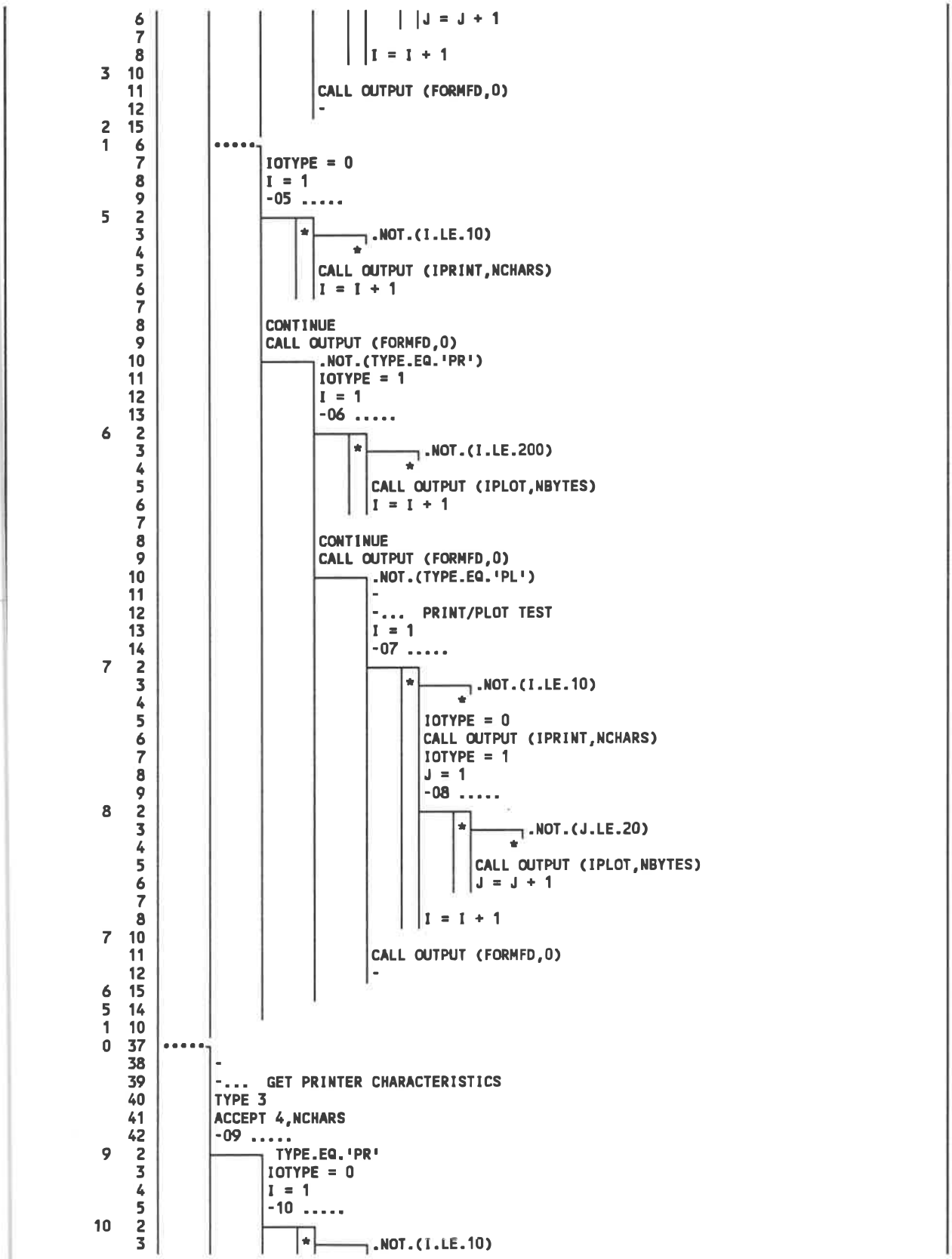
La figure 6.22 illustre les métriques évaluées par DATRIX sur le source produit par SCHEMACODE (sans duplication). Remarquer qu'il n'y a plus de bris de structure mais que le nombre de chemins indépendants et d'énoncés exécutables a augmenté (comparer avec la figure 6.19).

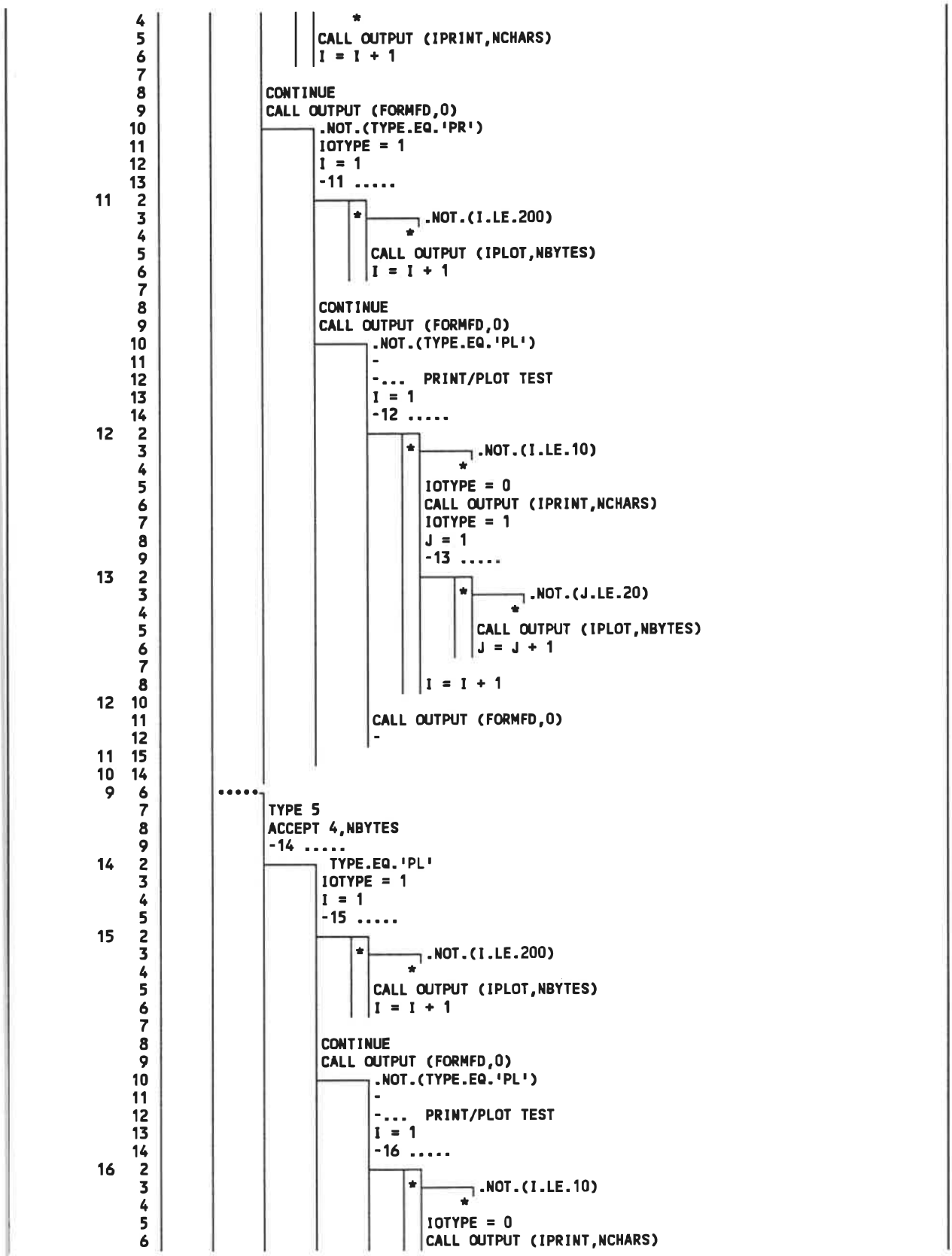
```
Routine:  printest
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)          4
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)         12
Somme des pondérations          (Vw)         47
Vol. de com. des declarations (Vcd)         265
Vol. de com. des structures    (Vcs)         50
Ratio du vol. de commentaires (RVc)         4.2

Nombre de bris de structure     (Bs)          0   <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)           4
Nombre cyclomatique             (Vg)         12
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants (Np)         243   <--

Ratio structurel de boucle      (Rls)      0.32
Ratio pondéré de boucle         (Rlw)      0.28
Niveau d'imbrication moyen      (Nel)       2.5
Nv d'imbrication maximal    (Nelmax)          5
Nv d'imbrication pondéré       (Nelw)       2.1
Nombre de sommets                 (v)        31
Nombre d'arcs                     (e)        41
Nb de sommets Conditionnels     (Ncn)        11
Complexité Moy. des Cond.        (Nc)       1.7
Complexité Max. des Cond.     (Ncmax)          2
Portée moyenne des cond.        (Psc)       3.9
Portée maximum des cond.     (Pscmax)         12
Longueur moy. des noms de var. (Ls)         4.7
Ratio des arcs commentés        (Rac)        21
Ratio des noeuds commentés      (Rnc)         0

Nombre total de lignes          (Nbt)       110   <--
Nombre de lignes exécutables    (Nbe)        74   <--
```

Fig. 6.22 - Métriques évaluées par DATRIX sur le source produit par SCHEMACODE (sans duplication).

La figure 6.23 illustre la procédure PRINTEST transformée en SPC avec duplication de code.


-SCHEMACODE 1.5 - Jean-Bernard Trouve
-Programme produit par : RFOR 4.1
-Nombre de variables introduites : 0
PROGRAM PRINTEST
- PRINTER/PLOTTER DRIVER TEST PROGRAM
COMMON /IOCOM/ IUNIT,LUNIT,LREC,IOTYPE
INTEGER*2 IPRINT(448),IPLOT(896)
INTEGER FORMFD,EOT
-DATA IPRINT/448*'AZ'/,IPLOT/896*-256/
-DATA FORMFD/0/,EOT/2/
-DATA LUNIT/5/
1 FORMAT (' Printer/Plotter type?'//,
... Printer only = PR'/, ' Plotter only = PL'/,
... ' Printer/Plotter = PP'//, '$Enter Printer/Plotter type: ')
2 FORMAT (A2)
3 FORMAT ('$Enter columns/line = ')
4 FORMAT (I3)
5 FORMAT ('$Enter number of bytes/scan = ')
10 FORMAT (' END OF DRIVER TEST')
IOTYPE = 0
CALL ATTACH
TYPE 1
ACCEPT 2,TYPE
TYPE.EQ.'PL'
TYPE 5
ACCEPT 4,NBYTES
-01 .....
TYPE.EQ.'PL'
IOTYPE = 1
I = 1
-02 .....
.NOT.(I.LE.200)
CALL OUTPUT (IPLOT,NBYTES)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PL')
-
-... PRINT/PLOT TEST
I = 1
-03 .....
.NOT.(I.LE.10)
IOTYPE = 0
CALL OUTPUT (IPRINT,NCHARS)
IOTYPE = 1
J = 1
-04 .....
.NOT.(J.LE.20)
CALL OUTPUT (IPLOT,NBYTES)


J = J + 1
I = I + 1
CALL OUTPUT (FORMFD,0)
-
IOTYPE = 0
I = 1
-05 .....
.NOT.(I.LE.10)
CALL OUTPUT (IPRINT,NCHARS)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PR')
IOTYPE = 1
I = 1
-06 .....
.NOT.(I.LE.200)
CALL OUTPUT (IPLOT,NBYTES)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PL')
-
-... PRINT/PLOT TEST
I = 1
-07 .....
.NOT.(I.LE.10)
IOTYPE = 0
CALL OUTPUT (IPRINT,NCHARS)
IOTYPE = 1
J = 1
-08 .....
.NOT.(J.LE.20)
CALL OUTPUT (IPLOT,NBYTES)
J = J + 1
I = I + 1
CALL OUTPUT (FORMFD,0)
-
-
-... GET PRINTER CHARACTERISTICS
TYPE 3
ACCEPT 4,NCHARS
-09 .....
TYPE.EQ.'PR'
IOTYPE = 0
I = 1
-10 .....
.NOT.(I.LE.10)


*
CALL OUTPUT (IPRINT,NCHARS)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PR')
IOTYPE = 1
I = 1
-11 .....
.NOT.(I.LE.200)
CALL OUTPUT (IPLOT,NBYTES)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PL')
-... PRINT/PLOT TEST
I = 1
-12 .....
.NOT.(I.LE.10)
IOTYPE = 0
CALL OUTPUT (IPRINT,NCHARS)
IOTYPE = 1
J = 1
-13 .....
.NOT.(J.LE.20)
CALL OUTPUT (IPLOT,NBYTES)
J = J + 1
I = I + 1
CALL OUTPUT (FORMFD,0)
-
.....
TYPE 5
ACCEPT 4,NBYTES
-14 .....
TYPE.EQ.'PL'
IOTYPE = 1
I = 1
-15 .....
.NOT.(I.LE.200)
CALL OUTPUT (IPLOT,NBYTES)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PL')
-... PRINT/PLOT TEST
I = 1
-16 .....
.NOT.(I.LE.10)
IOTYPE = 0
CALL OUTPUT (IPRINT,NCHARS)

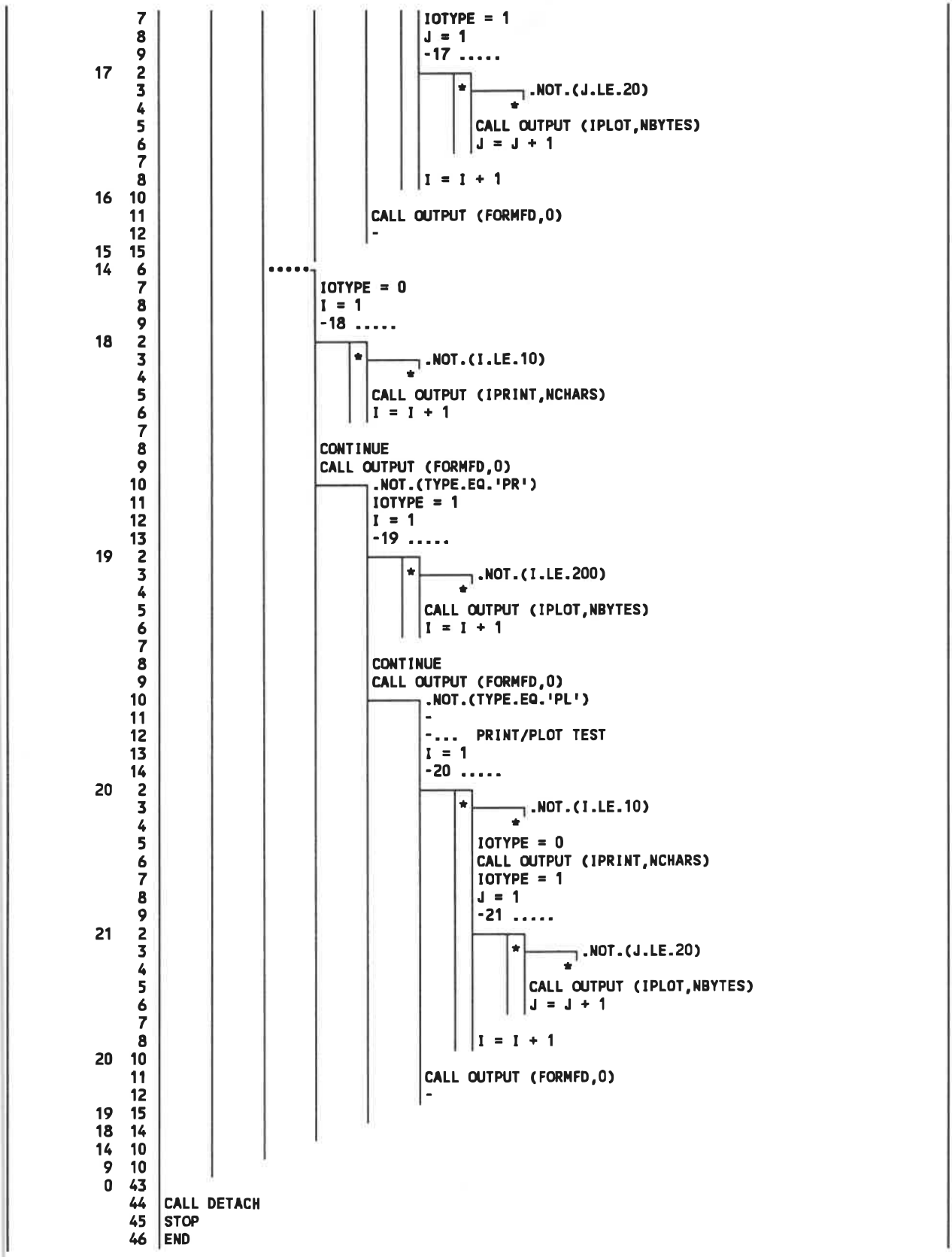

IOTYPE = 1
J = 1
-17 .....
.NOT.(J.LE.20)
CALL OUTPUT (IPLOT,NBYTES)
J = J + 1
I = I + 1
CALL OUTPUT (FORMFD,0)
-
.....
IOTYPE = 0
I = 1
-18 .....
.NOT.(I.LE.10)
CALL OUTPUT (IPRINT,NCHARS)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PR')
IOTYPE = 1
I = 1
-19 .....
.NOT.(I.LE.200)
CALL OUTPUT (IPLOT,NBYTES)
I = I + 1
CONTINUE
CALL OUTPUT (FORMFD,0)
.NOT.(TYPE.EQ.'PL')
-
-... PRINT/PLOT TEST
I = 1
-20 .....
.NOT.(I.LE.10)
IOTYPE = 0
CALL OUTPUT (IPRINT,NCHARS)
IOTYPE = 1
J = 1
-21 .....
.NOT.(J.LE.20)
CALL OUTPUT (IPLOT,NBYTES)
J = J + 1
I = I + 1
CALL OUTPUT (FORMFD,0)
-
CALL DETACH
STOP
END

Fig. 6.23 - Procédure PRINTEST transformée en SPC avec duplication.

La figure 6.24 illustre les statistiques sur la transformation de PRINTEST avec duplication de code. Remarquer le nombre important de lignes dupliquées (177).

```
RFOR 4.1

Duplication de code toujours utilisee si possible.

Fichier                          :  PRINTEST
Structuration reussie            :  OUI

GRAPHE

Nombre d'arcs original           :   29
Nombre d'arcs final              :    1

STANDARDISATION (avant restructuration)

Sommets proceduraux              :   15
Arcs paralleles                  :    1
Arcs reflexifs                   :    4
Boucles generalisees             :    0
                                   ----
Total                            :   20

MESURES

% arcs elimines avant restruct.  :   57 %   <----
% op. standard. avant restruct.  :   68 %

STANDARDISATION (total)

Sommets proceduraux              :   20
Arcs paralleles                  :    5
Arcs reflexifs                   :    4
Boucles generalisees             :    0
                                   ----
Total                            :   29

RESTRUCTURATION

Transformations ID-0             :    3
Transformations ID-1             :    0
Transformations OD               :    0
Transformations IL-0             :    0
Transformations IL-1             :    0
                                   ----
Total                            :    3   <----

Nombre de variables creees       :    0   <----
Nombre de lignes dupliquees      :  177   <----
```

Fig. 6.24 - Statistiques sur la transformation de PRINTEST (avec duplication de code).

La figure 6.25 illustre les métriques évaluées par DATRIX sur le source produit par SCHEMACODE (avec duplication). Remarquer que les bris de structure sont éliminés mais que le nombre d'énoncés a augmenté de façon dramatique (comparer avec la figure 6.22).

```
Routine:  printest
                                DATRIX 1.5.4(β)
Tableau récapitulatif           Projet:     ETC


Nom de la métrique                          Valeur

Nombre de sommets d'entrée      (Ni)          1
Nombre de sommets de sortie     (Ne)          1
Nombre de boucles               (Nl)         18
Nombre de sommets récursifs     (Nr)          0
Volume structurel               (Vs)         33
Somme des pondérations          (Vw)        106
Vol. de com. des declarations (Vcd)         265
Vol. de com. des structures    (Vcs)        132
Ratio du vol. de commentaires (RVc)           4

Nombre de bris de structure     (Bs)          0   <--

Nb de bris de str. pondéré      (Bsp)         0
Nombre de croisements des arcs  (K)          22
Nombre cyclomatique             (Vg)         31
Nombre de sommets pendants      (Vp)          0

Nombre de chemins indépendants (Np)          70   <--

Ratio structurel de boucle      (Rls)      0.54
Ratio pondéré de boucle         (Rlw)      0.58
Niveau d'imbrication moyen      (Nel)       4.6
Nv d'imbrication maximal    (Nelmax)          8
Nv d'imbrication pondéré       (Nelw)       4.4
Nombre de sommets                (v)        86
Nombre d'arcs                    (e)       115
Nb de sommets Conditionnels     (Ncn)        30
Complexité Moy. des Cond.        (Nc)       1.9
Complexité Max. des Cond.     (Ncmax)         2
Portée moyenne des cond.        (Psc)       6.5
Portée maximum des cond.     (Pscmax)        31
Longueur moy. des noms de var. (Ls)          4.6
Ratio des arcs commentés        (Rac)        15
Ratio des noeuds commentés      (Rnc)         0

Nombre total de lignes          (Nbt)       243   <--
Nombre de lignes exécutables    (Nbe)       180   <--
```

Fig. 6.25 - Métriques évaluées par DATRIX sur le source produit par SCHEMACODE (avec duplication).

6.2 Discussion des résultats

En se basant sur les résultats de la section précédente, nous tirons les quelques conclusions suivantes.

La méthode est fonctionnelle

Bien que ceci ne constitue pas une preuve mathématique du fonctionnement de la méthode implantée (qui déborderait de toute façon du cadre de ce travail), nous n'avons rencontré à ce jour aucun programme FORTRAN remplissant les conditions spécifiées qui ne puisse être transformé en SPC. Même des programmes de plusieurs centaines de lignes, qui avaient déjà été traduits "manuellement" après plusieurs dizaines d'heures (et d'erreurs), ont été convertis avec succès en quelques minutes, et ce automatiquement.

Le programme résultant est plus lisible

Bien que la lisibilité soit un critère subjectif, on peut dire que la représentation SPC donne au programme un accès immédiat à plus d'information.

Plusieurs exemples peuvent être donnés, mais les procédures BOUND et LOOKUP sont des cas patents d'amélioration de lisibilité.

Les structures similaires au SPC sont bien reconnues

Si la structure originale du programme se rapproche intimement de celles définies dans le SPC, elle est bien identifiée et se retrouve également dans le programme résultant. L'information contenue dans la structure originale est, dans ce cas conservée entièrement. Conséquement, un programmeur qui utilise le langage FORTRAN selon les concepts modernes de programmation n'aura pas de difficulté à visualiser son programme transformé.

Effets de l'introduction de variables de contrôle

Le cas de la procédure LOOKUP montre bien les inconvénients de l'introduction de variables de contrôle dans un programme. La complexité de la logique est augmentée à cause du plus grand nombre de sommets conditionnels.

Il appert cependant que ces variables ont quelquefois une signification, comme si le programmeur, au lieu de les créer, avait préféré «dé-structurer» son programme. On peut considérer, par exemple, le programme PRINTEST où la variable VAR_2 est vraie dans le cas d'une imprimante (TYPE.EQ.'PR') et VAR_1 est vraie dans le cas d'un traceur (TYPE.EQ.'PL'). Les variables introduisent donc parfois une certaine redondance de l'information servant au contrôle.

Effet de la duplication de code

La duplication de code entraîne bien évidemment une augmentation du volume du programme. Tout dépendant du nombre de ligne dupliquées, les résultats sont plus ou moins souhaitables. Dans le cas du programme PRINTEST, il est clair que cette alternative n'est pas recommandable.

Il arrive toutefois qu'elle soit intéressante lorsque le nombre de lignes dupliquées reste faible, comme dans le cas de la procédure LOOKUP. Il n'y a alors pas d'augmentation de la complexité de la logique et le programme résultant reste reconnaissable.

Il existe un compromis duplication de code / création de variables

Ce compromis est difficile à évaluer avant la restructuration et dépend habituellement du programme à traiter. La solution consiste à ajuster le paramètre de duplication et à procédeer par itérations successives jusqu'à ce qu'un résultat acceptable soit obtenu. Les temps d'exécution réduits permettent d'employer cette stratégie.

La transformation ne simplifie pas l'algorithme

Bien que la visualisation du programme soit améliorée, sa logique n'est en aucun cas simplifiée. Au mieux, elle est gardée intacte si aucune variable de contrôle n'est introduite. Le système de restructuration ne peut corriger un design mal réalisé. Cette opinion est partagée par plusieurs [49] [27].

Considérons encore une fois le programme PRINTEST. Si ce programme avait été écrit au départ en pseudocode schématique, le résultat aurait probablement ressemblé au programme de la figure 6.26.

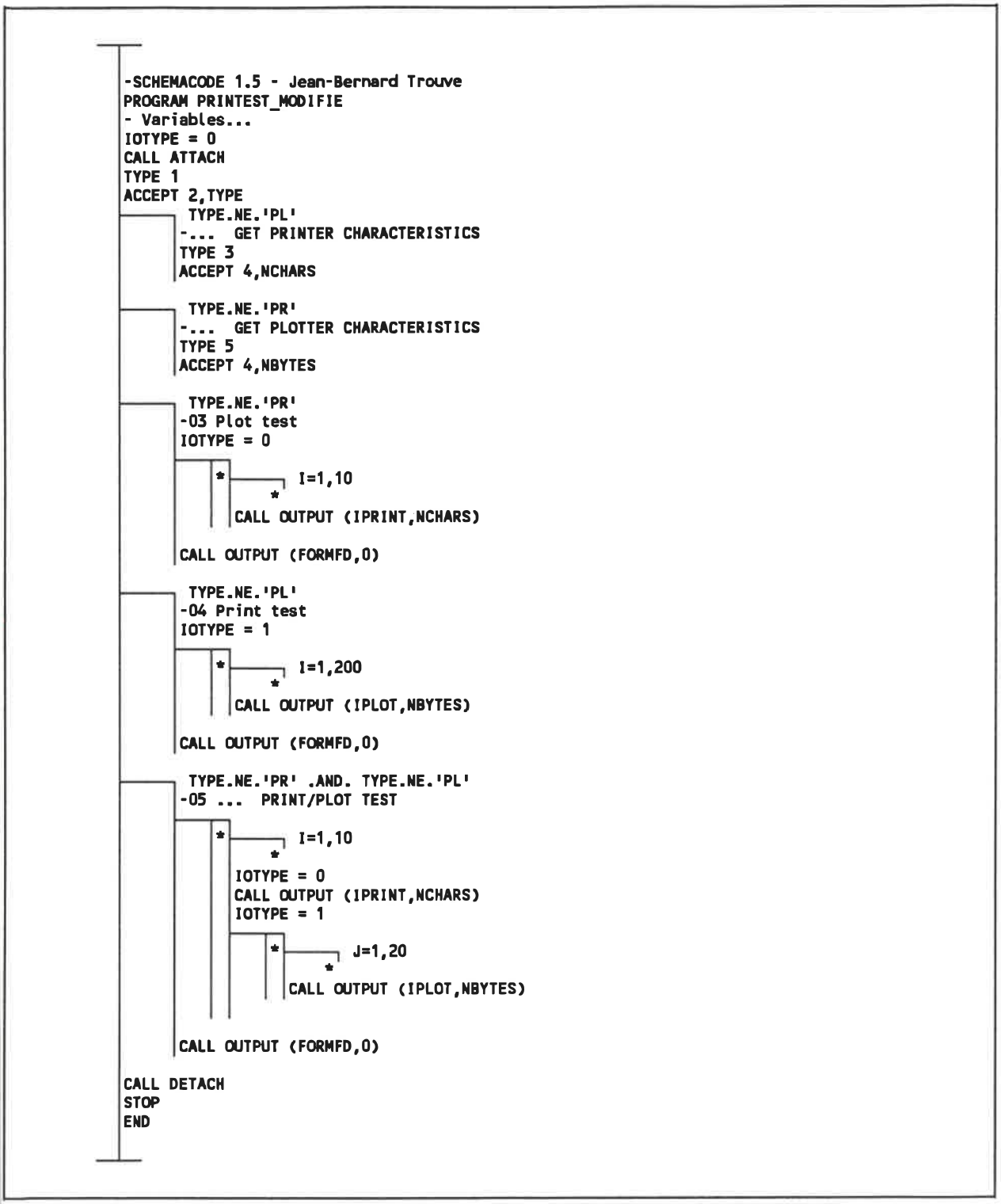


Fig. 6.26 - Autre version du programme PRINTEST en SPC

Ce qui est, d'après nous plus compréhensible que le programme original et que celui produit par notre système.

La documentation textuelle n'est pas meilleure, mais...

Comme nous l'avons expliqué, le système crée des commentaires opérationnels dans le programme résultant mais ne peut y adjoindre de texte significatif. Ceci relève d'un niveau de compréhension du programme qui dépasse totalement la simple analyse du flux de contrôle.

Par contre, la position de ces commentaires indique les endroits où une documentation textuelle est approriée. Ceci n'est toutefois aucunement limitatif et l'addition de commentaires opérationnels supplémentaires est sûrement souhaitable.

Notons qu'une de nos tentaives à ce sujet fut de retenir le commentaire le plus rapproché dans le programme original comme texte d'un commentaire opérationnel produit automatiquement. Les résultats furent toutefois rarement intéressants: le texte n'était pas souvent l'expression d'une abstraction et ne concernait pas toujours les énoncés contenus dans le raffinement relié au commentaire opérationnel.

Le système peut être amélioré

Le prototype réalisé est loin d'être idéal et nous en sommes conscients. Voici une courte liste des améliorations suggérées:

- dans les graphes produits par DATRIX, ajouter les conditions d'exécution d'arcs et les coordonnées précises de localistion du texte des arcs dans le programme source (en incluant les commentaires); créer également des graphes plus détaillés en évitant l'élimination de sommets (ceci permettrait entre autre l'analyse des IF THEN et une meilleure analyse des boucles DO);

- modifier CRT2ACG pour tenir compte de ces modifications et créer directement un fichier ACG complet (fusionner CRT2ACG et FILLACG); éliminer GBOOL;
- créer des branches SINON lors de l'élimination d'arcs parallèles dans RFOR;
- augmenter le nombre de possibilités d'améliorations du SPC dans MAKEUP;
- chercher des algorithmes pour créer des commentaires opérationnels significatifs;
- créer automatiquement des sous-programmes lorsque le nombre de lignes dupliquées devient trop important.

CONCLUSION

All is well that ends.

L'objectif initial qui, rappelons-le, était d'élaborer un mécanisme de traduction du FORTRAN au pseudocode schématique (SPC) puis d'automatiser ce mécanisme a été atteint. Le système réalisé permet la production du pseudocode schématique de telle façon qu'il puisse être entretenu à l'aide d'outils adaptés à cette représentation.

Basé sur une analyse complète du flux de contrôle plutôt que sur la simple translittération, notre système de traduction produit rapidement des programmes SPC à partir de programmes FORTRAN 77, que ces derniers soient structurés ou non. Lorsque des transformations de restructuration sont nécessaires, des paramètres permettent de contrôler le degré de duplication de code ainsi que l'introduction de variable de contrôle. L'algorithme utilisé est toujours convergent et assure la production d'un résultat.

Le prototype comprend un module d'analyse qui convertit le programme source en graphe étiqueté, un module de restructuration qui transforme le graphe et un module de synthèse qui convertit le graphe en SPC. Seul le module d'analyse dépend du langage de programmation source (FORTRAN).

Les performances du prototype au niveau rapidité d'exécution permettent l'exploration itérative de plusieurs alternatives de traduction lorsqu'une restructuration est nécessaire. Ces alternatives représentent différents compromis entre la duplication de code et l'utilisation de variables de contrôle. Les résultats montrent que ces compromis sont différents d'un programme à l'autre.

Les études effectuées lors de ce travail montrent toutefois que la méthode possède des limites et ne remplace pas toujours avantageusement la refonte d'un programme. Elle permet cependant d'isoler facilement les parties les plus susceptibles d'être récupérées telles quelles. Mais elle ne saurait en aucun cas améliorer un programme mal conçu; elle permet seulement de mieux le visualiser.

Le prototype réalisé permet, grâce à ses performances, d'évaluer rapidement et à coût très réduit le degré de structuration d'un programme de grandes dimensions (plusieurs dizaines de milliers de lignes de code source). Il permet également de le transformer dans une représentation plus abstraite et de le maintenir plus facilement, à coût réduit. Cette représentation, le pseudocode schématique, possède effectivement l'avantage de faciliter la communication en mettant en relief la logique des programmes et d'en organiser la documentation systématiquement.

Suite à ce travail, nous désirons émettre quelques recommandations concernant différentes avenues connexes et ouvertes à l'exploration.

Nouvelles métriques

Il y a tout d'abord l'aspect des métriques de structure, dont DATRIX ne fournit que deux exemplaires: Bs, qui est le nombre de bris de structure, et Bsp, qui est la somme des poids des arcs participant aux bris de structure. Ces deux définitions ont le désavantage d'être dépendantes de l'algorithme de traçage du graphe.

Certaines mesures sont évaluées par notre prototype. Celles-ci comprennent le nombre d'opérations de restructuration effectuées pour produire le SPC, le nombre de variables de contrôle créées ainsi que le nombre de lignes de code dupliquées. Ces mesures sont une quantification de la non-structure puisqu'elles évaluent le travail à effectuer pour restructurer un programme (elles valent zéro si le programme est structuré).

Une autre mesure vise à évaluer l'impact ou l'aspect de localité des bris de structure: le pourcentage d'arcs éliminés avant restructuration. Ceci reflète la quantité de structures simplifiables lors de la mise en forme standard avant qu'une restructuration ne soit nécessaire et cherche à mesurer ce qui est structuré plutôt que ce qui ne l'est pas. Ce pourcentage est égal à 100 dans le cas de programmes structurés.

Finalement, d'autres métriques de structure sont évaluées: le nombre total d'arcs parallèles simplifiés donne le nombre de structures conditionnelles et la somme des arcs réflexifs et boucles généralisées donne le nombre de structures répétitives.

Nouveaux langages sources

Parce que seul le module d'analyse est dépendant du langage, les modules de restructuration et de synthèse du SPC n'ont pas besoin d'être modifiés pour adapter le système à un autre langage source que le FORTRAN. Les algorithmes de mise en forme standard et de restructuration, qui forment le coeur du système, sont en effet basés sur des graphes et manipulent le code comme de simples chaînes de caractères. L'adaptation de la méthode à un autre langage source est donc relativement facile car elle a été prévue dans le design du système. Le domaine d'application des algorithmes reste conséquemment très général.

Commentaires opérationnels

L'aspect de la documentation textuelle, et en particulier la production de commentaires opérationnels significatifs, n'a pas été abordé dans ce travail. Ceci requiert une analyse sémantique qui est autrement plus complexe que l'analyse de flux de contrôle que notre système effectue. Il serait intéressant d'évaluer s'il est possible de récupérer certains commentaires du programme original afin d'en faire des commentaires opérationnels.

Même s'ils sont dépourvus de texte, les commentaires opérationnels produits indiquent néanmoins où certaines abstractions devraient être documentées.

Forme normalisée de représentation des programmes

Dans une perspective plus large de la maintenance, on pourrait considérer le pseudocode schématique comme étant une forme normalisée de représentation des programmes, la schématisation de la logique étant sûrement un avantage. Un système réalisant automatiquement la traduction d'un programme source quelconque en SPC permet en tout temps l'accès à une documentation uniforme et significative.

De plus, si le programmeur le désire, il dispose à ce moment d'outils plus puisssants pour l'assister dans le processus de maintenance. Ces outils lui permettent la manipulation directe du SPC en profitant de tous les avantages de cette représentation, y compris la possibilité de documenter ses abstractions grâce aux commentaires opérationnels. Notons que même si cela est possible, l'humain devra quand même valider le résultat.

Cette normalisation de représentation des programmes offre des avantages certains au niveau de la maintenance: moins de temps perdu en lecture et

compréhension des programmes, meilleure documentation, systématisation du processus de maintenance et réduction des coûts.

Dans ce contexte, nous croyons que le travail réalisé constitue une étape en vue de faciliter la maintenance du logiciel.

BIBLIOGRAPHIE

[1] AHO, A., KERNIGHAN, B. & WEINBERGER, P., "The AWK Programming language.". Addison-Wesley, Reading, Mass. (1988).

[2] AHO, A., SETHI, R., & ULLMAN, J, "Compilers: Principles, Techniques and Tools". Addison-Wesley, Reading, Mass. (1986).

[3] ANONYMOUS, "Clipper User Guide". Marina Information Systems, Marina Del Rey, California (1985).

[4] ANONYMOUS, "dBASE III User Manual". Ashton-Tate, Culver City, California (1984).

[5] ANONYMOUS, "Microsoft FORTRAN". Microsoft, Redmond, Washington (1985).

[6] ANONYME, "SCHEMACODE: manuel de l'usager". Schémacode Int'l, Montréal (1986).

[7] ANONYMOUS, "VAX-11 FORTRAN User's Guide". Maynard, Mass. (1983).

[8] BAINBRIDGE, E.S., "Minimal while programs" in "Lecture Notes in Computer Science". Springer-Verlag, Berlin (1976).

[9] BAKER, B., "An Algorithm for Structuring Flowgraphs". Journal of the ACM, pp. 98-120 (January 1977).

[10] BEAUCAGE, A. & ROBILLARD, P.N., "Conception et développement d'un logiciel d'évaluation de code source: prototype LECS". Centre de développement technologique, Montréal (1986).

[11] BERGE, C., "Graphes et hypergraphes". Dunod, Paris (1973).

[12] BOEHM, C. & JACOPINI, G., "Flow diagrams, Turing machines and languages with only two formation rules". Communications of the ACM, pp. 366-371 (September 1966).

[13] BRUNO, J. & STEIGLITZ, K., "The expression of algorithms by charts". Journal of the ACM, pp. 517-525 (October 1972).

[14] COALLIER, F., "La caractérisation de la structure des programmes sources". Mémoire de maîtrise, École Polytechnique (1987).

[15] DIJKSTRA, E., "1972 ACM Turing Award Lecture". ACM Press, New York (1987).

[16] DUNCAN, R. "Advanced MS DOS Programming". Microsoft Press, Redmond, Washington (1988).

[17] GRENIER, A., "Méthode de construction de traducteurs de pseudocode schématique" Mémoire de maîtrise, École Polytechnique (1989).

[18] KATZAN, H., "FORTRAN 77". Van Nostrand Reinhold, London (1978).

[19] KERNIGHAN, B. & PLAUGER, P., "The Elements of Programming Style". McGraw-Hill, New York (1974).

[20] LABELLE, J., "Théorie des graphes". Modulo Editeur, Mont-Royal, Qc (1981).

[21] LESK, M., "Lex - a lexical analyzer generator". AT&T Bell Laboratories, Murray Hill, N.J. (1975).

[22] LINGER, R., MILLS, H. & WITT, B., "Structured Programming: Theory and Practice". Addison-Wesley, Reading, Mass. (1979).

[23] LIPSCHUTZ, S., "Theory and Problems of Data Structures". McGraw-Hill, New York (1986).

[24] LYONS, M., "Salvaging your software asset (tools based maintenance)". Proceedings of the National Computer Conference, Arlington, Virginia, pp.337-341 (1981).

[25] McCABE, T.J., "A complexity measure". IEEE Transactions on Software Engineering, pp. 79-90 (March 1979).

[26] OULSNAM, G., "The Algorithmic Transformation of Schemas to Structured Form". The Computer Journal, Vol. 30, No 1, pp. 43-51 (1987).

[27] OULSNAM, G., "Unravelling Unstructured Programs". The Computer Journal, Vol. 25, No 3, pp. 379-387 (1982).

[28] PARTSCH, H. & STEINBRUGGEN, R., "Program Transformation Systems". Computing Surveys, pp. 199-236 (September 1983).

[29] ROBILLARD, P.N., "A Software Tool and a Schematic Notation that improve the use of programming languages". IEEE Proceedings of SOFTFAIR CONFERENCE II, San Francisco (December 1985).

[30] ROBILLARD, P.N., "L'entretien du logiciel: étude de cas". Rapport Technique EPM/RT-84-7, École Polytechnique de Montréal, Montréal (1984).

[31] ROBILLARD, P.N., "Le logiciel: de sa conception à sa maintenance". Gaëtan Morin, Chicoutimi (1985).

[32] ROBILLARD, P.N., "Schematic Pseudocode for Program Constructs and its computer automation by SCHEMACODE". Communications of the ACM, pp.1072-1089 (November 1986).

[33] ROBILLARD, P.N., "Schematic Construct Notation for Programming and its Automation by SCHEMACODE", Technical Report EPM/RT-84-13, École Polytechnique de Montréal (1984).

[34] ROBILLARD, P.N. & COUPAL, D., "Data on the Automation on Program's Comments". 6th Annual Pacific Northwest Conference, Portland, Oregon, pp.405-417 (September 19-20, 1988)

[35] ROBILLARD, P.N. & LEBLANC, D., "The Simulated Working Environment in a Project-based Software Engineering Course". Computers & Education, pp. 471-477, Vol. 12, No 4 (1988).

[36] ROBILLARD, P.N. & PLAMONDON, R., "Schemacode: an Interactive Schematic Pseudocode for Program Development, Documentation and Structured Coding". Proceedings of the NBS/IEEE/ACM Software Tool Fair, San Diego, California (1981).

[37] ROBILLARD, P.N., TROUVÉ, J.-B. & GRENIER, A., "A pre-processor for schematic pseudocode". IEEE Proceedings of the Second International Conference on Computers and Applications, Beijing, China (August 1987).

[38] ROBILLARD, P.N., TROUVÉ, J.-B. & GRENIER, A., "L'automatisation du raffinement successif et de sa documentation". Journées Internationales sur le Génie Logiciel et ses Applications, Toulouse, France (Décembre 1988).

[39] SAMSON, J., "Réalisation d'un convertisseur de fichier source 'C' en pseudocode schématique (RETROC)". Rapport de projet de fin d'études, École Polytechnique (1988).

[40] SCHNEIDEWIND, N., "The state of Software Maintenance". IEEE Transactions on Software Engineering, pp. 303-310 (March 1987).

[41] ST-DENIS, R. & ROBILLARD, P.N., "Une approche pour le développement et la maintenance de systèmes informatiques complexes". Rapport Technique EPM/RT-86/31, École Polytechnique de Montréal, Montréal (Août 1986).

[42] ULLMAN, J., "Principles of Database Systems". Computer Science Press, Rockville, Maryland (1982).

[43] WAITE, W. & GOOS, G., "Compiler Construction". Springer-Verlag, Berlin, (1984).

[44] WATERS, R.C., "Program Translation via Abstraction and Reimplementation". IEEE Transactions on Software Engineering, pp. 1207-1228 (August 1988).

[45] WHITTY, R.W., FENTON, N.E., & KAPOSI, A.A., "A generalised mathematical theory of structured programming". Theoretical Computer Science 36, pp. 145-171 (1985).

[46] WHITTY, R.W., FENTON, N.E., & KAPOSI, A.A., "A rigorous approach to structural analysis and metrication of software". Software & Microsystems, pp. 2-16 (February 1984).

[47] WHITTY, R.W., FENTON, N.E., & KAPOSI, A.A., "Structured programming: a tutorial guide". Software & Microsystems, pp. 54-65 (June 1984).

[48] WILLIAMS, M.H., "Generating structured flow diagrams: the nature of unstructuredness". The Computer Journal, pp.45-50 (February 1977).

[49] YOURDON, E., "Techniques of Program Structure and Design". Prentice-Hall, Englewood Cliffs, N.J. (1975).

[50] ZARRELLA, J., "Language Translators". Microcomputers Applications, Suisun City, California (1982).